A Dissertation Report on

Development and Implementation of Feedback controller and Obstacle Detection Algorithm for a Nonholonomic Car type Robot

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology
(Electronics Engineering)

Submitted By:

Akshay Adinath Dhotre
(2014MEN006)

Under Guidance of:

Dr. Mrs. S. S. Deshpande

Department of Electronics Engineering

MTE Society's
WALCHAND COLLEGE OF ENGINEERING, SANGLI.

2015 - 2016

CERTIFICATE

This is to certify that, the dissertation report entitled

"Development and Implementation of Feedback controller and Obstacle Detection Algorithm for a Nonholonomic Car type Robot"

Submitted by

Akshay Adinath Dhotre

(2014MEN006)

In the partial fulfillment for the requirement for the degree of

Master of Technology (Electronics Engineering)

is a record of students own work carried out under my supervision during the academic year 2015-2016.

Date:

Dr. Mrs. S. S. Deshpande
(Guide)

The candidate has completed all the required phases of evaluation and performed satisfactorily. Hence, recommended for the partial fulfillment of the requirement for the said degree.

Dr. V. B. Dharmadhikari (Chairman, DPGC)	(Member 1, DPGC)	(Member 2, DPGC)

(External Examiner)	(Dean Academics)

Pune Vidyarthi Griha's
COLLEGE OF ENGINEERING & TECHNOLOGY
(Under DST WOS-A Project Sanction No : 100/(IFD)/3732/2013-14)
S. NO. 44, Vidyanagari, Parvati, Pune-411009. Website: www.pvgcoet.ac.in

Certificate

This is to certify that the work carried out as part of this project was sponsored by Department of Science and Technology (DST), Government of India under the WOS-A grant (Sanction No :100/(IFD)/3732/2013-14 dated 29.08.2013) for project entitled: Cars that Drive and Park (CTDP)

The report is submitted as a partial fulfillment of the requirement of the Post-Graduate degree course in Electronics Engineering, Walchand College of Engineering, Sangli, during the academic year 2015–2016.

The project is carried out by Akshay Adinath Dhotre.

Prof. Dr. Mrs. V. A. Joshi
Mentor (WOS-A)
Professor, Dept. of Electrical Engg.
PVGs COET, Pune-9

Dr. Rahee A. Walambe
Principal Investigator, WOS-A
Dept. of Electrical Engg.
PVGs COET, Pune-9

Abstract

This dissertation work is related to hardware setup to implement feedback control and obstacle detection for car type robot. Car type robots can be auto driving autonomous robots. Auto-navigation is an essential feature in development of such robots. The motion planner developed in provides the reference states. The feedback algorithm which is currently under development uses the physical sensory data and compares it with these reference values to find and correct the actual states of the vehicle. The feedback algorithm is being developed to track the motion planner. The implementation is based on Raspberry Pi (RPi) board. This board reads physical data from sensors and processes it, focus in this project is the development of the hardware setup for implementation of the feedback control and obstacle detection algorithms.

Acknowledgements

This project is funded under the WOS-A scheme (SR/ET/WOS-34/2013-14) of Dept of Science and Technology, Govt of India.

I would like to express my gratitude towards my project guide Dr. Mrs. S. S. Deshpande for her constant guidance during my project. I would like to thank Dr. V. A. Joshi and Dr. Rahee Walambe (PVGCOE, Pune.) for allowing to work on this project, and helping me whenever I need it.

I would also like to thank our H.O.D. Dr. V. B. Dharmadhikari for his continuous encouragement. I take this opportunity to express my sincere thanks to all the staff members of Electronics Engineering Department for their help whenever required and for making its facilities available to me.

Finally I want to thank my colleagues Manoj Jagdamwar, Nitish Shete, Vaibhav Katkar, Manisha Sirsat, Pallavi Kumbhar, Amit Magdum and all those who helped me directly or indirectly in this project.

Akshay Adinath Dhotre
M.Tech (Electronics)

Contents

List of Figures

List of Abbreviations

RPi	Raspberry Pi
LIDAR	Light Detection and Ranging
CPU	Central Processing Unit
GB	Gega Bytes
RAM	Random Access Memory
HDMI	High-Definition Multimedia Interface
GPIO	General Purpose Input Output
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
PWM	Pulse Width Modulation
PPR	Points Per Revolution
ADC	Analog to Digital Converter
SSH	Secure Shell

Chapter 1

Introduction

The vision of self-driving and parking cars promises to bring fundamental change to one of the most essential aspects of our daily lives. Although automatic gears are common in cars, complete auto-drive car is still not a reality. Autonomous driving involves high machine intelligence and decision making capabilities for the vehicle.

For autonomous car robot auto navigation is essential, hence feedback controller and obstacle detection will help in this area and car movements will be more accurate. Feedback controller designed for car will ensure accurate path planning and motion planning. So, overall stability and accuracy of path trajectory will improve. Detection of static or moving obstacle is essential to ensure safety. Obstacle detection will prevent car from damaging others and getting damaged. This will make car safer in dense populated areas.

To implement both these functionalities in practical scenario, a powerful and intelligent hardware platform is essential. There are various single board computers are available which can be used for this purpose.

1.1 Significance

- For autonomous car robot auto navigation is must, hence feedback controller and obstacle detection will help in this area and car movements will be more accurate.
- Feedback controller designed for car will ensure accurate path planning and motion planning. So, overall stability and accuracy of path trajectory will improve.
- Obstacle detection will prevent car from damaging others and getting damaged. This will make car safer in dense populated areas.
- The same principle of auto-navigation can be used for industrial and military robots like auto fork lift trucks, intelligent drones etc.

For implementation of these features a strong hardware platform is necessary.

1.2 National and International Status

Although various automobile manufacturers are currently working on the auto-drive and auto park features, the solution is not yet readily available. Especially in Indian Market, no Car manufacturer has yet developed such kind of technology to be incorporated. From commercial point of view, the solution offered by International car manufacturers (Toyota, Lexus, and Audi) is expensive and does not involve complete autonomy.

1.3 Nonholonomic Systems:

A nonholonomic system is a system whose state depends on the path taken in order to achieve it. Such a system is described by a set of parameters subject to differential constraints, such that when the system evolves along a path in its parameter space (the parameters varying continuously in values) but finally returns to the original set of values at the start of the path, the system itself may not have returned to its original state.

In these systems there is one in which there is a continuous closed circuit of the governing parameters, by which the system may be transformed from any given state to any other state. So, feedback control is necessary to get desired operation in these systems. Hence, design of feedback controller for nonholonomic systems is a critical task.

1.4 Literature Review

Although various automobile manufacturers are currently working on the auto-drive and auto park features, the solution is not yet readily available. Especially in Indian Market, no Car manufacturer has yet developed such kind of technology to be incorporated. From commercial point of view, the solution offered by International car manufacturers (Toyota, Lexus, and Audi) is expensive and does not involve complete autonomy.

Review of other literature survey is as follows:

Mohammad Abdul Qayum and his collages [**1**] develop a unique approach for the integration of intelligent system control into transportation systems. Authors have implemented an algorithm that controls a car to track a predefined track which integrates a human driver with computer control to increase human performance while reducing reliance on detailed driver attention. Knowledge obtained from the optical tracking system about vehicle position and orientation provides the automatic decision making intelligence needed to follow a virtual vehicle moving on track.

Adeel Akhtar and his collages [**2**] presents an approach for designing path following controllers for the kinematic model of car-like mobile robots using transverse feedback linearization with dynamic extension. This approach is applicable to a large class of paths. According to authors transverse feedback linearization makes the desired path attractive and invariant, while the dynamic extension allows the closed-loop system to achieve the desired motion along the path.

Kichun Jo and Junsoo Kim [**3**] applied the distributed system architecture to the autonomous driving system, and proposes a development process and a system platform for the distributed system of an autonomous car. A time-triggered network protocol, FlexRay, is applied by authors as the main network of the software platform to improve the network bandwidth, fault tolerance, and system performance.

Jian-Min Wang; Sen-Tung Wu; Chao-Wei Ke; Bo-Kai Tzeng [**4**] provides a simple parking path programming strategy for automatic parking system (APS). The control strategy employs the minimum turning radius of the vehicle by means of a distance infrared sensor to determine the parking path. According to authors the distance between the vehicle and boundary of parking space be detected, referring to the minimum rotating radius. In addition, auto parking behavior will be accomplished without any control of extra rotating angle.

Kobayashi, T. Ozaki, [**5**] presented a method for design of feedback controller for nonholonomic systems. Authors introduce an approximation technique for the nonholonomic problems and construct a feedback controller based on the standard solution of the Hamilton-Jacobi-Bellman (HJB) equation. Some other methods are also present will be considered.

From above literature we can say that feedback controller is needed for autonomous robots for proper operation and present theory has implemented some methods of designing feedback controller. Also, for obstacle detection LIDAR based solution will be simple and easy for implementation.

For actual implementation of feedback control and obstacle detection some papers are referred. In [**6**], authors have presented the Differential Flatness Based Nonholonomic Motion Planner for a Car type Mobile Robot. In [**7**] authors have shown how the spline based optimization can be incorporated in this motion planner to avoid the singularities and generate the shortest path which also satisfy the nonholonomic and curvature constraints. However, the development discussed in both [**6**] and [**7**] uses the open loop control and hence there are certain issues in generating the exact trajectories on the hardware platform. These issues are highlighted in [**7**]. The way forward is to implement the feedback control algorithm which compares the expected states from motion planner with the actual states obtained from the sensors. The error is then processed using the nonlinear feedback controller such as [**9**] and [**6**]. Corrected control inputs are generated which will force the car to follow the path generated by the motion planner accurately.

This dissertation work is aimed towards hardware setup development for implementing the two essential algorithms viz; Feedback Control and Obstacle Detection. The feedback control requires the accurate knowledge of the states of the car (i.e. position in plane and orientation angle). These can be obtained from the sensors mounted on the vehicle. In our case, the three sensors which are under consideration and which generates the four car states are: Magnetometer(orientation angle-θ), Potentiometer(Steering angle-ϕ) and Optical Encoder(position of car-x,y). Acquiring the accurate data which can be directly used in a feedback controller is of at most importance for accurate path tracking.

The other aspect of work is the implementation of Obstacle Detection Algorithm. For the efficient collision avoidance in a static/dynamic environment, it is necessary that the obstacles are detected accurately. The LIDAR based OD algorithm [**8**] is currently under development.

1.5 Objectives

- Interfacing Magnetometer, optical encoder, potentiometer, LIDAR module, DC motor and servo motor with single board computer (SBC) and testing them.
- Implementing Open loop control on hardware and testing it.
- Implementing closed loop control on hardware and testing it.
- Obtaining obstacle data around the robot and using that in motion planning algorithm.

1.6 Study of Initial Stage of Project

This Dissertation work is a part of project "Auto-drive with Park Assist (CTDP) (cars that drive and park)".

The project work started with objective to develop and demonstrate the technology (inclusive of hardware and software) for autonomous navigation (auto-drive) and autonomous parking (auto-park) of ground vehicles, on a prototype laboratory model. It is also proposed that the technology be contained within a device or plug-in module.

This device, once validated, can be directly ported to the existing vehicles with a few minor changes in the existing systems (in terms of making the device customized based on available sensory data etc).

Until August 2015, the Matlab simulations and basic hardware was ready. Hardware Platform used is based on board "Navstik". Figure 1.1 and 1.2 shows the hardware developed.

This board has built-in gyroscope, magnetometer, accelerometer and barometric pressure system. But only pwm generation capability of board was used.

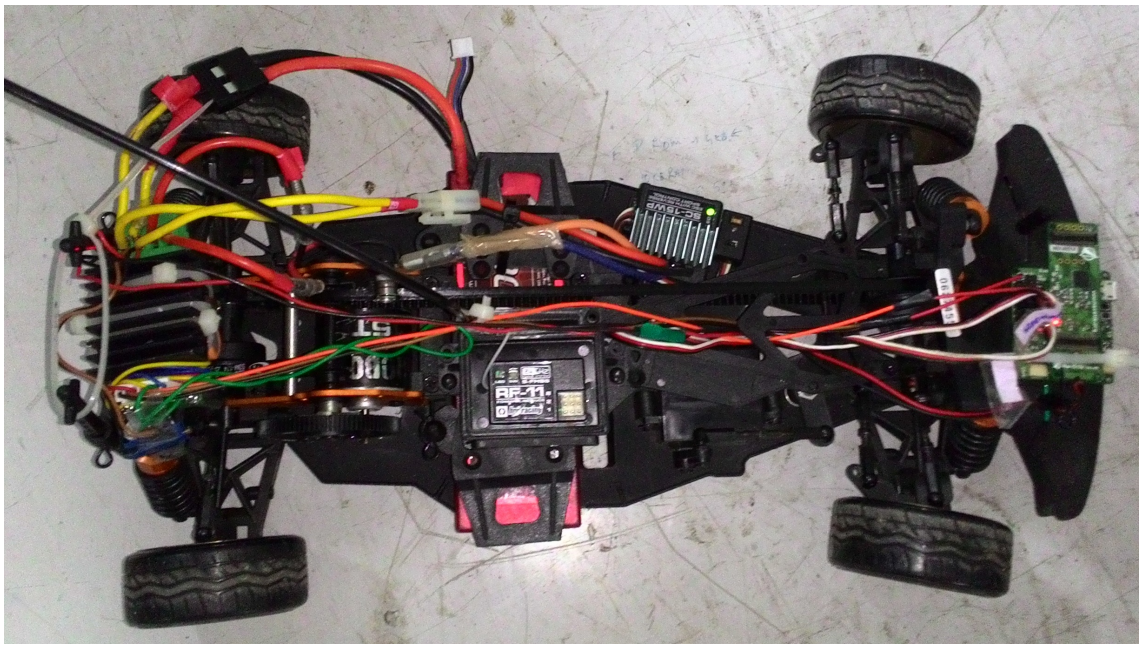

FIGURE 1.1: Car Model (Top view)

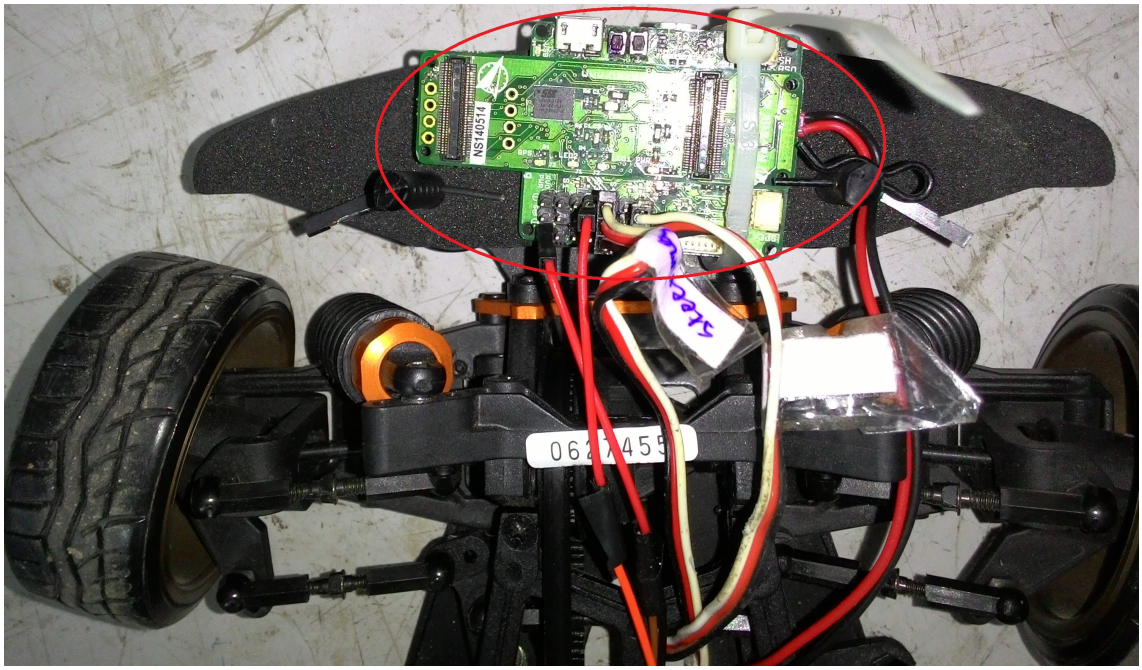

FIGURE 1.2: Navstik Board on Car

Initially the objective was to develop the hardware using single board computer and navstik board. The communication between SBC and navstik board was implemented using zig-bee modules. But problem with that approach was the overhead in communication data. Navstik board communicates using protocol named Mavlink. This overhead can create delay in processing feedback and obstacle data. So, this approach is not suitable for real time data handling and also it has some difficulties in implementation like compatibility with xbee and SBC.

The system was open loop, and it has errors in operation like, it does't stop at desired point, but moves some distance forward or stops behind. As in Figure 1.3 blue arrow shows start & end points and Red arrows show path taken by car.

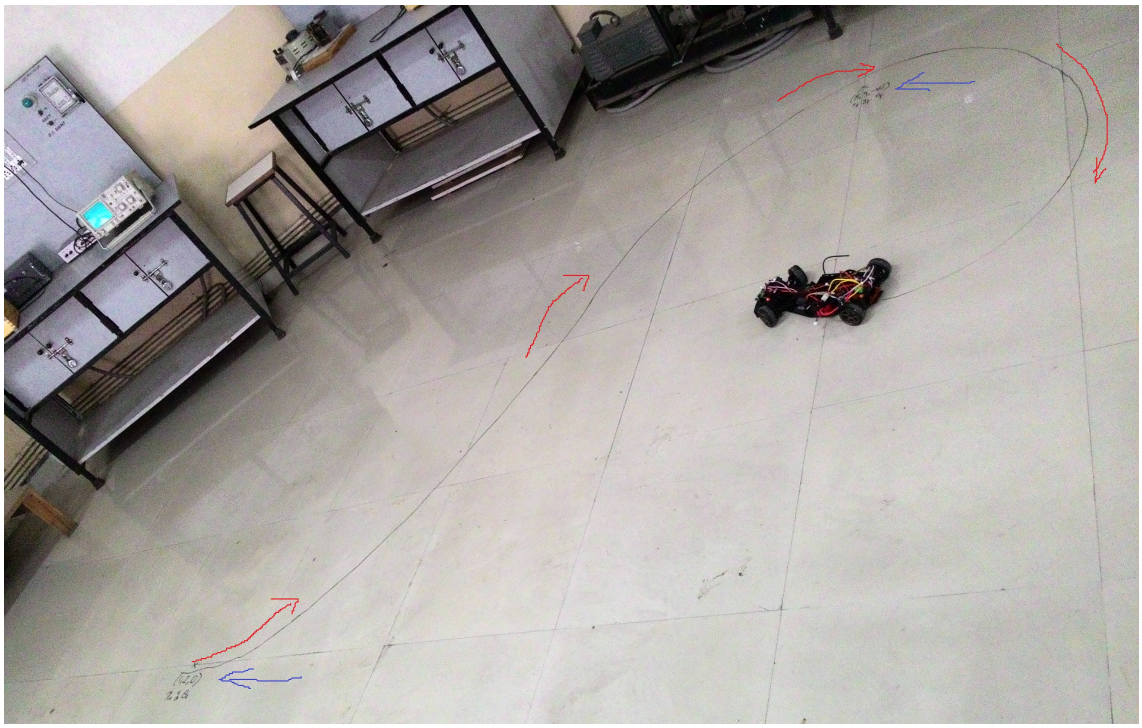

FIGURE 1.3: Traced Path by Car

Chapter 2

Proposed Work

For autonomous robots, control on each parameter is necessary for achieving good results. For this sensors play main role. Various sensors like encoder for distance measurement, compass magnetometer for finding orientation, obstacle detection sensors like LIDAR or Ultrasonic modules can be used. So, for nonholonomic robot, along with sensors data mathematical equations gives the next state of the movement of the robot.

For the project requirement is implementation of hardware. So here main part of consideration is interfacing of all sensors required to SBC. For this first selection of SBC is to be done.

Selection of SBC depends upon number and type of connections for sensor interfacing, processing power required, software availability and community support. To select SBC there was three options Raspberry Pi, Beagle-bone Black and Intel Edison. Due to strong community support, powerful features and low price, Raspberry Pi is selected to implement motion planning, path planning and obstacle detection algorithms.

After this the interfacing of sensors to raspberry pi SBC and its testing is need to be done.

2.1 Block Diagram

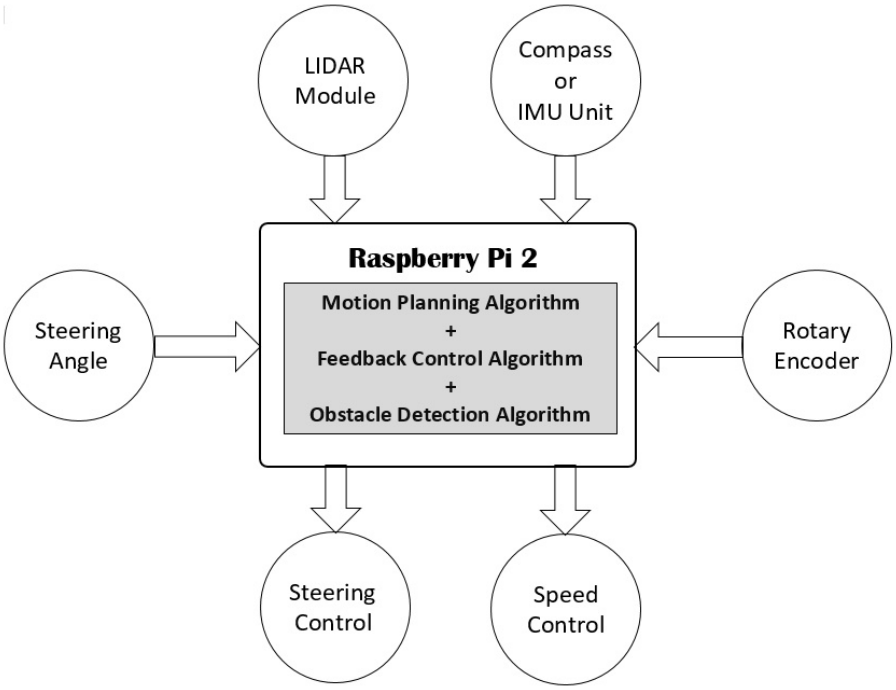

FIGURE 2.1: Block Diagram

Figure 2.1 shows the basic block diagram of system. Raspberry Pi will collect data from LIDAR (Light detection and ranging) module, compass magnetometer, rotary encoder and potentiometer on steering motor.

Input Section

Input section consists of sensors which collect data from vehicle and this data is used for finding position and behavior of robot.

1. **Steering Angle**: For sensing steering angle potentiometer is used, which is connected to steering motor. It is calibrated for steering angle of robot.
2. **Rotary Encoder**: For finding distance traveled by robot encoder is useful, by knowing number of pulses in particular time interval, speed of car can also be calculated.
3. **LIDAR Module**: Information about obstacle can be obtained using LIDAR. The module gives distance of obstacle and also we can calculate approximate size of obstacle by rotating it using motor and scanning the area around.
4. **Compass**: Electronic compass or magnetometer module gives orientation angle of module with respect to magnetic field. Here orientation angle w.r.t. earth's magnetic field is considered. Using this orientation angle and data from encoder we can calculate x and y co-ordinates of robot, which give exact position of robot in 2D plane.

Output Section

Output section is nothing but PWM generated for steering and driving motor. These PWMs are generated by motion planner algorithm and wiringPi library which considers data from all sensors. Output section mainly has two motors connected, one DC motor for derving and one servo motor for steering the car. Additionally, one servo motor can be included for LIDAR module to rotate in 180 degrees.

Processing Section

Processing section performs all mathematical and logical computations depending upon inputs and states. Data is taken from input section and results generated are given to output section. Processing is done using Raspberry Pi board. This runs motion planning, feedback control and obstacle detection algorithms. Flow charts for processing algorithms are shown below.

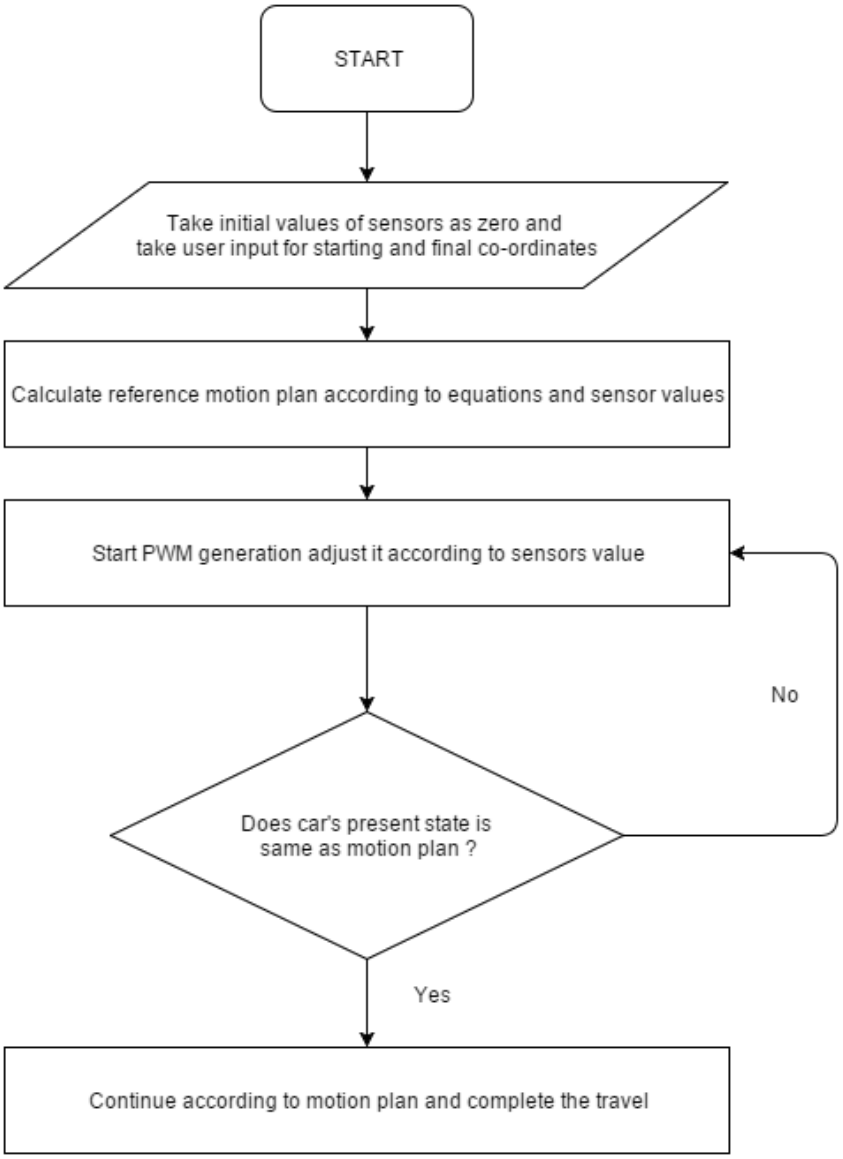

FIGURE 2.2: Flow Chart for Feedback control

From flow chart in Figure 2.2 first step is to take initial values of sensors as zero and generate first state using mathematical equations and inputs from users. Then start reading data from sensors and calculate new state variables. According to values of state variables output section generates PWM signals. Continue this process until entire motion plan is completed i.e. final values given by user are meet.

From flow chart shown in Figure 2.3, first step is to initialize the servo motor and LIDAR module. This means angle of LIDAR module should be at initial position. Then start scanning the area around the car, this is done by rotating servo motor in clockwise and anticlockwise direction. While doing that save distance of obstacle and angle of obstacle w.r.t. car in an array and copy that data into a .txt file. This file is read by motion planner for planning path and speed of car according to presence of obstacle.

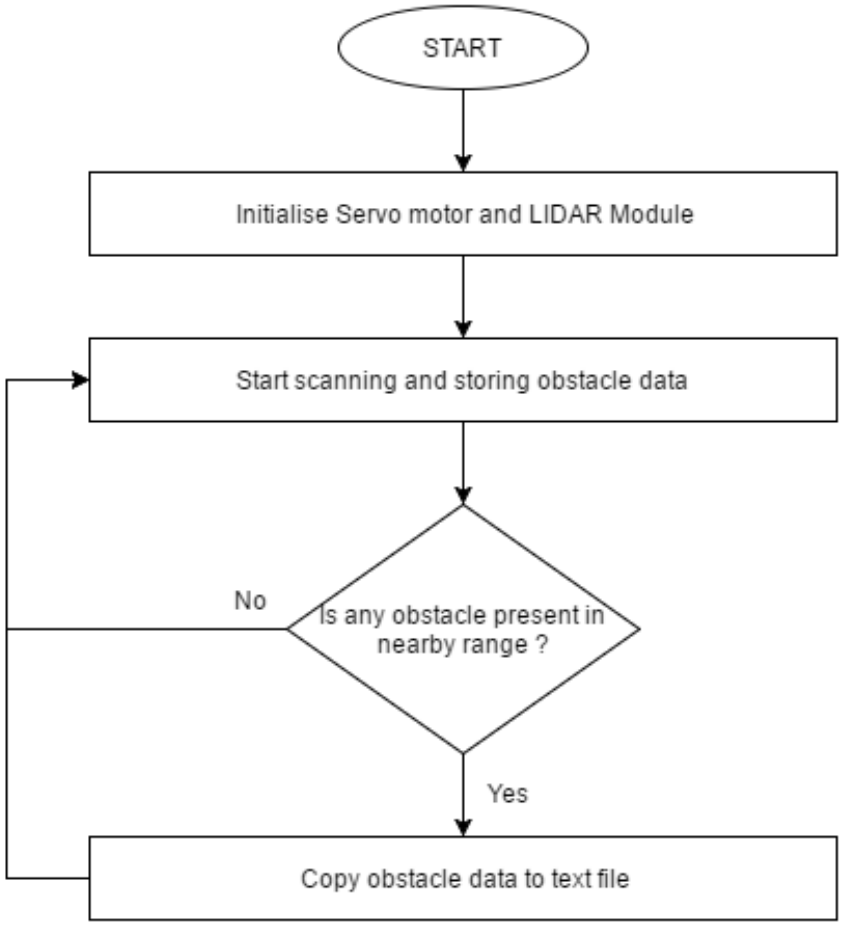

FIGURE 2.3: Flow Chart for Obstacle Detection

2.2 Feedback Control Algorithm

Feedback control is necessary for safety and accurate movement of robot. Figure 2.4 shows the flow chart which describes feedback control operation. This algorithm calculates control signal values in form of structures to travel from starting point to end point.

Initially the values are taken as default i.e. first value calculated by motion planner. While car starts motion, data from sensors is collected, viz. steering angle from potentiometer, orientation angle from magnetometer and distance traveled from encoder.

This data is collected in structures and then compared to initially calculated values by motion planner. Then the control signals in form of PWM are so generated that the error between calculated parameters and parameters obtained from sensors is minimum as possible [**4**].

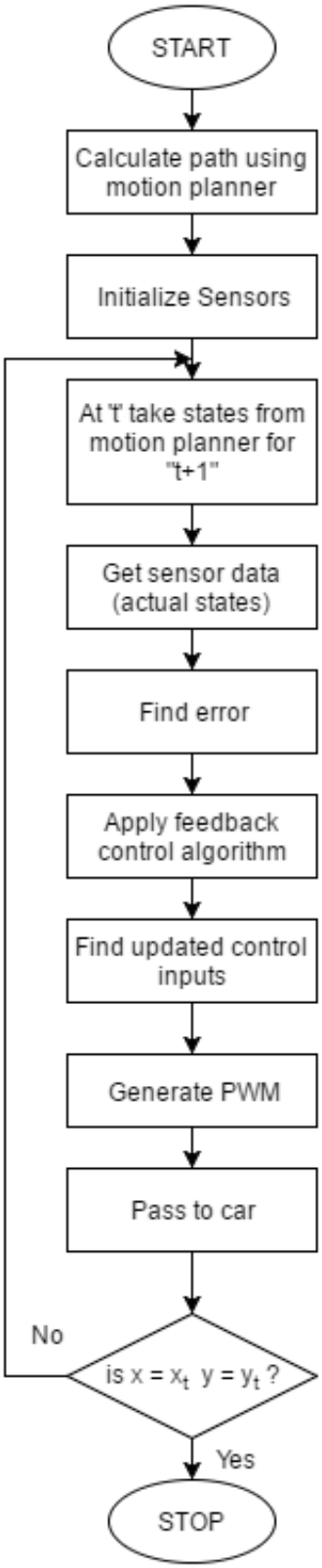

FIGURE 2.4: Feedback Control Flow Chart

Chapter 3

Hardware Implementation Using Raspberry Pi

Raspberry Pi is a popular single board computer, providing various features on credit card sized circuit board. It was developed in the United Kingdom by the Raspberry Pi Foundation. There are many revisions of these board starting from basic model A and higher specification model B to Raspberry Pi 3 model B.

All raspberry pi models have Broadcom System on Chip (SoC), which includes ARM compatible CPU and on-board graphics processing unit. CPU clock frequency ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board RAM range from 256 MB to 1 GB. SD (Secure Digital) cards are used to store the operating system and program memory. These boards have one to four USB ports, HDMI and composite video output, and a 3.5 mm jack for audio.

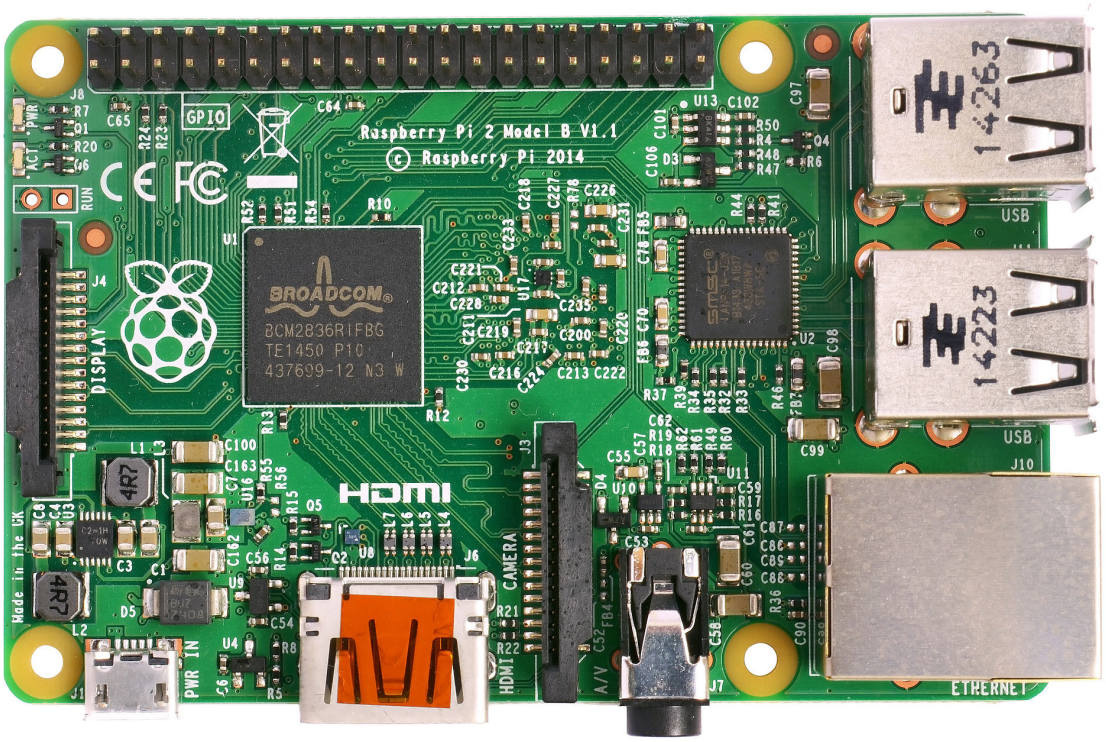

FIGURE 3.1: Raspberry Pi 2 Model B

Features of Raspberry Pi

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM
- 40 GPIO Pins
- I2C, SPI and Serial Ports
- Floating Point Math support
- Hardware PWM on 1 gpio pin, software PWM on all gpio pins.
- Good community support

It can run the full range of ARM GNU/Linux distributions, including Raspbian, Snappy Ubuntu Core, as well as Microsoft Windows 10. It has 4 USB ports and 40 GPIO pins. GPIO pins include dedicated pins for I2C, SPI interface.

It also has floating point math support which is essential for mathematical calculations done by motion planning algorithm.

Getting Raspberry Pi Ready

For using RPi as a Linux computer we need to do some initial setup.

1. Load the Linux image on SD card using win32diskimager utility.
2. Enable I2C and SPI interface.
3. Install essential libraries on it.
4. Setup the Wifi network connection.

After performing these steps we can use raspberry pi as a computer on which we can run all codes.

First of all we will look at I2C and SPI Protocols.

3.1 I2C Protocol

Serial communication is a easier way for communication between devices. But traditional approach in serial communication has some drawbacks.

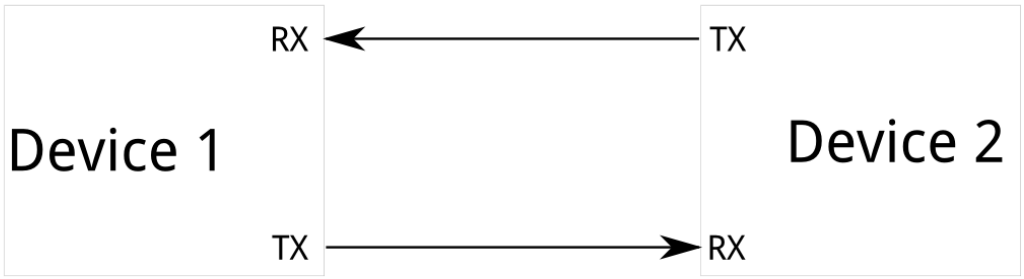

FIGURE 3.2: Serial Communication

Because serial ports are asynchronous (no clock data is transmitted), devices using them must agree ahead of time on a data rate. The two devices must also have clocks that are close to the same rate, and will remain soexcessive differences between clock rates on either end will cause garbled data.

Asynchronous type serial ports require hardware overheadthe UART at either end is relatively complex and difficult to accurately implement in software if necessary.

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address they must still be obtained from NXP (formerly Philips semiconductors).

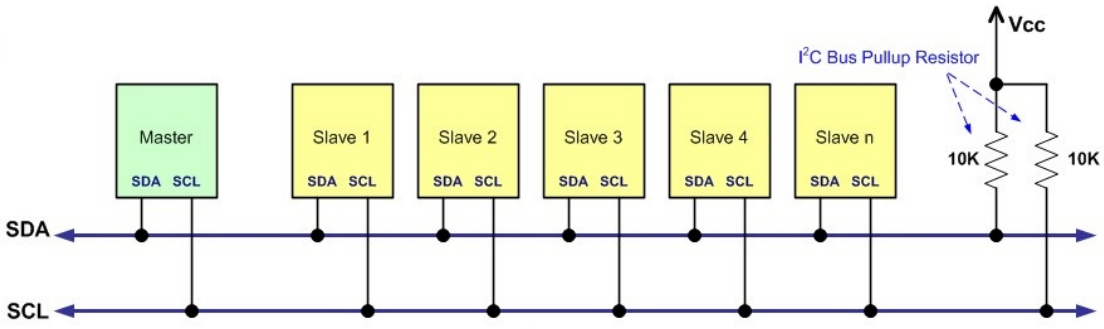

FIGURE 3.3: I2C Bus Architecture

Each I2C bus has two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master; some slave devices may force the clock low at times to delay the master sending more data (or to require more time to prepare data before the master attempts to clock it out). This is called clock stretching and is described on the protocol page.

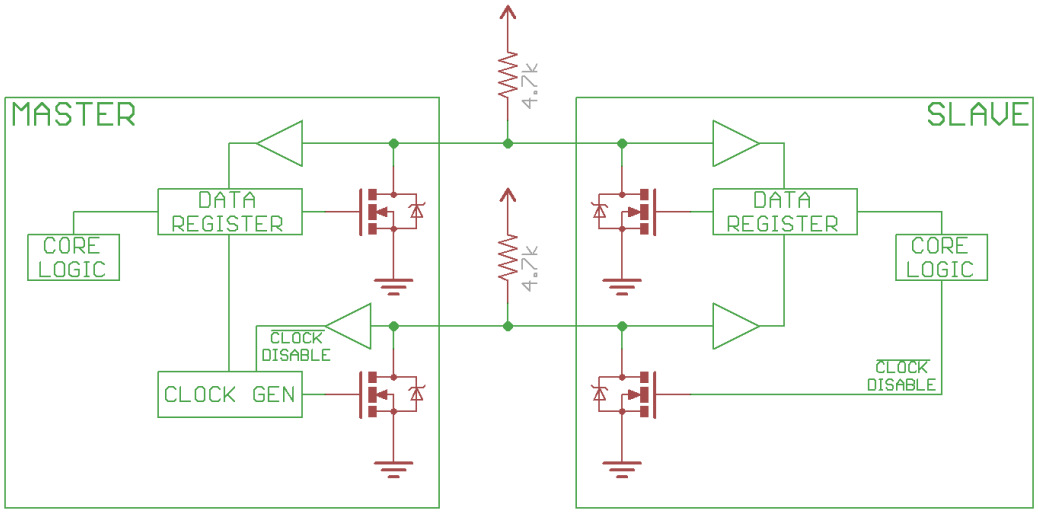

FIGURE 3.4: I2C Hardware (Notice pull - up resistors)

I2C bus drivers are open drain, meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device

is trying to drive the line high while another tries to pull it low, eliminating the power for damage to the drivers or excessive power dissipation in the system. Each signal line has a pull-up resistor over it, to restore the signal to high when no device is asserting it low.

Figure 3.4 shows internal hardware if I2C communication. Resistor selection varies with devices on the bus, but a good rule of thumb is to start with 4.7k and adjust down if necessary. I2C is a fairly robust protocol, and can be used with short runs of wire (2-3m). For long runs, or systems with lots of devices, smaller resistors are better.

Signaling in I2C

In normal state both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the Start condition (S) followed by the address of the slave device (B1). If the bit 0 of the address byte was set to 0 the master device will write to the slave device (B2). Otherwise, the next byte will be read from the slave device. Once all bytes are read or written (Bn) the master device generates Stop condition (P). This signals to other devices on the bus that the communication has ended and another device may use the bus.

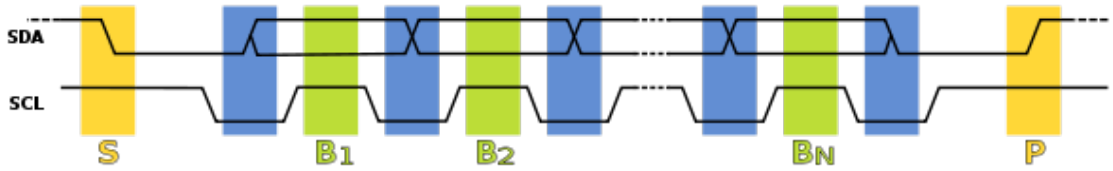

FIGURE 3.5: Signaling in I2C Protocol

Most I2C devices support repeated start condition. This means that before the communication ends with a stop condition, master device can repeat start condition with address byte and change the mode from writing to reading.

Configuring I2C

I2C is a communication protocol developed by Phillips. It is commonly used for communication between chips.

I2C bus allows multiple devices to connect to RPi, each with unique address. For enabling I2C on RPi first run these two commands:

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
```

Now to install i2c support for the ARM core and Linux kernel run

```
sudo raspi-config
```

After that go to Advanced Options > I2C and select it after that select **YES** for prompts appearing after that. Reboot the RPi for applying the changes made.

Now, to test I2C type following command and all connected devices will appear on the screen as shown in Figure 3.6 where red circles highlight address of devices.

```
sudo i2cdetect -y 1
```

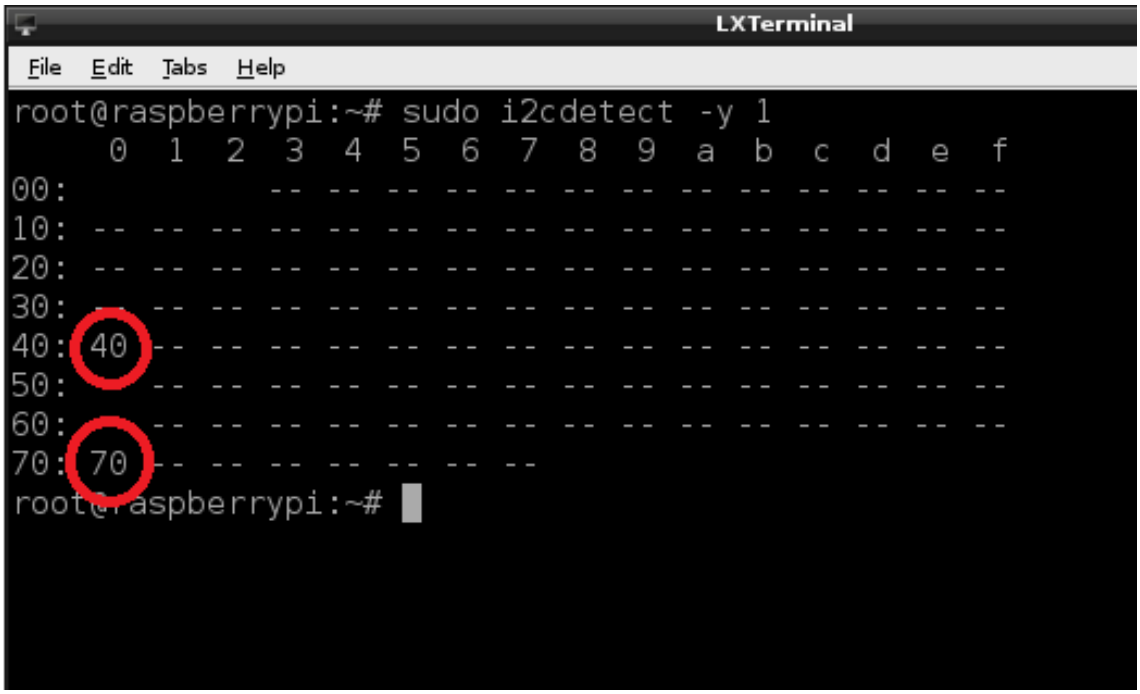

FIGURE 3.6: Address Map of I2C Devices Detected

3.2 SPI Protocol

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by micro-controllers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two micro-controllers. This bus was developed by Motorola to provide full-duplex synchronous serial communication between master and slave devices.

Figure 3.7 shows typical architecture of SPI bus. Typically there are three lines common to all the devices:

- MISO (Master In Slave Out) - The Slave line for sending data to the master,
- MOSI (Master Out Slave In) - The Master line for sending data to the peripherals,
- SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master
  and one line specific for every device:
- SS (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.

When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.

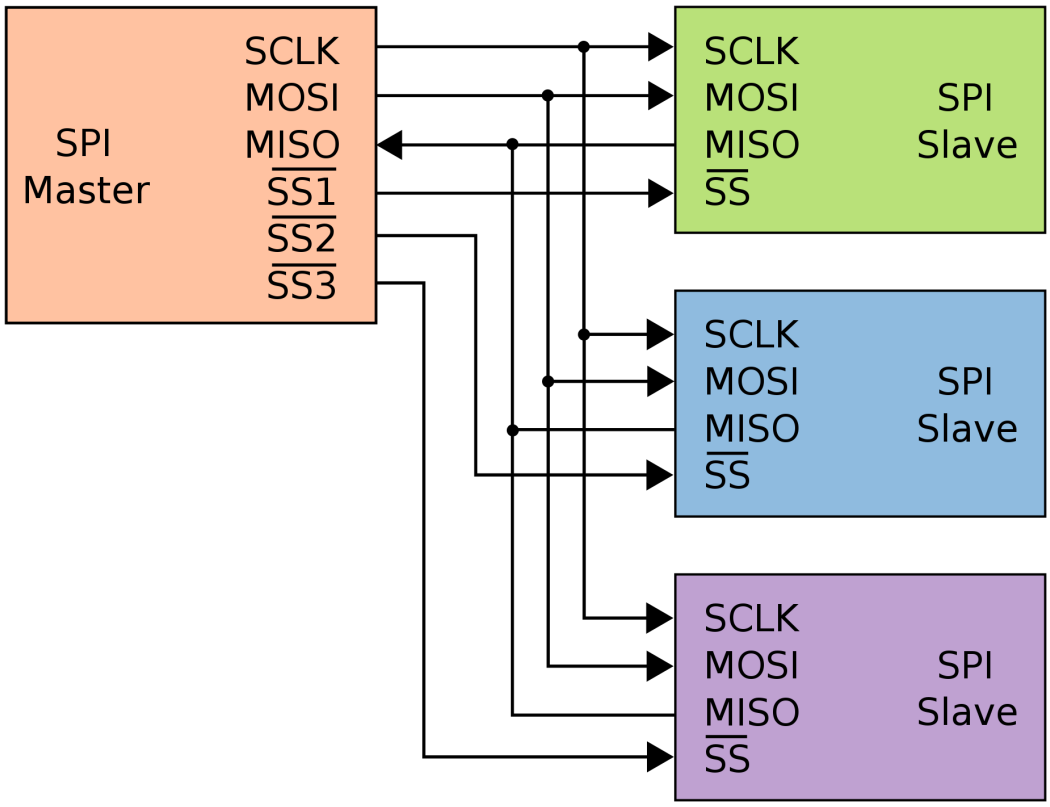

FIGURE 3.7: Typical SPI Bus Architecture

To write code for a new SPI device you need to note a few things:

- What is the maximum SPI speed your device can use?
- Is data shifted in Most Significant Bit (MSB) or Least Significant Bit (LSB) first?
- Is the data clock idle when high or low? Are samples on the rising or falling edge of clock pulses?

The SPI standard is loose and each device implements it a little differently. This means you have to pay special attention to the device's data-sheet when writing your code.

Figure 3.8 shows signaling in SPI bus. The Slave Select line is normally held high, which disconnects the slave from the SPI bus. (This type of logic is known as "active low, and youll often see used it for enable and reset lines.) Just before data is sent to the slave, the line is brought low, which activates the slave. When youre done using the slave, the line is made high again. In a shift register, this corresponds to the "latch input, which transfers the received data to the output lines.

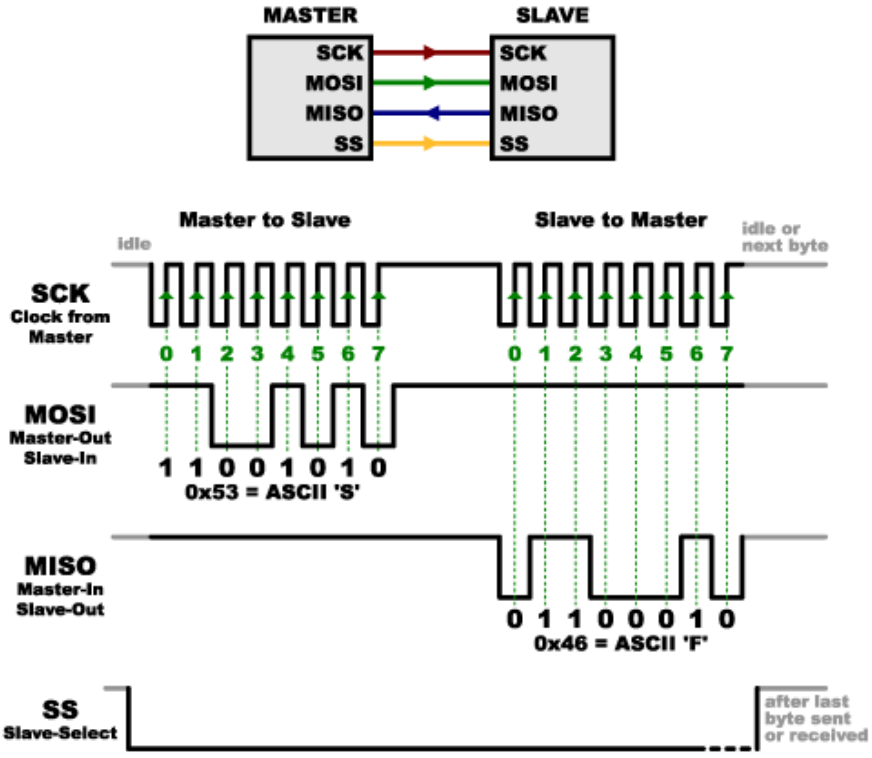

FIGURE 3.8: Signaling in SPI Bus

Configuring SPI

Configuring SPI in RPi is same as configuring I2C, only go for SPI options in Raspi-Config > Advanced options. Then after rebooting Pi, SPI will work.

To test SPI bus is working or not, type command as

```
ls -l /dev/spidev*
```

Address of connected SPI devices will be visible as shown in Figure 3.9.

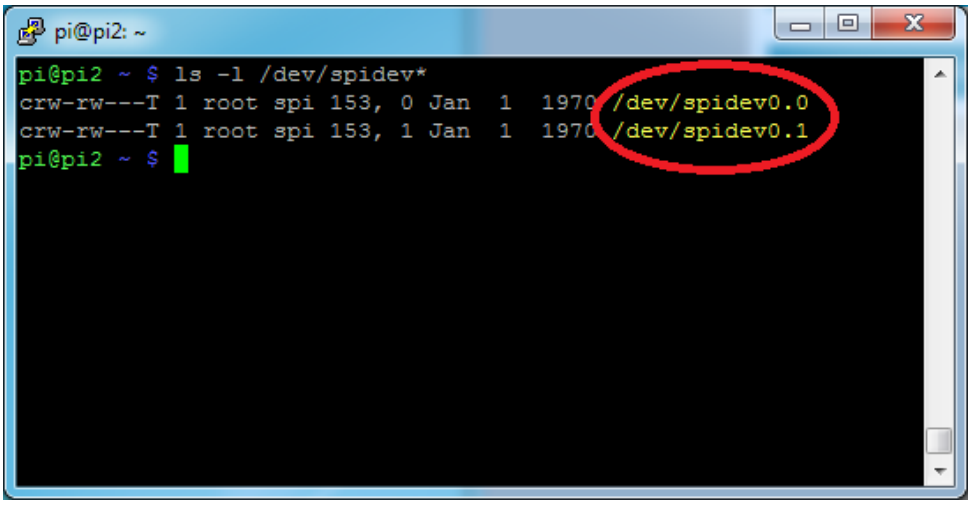

FIGURE 3.9: Addresses of Connected SPI Devices

3.3 WiringPi Library

WiringPi is a GPIO access library written in C for the SoC used in the Raspberry Pi. It is usable from C and C++ and many other languages with suitable wrappers. Its designed to be familiar to people who have used the Arduino wiring system.

Raspberry Pi GPIO Header

BCM	WiringPi	Name	Physical		Name	WiringPi	BCM
		3.3v	1	2	5v		
2	8	SDA.1	3	4	5V		
3	9	SCL.1	5	6	0v		
4	7	1-Wire	7	8	TxD	15	14
		0v	9	10	RxD	16	15
17	0	GPIO. 0	11	12	GPIO. 1	1	18
27	2	GPIO. 2	13	14	0v		
22	3	GPIO. 3	15	16	GPIO. 4	4	23
		3.3v	17	18	GPIO. 5	5	24
10	12	MOSI	19	20	0v		
9	13	MISO	21	22	GPIO. 6	6	25
11	14	SCLK	23	24	CE0	10	8
		0v	25	26	CE1	11	7
0	30	SDA.0	27	28	SCL.0	31	1
5	21	GPIO.21	29	30	0v		
6	22	GPIO.22	31	32	GPIO.26	26	12
13	23	GPIO.23	33	34	0v		
19	24	GPIO.24	35	36	GPIO.27	27	16
26	25	GPIO.25	37	38	GPIO.28	28	20
		0v	39	40	GPIO.29	29	21
BCM	WiringPi	Name	Physical		Name	WiringPi	BCM

FIGURE 3.10: WiringPi and BCM numbering scheme according to GPIO header of RPi.

WiringPi includes a command-line utility gpio which can be used to program and setup the GPIO pins. The I2C, SPI and UART interfaces can also be used as general purpose I/O pins when not being used in their bus modes. This library supports software PWM function. WiringPi supports its own pin numbering scheme as well as the BCM_GPIO pin numbering scheme provided by Broadcom. Figure 3.10 shows numbering schemes. Here BCM means Numbering given by Broadcom, which is physical pin numbering.[7]

This is the main library used for this project work as the project code is in C language.

3.4 Generating Hardware and Software PWM of Specific Frequency

Pin number 12 of raspberry Pi has hardware PWM support. There are many libraries written for it which support software PWM on other gpio pins. They can be useful for applications requiring more number of motor controls or LED controls. Here WiringPi library is used to generate PWM signals.

Types of PWM:

- Balanced mode - the duty cycle will control the frequency of the noise.
- Mark-Space mode - the frequency is the base PWM frequency and it is controlled by the clock.

FIGURE 3.11: PWM Generated with 100Hz frequency

Mark-space mode is useful for motor control applications, since it gives constant frequency with variable pulse width. While balanced mode varies frequencies to vary duty cycle, useful for tone generation applications.

In this project two PWM signals are required one for steering motor and another for driving motor. Two software PWM signals can be generated using WiringPi and can drive the motors according to requirement.

```
int softPwmCreate (int pin, int initValue, int Range);
```

This creates software controlled PWM pin. If we give Range as 100 and initial value 0, we can generate PWM with width anything from 0 to 100.

```
void softPwmWrite (int pin, int value);
```

This gives the value to the pin, to create required pulse width.

3.5 Interfacing of Sensors to Raspberry Pi

Reading Analog Voltages Using External ADC MCP 3008

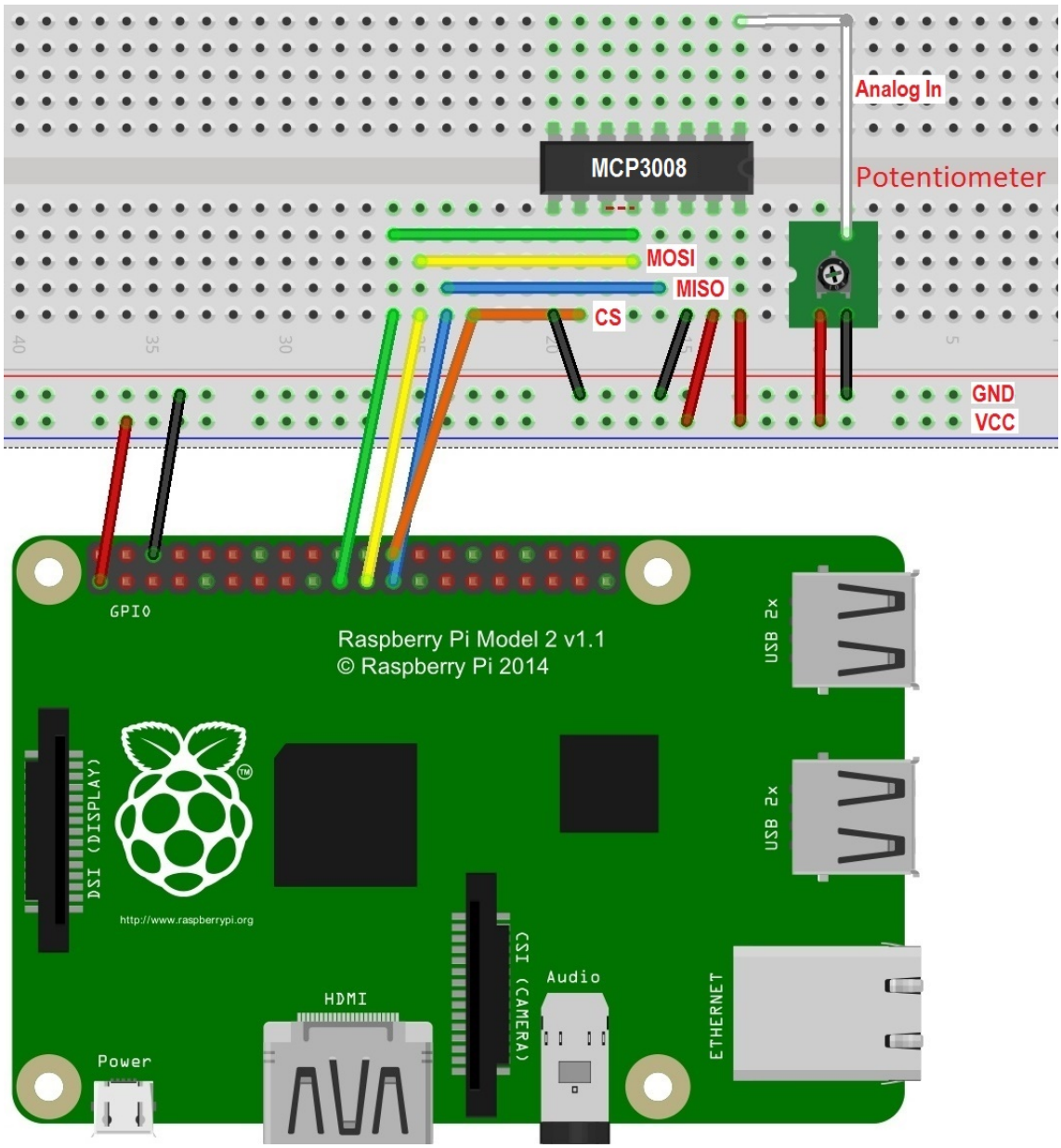

FIGURE 3.12: Potentiometer connected to R Pi via ADC

Besides having 40 gpios, Raspberry Pi 2 has one downside. It has no analog supporting pin. So, to measure analog voltages required in some applications is not directly possible. Here we have to use Analog to digital converter to read voltage from potentiometer.

Figure 3.12 shows the potentiometer connected to ADC IC and connection to RPi. MCP3008 is 8-Channel ADC used which gives capability to read analog values. This IC communicates using SPI (Serial Peripheral Interface) bus.[8]

By calibrating the potentiometer according to application we can process analog values to get desired results.

Interfacing Magnetometer to RPi

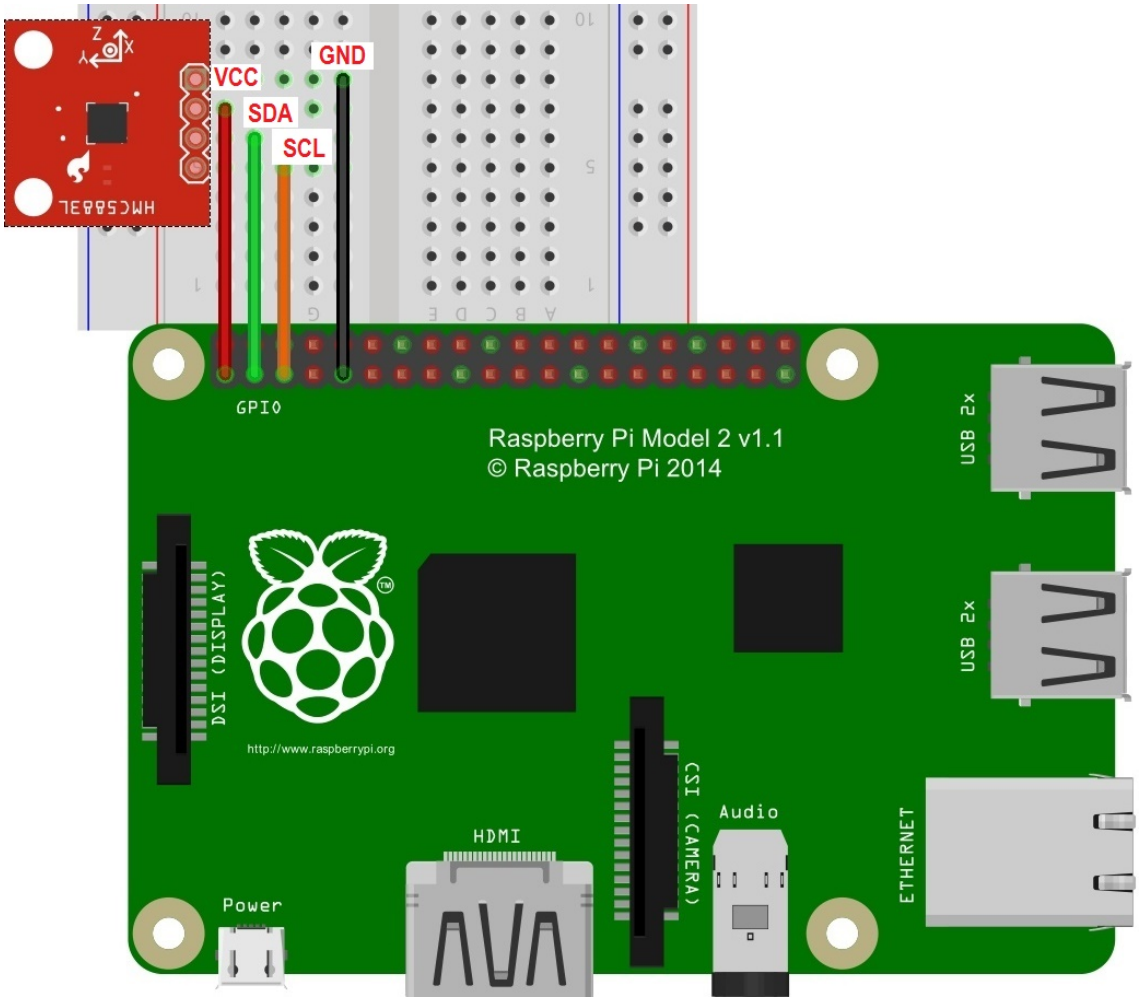

FIGURE 3.13: Magnetometer Connected to RPi

Magnetometer module measures the earths magnetic field in three axes. It provides individual readings for each axis, which may be used separately or together for 3D calculations. Also it can measure raw magnetic strength of nearby magnetic source. Generally it is used as electronic compass in electronic gadgets.

Here HCM5883L module is used to find orientation angle of vehicle. It has Integrated 12-bit ADC. This module communicates with RPi using I2C bus. Figure 3.13 shows the connection between RPi and magnetometer.

Bearing angle can be calculated from raw x, y and z values using formula:

$$Angle = \frac{tan^{-1}(\frac{y}{x}) \times 180}{\Pi}$$

Here atan is arc tangent function.

LIDAR Module Interfacing to RPi

LIDAR stands for Light Detection and Ranging, this is the technology similar to RADAR, only instead of radio waves it uses LASER for detection of obstacle. LIDAR is a remote sensing method that uses light in the form of a pulsed laser to measure ranges (variable distances) to the earth. These light pulses combined with other data recorded by the

airborne system generate precise, three-dimensional information about the shape of the Earth and its surface characteristics. LIDAR based sensor gives distances of obstacles nearby, so that we can use them to interpret environment around robot [**3**]. For using with RPi, PulsedLight LIDAR module in Figure 3.14 is useful which communicates to RPi using I2C protocol. Figure 3.15 shows the connections for I2C bus.

FIGURE 3.14: PulsedLight LIDAR Module

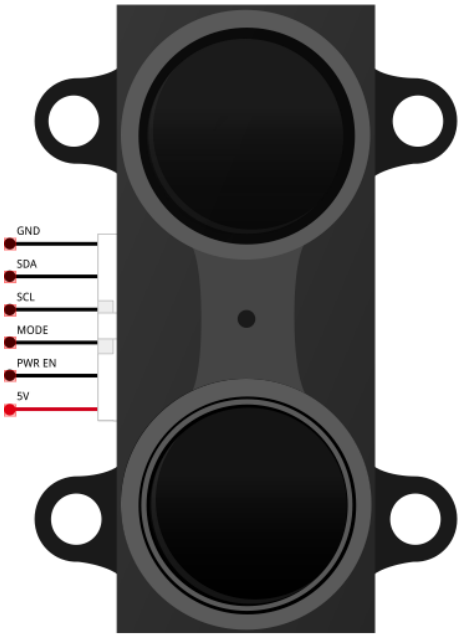

FIGURE 3.15: Lidar I2C Connections

This module gives direct distance of obstacle in front of it. So, to know about obstacles around robot, we can rotate module and get data for 180 degree around or for 360 degree around. We have mounted the LIDAR on servo motor as shown in Fig 7 and we measure the distance of LIDAR from its surrounding objects by rotating the servo motor. We get the distance reading in mm at every 1 degree rotation. The reading is the distance at which the laser beam is obstructed. By predetermined the safety perimeter (e.g. 10cm),

we can detect the objects inside that perimeter and accordingly plan our path so as to avoid them.

Interfacing Encoder to RPi

Encoders are often used in intelligent machines to control their movements. A rotary encoder, is in essence a pair of switches which change as a shaft is turned, and by reading those switches, one can tell whether and in which direction the shaft is turning. A rotary encoder cannot tell absolute position, but relative position. We can calculate distance traveled and speed using programming logic.

In case of car type robots rotary optical encoders are used to find distance traveled by it, so as to find its relative position. Also using encoder readings we can calculate speed of robot. Figure 3 shows encoder interfaced to R Pi through GPIO. The encoder used for the car model is from Jencoder [**10**]. The PPR rating of encoder is 2000 and it can be operated using 5V to 24V DC supply voltage.

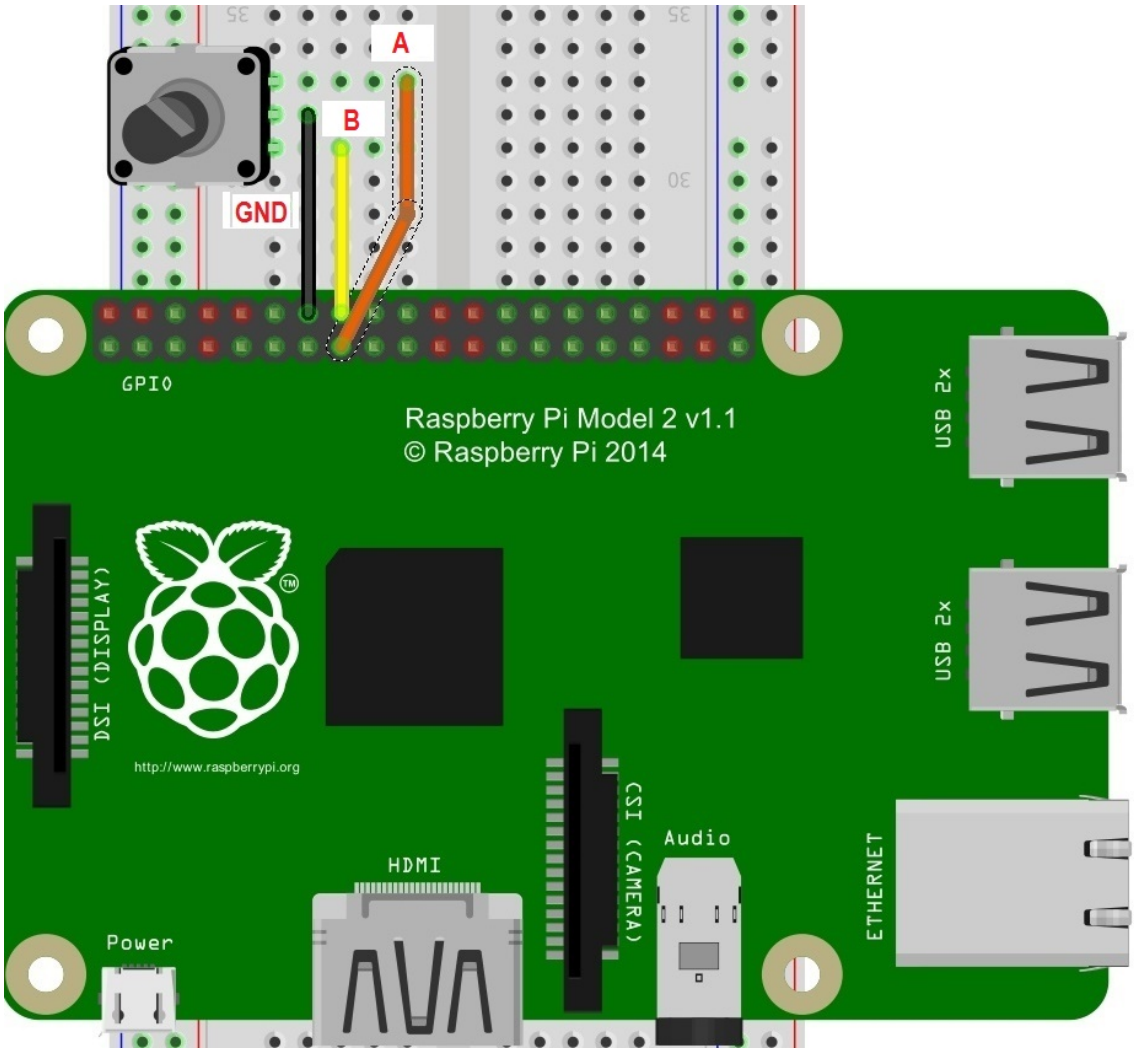

FIGURE 3.16: RPi and Encoder Connection

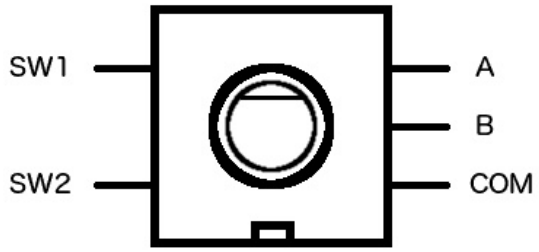

FIGURE 3.17: Simple Encoder Pinout Diagram

Using the built in pull-up resistors on the Raspberry Pi for the pins connected to A and B as shown in Figure 3.17, means that when the respective switches are closed, the pins will read low (0) and when the switches are open, the same pins will read high (1). Theres no need for any additional resistors or capacitors. As the encoder shaft is turned, the switches will alternatively open and close so that they progress through a series of states known as Gray Code. These states are as follows:

00, 01, 11, 10

By reading any two successive states, one can tell which direction the encoder is turning. A and B signals have fixed Phase shift between them, using this we can find direction of rotation of shaft as shown in Figure 3.18. 00 to 01 might be clockwise while 00 to 10 might be counter-clockwise. Using interrupts, we can get the Pi to record each change of a pin and to record the direction the encoder is turning.

Distance calculation is done using formula

$$Distance = \frac{(Pulse\_count) \times 2 \times \pi \times (Radius\_of\_wheel)}{(Pulses\_per\_rotation)}$$

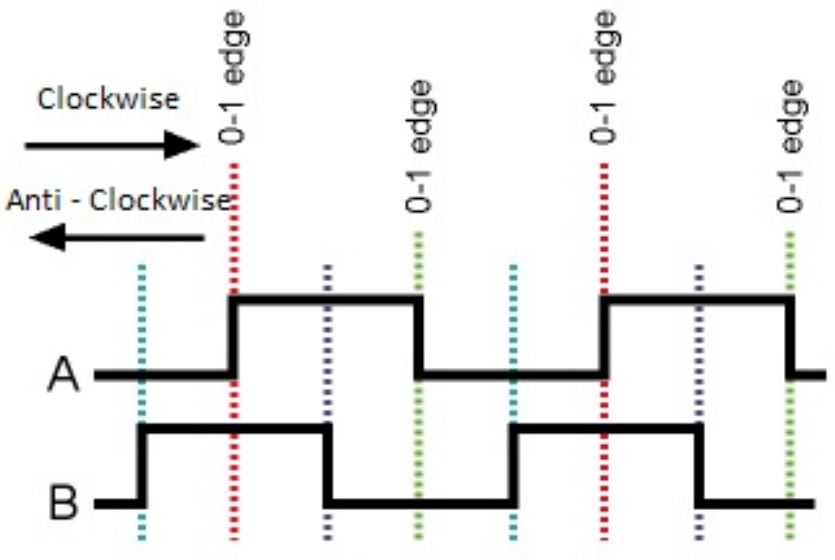

FIGURE 3.18: Encoder waveforms and Direction Indication

Wifi Module connection to RPi

Raspberry Pi has remote access capability, which enables it to be controlled through remotely located monitor. This is very useful feature which enable RPi to send and receive data remotely. For remote access SSH connection is used. RPi 2 has 4 USB ports which can me used to connect many plug and play peripherals to it. Wifi module is one of them which can be used for connecting RPi to the nearby wireless network.

FIGURE 3.19: Wifi Module

Wifi module used is RALink RT5370 Wireless-N USB 2.0 Adapter. It complies with IEEE 802.11n (Draft 2.0), IEEE 802.11g, IEEE 802.11b standards. Data rate for transmitting is up to 150 Mbps and for receiving is 150 Mbps. Also, supports 20 MHz/40 MHz frequency width and auto-detects and changes network transmission rate.

We need to do some modifications in configuration files of RPi for auto connecting the RPi to nearby wireless network [**13**].

3.6 Establish an SSH Connection to Raspberry Pi

In order to access the Raspberry Pi via remote SSH, we need to know the local IP address the router has given to it. Advanced IP Scanner can be used to scan network and generate a list of all connected devices and their local IP addresses. Just click Scan in the upper left hand area of the window, and a list of all connected devices will be generated with their associated local IP addresses:

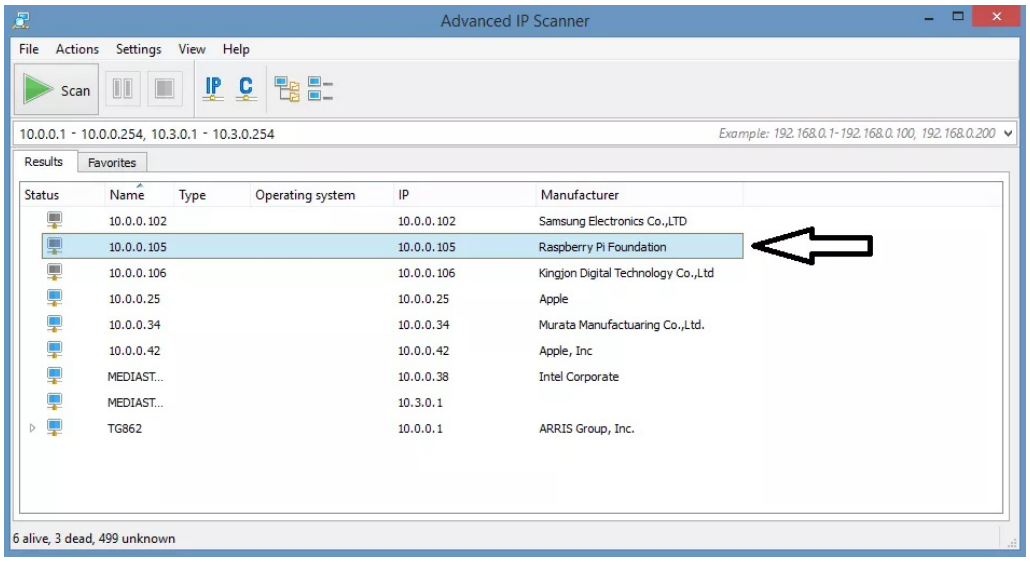

FIGURE 3.20: Advanced IP Scanner Window

Now SSH connection can be done to the Raspberry Pi with SSH client, PuTTY. Open up PuTTY, and enter the local IP address of the Raspberry Pi into the Host Name (or IP address) field and open the connection.

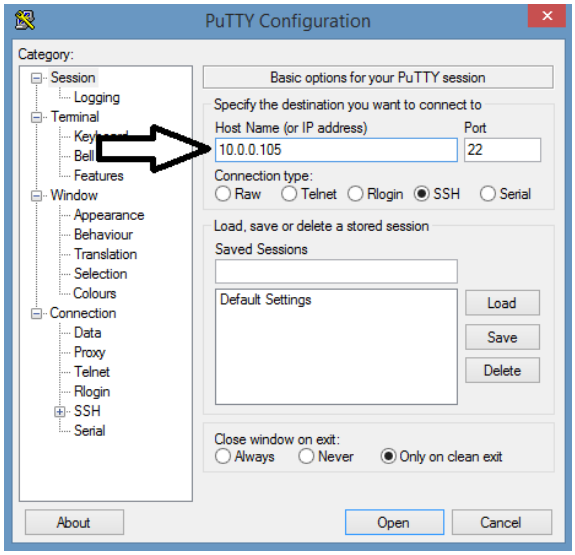

FIGURE 3.21: PuTTY SSH Client Window

If the SSH connection is successful login prompt will appear. After entering login name and password, console will appear.

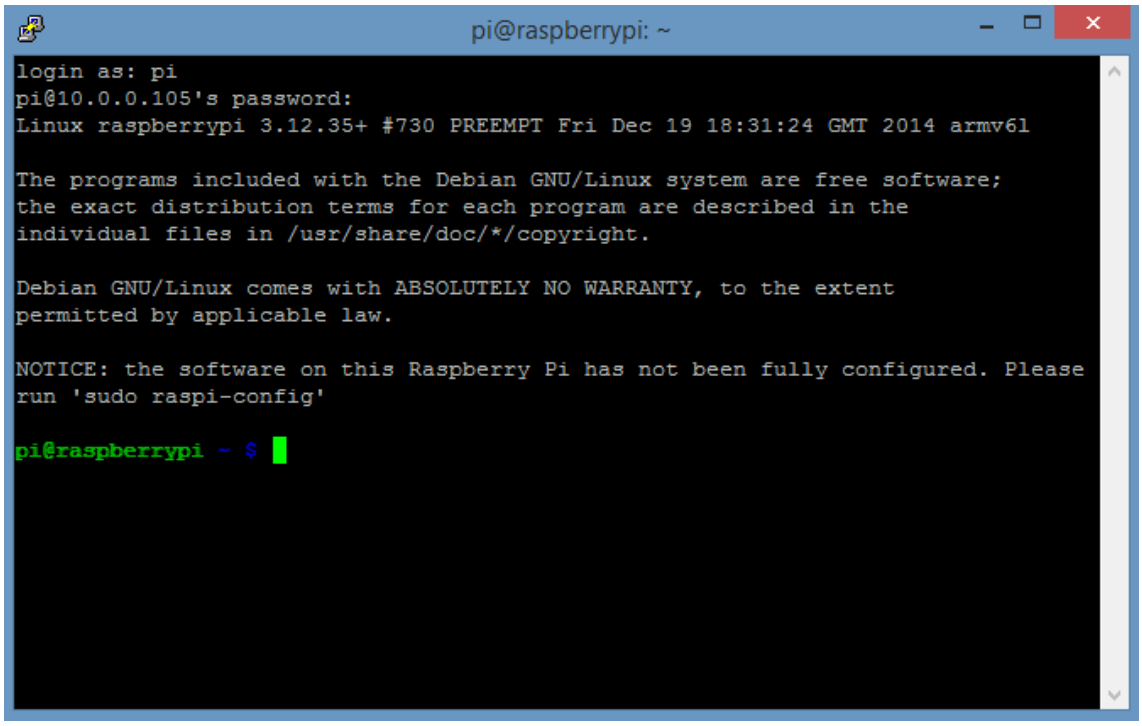

FIGURE 3.22: Raspberry Pi Console from PuTTY Client

If we need GUI of RPi, enable X11 forwarding in PuTTY Client. This needs Xming server installed on PC.

3.7 Running a program at boot-up of Raspberry Pi.

For standalone system like car type robot, the programs should start running automatically when device boots up. This can be done by writing a command which calls programs to run.

Procedure for making auto running programs:

1. Make the desired program as executable by changing its permissions.
2. Edit rc.local file by executing this command in terminal
   sudo nano /etc/rc.local
3. After the initial comments add the following line
   sudo /home/pi/projects/name_of_executable &
   Here & is for running program in background.
4. If you need to kill your program running in the background this command is used
   sudo killall name_of_executable

Chapter 4

Results

4.1 Board For Connecting All Sensors to RPi

For easy connection of all sensors to raspberry pi, a shield type circuit is designed on general purpose board as shown in Figure 4.1 & 4.2 and also on Printed Circuit Board (PCB) using KiCad Software.

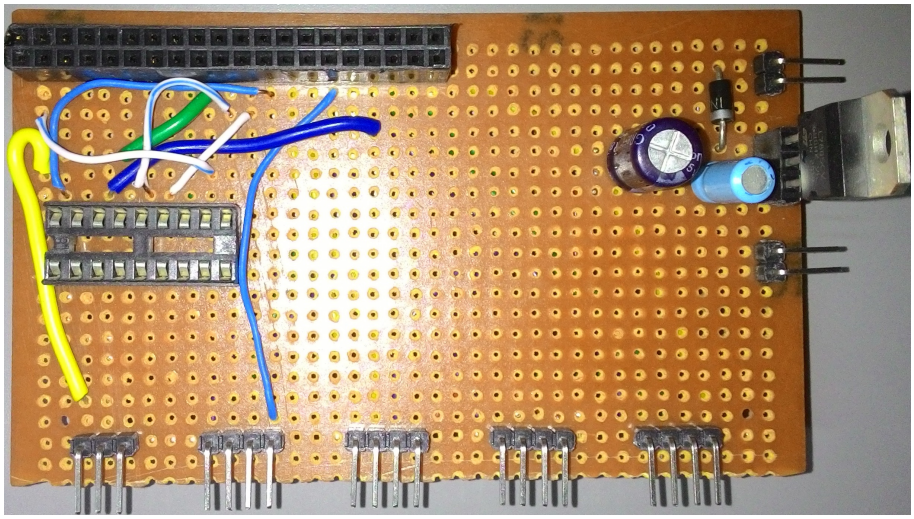

FIGURE 4.1: Circuit on General Purpose Board (View 1)

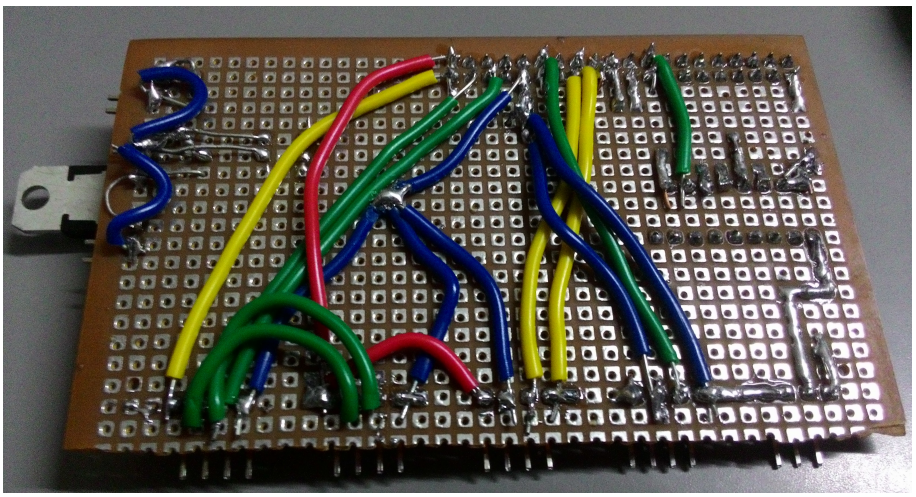

FIGURE 4.2: Circuit on General Purpose Board (View 2)

This PCB is designed to fit on top of the RPi so it contains connections from RPI header to other connectors. Figure 4.3 shows schematic diagram, in this schematic 40 pin connector fits on GPIO header of RPi.

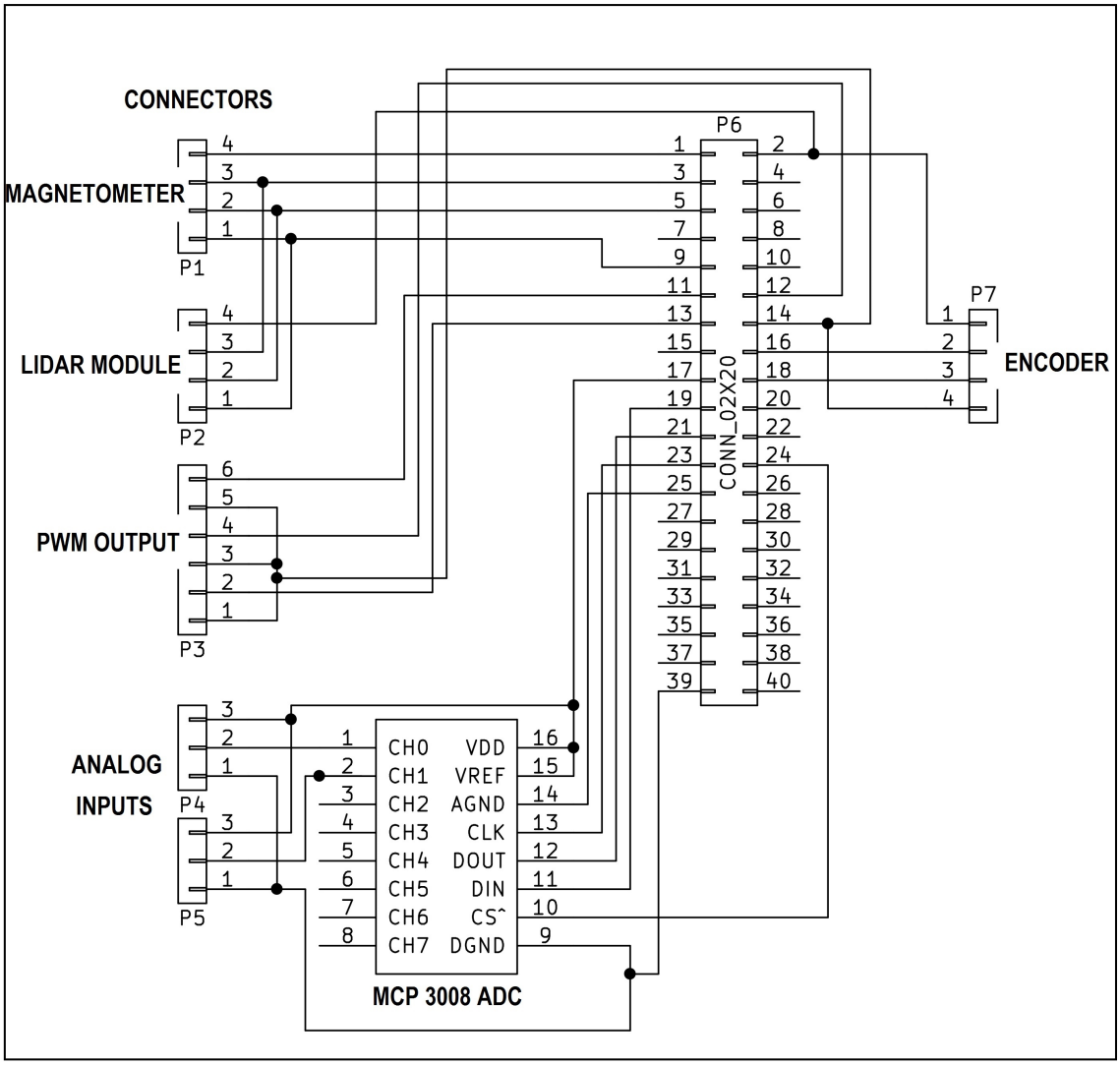

FIGURE 4.3: Schematic for making PCB layout

After manually routing all connections PCB layout obtained is shown in Figure 4.4.

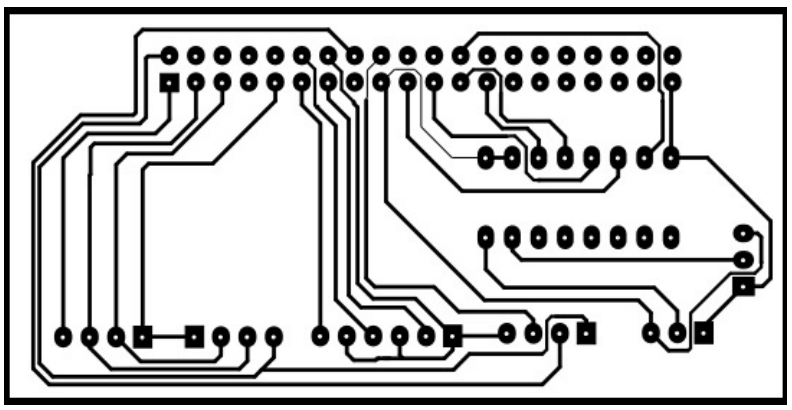

FIGURE 4.4: PCB Layout

Figure 4.5 and 4.6 show PCB after Printing and etching tracks.

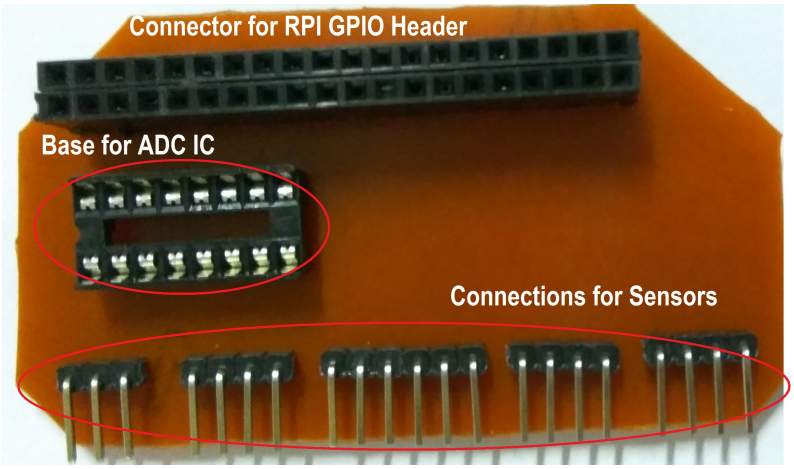

FIGURE 4.5: PCB Front View

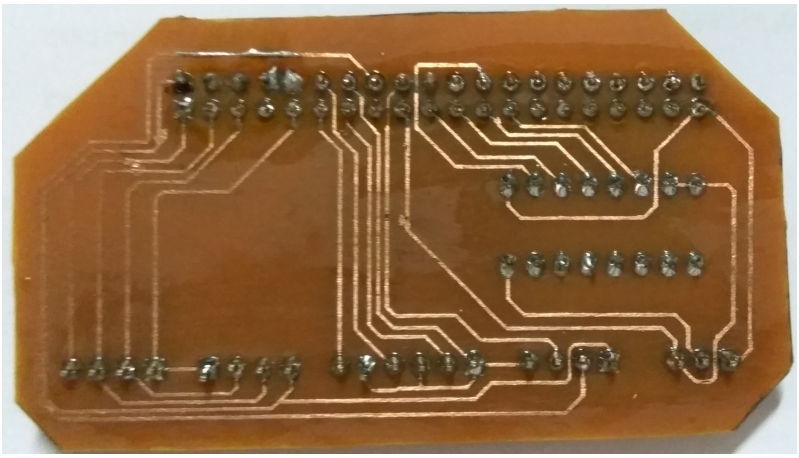

FIGURE 4.6: PCB Back View

There is one 40 pin connector which fits on RPi Gpio header, one 16 pin IC base for placing MCP3008 ADC IC and L-shaped male connectors for connecting sensor wires to the shield.

Figure 4.7 shows the PCB placed on RPi.

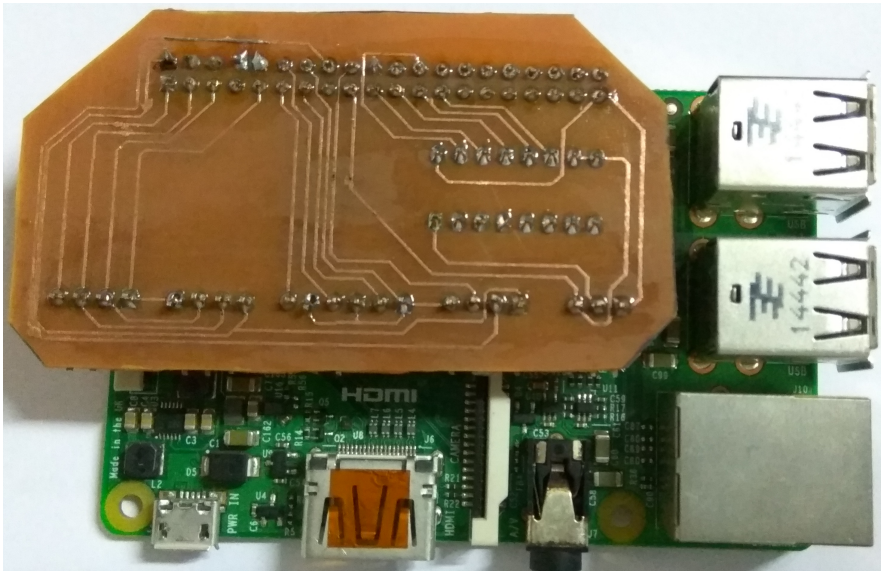

FIGURE 4.7: RPi with PCB Shield

4.2 Robot Model after connecting all sensors

Figure 4.8 and 4.9 show the car model with all sensors except LIDAR module.[**12**] Steering servo motor is connected to potentiometer which gives change in voltage as motor rotates. Encoder is connected to drive motor through gears and this motor is driven by motor driver circuit. In Figure 4.9 we can see RPi is connected to all sensors and an Wifi module.

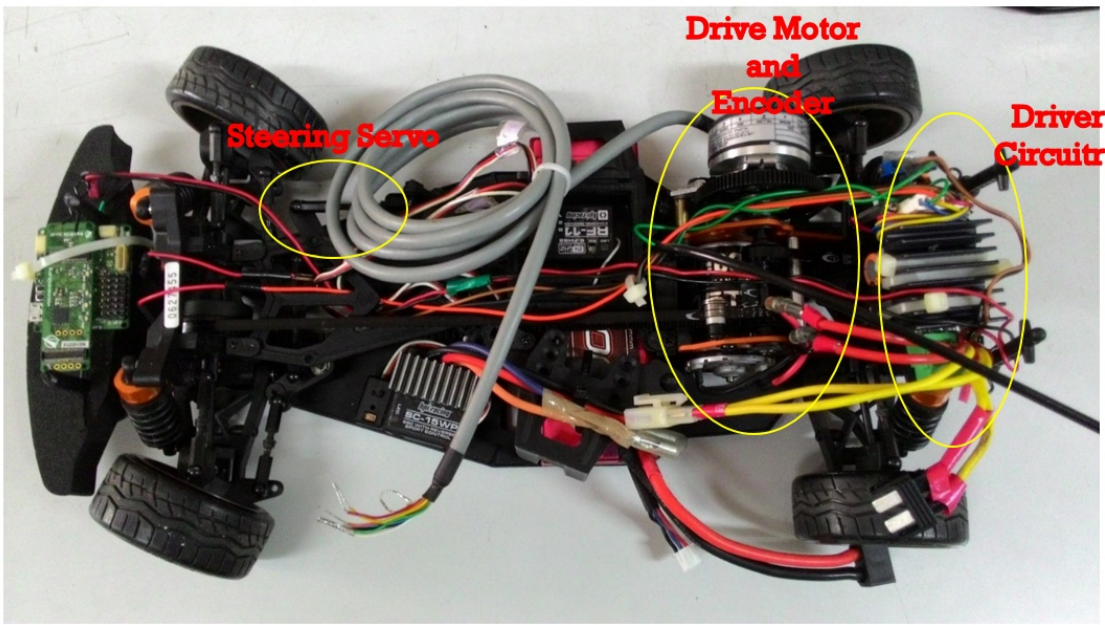

FIGURE 4.8: Controls Present on Car Model

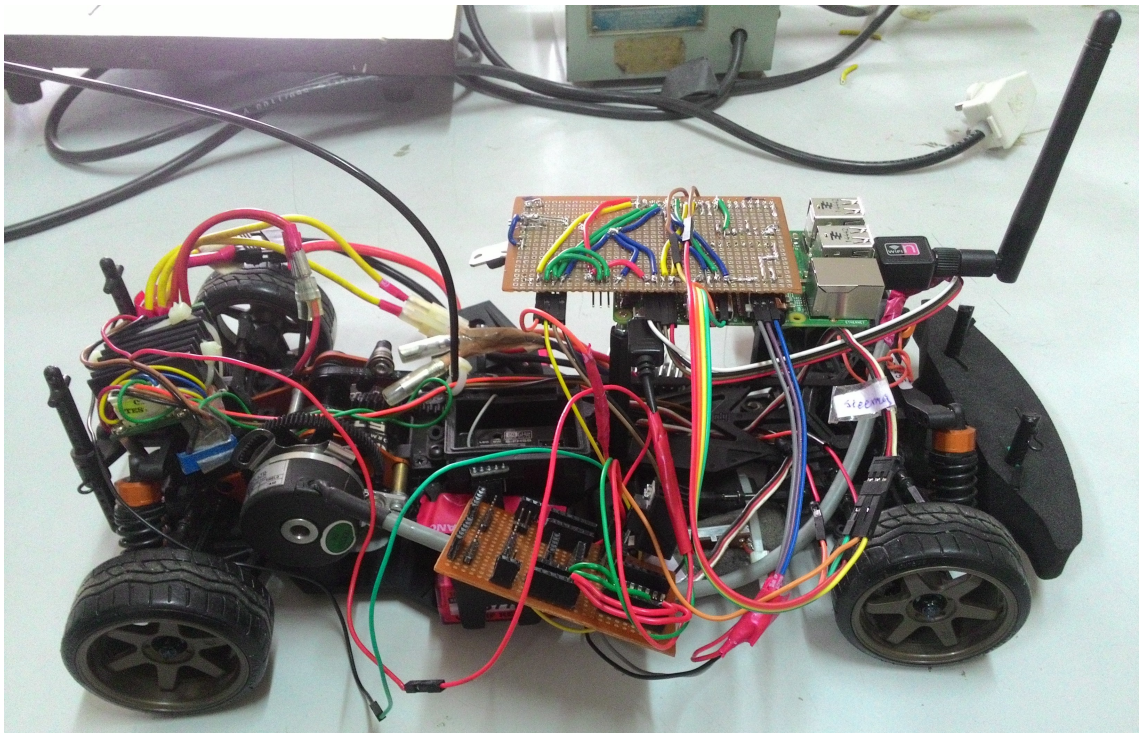

FIGURE 4.9: Car Model with All Sensors Connected

4.3 Detection of Obstacles

For autonomous driving car the detection of static or moving obstacle is essential to ensure safety. To implement such obstacle detection feature there are various methods like use of Infra-Red light based detectors, use of camera, using LIDAR etc. Here obstacle detection is done using LIDAR based Sensor.**[6]**

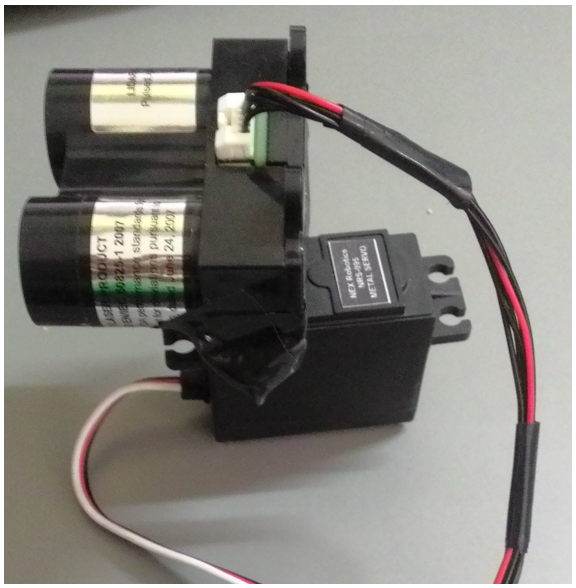

FIGURE 4.10: LIDAR Module Mounted on Servo Motor

For setup shown in Figure 4.10, we can precisely control the angle of servo motor using PWM, and collect data of obstacle distance corresponding to angle. Then this data is communicated to motion planner so that it avoids the angles which have obstacle nearby.

4.4 Results of Data Acquisition From Hardware Board

LIDAR Data

A program is implemented to rotate servo motor and map the obstacle data, the output of which is shown in Figure 4.11.

```
pi@raspberrypi ~/Project/lidar_servo $ sudo ./obs
Angle = 0 degree    Obstacle Distance = 43 c.m.
Angle = 8 degree    Obstacle Distance = 46 c.m.
Angle = 17 degree   Obstacle Distance = 42 c.m.
Angle = 25 degree   Obstacle Distance = 55 c.m.
Angle = 34 degree   Obstacle Distance = 59 c.m.
Angle = 42 degree   Obstacle Distance = 55 c.m.
Angle = 51 degree   Obstacle Distance = 55 c.m.
Angle = 59 degree   Obstacle Distance = 9999   Nearby Obstacle is at = 59
degree
Angle = 68 degree   Obstacle Distance = 9999   Nearby Obstacle is at = 68
degree
Angle = 76 degree   Obstacle Distance = 9999   Nearby Obstacle is at = 76
degree
Angle = 85 degree   Obstacle Distance = 9999   Nearby Obstacle is at = 85
degree
Angle = 93 degree   Obstacle Distance = 9999   Nearby Obstacle is at = 93
degree
Angle = 102 degree  Obstacle Distance = 9999   Nearby Obstacle is at = 102
degree
Angle = 110 degree  Obstacle Distance = 65 c.m.
Angle = 119 degree  Obstacle Distance = 57 c.m.
[6]+  Stopped                 sudo ./obs
```

FIGURE 4.11: Obstacle Distance Finding Program

Magnetometer Data

Magnetometer gives readings for x, y and z directions. Using these values we can calculate orientation angle.

Formula used to calculate orientation angle is using inverse tangent function.

$$angle = \frac{atan(\frac{x}{y}) \times 360}{\pi}$$

As car has movement is in x-y plane, no need to consider z axis value produced by magnetometer. Figure 8 shows output generated by magnetometer.

```
**************************************
x=-4096, y=-786, z=-2
angle = -169.1


**************************************

**************************************
x=-4096, y=-786, z=1
angle = -169.1


**************************************
```

FIGURE 4.12: Screen-shot of Magnetometer Program Output

Encoder Data

Rotary encoder generates two pulse trains as it rotates. Using these two pulse trains the distance and direction (i.e. forward or reverse) can be found out. Also using distance traveled we can calculate speed of robot. Every encoder has its ppr value which means pulses per revolution. Using this value we can calculate distance traveled.

Formula for calculating distance traveled is:

$$distance = \frac{count \times pi \times diameter}{ppr}$$

here diameter is diameter of wheel attached t shaft of encoder. Figure 9 shows data produced by encoder on terminal window, when its code is running.

```
Hello!
Distance Travelled = 0.000013 meters
Distance Travelled = 0.000026 meters
Distance Travelled = 0.000039 meters
Distance Travelled = 0.000052 meters
Distance Travelled = 0.000066 meters
Distance Travelled = 0.000079 meters
Distance Travelled = 0.000092 meters
Distance Travelled = 0.000105 meters
Distance Travelled = 0.000118 meters
Distance Travelled = 0.000131 meters
Distance Travelled = 0.000144 meters
Distance Travelled = 0.000157 meters
Distance Travelled = 0.000171 meters
Distance Travelled = 0.000184 meters
Distance Travelled = 0.000171 meters
```

FIGURE 4.13: Screen shot of Distance Measurement Program Output

Potentiometer Data

Potentiometer is interfaced using 10 bit ADC and the voltage is calibrated in steering angle, therefore the output is in form of angle in degrees.

To measure voltage of potentiometer we need to multiply ADC count output by ADC reference voltage and divide it by resolution of ADC.

$$voltage = \frac{count \times reference\_voltage}{2^{10}}$$

Here 10 is used because MCP3008 is 10 bit ADC.

Steering angle is calculated from voltage value by multiplying with coefficient obtained by calibrating angle and voltage physically. Figure 10 shows the output of potentiometer program, which gives voltage and steering angle.

```
************************************
Voltage of Potentiometer:  1.801529 V
Steering angle from Potentiometer:  26.059483 steering_angle

************************************

************************************
Voltage of Potentiometer:  1.801819 V
Steering angle from Potentiometer:  26.070005 steering_angle

************************************

************************************
Voltage of Potentiometer:  1.801755 V
Steering angle from Potentiometer:  26.067665 steering_angle

************************************

************************************
Voltage of Potentiometer:  1.801465 V
Steering angle from Potentiometer:  26.057142 steering_angle

************************************
```

FIGURE 4.14: Screen-shot of Steering Angle Measurement Program Output

Chapter 5

Conclusion & Future Scope

The dissertation work is based on the development of hardware setup for the implementation of the Feedback Control algorithm and Obstacle Detection Algorithm.

The hardware setup is developed using raspberry pi as main processing board. It is interfaced with all sensors which help in auto navigation and obstacle detection. This hardware setup helps for testing the algorithms for motion planner and obstacle detection.

Interfacing with encoder, gives distance traveled by robot and using mathematical formula speed can be calculated. Interfacing with magnetometer gives orientation angle of robot. Potentiometer connected to steering servo motor gives various values of voltages for change in steering angle. LIDAR module gives distance of obstacle in centimeters, by giving threshold distance for nearby and far obstacles, nearby obstacles can be differentiated.

From the sensor data acquired using these setup consisting of the magnetometer, potentiometer and optical encoder, the feedback controller is currently being implemented.

The LIDAR based obstacle detection algorithm is also under development. the obstacles will be marked on the map in run time and accordingly the path of the vehicle will be recalculated by the motion planner.

References

[1] Qayum, M.A., Siddique, N.A., Haque, M.A., Tayeen, A.S.M. Control of autonomous cars for intelligent transportation system, *International Conference on Informatics, Electronics and Vision (ICIEV)*, 2012.

[2] Adeel Akhtar, Christopher Nielsen, and Steven L. Waslander; 'SPath Following Using Dynamic Transverse Feedback Linearization for Car-Like Robots', *IEEE Transactions On Robotics*, VOL. 31, NO. 2, APRIL 2015.

[3] Kichun Jo, Junsoo Kim, Dongchul Kim, Chulhoon Jang and Myoungho Sunwoo, Development of Autonomous CarPart I: Distributed System Architecture and Development Process, *IEEE Transactions on Industrial Electronics*, Vol. 61, No. 12, December 2014.

[4] Jian-Min Wang; Sen-Tung Wu; Chao-Wei Ke; Bo-Kai Tzeng, Parking Path Programming Strategy for Automatic Parking System, *Vehicle Engineering (VE)* Volume 1 Issue 3, September 2013.

[5] Kobayashi, T. ; Dept. of Mech. Eng., Osaka Prefecture Univ., Sakai, Japan ; Ozaki, T. ; Imae, J. A New Control Design for Nonholonomic Systems Based on Controllability of Linear Systems, *IEEE/SICE International Symposium on System Integration (SII)*, 2011.

[6] Agarwal N, Walambe R., Joshi V, Rao A. "Integration of Grid Based Path Planning with a Differential Flatness Based Motion Planner in a Non-holonomic Car-type Mobile Robot" in Institution of Engineers Journal, November 2015, Pune, India

[7] Walambe R.A, Agarwal N. , Kale S. Joshi V. "Optimal Trajectory Generation for Car-Type Mobile Robot Using Spline Interpolation" in proceedings of ACODS 2016, Feb 2016, Trichy, India

[8] Jeffery Young, Milan Simic, "LIDAR and Monocular Based Overhanging Obstacle Detection", *Knowledge-Based and Intelligent Information & Engineering Systems 19th Annual Conference, KES-2015, Singapore*, Volume 60, 2015, Pages 14231432, September 2015.

[9] LU Jiangzhou, Sepanta Sakhavat, XIE Ming and Christian Laugier, "Sliding Mode Control for Nonholonomic Mobile Robot", *In Proc. of the Int. Conf. on Control, Automation, Robotics and Vision, Singapore*, December 2000.

[10] A. De Luca, G. Oriolo and C. Samson, "Feedback Control of a Nonholonomic Car-Like Robot", *Lectures Notes in Control and Information Sciences 229*. Springer, ISBN 3-540-76219-1, 1998, 343p.

[11] PulsedLight, Makers of LIDAR-Lite: 500 Hz, Assignable I2C Addresses & More, [Online] 2016, http://pulsedlight3d.com (Accessed: 13 April 2016).

[12] Wiring Pi, [Online] 2016, http://wiringpi.com (Accessed: 13 April 2016).

[13] MCP3008 8-Channel 10-Bit A/D Converter, [Online] 2016, http://shaunsbennett.com/piblog/?p=266 (Accessed: 13 April 2016).

[14] James Brundell: HMC5883L magnetometer to Raspberry Pi connection notes, [Online] 2016, http://brundell.blogspot.in/2015/06/hmc5883l-magnetometer-to-raspberry-pi.html (Accessed: 13 April 2016).

[15] Jencoder,[Online] 2016, http://jencoder.com (Accessed: 13 April 2016).

[16] Raspberry Pi 2 Model B, [Online] 2016, https://www.raspberrypi.org/products/raspberry-pi-2-model-b (Accessed: 13 April 2016).

[17] 762 RTR Sprint 2 Drift Sport, [Online] 2016, http://hpiracing.world/en/kit/762

[18] How to Setup Wi-Fi On Your Raspberry Pi via the Command Line,[online] 2016, http://www.howtogeek.com/167425/how-to-setup-wi-fi-on-your-raspberry-pi-via-the-command-line/ (Accessed: 13 April 2016).

Appendix A

Paper Presented at The International Conference

Department of Robotics and Automation Engineering
PSG College of Technology
Coimbatore - 641 004

ICAARS 2016
Make in India by Robots

International Conference on
Advancements in Automation, Robotics & Sensing
(ICAARS - 2016)

CERTIFICATE

This is to certify that Dr./Mr./Mrs./Ms. AKSHAY ADINATH DHOTRE has presented a paper in the technical session on 23.06.16 entitled DATA ACQUISITION AND HARDWARE SETUP DEVELOPMENT FOR IMPLEMENTATION OF FEEDBACK CONTROL AND OBSTACLE DETECTION FOR CAR TYPE ROBOT in the **International Conference on Advancements in Automation, Robotics and Sensing (ICAARS 2016)** organised by the Department of Robotics and Automation Engineering, PSG College of Technology, Coimbatore during 23 - 24 June 2016.

Dr. B. Vinod
Convener

Dr. R. Rudramoorthy
Principal