MAEER's

BE Project Stage-II Report

ON

PATH PLANNING FOR AUTONOMOUS VEHICLE

SUBMITTED BY:

Aditya Shahapure (71700549G)

Khemraj Ubale (71700632J)

Rushikesh Shinde (71700562D)

Project Guide:

Prof. Dr. Arti Khaparde (Internal Guide)

Sponsored by
College

Year: 2019-2020

Department of Electronics and Telecommunication
Maharashtra Institute of Technology, Pune - 38.

MAEER's

MAHARASHTRA INSTITUTE OF TECHNOLOGY, PUNE.

CERTIFICATE

This is to certify that the Project Stage-II entitled

PATH PLANNING FOR AUTONOMOUS VEHICLE

has been carried out successfully by

Aditya Shahapure (71700549G)

Khemraj Ubale (71700632J)

Rushikesh Shinde (71700562D)

during the Academic Year **2019-2020**
in partial fulfilment of their
course of study for bachelor's degree in
Electronics and Telecommunication as per the syllabus prescribed by the
SPPU.

Prof. Dr. Arti Khaparde
(Internal Guide)

Head of Department
(Electronics & Telecommunication)

DECLARATION

We the undersigned, declare that the work carried under

Project Stage-II entitled

PATH PLANNING FOR AUTONOMOUS VEHICLE

has been carried out by us and it is being not implemented by any external agency/company that sells projects. We further declare that work submitted in the form of report is not been copied from any paper/thesis/site as it is. However existing methods/approaches from any paper/thesis/site are being cited and are been acknowledged in the reference section of this report.

We are aware that our failure to adhere the above, the Institute/University/Examiners can take strict action against us. In such a case, whatever action is taken, it would be binding on us.

Roll no	Name of student	Signature with date
405072	Aditya Shahapure	
405079	Khemraj Ubale	
405073	Rushikesh Shinde	

Academic Year **2019-2020**
MIT, Pune

ACKNOWLEDGMENT

Final project is one of the most crucial parts of an engineering degree course. It is a culmination of the education we have received in the course of getting our degree. Undertaking this project has given us the ability to put the skills and technologies we have learned to good use. In doing so, we also came across many new things which provided us with an opportunity to discover and learn. However, the project would not be complete without the guidance that we have received.

We would like to thank our internal guide Prof. Dr. A. A. Khaparde for her suggestions, for sharing examples and resources of her own experience, and constant guidance. Further, we would like to thank our HOD Prof. Dr. B. S. Chaudhari for taking the time to give us advice and suggestions regarding the project. We would also like to thank Prof. Dr. A. D. Anuse for his valuable guidance and advice. Lastly, we would like to thank all the faculty members who have taught us and helped us through the years.

ABSTRACT

In recent years, autonomous/self-driving cars have drawn much interest as a topic of research for both academia and industry, which helps in better time utilization, fatigue reduction, human comfort and safety.

For a car to be truly autonomous, it must make sense of the environment through which it is driving. The car gets information about the environment using sensors such lidar, cameras, inertial sensors etc. The information from these sensors can be used together and fused to localize the car and track objects in its environment, allowing it to travel successfully from one point to another.

The process of path planning and autonomous vehicle guidance depends on three things:
Localization, mapping, and tracking objects. With advancements in the area of deep learning and incremental improvements in computing power, object detection using images outperforms other methods for the detection and classification of objects. Also, several techniques pertaining to computer vision and image processing can be used. This is possible due to various image processing algorithms.

The tasks performed by an autonomous vehicle can be even furthered by including the
detection of traffic signs. The area of image processing can be explored to develop several techniques which can detect basic traffic signs and hence make the vehicle move, stop or turn accordingly.

In our project, we aim to use various concepts from the field of image processing to achieve the dynamic detection of lane lines and traffic signs in order to enable the vehicle to move autonomously. Also, we aim to use a GPS module to set start and stop destination coordinates so that the vehicle can move accordingly to the desired location. Finally, we wish to integrate all the aforementioned three aspects of our project into one system, with which the basic objectives of path planning would be met.

INDEX

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 OVERVIEW

Autonomous Driving is one of the primary research areas in Silicon Valley as well as in top companies such as Google, NVIDIA and most famously Tesla. Autonomous vehicles include various latest technologies from different fields such as AI, electronics, communications, mechatronics etc. There is a lot of work being done in computer vision; to get the vehicle to interpret its environment similarly to humans.

In this project we want to combine the input of sensors and camera and to output proper commands to the vehicles steering and driving actuation systems. To demonstrate our methods, we are building a prototype vehicle with a single front facing camera as its primary input. To control the car along a track, there is a need to detect the line and anticipate turns. This allows to go at full speed in straight line and reduce speed before turning. This is done in two steps by processing the image coming from the camera. This is done by the system being able to predict the line curvature after its detection and then focus the approaches to spot the line center.

The image processing and control is done on the Raspberry Pi. It communicates with the Arduino that sends orders to the motors (direction and speed) using Pulse-Width Modulation (PWM). To control the car along the track, we need to detect the line and anticipate turns: this allows to go at full speed in straight line and reduce speed before turning. This is done in using the PID control concepts.

For this, the lane lines are to be detected. A generalized algorithm is used to do the same. The input in the form of an image taken from the camera is fed to an image analyser block which basically puts the captured image in a queue. The output of this block is further given to the main image processing thread which in turn performs the task of lane and object detection. The output of the image processing thread is given to the Raspberry Pi.

As far as the detection of traffic signs is concerned, a technique called “K-means Clustering” has been used. We basically have focused on the core four signs namely, “Forward”, “Stop”,

"Left" and "Right". For demonstrating the use of GPS, we aim to provide a "fork" in the straight path. Based on the current coordinates and the destination ones, the navigation algorithm shall calculate the smallest distance required to reach the target and accordingly decide whether to turn left or right.

1.2 SCOPE OF THE PROJECT

The aim of this project is to exploit the domain of image processing in order to detect lane lines and traffic signs to be further able to mechanically control the movement of the prototype vehicle, without any aid from humans. Also included in our project is the application of GPS which would make it possible for the vehicle to automatically go to the set location, calculating the least distance required to do so. We wish to finally integrate all these three aspects into one comprehensive system, thus meeting the basic requirements of path planning.

This project can be envisioned to truly run on an embedded platform, equipped with state-of-the-art high-end electronic processors and various sensors including a much more capable and powerful camera. The power of several domains such as Artificial Intelligence and Deep Learning can be also harnessed with these complex processors (for example, Nvidia's Jetson platform). Hence, the scope of our project is limited by the processing power of the central unit. Furthermore, budgetary restrictions also apply. This is why were not able to implement LIDAR and RADAR sensors even though they may prove to be more than useful in the process of path planning.

The process of traffic signs detection can be made much more efficient and expedited with the use of neural networks, which could be trained to do the same. This would have involved the use of much more complex and powerful hardware which in turn would have incurred a very high cost. All this was beyond the scope of this project, with our main computing unit, the Raspberry Pi system; falling short of achieving such a feat. We have thus limited ourselves to implement basic computer vision algorithms to come up with a way to recognize four basic traffic signs, viz. "forward", "stop", "left" and "right". But the scope could be always extended to combine the concepts of image processing with those of deep neural networks in order to come up with a much more superior and complex system.

Further, the use of GPS could be enhanced with the inclusion of an Inertial Measurement Unit (IMU). The GPS system when used alone proves to be rather disadvantageous as it provides a very low stability and

also faces a lot of issues regarding accuracy. The IMU contains three sensors, viz. accelerometer, gyroscope and magnetometer; which are capitalized on to increase the efficiency of the GPS unit, thereby increasing its accuracy and making it much more stable and less prone to random shift in its output. The usage of Kalmann Filters can also prove to be quite beneficial for achieving the same.

The Raspberry Pi also gives a live feed of the video to a nearby computer over wireless connection. An Arduino UNO board serves as an additional piece of hardware to which the motor drivers and GPS module are connected. It is the Raspberry Pi which communicates with the Arduino board; in response to input in the form of video frames. The use of a proportional-integral-derivative (PID) control logic also comes in handy when it comes to speed control whilst making turns, thereby making the process smoother and less abrupt.

Finally, we wish to demonstrate all this in an indoor environment, making use of hand-cut traffic signs and a path made of a coloured tape. The scope of this project can further be expanded to real life-sized actual vehicles, albeit it incurring a high cost and need of a much more capable computational system.

1.2 ORGANISATION OF REPORT

This report contains 8 chapters. The first one serves as an introductory part, aimed at providing an overview and scope of our project. The second one illustrates the review of the literature and present scenario in the industrial environment. The third chapter starts with the list of hardware and software used, moving on to the description and block diagram of the whole system. It also discusses the many complexities which were involved in this project. The fourth chapter deals with the system design. It also includes specifications of the components used, a circuit diagram and the related PCB layouts. Further, the fifth chapter shows the system flowchart. The sixth one includes observations of the system. The conclusion is provided in the seventh chapter. Finally, the eighth chapter talks about future scope of the project. References too have been mentioned at the end.

Chapter 2

REVIEW OF LITERATURE

2.1 LITERATURE SURVEY

The vast and ever-expanding topic of self-driving vehicles is at the forefront of research and development in many of the world's leading automobile industries. Being a diverse field, it involves the disciplines from various areas of science, design and engineering. Design of such autonomous systems have been spearheaded by the recent developments in the field of Artificial Intelligence, machine learning and image processing, pertaining to the field of computer science and engineering. The sources of our study range from various research papers and journals to several internet articles, videos and websites.

AutoRally [1] is an open source self-driving car platform based on a vehicle which uses predictive control to steer the vehicle. It primarily uses a camera and IMU. [2] is one of the first autonomous vehicle implementation projects, which uses a neural network architecture with camera input to drive the vehicle.

A lot of research has gone into path planning of mobile robots. [3] is a great survey paper which discusses and compares a variety of path planning techniques. [4] uses an ROS (Robot Operating System) and a TEB (Time Elastic Bands) based local planner on a campus vehicle. [6] has plenty of relevant information as well, especially about localization and different methods for MOT (Moving Objects Tracking). [7] is another autonomous robot-based student project which was very useful as a reference. [8] is a paper that describes Stanley, the autonomous vehicle which won the 2004 DARPA challenge. [9] was a great starting point to learn about real time lane detection.

[11] provides some of the best explanations for a GPS-based path planning for a rover. It covers the basics of related math and also talks about the Haversine formula. The concept of image processing is one of the most crucial topics when it comes to the use of computer vision in autonomous vehicles. [12] is a research paper which delves into this topic and provides an insight into the real time lane detection using image processing.

2.2 PRESENT SCENARIO

The field of autonomous vehicles and the research work concerning the same has become one of the most sought-after topics in today's era. An American automotive giant named Tesla Motors has become one of the most prominent companies spearheading the research and development of self-driving cars. Unlike its contemporaries, Tesla does not use LIDAR sensors and relies mainly on its vision system and RADAR. Tesla Autopilot is a suite of advanced driver-assistance system features offered by Tesla that has lane centering, adaptive cruise control, self-parking, automatic lane changes, semi-autonomous navigation on limited access freeways, and the ability to summon the car from a garage or parking spot. As an upgrade to the base Autopilot capabilities, the company's stated intent is to offer full self-driving (FSD) at a future time, acknowledging that legal, regulatory, and technical hurdles must be overcome to achieve this goal. Various universities such as CMU also have their own self-driving car research projects.

The research for self-driving cars kicked off with the 2004 DARPA (Defense Advanced Research Projects Agency) challenge which is an autonomous vehicle racing competition. The SDVs (Self Driving Vehicles) gained public recognition through the three DARPA challenges in 2004, 2005, and 2007; which resulted in the establishment of four SDV characteristics: sensing, perception, planning and control. Sensors are used to take raw data measurements, which are transformed by the perception component into usable information. The planning component creates a path based on that information, and the control component contains the actuators to drive the car (based on the planned path, through direct sensing, in order to avoid obstacles). A combination of camera, radar and laser systems are used to retrieve data about the environment. For the position and motion of the car, SDVs are equipped with satellite navigation, inertial and odometry measurements. Most of the teams were failures but in the subsequent such challenges, the autonomous vehicles became better and better. In 2009, the tech megacompany Google hosted their own DARPA challenge.

Since then self-driving cars research has accelerated at a rapid pace with lots of Robot race competitions, hobby groups and many open source projects promoting the same.

Great advancements are being made towards road perception by transforming road shapes and markings from 3D estimates into 2D images. There have been developments in computer vision to interpret traffic lights and signals, but more work is required. Advancements in computer vision, combined with LiDAR technology, has enabled self-driving vehicles to overcome many issues related to poor performance at night, ambient lighting conditions, and bad weather conditions. Developments in electronic mapping have aided the car's navigation by incorporating geographical characteristics, traffic information, building information, and traffic signs. The satellite navigation systems – GPS, GLONASS, Galileo etc. have spurred developing of electronic maps. The navigation process of SDVs can be classified into four sections: route planning, behavioural decision-making, motion planner, and vehicle control.

There have been vast improvements in higher machine learning capabilities, with much of the cognitive automation being done with advanced deep learning and neural network-based models. There have been great developments used in hardware of the past number of years, allowing for faster processing of neural networks. Motion planning determines the best path for a car to take, comfortable for the passenger, while avoiding collision.

The field of autonomous vehicles is a buzzing with a lot of technological development already, but its best days are yet to come, taking into accounts multiple barriers such as safety, security, technical, political, economic, geographic etc.

Chapter 3

SYSTEM DEVELOPMENT

3.1 SYSTEM SPECIFICATIONS

Software:

- Python/C++
- Anaconda IDE
- Jupyter Notebook
- Libraries such as OpenCV cv2, Numpy, Matplotlib, Keras, scipy, time, math
- Pyfirmata
- VNC
- Arduino IDE
- EasyEDA

Hardware:

- Raspberry Pi 3B+
- Pi Cam
- 4WD RC rover
- Arduino UNO
- Motor Driver L298D
- DC motors
- GPS NEO-6M
- HMC5883L magnetometer compass
- LM 2596 buck convertor
- 11.1V/1500mAh Li-Po Battery
- Portable power bank

3.2 SYSTEM DESCRIPTION

Our main system is a concept vehicle built by modifying an RC car, making its functioning autonomous. The primary input to the system is a single front facing camera (pi cam), connected to the raspberry pi central processor. This setup is used to detect lane lines and traffic signs using computer vision. The GPS module along with an Inertial measurement unit forms another input device, which provides coordinates of the location set for the car to move to. The data from these devices are used by the vehicle to navigate autonomously. The camera is also used for detection of basic traffic signs, using the concepts from image

processing and computer vision. This enables the car to either move forward, turn right and left; or stop completely, according to the sign detected. Testing of the system has been done on a prebuilt track.

3.3 BLOCK DIAGRAM:

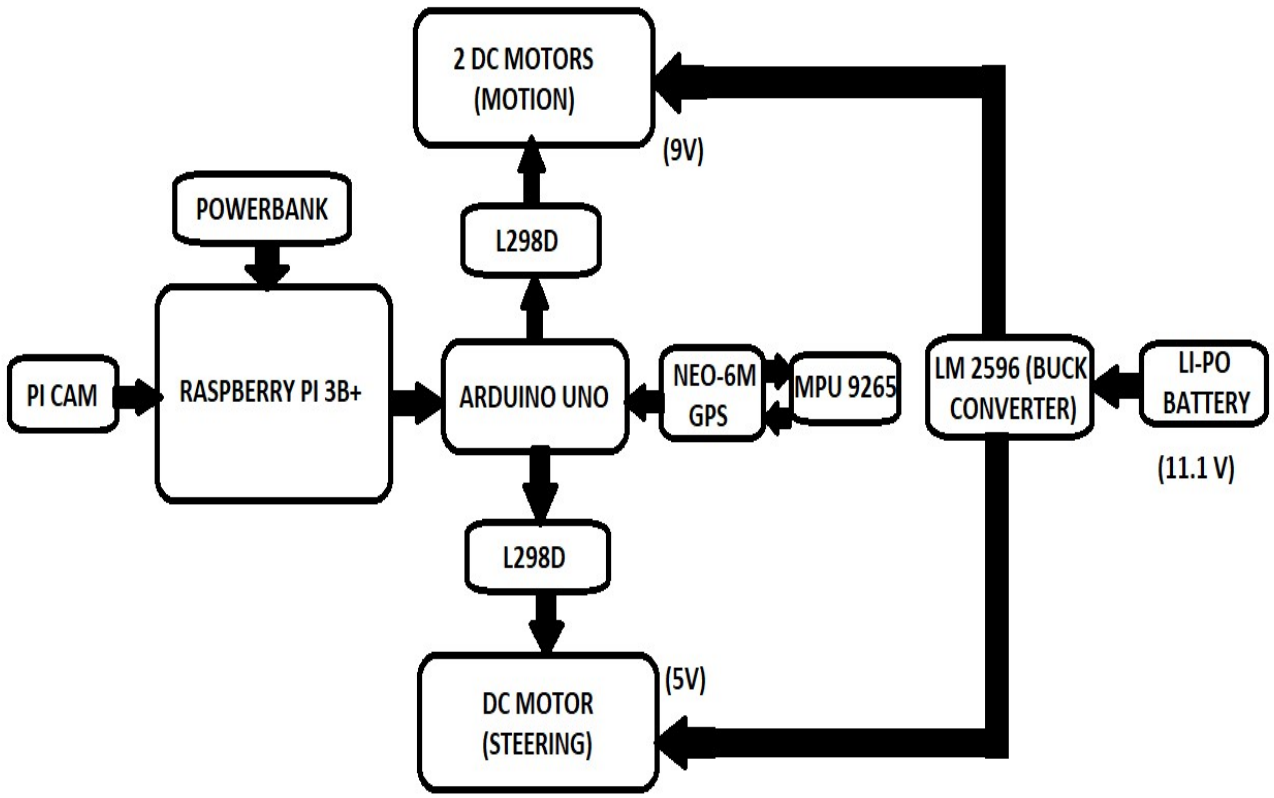


Figure 3.2: System Block Diagram

3.4 COMPLEXITIES INVOLVED:

1) Selection of main controller:
   We initially used a raspberry pi 2B but soon realised that it proved to be quite inefficient when it came to testing. Switching to a raspberry pi 3B+ version made it possible to use Wi-Fi so that the live feed given by camera could be seen sitting at a reasonable distance, without any need to use cables. Also, the 3B+ boasts of a higher processing power as compared to the earlier 2B version.

2) Faulty picam:

   We had to purchase the camera module twice as the first one we used turned out to be faulty after two or three experiments.

3) Camera attachment:

   We faced several difficulties while trying to figure out a way in which we could fix the camera in front of the vehicle. Also, the field of view needed to be taken into consideration along with an optimum height from the ground.

4) Failure to generate the required starting torque for motors:

   We initially used the l293d motor driver, which kind of failed to meet the requirements.

   It was incapable of giving a high starting torque to the motors, meaning that the vehicle needed to be given a slight push in order to start moving.

5) Damage to l293d motor driver:

   Our first motor driver, the l293d was subjected to failure due to overheating, which went previously unnoticed. We then switched to a new motor driver, the l298d, which has its own advantages. The latter turned out to be quite superior to the former in terms of power handling capacity and efficiency.

6) Vibrations while moving:

   The vehicle was subjected to a lot of vibrations whilst moving. This caused the camera to inaccurately detect lane lines.

7) GPS related issues:

   The GPS accuracy proved to be quite low, thanks to the large circle of error, the radius of which went upwards to some 5 or 6 metres. The accuracy also depends on the availability of satellites.

8) IMU related issues:

   The magnetometer readings keep on accumulating error over time. Also, the magnetometer itself is quite susceptible to magnetic forces.

Chapter 4

SYSTEM DESIGN

4.1 Hardware:

The prototype vehicle should have characteristics which are as close as possible to a real vehicle. This includes realistic drive mechanism, steering mechanism (Ackermann steering), proper sensors for autonomous driving and a central processor which can handle all the sensor input data as well as perform operations on it in real time.

For this purpose, our system design is as described in Figure 3.2. We use the base of an RC car since it has the mechanical structure required for Ackermann steering already built in.

Sensors-

- Since the autonomous function of the car will be built around vision, the camera is the most important sensor. The important specifications of the camera are the FOV, resolution and the bit rate. It also has to be compatible with the control unit (Raspberry Pi). For this purpose, we have used the Pi cam.

- IMU sensor (Inertial Measurement Unit) is a device that is used to obtain information about movement of a vehicle. It provides the vehicle's orientation, angular rate using accelerometers and gyroscopes. The MPU-6050 is a cheap IMU which contains 3-DOF accelerometer and gyro. The MPU 9265 can also be used. These IMU devices are used for odometry.

- GPS sensor can be used for localization and global path planning.

Actuation-

- A DC motor is used to control the steering mechanism of the vehicle. It can be directly controlled by analog pins of the Arduino. The Raspberry Pi will provide results of its computation to I2C slave device Arduino which in turn will provide signal to servo motor as required.

- Generic 120rpm geared DC motors along with PD control are used for actuation of wheels of prototype. They are controlled using Arduino through a motor driver. Motor driver L298d is a suitable device for this purpose. These motors require up to 12V/2A input.

Control Unit-

- We use Raspberry Pi 3B+ as our main control unit to which the pi cam is directly connected. Input from this camera is sent directly to the Pi. Most of the computations pertaining to image processing take place inside the Raspberry Pi itself.
- Raspberry Pi is not suitable for real-time applications such as actuation. For this reason, an Arduino Uno is connected to it as an I2C slave device. The Raspberry Pi feeds the results of its computations to the Arduino slave which in turn controls the motors.
- Also connected to the Arduino are the GPS and IMU units.

Power Supply-

- An 11.1V Li-Po battery is used, which is connected to a buck converter.
- The buck converter gives two outputs, one of 9V for the two DC motors used for forward motion and the other one of 5V for another DC geared motor used for steering wheels.
- Also used are two motor drivers, each one capable of driving two motors. These motor drivers are connected to the Arduino board, which in turn is powered by the Pi.
- A portable power bank is used to power the raspberry pi system.

4.2 Computer vision:

The vast field of image processing has been explored and put to use for achieving majority of our objectives. This is possible thanks to a powerful computer vision library called OpenCV which works with python programming language.

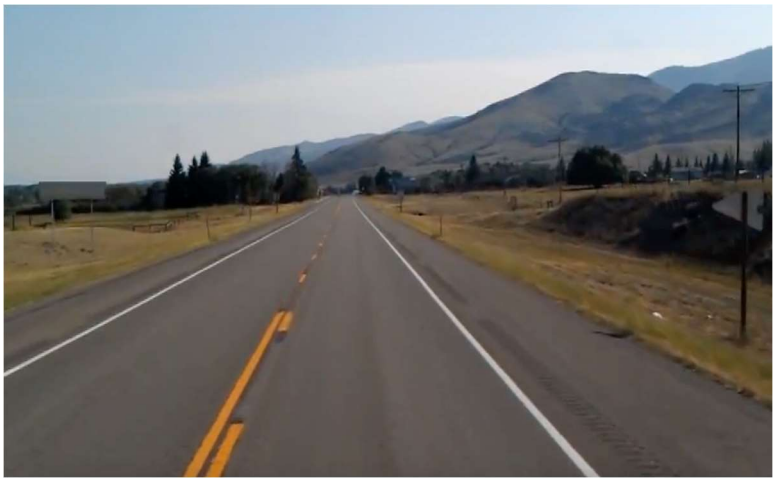
4.2.1 Lane Detection-

Detection of lane lines is the foremost of all objectives as these lines serve as a reference for the vehicle's motion planning. Several steps are involved in achieving the same.

STEP 1) To load the test image:

The camera starts recording live video of what is in front of it and sends the data to the raspberry pi in the form of individual frames. These individual frames are then processed to identify the lane lines.

For the sake of testing, the sample image used was:



STEP 2) To convert the sample image to HSV color space:

After taking video recording as frames from the camera, the next step is to convert each frame into Hue, Saturation, and Value (HSV) color space. The main advantage of doing so is to be able to differentiate between colors by their levels of luminance.

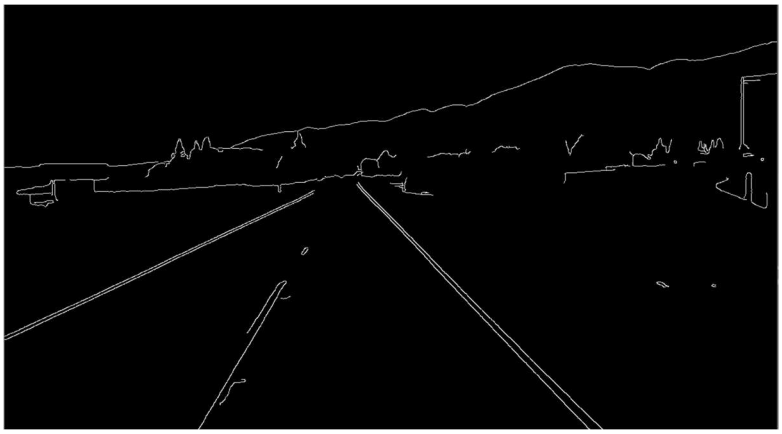
The test image after being converted into HSV space:

STEP 3) Detecting Blue Color and Edges:

After converting the image into HSV color space, we need to detect the color of our interest, which is blue in this case. To extract blue color from an HSV frame, a range of hue, saturation and value should be specified. Further, to reduce the overall distortion in each frame, edges are detected using canny edge detector.

For the purpose of testing, we did not consider to extract the blue color as the test image contained yellow lanes. Using the Canny () function, derivative of smoothened image is obtained. The image obtained post the derivative operation basically represents gradient. An upper and lower threshold limit is manually set and if the gradient is higher that upper threshold, it is an edge. If the gradient is between upper and lower thresholds, then it is accepted as an edge only if the neighboring pixel is an edge.

The edge-detected image is:

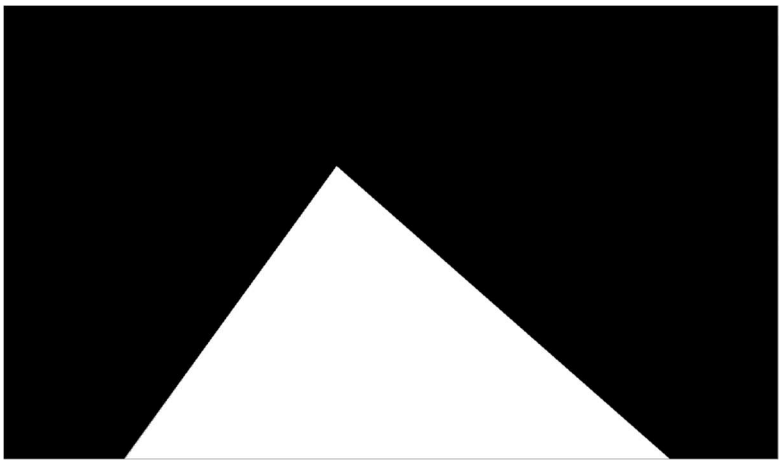


STEP 4) To find the ROI ("Region of Interest"):

Region of interest is basically isolating that area of the image which will contain the required lane lines. This can be decided by testing based on camera angle. By isolating only the required area, we can remove excess noise and edges that are irrelevant to our problem.

We use the "matplotlib" library included within OpenCV to clarify how we isolate this region of interest. The sub-package used is "pyplot", which displays the image along with the required graph. We basically limit our field of view based on the ROI; which ultimately traces a triangle with the calculated vertices. This triangle isolates the region where we find the lane-line.

The region of interest is, given by the white triangle is:



STEP 5) Line detection using the Hough Transform:

The Hough Transform is a technique that detects straight lines and thus identifies the lane-lines. The idea of identifying possible lines from a series of points is how we find lines in the gradient image. The gradient image is basically a series of white points which represent the edges in the image-space.

1. Split the Hough-space into a grid. Each bin inside the "bin" corresponds to the slope(m) and y-intercept(b) value of a candidate line.
2. All the points of intersection of the lines representing those four points of cartesian space (image space) lie in a single bin of the grid present in Hough-space.
3. For every point of intersection, a "VOTE" of the bin in which that point belongs to, is cast. The bin with the maximum number of votes will represent the slope(m) and y-intercept(b) of the line we desire. The line we need is the one which passes through the maximum number of points (or nearly all) , present in the image plane.
4. Whatever the "m" and "b" value this belongs to, that's the line we are going to draw since it was voted as the line of "BEST-FIT"; in describing our data.

The below figure 4.2.1 shows Hough transform. A point in image space gets converted to its corresponding curve in Hough space. In our case, a line in image space is given by its polar equation. The 'rho' and 'theta' values corresponding to the point of intersection in the Hough space (red dot as seen in the figure below) gives the parameters of the detected line in the image space.

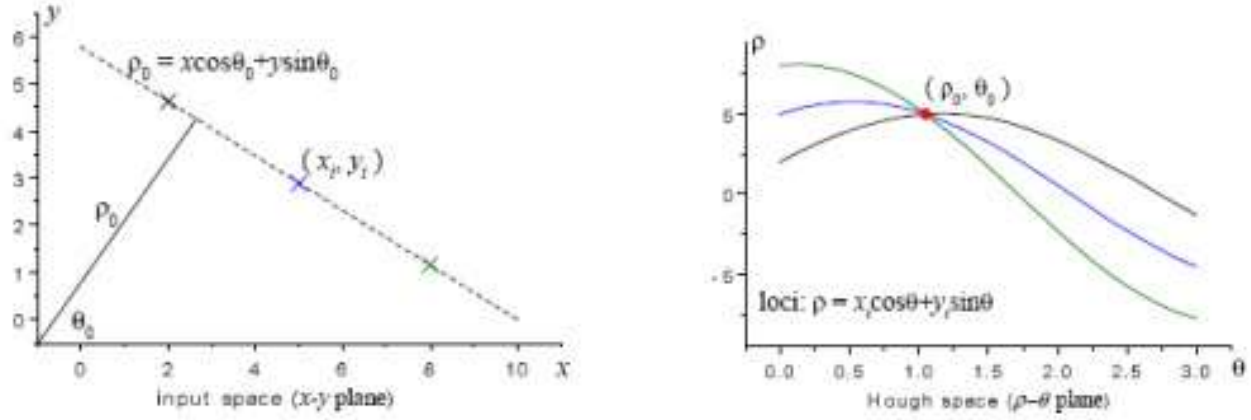


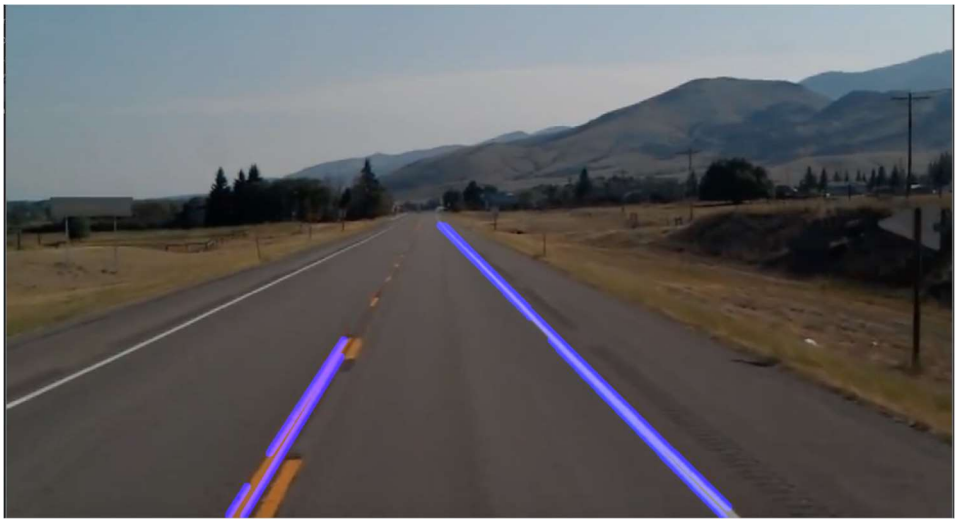
Figure 4.2.1: Hough transform

When vertical line is encountered in cartesian space, its slope would be infinity. That is why polar space is used instead of cartesian. In this case the intersection point in Hough space gives the value of rho and theta for line with most collinear points in image space and that line is chosen as 'BEST-FIT'.

The equation of a line is given by y = mx + b, where 'm' is the slope and 'b' is the y-intercept. The average of slopes and intercepts of line segments detected using Hough transform is to be calculated. The left lane appears to be going upwards so it has a negative slope. The left lane line has x1 < x2 and y2 < y1 and the slope = (y2 - y1) / (x2 - x1) which will be negative. Therefore, all lines with negative slopes are considered left lane points. The right lane is the complete opposite, it can be seen that the right lane is going downwards and will have positive slope. Right lane has x2 > x1 and y2 > y1 which will give a positive slope. So, all lines with positive slope are considered right lane points. In case of vertical lines (x1 = x2), the slope will be infinity. In this case, we will skip all vertical lines to prevent getting an error. To add more accuracy to this detection, each frame is divided into two regions (right and left) through 2 boundary lines. All width points (x-axis points) greater than right boundary line, are associated with right lane calculation. And if all width points are less than the left boundary line, they are associated with left lane calculation. To prevent dividing by 0, a condition is presented. If slope = 0, which means y1 = y2 (horizontal line), we give the slope a value near 0. This will not affect the performance of the algorithm as well as it will prevent an impossible case to occur (dividing by 0).

The OpenCV function called "HoughLinesP" does the job of only detecting the lines. We need to define one more function which shall display those lines. The idea here is to create a black-colored image mask again, which is of the same size as that of the original image; and display the detected lines on it.

The image thus obtained post the application of Hough transform:



STEP 6) Calculating and displaying heading lines:

This is the final step before we apply speeds to the motors. The heading line is responsible to give the steering motor the direction in which it should rotate and give the throttling motors the speed at which they will operate. Calculating heading line is pure trigonometry. Some extreme cases are when the camera detects only one lane line or when it doesn't detect any line.

Case 1: turning right:

As can be seen in the figure 4.2.1 below, 'x_offset' is basically how much the average ((right x2 + left x2) / 2) differs from the middle of the screen. 'y_offset' is always taken to be height / 2.

x_offset = (x2_l + x2_r)/2) – (mid of frame)

y_offset = (height/2)

The angle "theta", as can be seen the figure below, is basically the one formed by the central vertical line and the segment joining the frame's center (M) to the point ((x2_l + x2_r)/2, height). The "theta" basically is the angle of deviation. It is given by:

Theta = tan^-1(x_offset/y_offset),

The actual steering angle is 90 degrees added to theta (after converting theta to degrees).

Steering angle = theta(degrees) + 90(degrees)

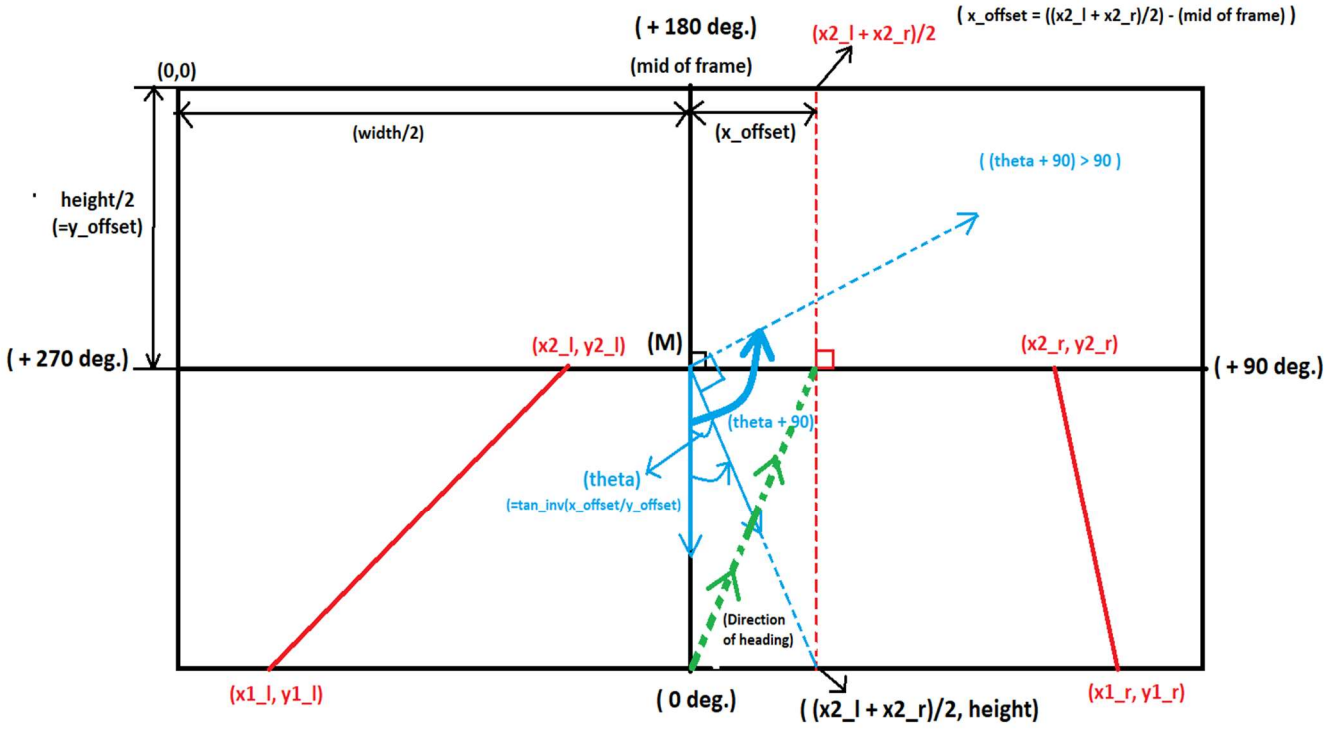


Figure 4.2.2: Heading direction in case of right turn

Now, if the steering angle is greater than + 90 degrees, then the car should take a right turn. (The direction of heading is shown by the dashed green line in the above figure). This theory is supported by the fact that had theta been 0 degrees, then the steering angle would have been (theta + 90) = (0 + 90) = 90 degrees, in which case the steering angle would have been perpendicular to the central vertical line. This would have made the dashed green line perpendicular to the frame's width, meaning that it would have coincided with the central vertical line, causing the vehicle to move forward without taking any turns.

The two end coordinates of the dashed green line giving the heading direction are given as:

(((width/2), height), ((mid of frame + x_offset), (height/2))).

Case 2: turning left:

Likewise, if the steering angle turns out to be less than + 90 degrees, then the car should make a left turn, as shown in the figure 4.2.3 below. In this case, theta will be negative and hence when added to 90 will yield a value less than + 90 degrees. This is because x_offset will be negative as in this case ((x2_l + x2_r)/2) is less than the mid of frame.

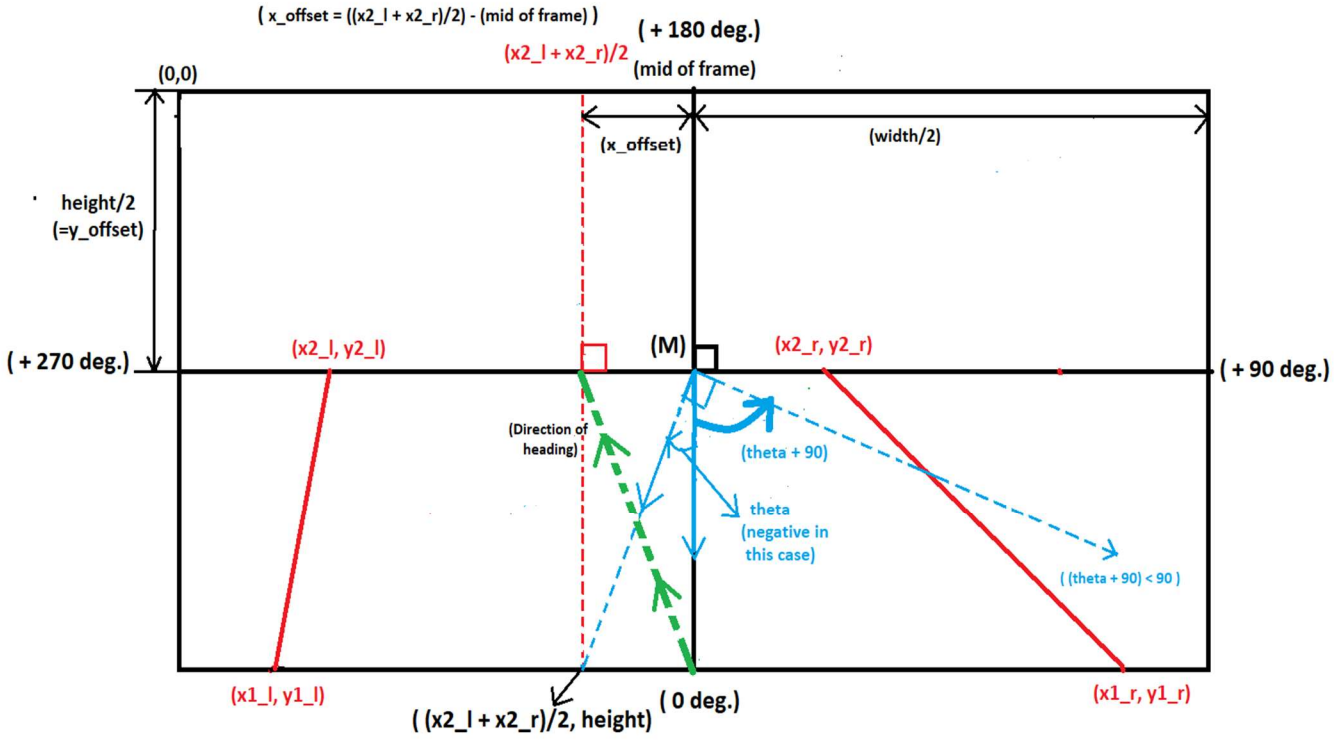


Figure 4.2.3: Heading direction in case of left turn

STEP 7) Applying PD control:

After getting the steering angle, it needs to be fed to the motors. As mentioned earlier, if steering angle is greater than 90, the car should turn right otherwise it should turn left. We applied a simple code that turns the steering motor right if the angle is above 90 and turns it left if steering angle is less than 90 at a constant throttling speed of 10% of PWM but got a lot of errors. The main error was when the car approached any turn, the steering motor acted directly but the throttling motors (the ones responsible for motion) got jammed. We then tried to increase the throttling speed to 20% of PWM at turns, but ended with the vehicle getting out of the lanes. There arose a need for something that increases the throttling speed a lot if the steering angle is very big and increases the speed a bit if the steering angle is not that big; then decreases the speed to an initial value as the car approaches forward direction (moving straight). The solution was to use a PD controller.

PID controller stands for Proportional, Integral and Derivative controller. This type of linear controller is widely used in robotics applications. The figure 4.2.4 below shows block diagram of a typical PID controller. The goal of this controller is to reach the "setpoint" (desired value for the system to reach) with the most efficient way unlike "on/off" controllers which turn on or off the system according to some conditions.

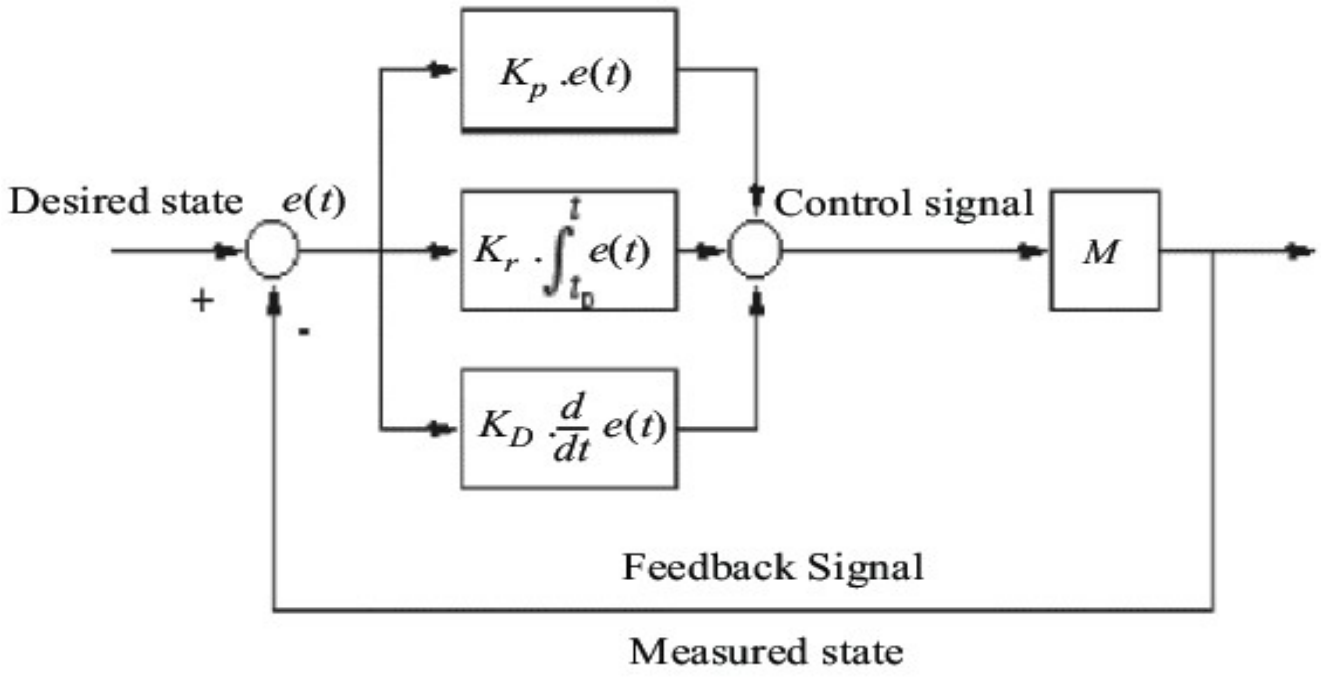


Figure 4.2.4: PID controller block diagram

The controlled variable in our case is the throttling speed (since steering has only two states, either right or left). A PD controller is used for this purpose since ‘D’ (derivative) action increases the throttling

speed a lot if the error change is very big (i.e. large deviation) and slows down the car if this error change approaches 0. Following are the steps to implement a PD controller:

1) Set the setpoint to 90 degrees (The car needs to always move straight)
2) Calculate the deviation angle ("theta" above)
3) The deviation gives two information: How big the error is (magnitude of deviation) and what direction the steering motor has to take (sign of deviation). If deviation is positive, the car should steer right otherwise it should steer left.
4) Since deviation is either negative or positive, an "error" variable is defined and is always equal to the absolute value of deviation.
5) The error is multiplied by a constant 'Kp'
6) The error undergoes time differentiation and is multiplied by a constant 'Kd'.
7) The speed of the motors is updated and the loop starts again.

If the error is noticeably big (or the deviation from middle is large), then the proportional and derivative actions are high, resulting in a high throttling speed. When the error approaches 0 (deviation from middle is less), the derivative action acts in a reverse fashion (slope is negative) and the throttling speed gets low to maintain stability of the system.

4.2.2 Traffic sign detection-

The concepts from the field of image processing can be further used to detect basic traffic signs. In our case, we have designed our code to identify four basic signs which include "forward", "stop", "left" and "right". After processing these images, the raspberry pi gives commands to the Arduino, which accordingly controls the motors.

This is accomplished by using "k-means" clustering algorithm, one of the most fundamental algorithms in machine learning.

In k-means clustering, the goal is to partition 'n' data points into 'k' clusters. Each of the 'n' data points are assigned to a cluster with the nearest mean. The mean of each cluster is called its "centroid" or "center". Overall, applying k-means yields k separate clusters of the original n data points. Data points inside a specific cluster are considered to be more similar to each other than data points that belong to other clusters.

In case of computer vision applications, we cluster the pixel intensities of an RGB image. Given an MxN size image, we thus have MxN pixels, each consisting of three components: Red, Green, and Blue respectively. These MxN pixels are treated as data points and are to be clustered using k-means. Pixels that belong to a given cluster will be more similar in color than pixels belonging to a separate cluster.

The main driving force behind all this is an OpenCV function called "kmeans()". The k-means clustering algorithm is used to extract the dominant color in the desired area. Since this function takes a 2D array as input, we need to flatten our input image which is given by 3 dimensions, which are width, height and depth of 3 RGB values, into a single vector of three pixel values Red(R), Green(G) and Blue(B). There are 3 output parameters in case of this function:

1. **compactness**: It is the sum of squared distance from each point to their corresponding centers.

2. **labels**: This is the label array (same as 'code' in previous article) where each element marked '0', '1', '2'….'k'.

3. **centers**: This is array of centers of clusters

We then list the labels and their frequency. The centroid corresponding to the label with maximum frequencies is returned by using the "itemfreq()" function.

All the above is done by a user-defined function named "get_dominant_color()", which takes in two arguments, the image itself and the desired number of clusters.

Another OpenCV function which plays a crucial role is the "HoughCircles()", which is used for detecting circles in an image. This works in a manner similar to that of function which was used for detecting straight lines. But in this case, we are left with 3 unknown parameters namely, x-coordinate of centre, y-coordinate of centre and the radius, as against 2 in case of straight lines (y-intercept and slope). This means that the parameter space in this case is 3D, as against 2D (in the earlier case of lines). Each horizontal plane in the parameter space would be equivalent to a 2D parameter space where R is known. The Circle Hough Transform is a bit inefficient at detecting circles, so it uses the gradient method of detecting circles using hough transform. This function considers parametric form of the equation of a circle.

A circle is given by the following equation: (x-a)^2 + (y-b)^2 = r^2.

The same equation when written in parametric form becomes:

$x = a + R\cos\theta$

$y = b + R\sin\theta$

The following figure shows a circle in both x-y cartesian plane (image space) and in hough or parametric a-b plane (parameter space):

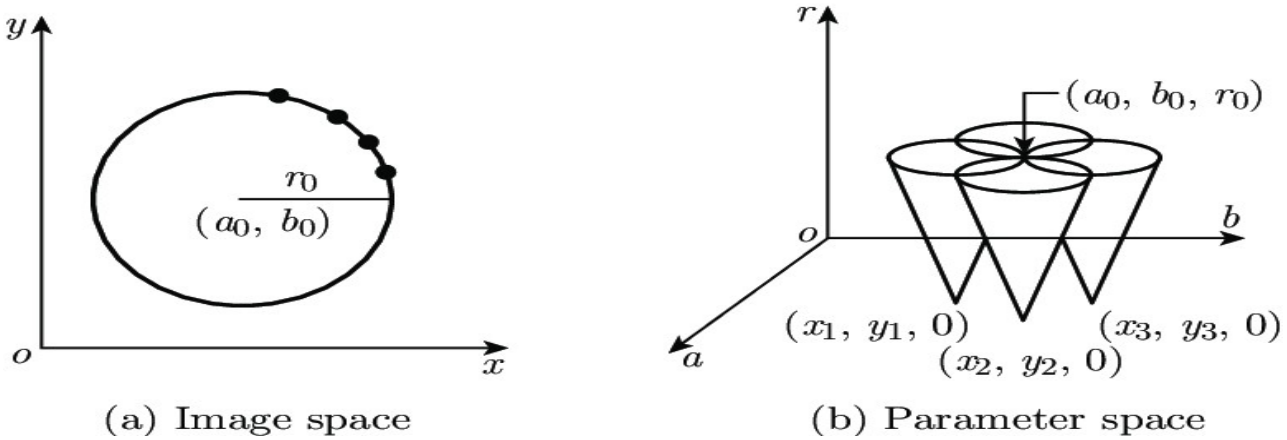


Figure 4.2.5: Image & para. space for circle hough transform

The output given by this function is a vector which gives the x and y coordinates of the centre along with the radius of the detected circles ([x,y,radius]). The below figure shows an example of a traffic sign (forward):

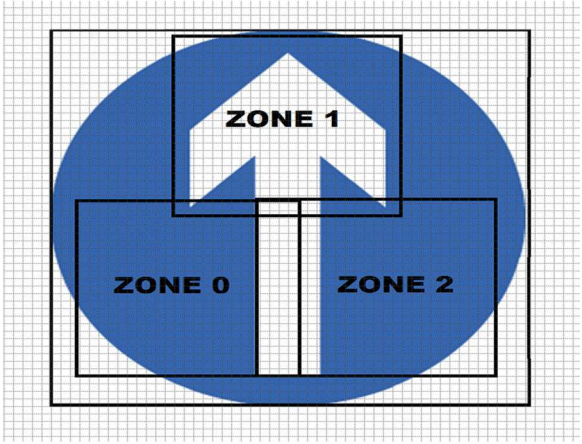


Figure 4.2.6: "Forward" sign (example)

Algorithm:

1) Perform the following steps as long as the camera is recording.

2) Convert the current image frame into its grayscale version.

3) Apply a blur filter to smoothen the image, so that it doesn't get subjected to noise while the process of edge detection.

4) Use the HoughCircles() function to detect circles in the image.

5) Find the circle having the largest radius which would basically be the traffic sign's shape.

6) If the x coordinate and y coordinate values of the above circle's center are both greater than its radius, then define a variable "square" which inscribes this circle. (There cannot be a case where y<r or x<r as then the circle would exceed the bounds of the frame, meaning that the whole circle can't get detected due to not being able to fit inside the frame).

7) Call the user-defined function “get_dominant_color()” and let the value returned by this be assigned to a variable named “dominant_color”. The "get_dominant_color" function returns the coordinates of the centroid (B,G,R) of the cluster containing the pixels with the dominant color. This is an array [blue_value, green_value, red_value].

8) if dominant_color[2] > 100, then print “STOP”. (if red_value > 100, then the image is dominantly red in color)
   else if dominant_color[0] > 80 (blue is the dominant color), divide the image into 3 zones (as can be seen in the above figure 4.2.5):
   Call the “get_dominant_color()” function again to extract the dominant color from each of the 3 zones.
   if zone_1_color[2] < 60 : (Red is negligible)
   if sum(zone_0_color) > sum(zone_2_color), then print “LEFT”
   (in case of “left” arrow, the B+G+R value of zone0 will be greater than that of zone2)
   else, print “RIGHT”
   else if sum(zone_1_color) > sum(zone_0_color) and sum(zone_1_color) > sum(zone_2_color), then print “FORWARD”.
   Else, print “N/A”.

9) Give commands to the vehicle according to the detected signs.

4.3 GPS navigation:

GPS navigation requires two major components: A GPS sensor to determine latitudinal and longitudinal position and a compass, to give direction of robot in earth plane. The sensors used are GPS NEO-6M and HMC5833L magnetometer. The GPS sensor gives position in various formats such as degrees or minutes and seconds. The GPS sensor receives data in NMEA (National Marine Electronics Association) format from satellites. The positional accuracy of GPS depends on number of satellites it can connect to. Our program extracts the GPS coordinates from the NMEA format and gives us latitude and longitude in degrees. From here we use these coordinates in our GPS navigation algorithm.

Magnetometer HMC5883L measures the direction and magnitude of the Earth’s magnetic field and hence is used for low cost compassing and magnetometry. It measures the Earth’s magnetic field value along the X, Y and Z axes from milli-gauss to 8 gauss. It can be used as a compass to find direction or to find the direction of heading of a device.

The HMC5883L uses a magnetoresistive sensor arranged in a bridge circuit, which is made of nickel-iron (Ni-Fe magnetic film) material. Magnetoresistance is the tendency of a material to change the value of its electrical resistance in an externally applied magnetic field. The correspondent movement of the nickel-iron material in space experiences earth’s magnetic field which changes the material’s resistance, and hence we get resultant voltage changes across the bridge. This change in voltages is used to get the magnetic field direction in space.

Algorithm:

1) Extract current location in degrees

2) Enter goal location

3) The distance and deviation between these two points is calculated

4) Although Latitude and Longitude are like x, y coordinates in the graph, two-dimensional formulas cannot be used because Earth is a sphere, not a plane. To overcome this, Haversine formula is used. This has some degree of approximation as Earth is not a perfect sphere. The Haversine formula determines the great-circle distance between two points on a sphere, given their latitudes and longitudes.

$$d = 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)}\right)$$
$$= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Where, ‘d’ is the distance between the two points along a great circle of the sphere and ‘r’ is the radius of the sphere. Also, ‘φ1’, ‘φ2’ are the latitude of point 1 and latitude of point 2 (in radians) respectively and ‘λ1’, ‘λ2’ are the longitude of point 1 and longitude of point 2 (in radians) respectively.

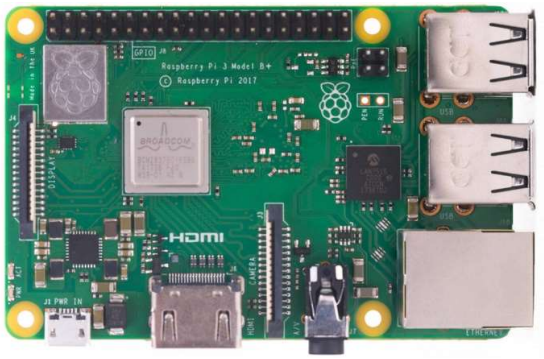
5) The compass gives the current heading of vehicle.

6) The following steps are looped till distance between current location and goal is less than a buffer of 1.5-2 meters at which goal location is ‘REACHED’:

   - Distance between current location and goal location is calculated
   - If distance < buffer: STOP or else CONTINUE
   - Deviation angle is calculated as such: current heading – bearing
   - If 0 < deviation < 90 or (– 270) < deviation < (-180)
     - Turn left
   - If (-90) < deviation < 0 or (-180) < deviation < (-90)
     - Turn right
   - Else
     - Go straight

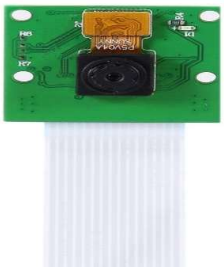
4.4 Component specifications:

1) Raspberry Pi 3B+:



- Quad core ARM Cortex-A53 64-bit processor clocked at 1.4GHz
- 1GB LPDDR2 SRAM
- Dual-band 2.4GHz and 5GHz wireless LAN
- Bluetooth 4.2
- Higher speed ethernet up to 300Mbps
- GPU: Broadcom Videocore-IV
- 40-pin GPIO
- Storage: Micro-SD
- Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- Operating voltage: 5V
- Recommended PSU current capacity: 2.5A

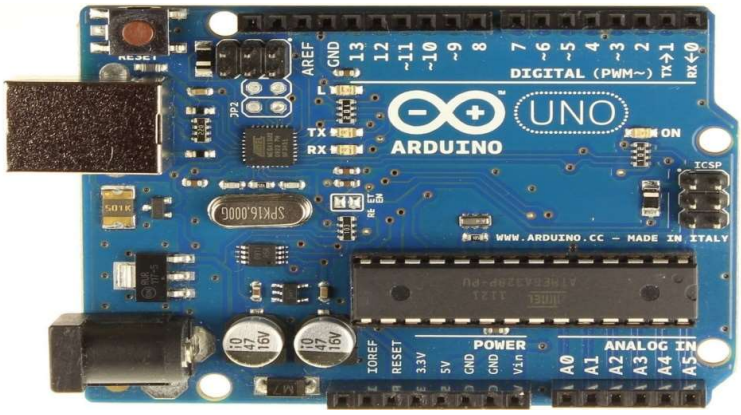
2) Pi camera module:



- Resolution: 5 Megapixels
- Video modes: 1080p/30fps, 720p/60fps and 480p/60/90fps
- Sensor: OmniVision OV5647
- Sensor resolution: 2592 x 1944 pixels

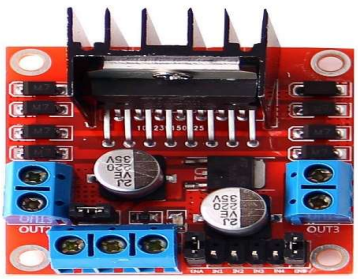
- Sensor image area: 3.76 × 2.74 mm
- Pixel size: 1.4 µm × 1.4 µm
- S/N ratio: 36 dB
- Focal length: 3.60 mm +/- 0.01
- Horizontal field of view: 53.50 +/- 0.13 degrees
- Vertical field of view: 41.41 +/- 0.11 degrees

3) Arduino UNO:



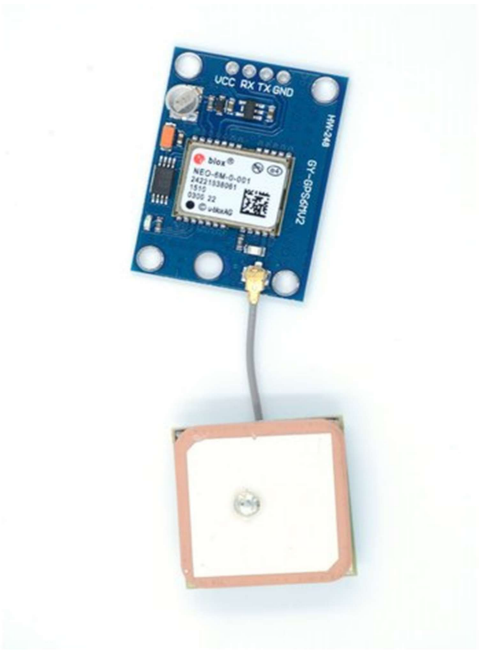
- Microcontroller: 16Mhz 8-bit Atmega 328p
- Operating Voltage: 5V
- Recommended Input Voltage: 7-12V
- 6 Analog Input Pins (A0-A5)
- 14 Digital I/O Pins (Out of which 6 provide PWM output)
- DC Current on I/O Pins: 40 mA
- DC Current on 3.3V Pin: 50 mA
- Flash Memory: 32 KB (0.5 KB is used for Bootloader)
- 2 KB SRAM
- 1 KB EEPROM

4) L298d Motor driver:



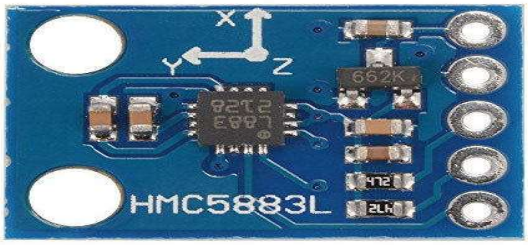
- Driver Chip: Double H Bridge L298N
- Motor Supply Voltage (Maximum): 46V
- Motor Supply Current (Maximum): 2A
- Logic Voltage: 5V
- Driver Voltage: 5-35V
- Driver Current: 2A
- Logical Current: 0-36mA
- Maximum Power (W): 25W
- Current Sense for each motor
- Heatsink for better performance
- Power-On LED indicator

5) GPS NEO-6M:



- 5Hz position update rate
- Operating temperature range: -40 TO 85°C
- EEPROM to save configuration settings
- Rechargeable battery for Backup
- The cold start time of 38 s and Hot start time of 1 s
- Supply voltage: 3.3 V
- Configurable from 4800 Baud to 115200 Baud rates. (default 9600)

6) HMC5883L magnetometer compass:



- Supply voltage: 3.3V-6V
- Heading accuracy: 1° to 2°
- Magnetic field range: -8 to +8 Gauss
- Max data rate: 160 Hz
- 12-bit ADC

7) LM 2596 buck converter

- Switching frequency: 150KHz
- Load Regulation: ± 0.5%
- Voltage Regulation: ± 0.5%
- Input voltage: 4.75-35V
- Output voltage: 1.25-26V(Adjustable)
- Output current: Rated is 2A, maximum is 3A (Additional heat sink required)
- Conversion Efficiency: Up to 92%
- Operating Temperature: Industrial grade (-40 to +85) (output power 10W or less)

4.5 Circuit diagram:

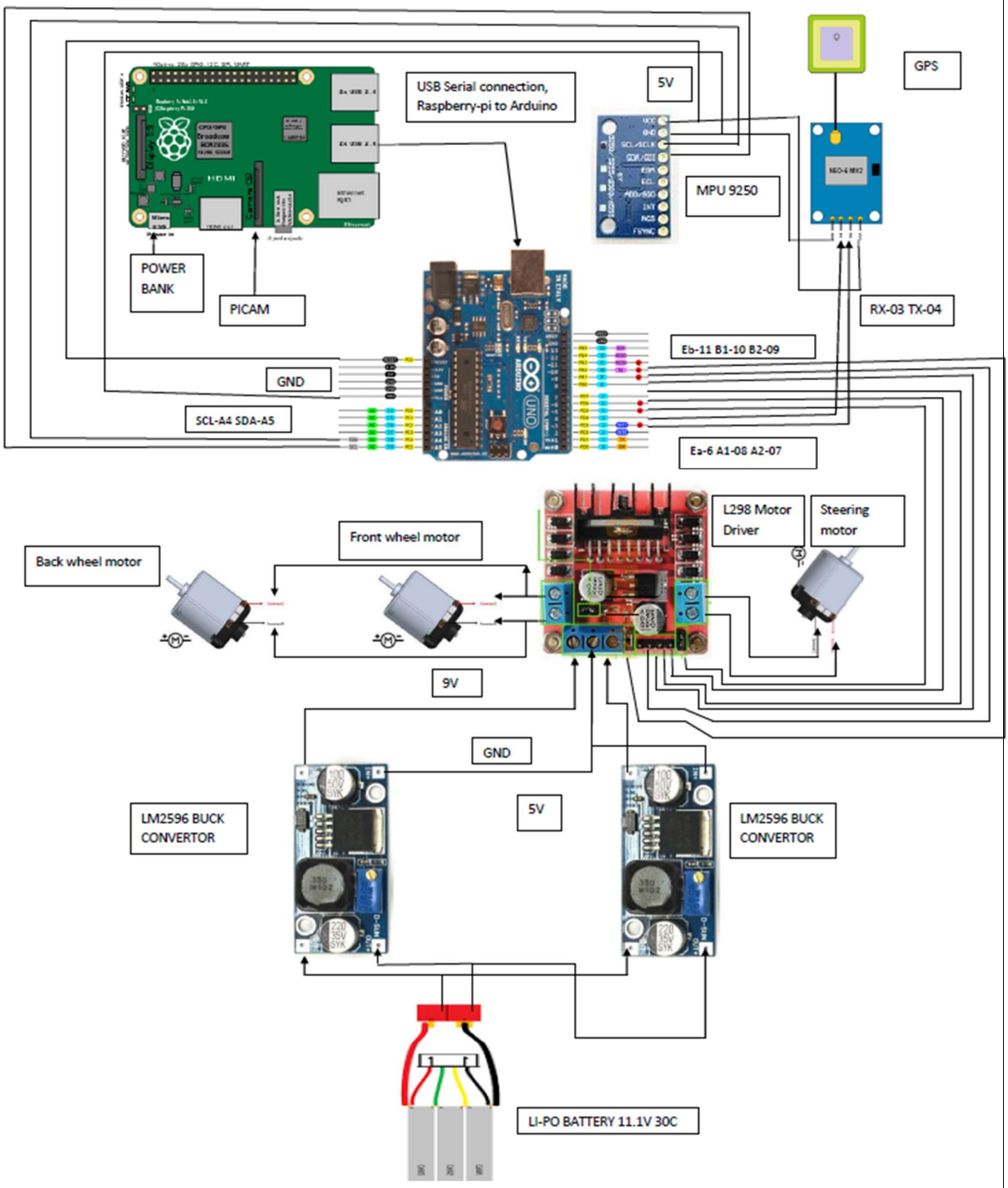


Figure 4.5: Circuit diagram

4.6 PCB layouts:

4.6.1 PCB layout for motor driver circuit:

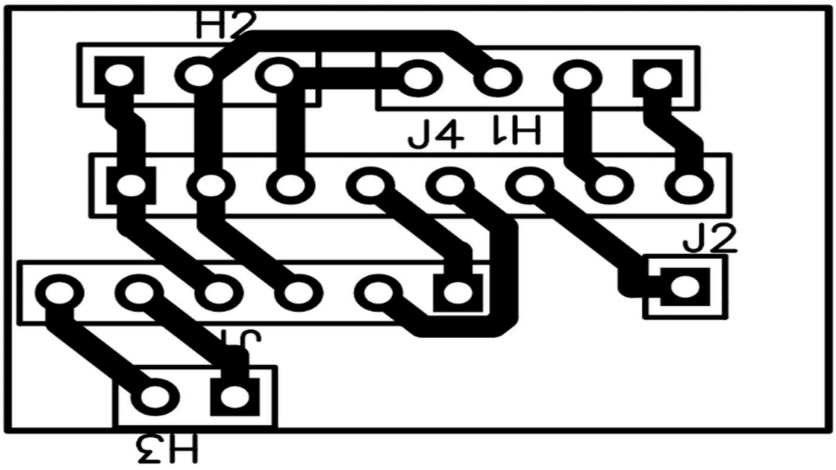


Figure 4.6.1: Motor driver circuit PCB layout

4.6.1 PCB layout for GPS and HMC 5883L circuit:

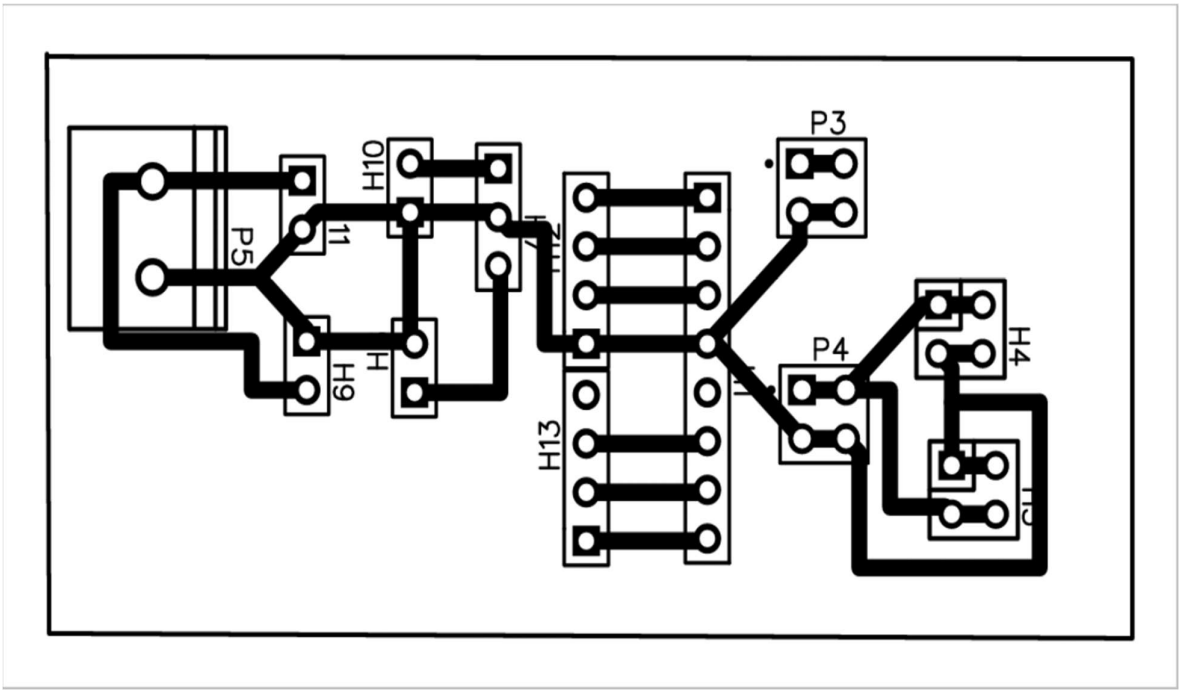


Figure 4.6.2: GPS and HMC 5833L circuit PCB layout

Chapter 5

FLOWCHARTS

5.1 Lane detection:

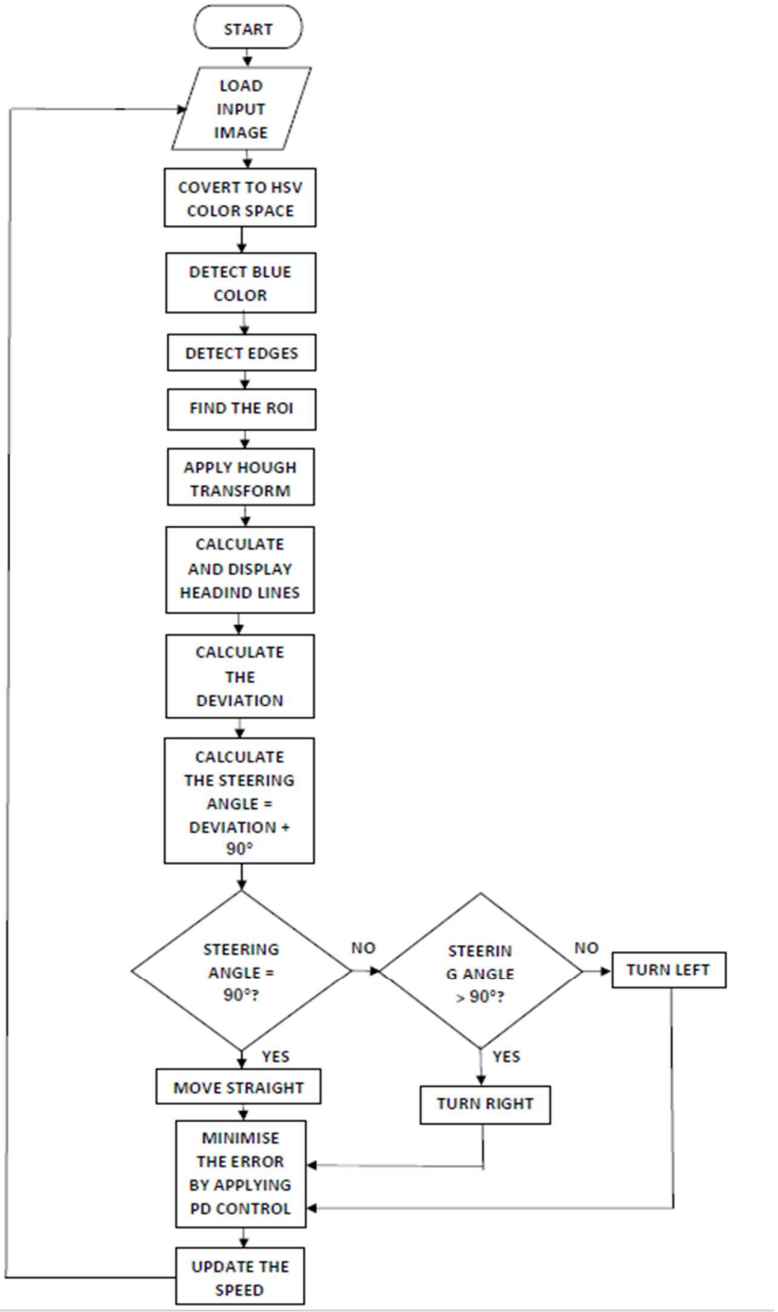


Figure 5.1: Lane detection flowchart

5.2 Traffic signs detection:

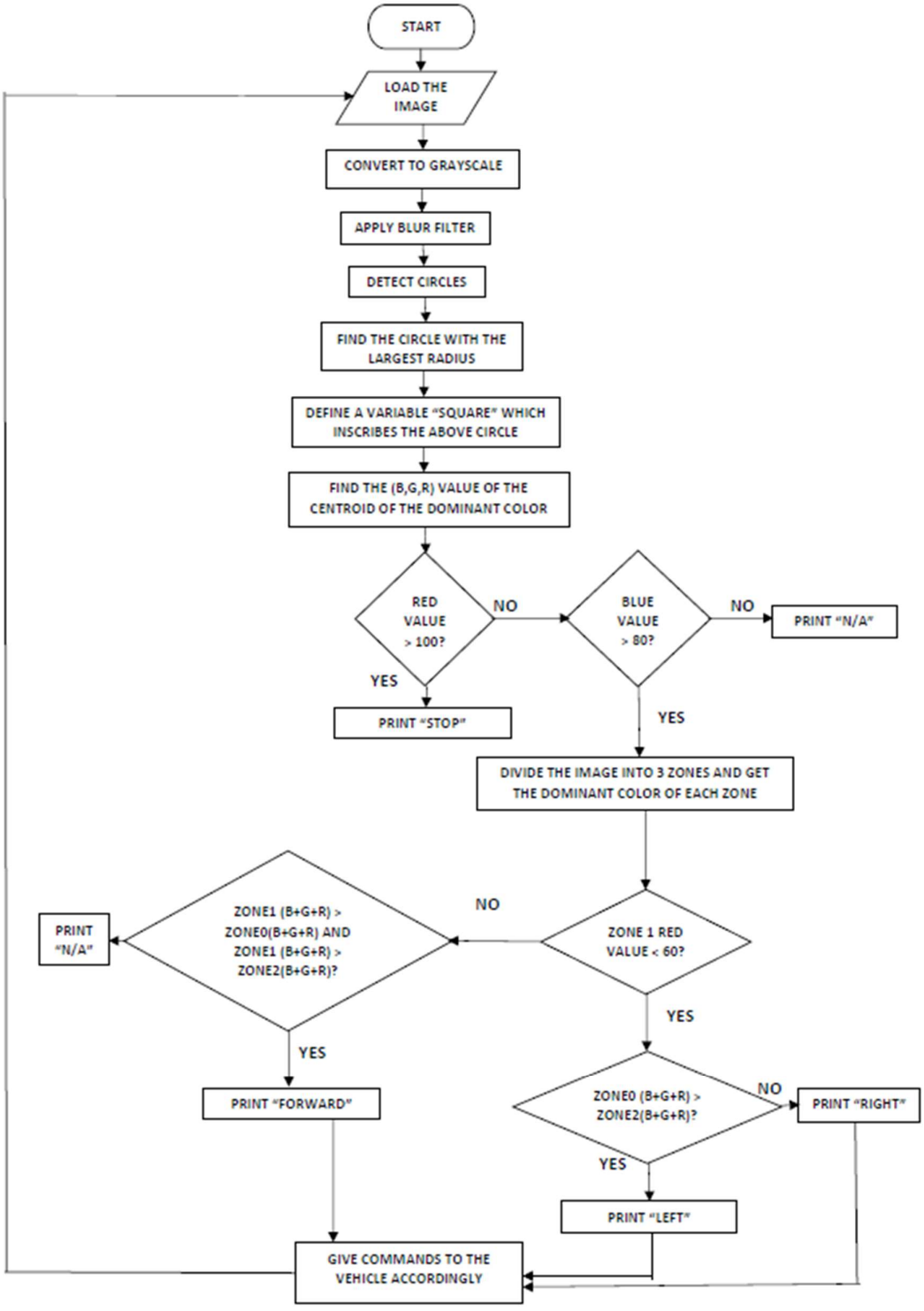


Figure 5.2: Traffic sign detection flowchart

5.3 GPS navigation:

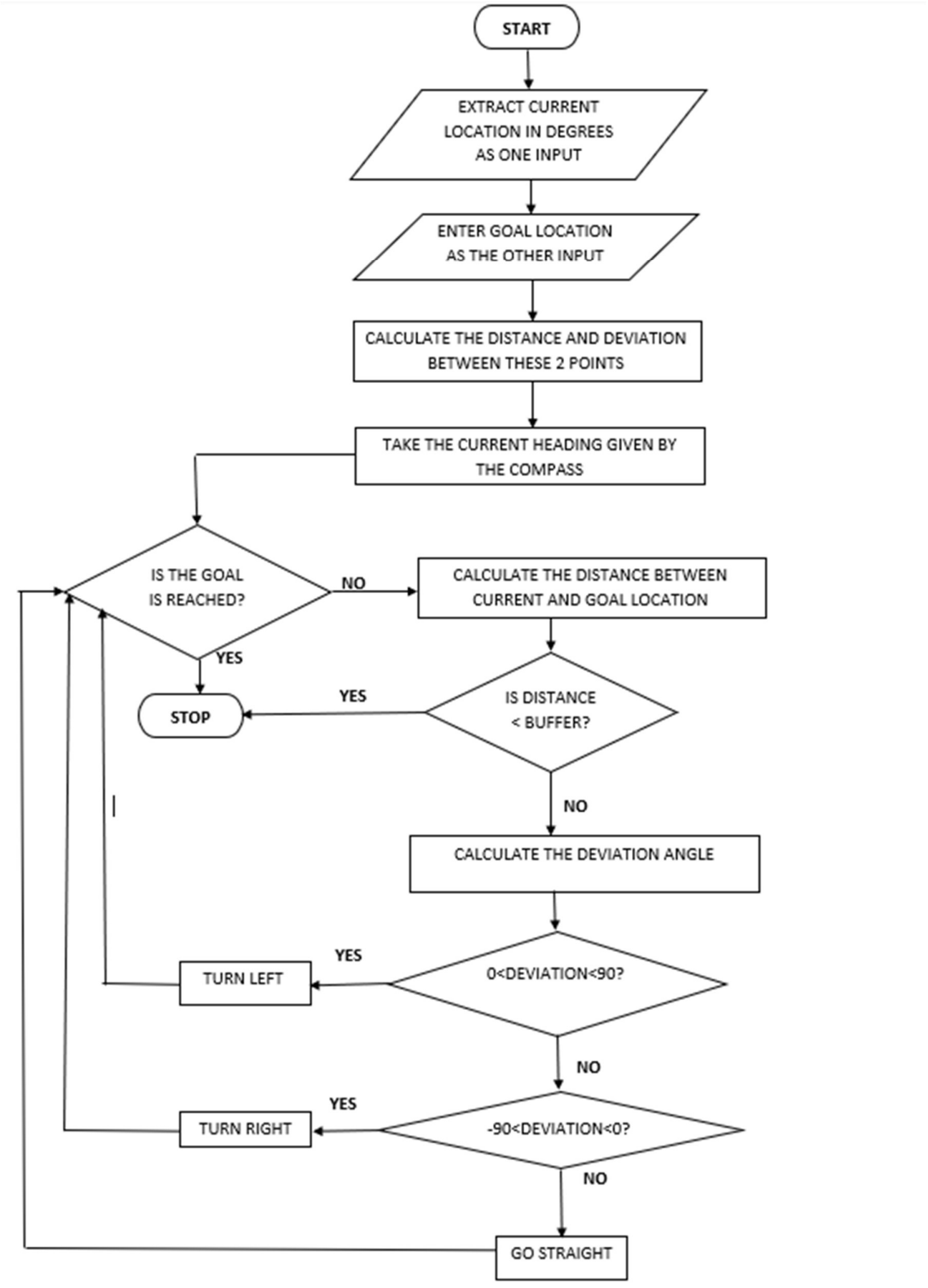


Figure 5.3: GPS navigation flowchart

Chapter 6

OBSERVATIONS

6.1 Lane detection:

The system was able to detect the lanes lines successfully in real time, though there was an issue of random vibrations wherein the camera was not able to properly capture the video, resulting in an incorrect result. This problem was particularly observed during turns.

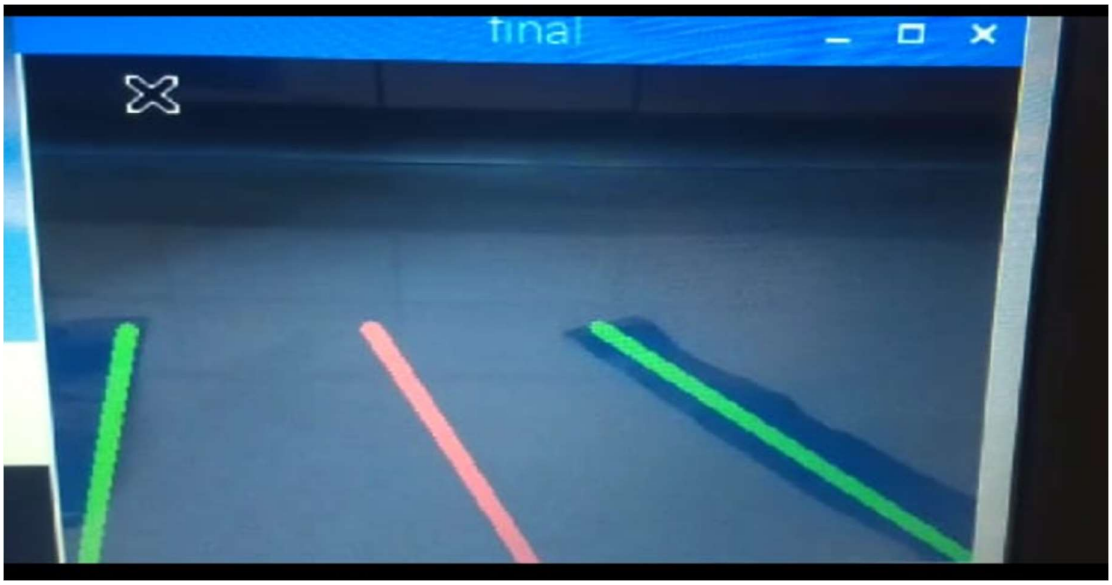


Figure 6.1: Detected lane lines along with the heading line

The inability of the L293d motor driver to provide the required starting torque to the motors was also observed. It was then replaced with a superior L298d driver, which has its own advantages over the prior. It has a higher current and voltage handling capacity as compared to the L293d. Further, the L298d also comes with a number of other features such as thermal shutdown, meaning that it will slow down and stop if overloaded, rather than burning out.

6.2 Traffic sign detection:

The algorithm which was developed for traffic sign detection worked well. But it was observed that at times, the sign detected did not match with what was seen by the camera. This problem however persisted for a short period, supposedly due to processing limitations. The traffic signs detection was initially tested on the computer using the anaconda software platform. The following was observed on the same:

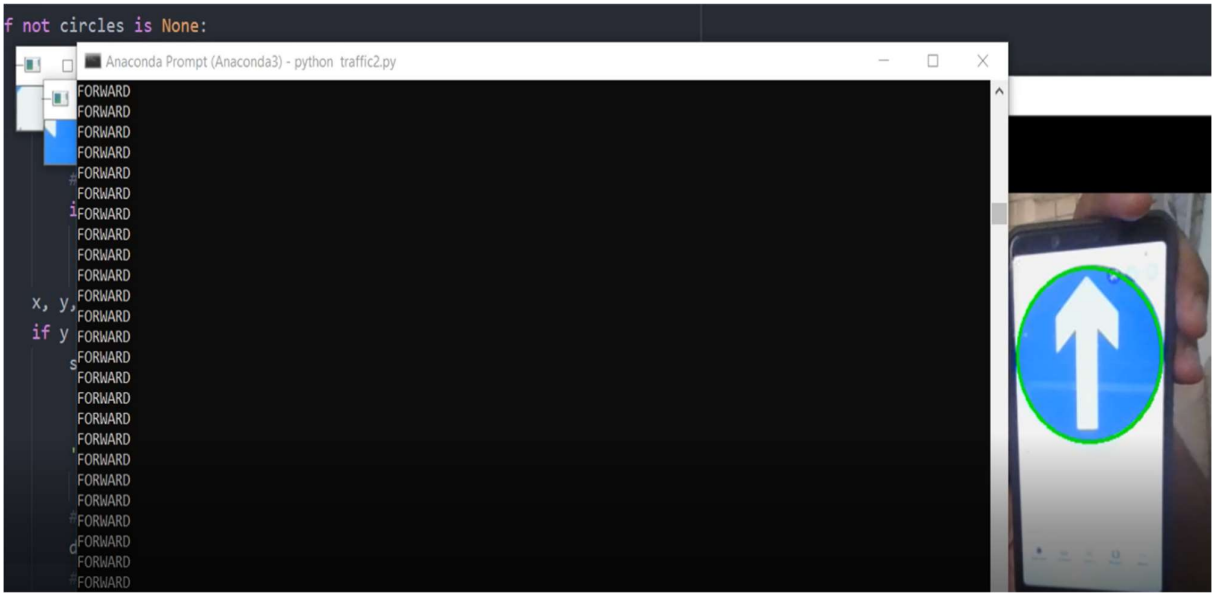


Figure 6.2.1: "Forward" sign

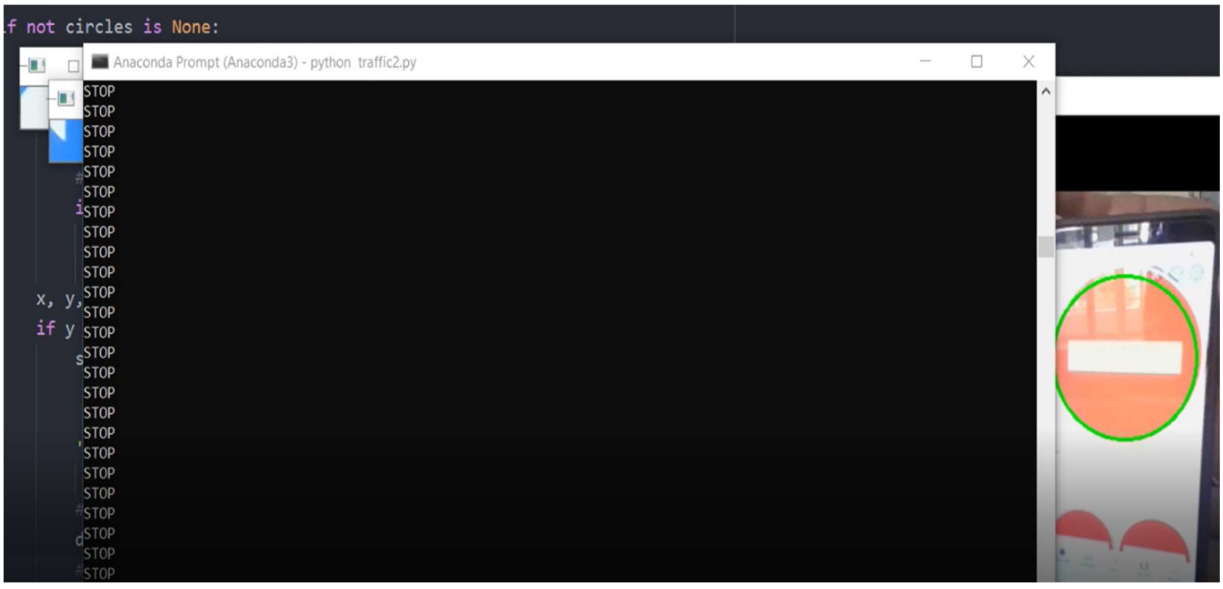


Figure 6.2.2: "Stop" sign

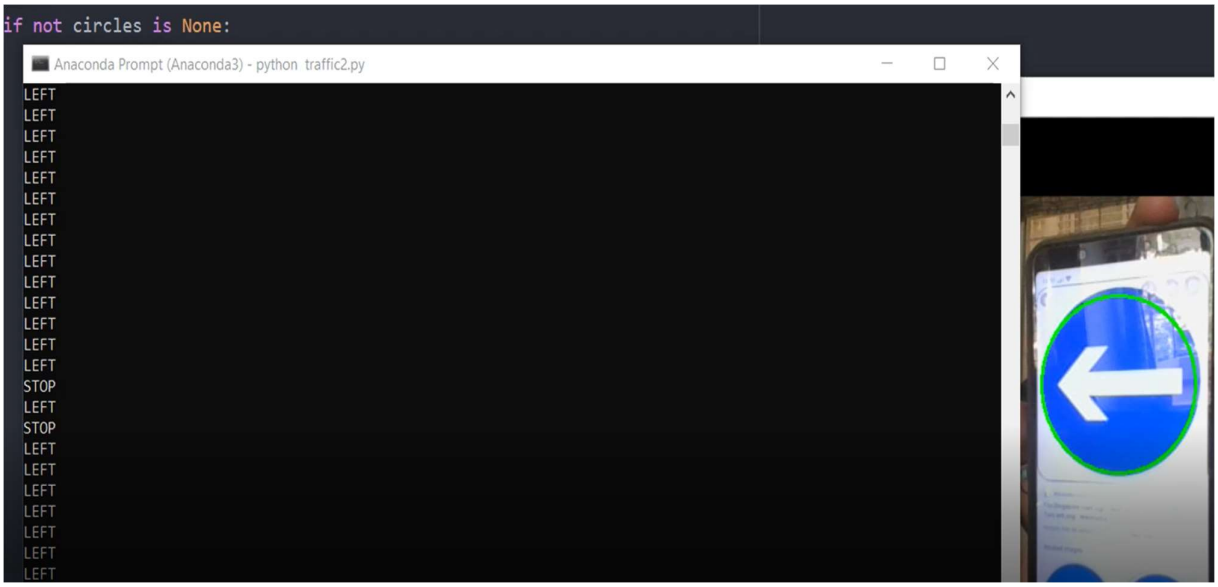


Figure 6.2.3: "Left" sign

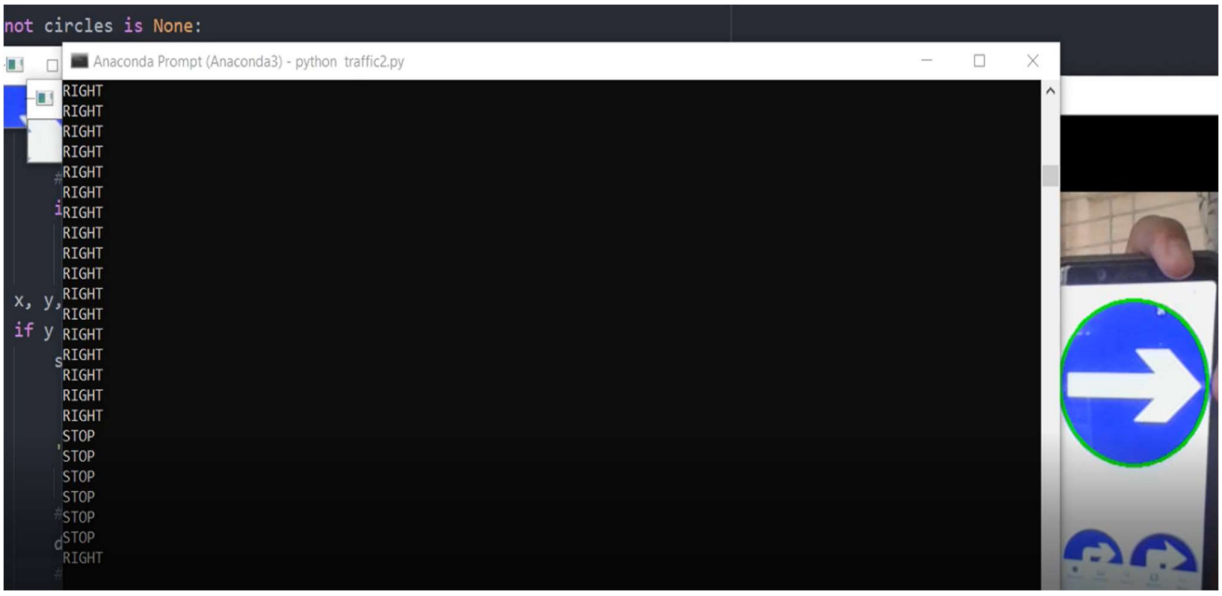


Figure 6.2.4: "Right" sign

6.3 GPS navigation:

The following figure shows the observations for GPS navigation part. As can be seen, it shows the readings for required distance, required heading, current heading, current and goal coordinates. Also shown by the algorithm is the direction in which the vehicle needs to be turned.

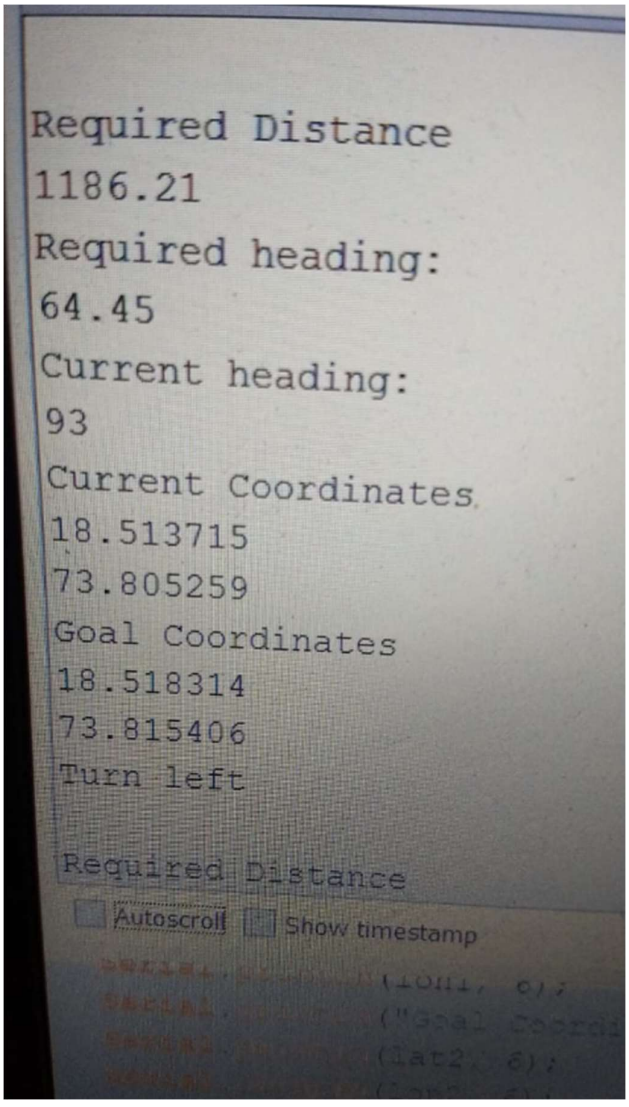


Figure 6.3: GPS navigation

Chapter 7

CONCLUSION

The use of image processing algorithms in order to achieve lane detection and traffic sign detection gave good results in standard light conditions. The Hough Transform was used to detect lane lines in the image. Based on the detected lines, a central line giving the direction of heading was calculated. Further, the speed of motors was adaptively controlled using a Proportional Derivative (PD) algorithm. This was done to ensure that the car does not go off the track or lose its direction whilst making turns.

For the purpose of demonstrating the detection of traffic signs, an algorithm to do the same was developed keeping in mind the four basic signs, viz. “forward”, “stop”, “left” and “right”. This again was achieved using concepts from the field of image processing. The k-means clustering technique was put into use to extract dominant colour in the frame of captured video. Accordingly, a user-defined function was created which returned the (B, G, R) coordinate values of centroid of the cluster containing pixels of nearby values of the dominant colour. The image was then divided into three zones. Based on the dominant colour present in these zones and some mathematical calculations, the appropriate sign was detected and displayed.

Moving ahead, the third part of GPS navigation was achieved using an algorithm which took two inputs in the form of current and goal location points, trying to minimise the distance between the same. The GPS sensor was used to determine latitudinal and longitudinal position, while the magnetometer compass was used to give direction in earth plane.

Finally, as far as hardware is concerned, the raspberry pi gave a satisfactory performance when it came to processing image related data. The Arduino UNO played its part in giving apt commands to the motor drivers, which in turn were responsible for controlling the DC motors. The pi cam used also gave good results while capturing video data. The Li-Po battery served as a good source of power while the buck converter made it sure that adequate voltage was supplied to the motors.

All in all, this project helped us delve into the interesting topic of self-driving vehicles and also enabled us to learn and discover many new things.

Chapter 8

FUTURE SCOPE

The detection of lane lines and traffic signs can be made more efficient and expedited through the use of machine learning and deep learning algorithms. One such concept from the field of deep learning learning and artificial intelligence is the CNN or “Convolutional Neural Networks”. These CNNs could be exhaustively trained to accurately detect the traffic signs with minimum error. These network could reach the accuracy level of more than 98.5%, speaking roughly. The neural networks could also be put to use when it comes to real-time detection of never-before-seen lane lines. The system could even be expanded to dynamically detect various objects. Going further, it could be enhanced to detect pedestrians so as to avoid accidental collisions.

Artificial Intelligence (AI) too is one of the most rapidly growing streams as far as research and development is concerned. AI based computer vision methods like semantic segmentation can be used to improve vision system of the vehicle. LIDAR and RADAR sensors can be used to detect and track other dynamic objects using Particle Filters and Kalmann Filters. These algorithms can be fine-tuned using AI. Model Predictive Control algorithms can be used for driving the car. Nvidia’s End-to-End Deep Learning method can be used to combine vision, steering and throttle control into a single program using deep learning methods. Deep Reinforcement Learning and Behavioural cloning are two other methods that can be used to autonomously drive the car. For all of these methods, more powerful processors are required. CUDA and parallel processing can be integrated into this project to improve processing power so more complex and accurate autonomous driving can be implemented with the aim of vastly improving the safety factor. The system could also be implemented on a large scale, involving real on-road vehicles, through the use of more powerful and advanced computing platforms such as the “Jetson”, developed by Nvidia.

Navigation using GPS could be further enhanced by incorporating a way in which the system would automatically come up with the shortest route possible, given the availability of multiple paths. The three aspects of lane detection, traffic sign detection and GPS navigation could be integrated into one complete system.

REFERENCES

[1] AutoRally: An Open Platform for Aggressive Autonomous Driving. Brian Goldfain, Paul Drews, Changxi You, Matthew Barulic, Orlin Velev, Panagiotis Tsiotras, James M. Rehg. *Control Systems Magazine (CSM)*, 2019

[2] Alvinn: An autonomous land vehicle in a land network. Dean A. Pomerleau, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213

[3] Paden, Brian, et al. "A survey of motion planning and control techniques for self-driving urban vehicles." *IEEE Transactions on intelligent vehicles* 1.1 (2016): 33-55.

[4] Marin-Plaza, Pablo, et al. "Global and local path planning study in a ros-based research platform for autonomous vehicles." *Journal of Advanced Transportation* 2018 (2018).

[5] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J. and Zhang, X., 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

[6] Badue, Claudine, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Ferreira Reis Jesus et al. "Self-driving cars: A survey." *arXiv preprint arXiv:1901.04407* (2019).

[7] Rosenschein, Jeff, Nir Pochter, and Zinovi Rabinovich. "HANS-HUJI's Autonomous Navigation System." (2008).

[8] Thrun, Sebastian, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong et al. "Stanley: The robot that won the DARPA Grand Challenge." *Journal of field Robotics* 23, no. 9 (2006): 661-692.

[9] Assidiq, A. A., Khalifa, O. O., Islam, M. R., & Khan, S. (2008, May). Real time lane detection for autonomous vehicles. In *2008 International Conference on Computer and Communication Engineering* (pp. 82-88). IEEE.

[10] Santos, D.S., Nascimento, C.L., & Cunha, W.C. (2013). Autonomous navigation of a small boat using IMU/GPS/digital compass integration. 2013 IEEE International Systems Conference (SysCon), 468-474.

[11] A. Al Arabi, H. Ul Sakib, P. Sarkar, T. P. Proma, J. Anowar and M. A. Amin, "Autonomous Rover Navigation Using GPS Based Path Planning," 2017 Asia Modelling Symposium (AMS), Kota Kinabalu, 2017

[12] M. B. Shahid, M. U. Shahzad, S. M. Rameez Bukhari and M. A. Abasi, "Autonomous vehicle using GPS and magnetometer with HMI on LabVIEW," 2016 Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)