Chartopolis - A Self Driving Car Test Bed

by

Rakshith Subramanyam

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Spring Berman, Co-Chair, Chair
Hongbin Yu, Co-Chair
Suren Jayasurya

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

This thesis presents an autonomous vehicle test bed which can be used to conduct studies on the interaction between human driven vehicles and autonomous vehicles on the road . The test bed will make use of a fleet of robots which is a microcosm of an autonomous vehicle performing all the vital tasks like lane following, traffic signal obeying and collision avoidance with other vehicles on the road. The robots use real time image processing and closed loop control techniques to achieve automation. The test bed also features a manual control mode where a user can choose to control the car with a joystick by viewing a video relayed to the control station. Stochastic rogue vehicle processes will be introduced into the system which will emulate random behaviours in an autonomous vehicle. The test bed was experimented to perform a comparative study of driving capabilities of the miniature self driving car and a human driver.

ACKNOWLEDGMENTS

I would like to thank Professor Spring Berman, for giving me a wonderful opportunity to work in her lab. She has been supportive all through out. I should also thank her for her patience for putting up with my ignorance in many areas.

I would like to thank Professor Hong Bin Yu for constantly supporting me and encouraging the work that I was doing. I also thank all his capstone students who have been helpful in setting up the test-bed.

I would like to thank all my collaborators, mainly Ruben Gameros who helped in constructing the test bed and i also thank all my fellow lab members of the ACS lab who have been a constant support, mainly Aniket Shrishat, Zahi Kakish, Sriram, Madlyin for sharing their valuable inputs.

I would like to thank Kowshik Thoppalli and Sai Pratyusha Gutti for helping me document this thesis. I am also thankful to the many friendships that I made in this phase which helped me stay focused on the job.

I am thankful to my graduate advisor, Sno Kleespies from the department of Electrical Computer and Energy Engineering at ASU, for helping me make my stay at ASU, a comfortable one.

This thesis would not have been possible without the love, guidance and unconditional support of my parents and friends.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

In the past decade the word 'self driving car' has become a part of our everyday vocabulary. Self driving cars have now hit the roads to the point where our cab rides are given by autonomous drivers. The advancements in the computational technology have made companies like Tesla (Reese (2017)), Waymo, Mercedes-Benz (Logan (2015)) and the Uber(Dwoskin (2016)) operate their autonomous taxis around United States since 2016 making them accessible to millions of users. These advancements of the autonomous vehicles have changed urban transportation by making it much more easier and convenient. Along with these developments were the troubles that every technology brings with it. It's the cost that is paid when the technology doesn't operate the way it should. The first fatal autonomous car crash happened in China's Hubei province(Boudette (2016)). Following this, within four months Tesla faced another fatal incident in Florida(Fleming (2016)) and the latest fatal accident happened in March 2018 at Tempe.

These incidents that tail along with the advancements bring in the requirement of experimenting and validating the performance of the self driving cars in an urban environment. Though this is a necessity it is becoming an issue with the stringent laws in many states in the US that restrict the accessibility of the roads for a self driving car. In Urban environments the traffic flow is mostly happening based on a non verbal communication between the drivers and the pedestrians on the road. With the removal of the driver from a car it is going to be tough to establish this kind of communication where there is an affirmation for both the pedestrian and the driver about each others' action. Now the question raises

on how these situations are going to be handled in the world of automation. Experiments are to be conducted where the interaction between the humans and the self driving cars are focused.

1.2 Problem Statement

Current testing and validation methods include on road experimentation by driving the self driving car around in a city. There are also experiments conducted in simulated and graphical environments. In a manner, to find solutions to these problems this thesis attempts to address the following question.

Can a self driving car test bed be constructed to emulate the conditions of the urban traffic and perform experiments related to human robot interaction.

This problem statement is examined in the context of real time machine vision in robotics and a suitable solution is discussed in this work.

1.3 Organization of Thesis

The remainder of the thesis is organized as follows: Chapter 2 briefly introduces the existing work in this research field. Chapter 3 explains the working and the algorithms behind a self driving car. Chapter 4 covers the proposed methods to emulate a self driving car in the Pheeno robot. Chapter 4 and 5 talks about the experiments carried out to show the effectiveness of the test bed and concludes the discussion with possible directions for future work.

Chapter 2

RELATED WORK

After the start of the self driving cars there have been multiple test platforms that have been used to experiment with the algorithms and the working of the autonomous vehicles. These test environments are based off simulations and emulations. Driving simulators are not new, there are numerous realistic and racing simulators, many autonomous vehicle manufacturing industries are using them. Given all of this, most of them do not provide enough data to the autonomous driving system in the aspect of human machine interaction. The CARLA (Dosovitskiy *et al.* (2017)) from the Intel labs and the Toyota research center is an simulation of an urban scenario which was developed to train self driving cars by experimenting the algorithms with different data sets and different test scenarios. The CARLA incorporates a library of assets that can be arranged into various weather settings and lighting conditions. Even this environment lacks the ability to have multiple human drivers drive alongside the autonomous car to study the behaviour of the humans and the self driving car next to each other.

The Duckietown(Paull *et al.* (2017)) was developed as an open-source platform for education and research in automation. This paper also includes building of vehicles called Duciebots which serve as miniature autonomous cars. The duckiebots are equipped with a Raspberry pi 2 and a monocular camera for sensing the objects in front. This combination of raspberry pi and camera makes the duckiebot to run complex software architecture like low level perception, object recognition, decentralized coordination etc. The duckiebot performs lane following by using active image processing techniques, it also detects the street names and the traffic signs using fiducial markers (Olson (2011)). Each fiducial marker is attributed to a different street name or traffic sign. The duckiebot also uses

the fiducial markers for localizing itself in the town. The size and the location of each fiducial markers are programmed prior and the duckiebot uses this data to estimate its current position. The duckiebot is also capable of performing path planning and navigation, the duckibot uses A* search algorithm to generate an optimal route to traverse the map. Though these robots have autonomous capabilities they lack the ability to establish V2X communications which is a growing research field of the self driving cars. The proposed solution establishes a V2x communication by establishing an AD-Hoc network where all the vehicles and the infrastructure can be connected.

Paralleling the Duckietown the students of University of Delaware also developed a smart city, called University of Delaware Scaled Smart city(UDSSC)(Stager *et al.* (2017)) This smart city is intended towards experimental validation of connected and autonomous vehicles. The UDSSC cars perform lane tracking and operations related to connected vehicles. Their research focuses more on the communication of connected vehicles and scheduling their movements when there is a road merging with another. But the UDSSC doesn't address the traffic lights and the ability of a human to drive alongside the self driving cars. These abilities pave way to design experiments around human vehicle interactions. The proposed solution solves this issue by modelling a city with traffic lights and lanes and making the self driving car operate in this setup.

Chapter 3

AUTONOMOUS CARS

This chapter provides a detailed explanation on how an autonomous car operates, the hardware components required and the software architecture. One of the important tools to know about in this exploration is Image processing, sensor fusion and Machine learning which will be presented in detail.

3.1 Autonomous Cars - Introduction

The term autonomous car may have been made common after the 2005 DARPA challenge. But the first efforts of making an autonomous car dates back to 1925(Felton (2017)) with the 1926 Chandler driving through the streets of New York city. The car was controlled by a radio signal that was transmitted by a tailing vehicle. The radio signal was sent to few circuit breakers which in turn controlled small motors attached to the throttle and the steering. There were multiple attempts after this to make an efficient autonomous car, but all of them depended on a device or circuitry embedded within the road.

The first autonomous car that was not dependent on the infrastructure of the road was developed by Ernst Dickmanns, in the early 1980. His team equipped a Mercedeze Benz van with a camera and sensors. This was the pioneer for the use of dynamic vision. The vehicle employed Kalman filtering techniques to perspective imaging so as to achieve autonomous driving with noisy data from the camera and sensors. But the first vehicle to start using depth perception was the DARPA funded Autonomous Land Vehicle (ALV)(Leighty (1986)) which demonstrated the use of LIDAR and a camera for performing self driving.

This vehicle encapsulated a RCA color video CCD camera and a EIRM Laser Range scanner. The camera captures RGB analog intensity image at 30 frames per second and delivers it to a VICOM video processor. The laser scanner employed an amplitude modulated(AM) light source which scanned the entire road ahead, the phase shift of the reflected wave is measured with respect to an internal reference and the distance of the objects in front is calculated with this measurement.

With all these different autonomous vehicles through history, there are defined functions that are attributed to certain levels of automation. A brief overview of these levels is enumerated below.

1. Level 0 autonomy is when the driver of the vehicle controls everything in the vehicle ranging from steering to throttle.

2. Level 1 autonomy is the state where there is assistance for the driver like cruise control etc.

3. Level 2 autonomy is when partial automation is achieved by letting at least one system like lane centering to operate with full automation. In this level the driver should monitor the vehicle at all times.

4. Level 3 autonomy is when minimal driver intervention is required. The system prompts the driver to take over if it encounters a scenario that it can navigate through.

5. Level 4 autonomy is highly automated, where there is no assistance required from the driver but the vehicle cannot operate in all the scenarios

6. Level 5 autonomy is where the vehicle is fully automated and there is no requirement of a driver. The vehicle can perform all the task from steering to throttle

3.2 Autonomous Cars - Hardware

There are multiple hardware components facilitating the automation of an autonomous car, but the most primary ones are the perception sensors, the GPS and the central processing unit. A brief operation of these sensors are elucidated below.

One of the most important sensory component is the camera which gives the car with live video feed of the road ahead of the vehicle, this video feed is used by the car to understand the surrounding like lane markings, traffic lights, traffic signs etc. Two techniques are primarily used to capture video feed from the surroundings - mono vision and stereo vision. In mono vision, there is a single camera lens that captures the environment ahead. The drawback is that the pixel density is low which throws the challenge of sophisticated image processing techniques to recognize the markings and signs on the road using a low resolution image and also this camera cannot capture depth of the objects in the frame. On the other hand, stereo vision(Bertozzi and Broggi (1998)) exploits the concept of depth perception by using two video cameras analogous to human vision. This perception of depth aids in differentiating the moving objects from static objects and empty spaces. For capturing the detail at farther distances, usually a camera is placed on the roof area (Thrun *et al.* (2006)).

Another important component in a self driving car is the LIDAR, It equips the car with a three dimensional point cloud of the objects surrounding the car. It shoots a beam of laser and inspects the reflection and measures the time of flight of the light to gauge the distance of the objects that is being scanned. The LIDAR gives a very accurate depth perception for the autonomous car and creates a 3D map which the car can use to navigate. The use of LIDAR eases the requirement on the machine vision of the autonomous car by enabling the processor with the knowledge of the relative position and speed of other vehicles on the road along with the locations of traffic signals and traffic signs. One of the

main drawbacks of the LIDAR sensor is that it is very expensive which makes it difficult to be used in commercial cars. Though there are multiple autonomous car manufacturing industries which depend heavily on LIDAR to achieve the required automation in a car there are also few industries which do not depend on LIDAR and rely purely on computer vision. One example is the Tesla which is purely dependent on the camera and computer vision though there have been multiple criticisms on this approach(Lohor (2016)).

Another most crucial sensor is the Global Position Sensor(GPS). This sensor enables the autonomous car to know where it is in the world with an accuracy of 4 inches. The GPS aids the autonomous car to navigate to the desired destination. Along with the information provided by computer vision the positioning of the vehicle on the lane is also done using the GPS data along. When it is snowing or there are no lane markings the autonomous car solely depends on GPS to stick to the lanes. The GPS data is attributed to a accurate map that was developed by the industry that makes the autonomous car. If the industry doesn't have provision to build a map of the entire world it purchases from another companies like Google Maps, NOKIA's Here maps etc. The problem with this type of mapping is that, with constantly changing roadways the maps should be updated very frequently to include the new changes made to the road ways. This problem might be solved in the future by crowd sourcing the map information. Another drawback of the GPS is when the sky is not clear or if the sensor is not able to receive satellite signal the localization accuracy decreases drastically.

Apart from these sensors the autonomous vehicle also uses various other sensors to aid its decision making process. It uses a RADAR to assist the LIDAR in generating a 3D point cloud. There are few manufacturers who just use RADAR for generating a 3D map. The advantage of RADAR is that it's relatively cheaper and has a very long range as compared to LIDAR. The car also uses an odometer and an inertial measurement unit to calculate the heading, velocity and distance travelled. Another most important hardware

component of the car is the Control unit which receives the data from all the various sensors and controls the vehicle accordingly. Along with these there are the actuators which control the movements of the pedals and the steering.

3.3 Autonomous Cars - Software

To use all the hardware components efficiently and to induce automation in the car there is a requirement of a strong software component running in the back end. All the sensory data is used hand in hand by the sensor fusion algorithm to estimate the next step that is to be taken by the autonomous car. The primary segment of self driving is lane following.

The lane following can be performed using computer vision techniques or using Machine Learning algorithms. The computer vision technique was the first algorithm to be used for the establishing automation in a car. This algorithm depends on digital image processing techniques. This method would be similar to how human processes an image to detect lanes on the road. Every image that a camera capture is captured as intensities of Red, Green and Blue, generally called as the RGB image. The RGB image is first converted to gray scale which reduces the image to a single color channel image. This reduces the processing time by a huge factor as the machine vision operations are performed on a two dimensional matrix rather than a three dimensional matrix. The gray scale image is smoothened to suppress any noise from the image. Only a small segment of this image contains the useful information regarding the lanes. To extract this a region of interest is defined and the corresponding segment is extracted. An edge detection algorithm is used to extract the edges in the image, these edges are run through a line extraction algorithm to extract the lines. The slope of the lines is used to characterize them as left and the right lanes. This algorithm is very reliable and operational but when the conditions of the road are very harsh this algorithm fails.

Another most commonly used technique is the Machine Learning algorithm. In this a huge quantity of data is collected and labelled for training a neural network that learns to identify a lane and the curvature of a lane. The accuracy of this method is very high and this method is also reliable in some harsh conditions. The drawback of this method is the requirement of huge training data and a very laborious process of labeling the data. Apart from lane detection, the autonomous car uses Machine Learning to detect the traffic signs, existence of a traffic light and other cars on the road.

Another crucial software segment in the car is to control the vehicle, for this purpose there are various control algorithms like the Proportional Integral Derivative(PID) controller, Linear Quadratic regulators(LQR) or Model Based Controller. But these algorithms heavily depend on the existence of a model for the system. Though these models are obtained and used for controlling the drive train of the vehicle, the decision making process that commands the controller still rests on the machine learning approach. The autonomous car is trained on a data of many million hours of driving by a human so that it can understand the command that is to be given to the steering wheel with respect to the traffic state.

Object tracking and trajectory estimation is a critical operation for an autonomous car.There are various sensors that gives data for a single object of interest. For example, the camera, LIDAR and RADAR all give the same data of what is present around the vehicle. These data sets are used to estimate the type of object, distance to the object, the velocity of the object and possibly the trajectory of the object. But inherently due to inaccuracies and errors in measurements these data sets are usually a probabilistic estimates. This inaccuracy will affect the control action that is being performed. To reduce the inaccuracies in processing the data, a sensor fusion algorithm which takes in data from various sensors is being used. This algorithm takes in the noisy input and generates an estimate of the states with less error.

3.3.1 Observer - Kalman Filter

The most commonly used sensor fusion algorithm is the Extended Kalman Filter (EKF) (Kala (2016)). The object of interest will be modelled in state space representation(Friedland (2012)). The state of the system at time step t can be estimated from information from time step $t-1$. The model of a system is represented using A,B and C matrix as shown below.

$$X_k = Ax_{k-1} + B_t u_t + W_k$$

$$Z_k = Cx_k + V_k$$

where

X_k is the state vector

A is the state transition matrix, this defines how the state values evolve over time

u is the input vector

B is the input matrix

W_k is the Gaussian noise that is acting on the state

V_k is the Gaussian noise acting on the sensor measurements

Z_k is the observation of the state

C is the observation matrix or the sensor matrix

The current state value is predicted by using the state transition matrix and the previous state estimate

$$\hat{x}_k = A\hat{x}_{k-1}$$

From the variance in the Gaussian noise of the sensors the R matrix is defined as the co-variance between the noise from various sensors. The diagonal elements of this matrix represents the sensors variance with itself. Usually the standard deviation of the sensors

can be measured and from this the variance is calculated by squaring the standard deviation. Similarly the co-variance matrix of the process noise is represented by Q. The estimation noise at time k of the system is represented by P_k. The estimation noise is the co-variance of the estimates made for the states at time step k. The Kalman Gain matrix is represented by G_k, this gain matrix is utilized to weigh the state estimate made by the observer and the state measurement made by the sensors. This gain matrix reduces the state estimation error for the next iteration and the state estimate value is used to calculate the the control input.

After estimating the current state values using the state estimate equation the process noise P_k is calculated.

$$P_k = AP_{k-1}A^T + Q$$

. This process noise along with the observation matrix is used to calculate the Kalman gain matrix G_k

$$G_k = P_k C^T (CP_k C^T + R)^{-1}$$

$$\hat{x}_k = \hat{x_k} + G_k(Z_k - C\hat{x}_k)$$

The estimation uncertainty is updated for the next iteration by

$$P_k = (1 - G_k C)P_k$$

After this step the $\hat{x}_k$ is fed to a controller which uses this observation of the state and would define a control input to the system. These type of systems are called full state feedback systems.

3.3.2 Controller

There are various controllers that have evolved over the years. But the autonomous cars are constrained by computational power and every disturbance from the surroundings. These constraints beget a fast and reliable controller, the most commonly used controllers

are MPC (Model Predictive Controller), PID (Proportional Integral Derivative) and the LQR (Linear Quadratic regulator).

PID controller is a commonly used feedback controller widely used in robotics and industrial control system. The PID controller actively calculates an error value between a set point and a measured variable and would modify the control input based on proportional integral and derivative term. The PID controller requires the user to have a model of the system. A model is a mathematical expression of the relation between the input and the output of a system based on differential equations. The gains of the controller are calculated with this knowledge. The sensor measures a value and feeds it back to the controller. The feedback loop calculates a difference in the current state and the desired state of the system, this is the error term. The error is being tracked by the proportional component of the controller. The controller commands a control input proportional to the error in the opposite direction. The proportional response can be adjusted by multiplying the error with the proportional constant K_p. This simplistic proportional control can be used independently, but this would suffer with the problem of overshoot. This is a condition where the control action might push the vehicle far to left of the lane when trying to correct an error due to the vehicle being more to the right side of the lane. To avoid this situation an derivative term is used.

The derivative term tries to flatten the error curve to a straight line. It keeps track of the previous error and would compute the rate of change of error or the slope of the error and would multiply this with a derivative gain K_p. The magnitude of the derivative gain dictates the contribution of the derivative term in the control action. Based on this estimate in the trend of the error, future overshoots can be avoided and the error can be reduced faster. But if the derivative constant is too high this control would induce oscillations, which will make the car oscillate around the set point. The integral term observes any residual errors in the system after applying proportional and derivative control. Eliminating this error is

crucial as in long period of time this error would cause the vehicle to drift off. The integral term keeps track of all the past errors, it does this by integrating the error from t_o to the current time step. This accumulated error is multiplied by an integral gain K_i and added to the controller output. The contribution of the integral term is proportional to the magnitude of the error and the duration of the error.

The PID equation is given below.

$$u(t) = K_p e(T) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

where,

$u(t)$ is the control input or the output from the controller

$e(t)$ is the error between the set point and the measurement at time t

K_p is the proportional gain

k_i is the integral gain

K_d is the derivative gain

τ is the variable of integration which takes values from time 0 to the present τ

The most import part in PID control algorithm is tuning the gain parameters. These gain parameters decide how much contribution each of the control term offers. There are multiple ways to tune the PID parameters. But all these can be categorized as tuning with the model or without the model. Once the system is modeled pole placement techniques can be used to get the gain value, This method is not very native to PID controller tuning. Other commonly used method is the manual tuning where the K_i and K_d values are set to zero and the K_p is increased till the system oscillates, then the K_p is set as half of that value. After this the K_i is increased till the offset is corrected in the least required time. After this the K_d is increased till the overshoot is eliminated and has a quick response.

Another method that is used for tuning PID controller is the Ziegler-Nichols method(Ziegler and Nichols (1942)). It is a heuristic method of tuning where the integral gain and the dif-

ferential gains are set to zero. The proportional gain is increased until it reaches the ultimate gain K_u, At this stage the control loop has stable and consistent oscillations. The oscillation period is measured and labelled as T_u which along with the ultimate gain K_u are used to set the gains for the PID. The proportional gain K_p will be set to 0.6 of K_u, the integral gain K_i will be set to $\frac{2}{T_u}$ and the differential gain K_d will be set to 0.125 of T_u

This method can also be used for tuning P, PI and PD controllers, the method of calculating the gains is euclidated in table

Control Type	K_p	K_i	K_d
P	$0.5K_u$	-	-
PI	$0.45K_u$	$\frac{0.833}{T_u}$	-
PD	$0.8K_u$	-	$0.125T_u$
PID	$0.6K_u$	$\frac{2}{T_u}$	$0.125T_u$

Table 3.1: Ziegler-Nichols Hyper Parameters Tuning

The Linear Quadratic Regulator (LQR) controller is used for operating a dynamic system at minimum cost, which is described by a quadratic function. The weights of the input cost matrix and state matrix determine the allowable magnitude of control action that can be used. The LQR is an important part of the solution to the Linear Quadratic Gaussian (LQG) problem. The LQG observes the system using a Kalman filter and solves the Algebraic Ricatti equation to obtain the controller. This computation is performed every time the input cost matrix and the state matrix change. Hence, the computation of the controller is not performed in real time. Another disadvantage of the LQR is the requirement of a dynamic model for the system. The LQR's transient response is faster than that of the PID controller.

Another control algorithm that is used on self driving cars is the Model Predictive Controller (MPC). This is a finite-time horizon problem where the controller is computed at

every time step, based on the system states at the previous time step. This process is very computationally intensive with $O(n)$ complexity where n is the number of variables in the Ricatti equation, which must be solved at every time step. The MPC dynamically adapts to the changes in the system parameters, whereas the LQR only accounts for parameter changes when the input cost matrix and state matrix are updated and the PID controller does not account for parameter changes unless the user manually updates the controller. Like the LQR, the MPC requires a dynamic model for the system, whereas the PID controller does not need a model of the system.

Chapter 4

CHARTOPOLIS

In this section a detailed explanation of the proposed test bed is provided along with the explanation of the key image processing components. The functions and capabilities of the robot that have been used to emulate the self driving car is also explained in derail.

The self driving car test bed can be divided into two major subsections - the hardware subsection and the software subsection. The hardware subsection deals with the construction of the test bed, and hardware components of the robot that is being used. The software subsection discusses about algorithms that are powering the robot and the algorithms that are running on the test bed and the intersections.

4.1 Hardware components

The hardware is the most important part of the test bed as it is facilitating a platform to test out the algorithms that are being developed for emulating a self driving car. It starts with constructing the physical layout of the test bed with lanes marked and the traffic lights setup as shown in Figure 4.1. The layout of the test bed was first designed in CAD respecting the dimensions of the robot and the lane width this is shown in Figure 4.2.

The robot that was used to emulate a self driving car is the Pheeno Robot (Wilson *et al.* (2016)) developed in ACS Laboratories. The pheeno robot was developed as a low cost alternative for performing swarm experiments. These robots are modular platforms, constructed using commercially available actuators and sensors. The structure of the robot is 3D printed and laser cut. The Pheeno has the modularity to be scaled up or scaled down and can be added with many different sensors or actuators. These capabilities made the author to choose the pheeno robot for constructing the test bed.

Figure 4.1: Construction of the test bed with the intersection and the traffic lights mounted

The Core support structure of the pheeno robot is 3D printed with ABS plastics. The circular base and cap of the housing are poly-carbonate . The 3D printed parts of the pheeno robot uses an infill of 12 which is dense enough to run the experiments for the test bed. The robot uses a differential drive to move around and this is facilitated with two micro metal gear motors with extended back shaft. The motor of choice was a micro gear motor with 51.45:2 gear ratio, a wheel of 32mm diameter, with this configuration the pheenos can move with a speed ranging from 4cm/sec to 42cm/sec. A magnetic encoder is attached to the extended back shaft of the robot which has a linear resolution of 0.163 mm/tick. The robot also features a LED ring around its periphery. This enables the users to programs the robot to display multiple states.

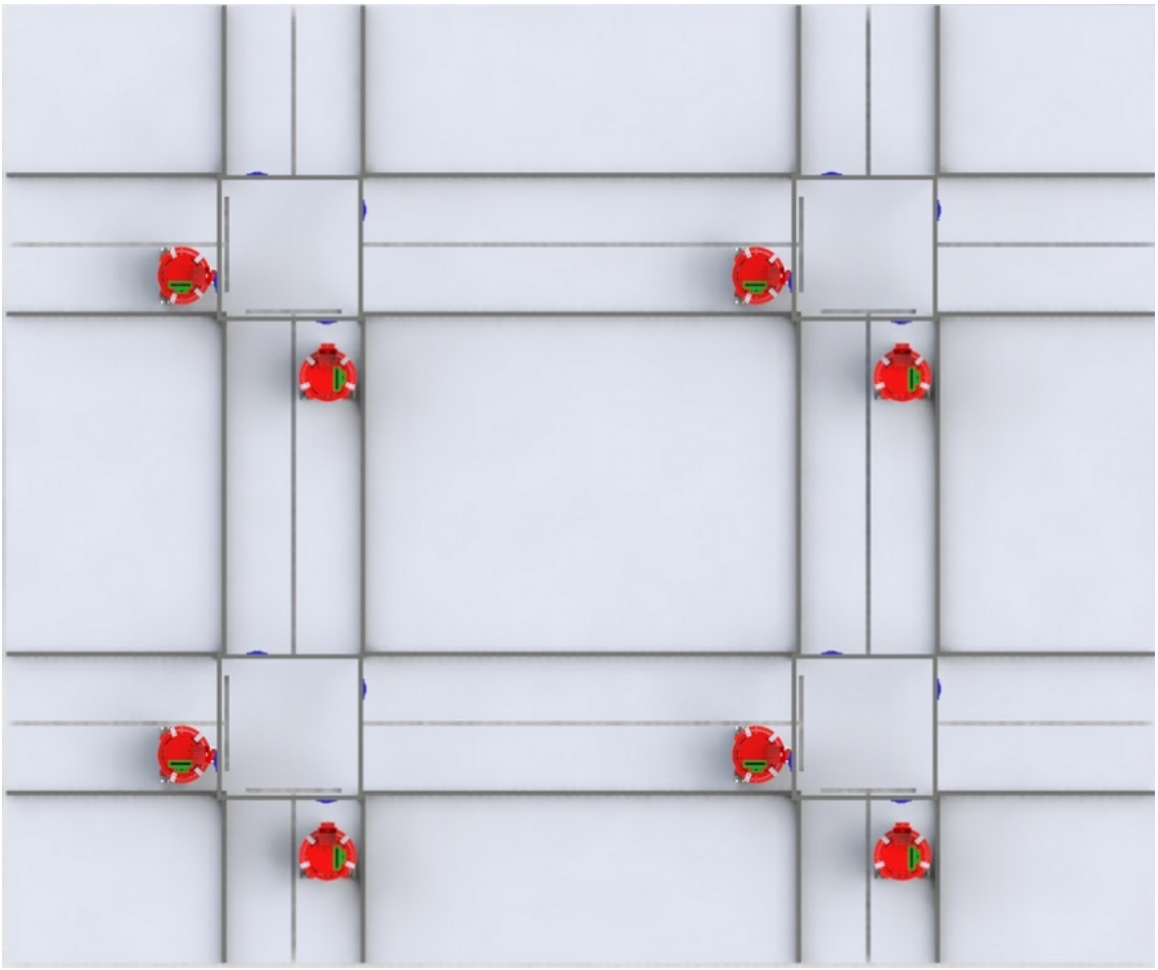

Figure 4.2: CAD model of the test bed representing a block

Along with this actuator there are a couple of sensors which are installed in the robot. The robot has six Sharp GP2YOA41SKOF IR sensors with a range of 4 - 30 cm. IR sensors with different ranges could be substituted with the interface pins that are provided with the robot. The IR sensor on the periphery of the robot gives a measurement of distance of the objects around the robot. It also has an Inertial Measurement Unit (IMU) which has three degrees of freedom magnetometer, three degrees of freedom gyroscope and three degrees of freedom accelerometer. The IMU measures acceleration, magnetic intensity and rate of change of angular position of the robot in all three dimensions. The robot also features a camera which has a 5MP Omnivision 5647 sensor in a fixed focus module.

The robot has a Raspberry Pi 3 computer as the primary on board processing unit. The Raspberry Pi is a credit card sized Linux computer the gives the flexibility for the user to program in multiple programming languages and run all the programs that would run on a native Linux system. It has a Quad Core 1.2Ghz Broadcom BCM2837 64 bit CPU and 1 GB of RAM, BCM43438 wireless LAN and Bluetooth Low, Energy on board. It has 40 GPIO pins, four USB 2.0 ports, a full size HDMI and a CSI camera port. Along with this the robot has a secondary processing unit which is Teensy 3.2. The Teensy board has a 32 bit Cortex M4 processor with a clock speed of 72 MHz, it has a RAM of 64 KB and 34 digital input output pins, 21 analog input pins. The Teensy board supports USB, serial, SPI, I2C and CAN communication protocols

4.1.1 Motor Model

The motor mentioned above is to be modelled to implement a controller to have a closed loop system which will make the robot follow the commands given by the algorithm. Two modelling techniques were used to study the model of the motor, an analytical method where the model of the motor is derived by differential equation of the mechanical systems and the electrical systems.

A DC motor has an armature, a permanent magnet and a shaft. The armature is connected to the electrical circuit which magnetizes and demagnetizes the armature. This will induce rotation in the armature in accordance with law of Electromagnetic Induction. This rotation is a result of the torque produced on the shaft of the motor, which is proportional to the current flowing through the armature. The rotation of the motor would produce a back emf on the armature which is proportional to the angular velocity of the motor. With these equations a relation between electrical power and mechanical power can be derived.

$$\tau = K_t I$$

$$E = K_b\omega$$

$$\tau = J\frac{d\omega}{dt}$$

where, τ is the torque on the shaft of the motor

k_t is the torque constant of the motor

I is the electric current flowing through the armature

From Kirchhoff circuit law a differential equation of the electrical system can be written as

$$V - IR_a - L_a\frac{dI}{dt} - E = 0$$

The armature inductance is very low which can be approximated to zero

$$V - \frac{\tau}{K_t}R_a - K_b\omega = 0$$

Rewriting the torque equation we have

$$V - \frac{J\frac{d\omega}{dt}}{K_t}R_a - K_b\omega = 0$$

Taking Laplace transform and rearranging the equations would give,

$$\omega = \frac{V}{\frac{Js}{K_t} + K_b}$$

This is the transfer function that relates the angular velocity of the shaft of the motor to the voltage applied to the motor. To get the relation between the linear velocity of the robot and the voltage applied to the motors the equation is modified slightly.

$$v = \frac{r_a V}{\frac{Js}{K_t} + K_b}$$

where, V is voltage applied to the armature

I is the current flowing through the armature

R_a and L_a are resistance and inductance of the armature

E is the back emf flowing through the armature due to the rotation of the shaft

r_w is the radius of the wheel

v_w is the linear velocity of the wheel

By getting the motor parameters from the data sheet of the motor, the transfer function of the motor is found to be

$$v = \frac{0.015V}{\frac{Js}{0.0187} + 0.0192}$$

An experimental method was also used to obtain the model of the motor. In this model a step voltage was given to the motor and the response of the motor was plotted as shown in Figure ( ). This plot is approximated to a first order transfer function, the pole location was obtained using curve fitting method.

This transfer function can be used to design a controller for making the wheel of the robot to follow the commands given by the processor.

4.2 Software components

With the given hardware the software segment plays the crucial role in realizing the autonomy of the robot, It runs complex algorithms to define the motions and allocates a trajectory for the robot by utilizing all the sensors and actuators. The control diagram is shown in Figure 4.3

4.2.1 Camera Calibration

The camera of the robot plays a crucial rule in emulating the robot as an autonomous car. Like the Tesla autonomous cars, this robot as well does not heavily depend on LIDAR data since camera being the major part in deciding the actions to be taken. The image from the camera should not be distorted. For this purpose the camera calibration procedure is

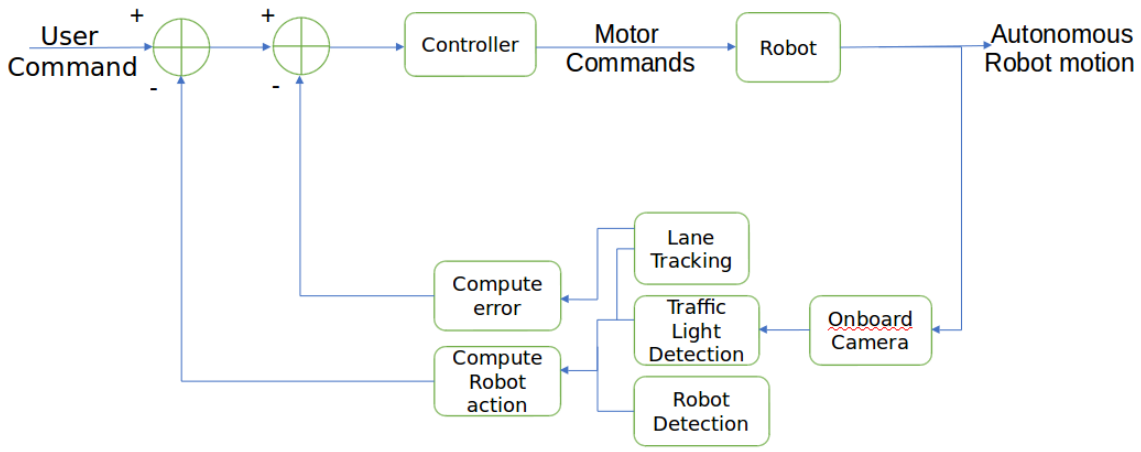

Figure 4.3: Block diagram of the closed loop controls for the robot

carried out.

Geometric camera calibration, also referred to as camera resectioning, estimates the parameters of camera. These are the intrinsic and extrinsic parameters. Intrinsic parameters are related to the optics of the camera. These parameters encompass focal length, image sensor format and principle point. The extrinsic parameters denote the coordinate system transformations from the world coordinates to the camera coordinates. These parameters define the position of the camera centre and the cameras heading in the world coordinates.

The major distortions in an image are the radial and the tangential distortion. Due to the radial distortions, straight lines will appear curved. There are two types of radial distortion - Barrel distortion where the image would barrel outwards making the pixels in the centre to distort more relative to the pixels in the edges, and Pincushion distortion where the pixels in the edge would distort more relative to the centre.The tangential distortion occurs when the lens is not aligned parallel to the imaging plane.

An image is a two dimensional representation of the three dimensional world. The world coordinates are transformed to pixel coordinates. This is referred to as forward

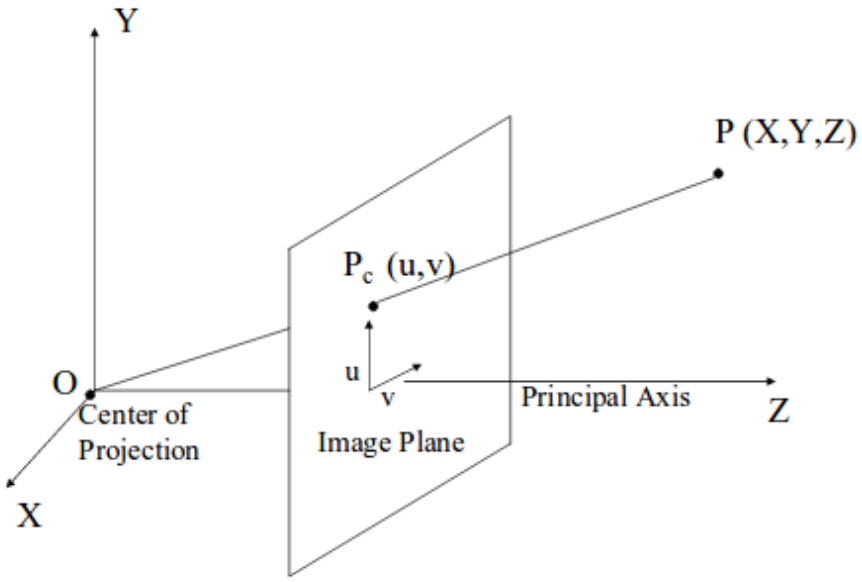

Figure 4.4: Projection of an object in the world frame represented by X,Y,Z on to an image plane,(Majumder (2010))

projection. The projection of a scene depicted in Figure 4.4 follows the following formula

$$u = f\frac{X}{Z}$$
$$v = f\frac{Y}{Z}$$

where, u is the x coordinate of the object in image plane

v is the y coordinate of the object in image plane

X is the x coordinate of the object in world plane

Y is the y coordinate of the object in world plane

f is the focal length of the camera

When the origin of the two dimensional image coordinate system doesn't coincide with where the principle axis meets the image plane, a translation of the image is to be performed. Let this translation be defined as t_u, t_v. Now, u and v becomes,

$$u = f\frac{X}{Z} + t_u$$
$$v = f\frac{Y}{Z} + t_v$$

The same equations can be written in an perspective transformation matrix

$$P_c = MP$$

To express this camera image in inches, the resolution of the camera is to be known. This can be denoted as m_u and m_v. So, the transformation matrix is dependent on the focal length of the camera, X, Y and Z coordinates of the object, the translation vectors t_u, t_v, the scaling vectors m_u and m_v. This is denoted by K, where K is called the intrinsic parameter matrix for the camera. So the projection of the object on the image plane follows the following equation

$$P_c = KP$$

When the camera does not have its projection centre at the origin the image will be rotated and translated to correct this. The image should be translated and rotated accordingly to align the projection centre. Let the translation vector be denoted by T_x, T_y, T_z and the rotational matrix be denoted by R. The extrinsic parameter E is given by

$$E = (R|RT)$$

The complete camera calibration matrix can be represented by

$$C = KR(I|T)$$

the projection on the image plane of object P after calibration is given by

$$P_C = CP$$

This calibration matrix is obtained by getting multiple images of a checker board in different angles of known dimensions and calculating the matrix parameters.

The image that is captured by the camera is passed to the distortion removal algorithm to remove any distortions in the image. This image is then passed to the process pipeline which performs the image processing algorithms to detect the elements of a traffic scenario.

4.2.2 Lane Detection

The lane detection algorithm taken in an undistorted image to estimate the location of lanes in a given frame. This estimate is then used to control the trajectory of the robot. The image that is received by the lane detection algorithm is in RGB color space, this RGB image is first converted into HSV color space (Schwarz *et al.* (1987)). HSV color space is an alternative representation of the RGB color model designed in the 1970s. The H corresponds to the hue value which represents different colors, the S corresponds to saturation dimension of the various shades of the color represented by the Hue and the V denotes the Value of white or black mixed with the color represented by hue and saturation. Thus HSV gives a cylindrical dimension to color and much more control of the choice of color as shown in Figure 4.5

The hsv of the test bed is shown in Figure 4.6 , this HSV image is masked for extracting color with high brightness or 'V'(value). This would result in an image with lanes and other noise over the horizon as depicted in Figure 4.7. The resulting image was too noisy which made random white spots appear all over the image. Gaussian smoothing was done over the image to remove this noise. Gaussian smoothening (Fisher *et al.* (2003)) uses a kernel which convolutes through each pixels of the image and would smoothen the image. This is a slow process which is depended on the size of the kernel and the size of the image. The

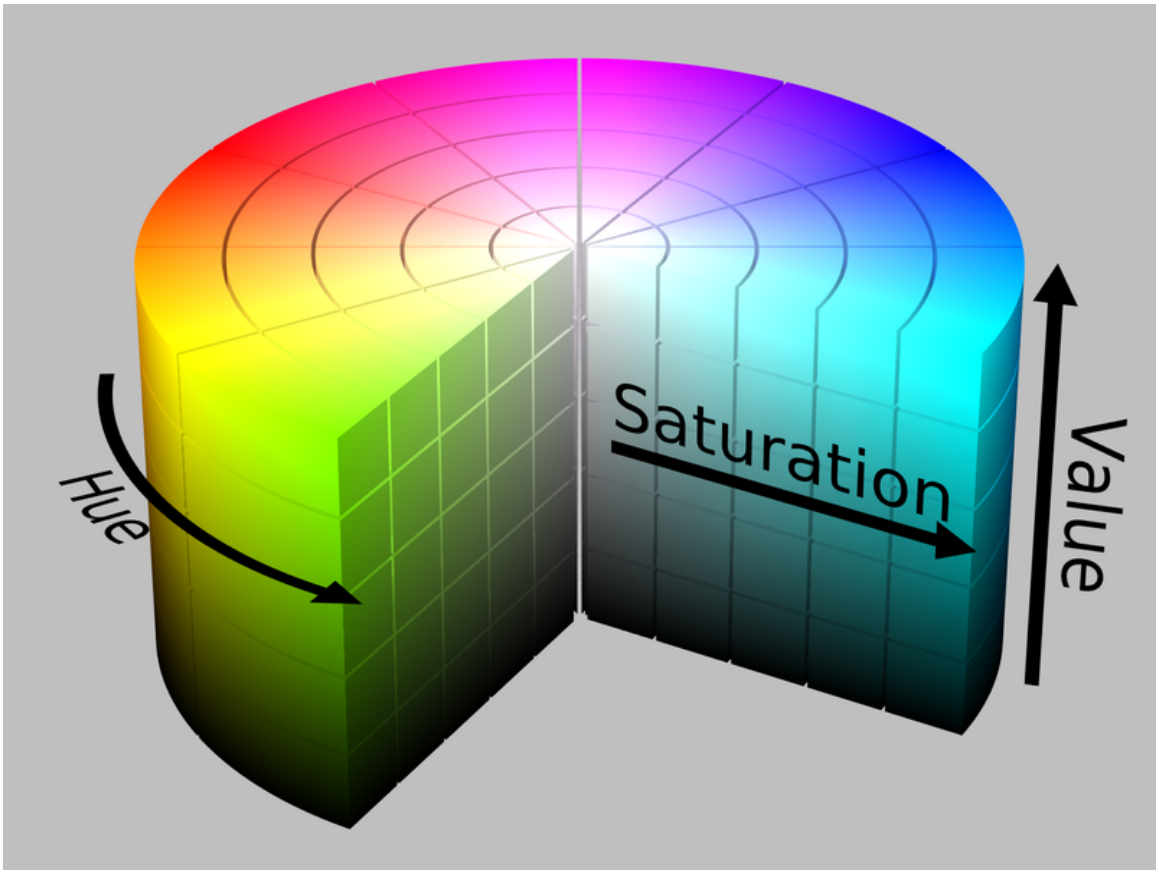

Figure 4.5: HSV color solid cylinder representing the Hue Saturation and Value parameters and their effect in the choice of color

kernel is a gaussian distribution matrix. An example of a gaussian kernel is depicted below. The image after applying the kernel is represented by the Figure 4.8

$$3X3Kernel = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

After the gaussian blur the image would still have some noise and big unwanted segments. These segments were removed by eroding the image. The erosion operation(Works (1999)) is similar to gaussian blur, it uses a kernel to perform the operation. The kernel is usually a odd square unity matrix. This matrix is convoluted over each pixel. The pixel

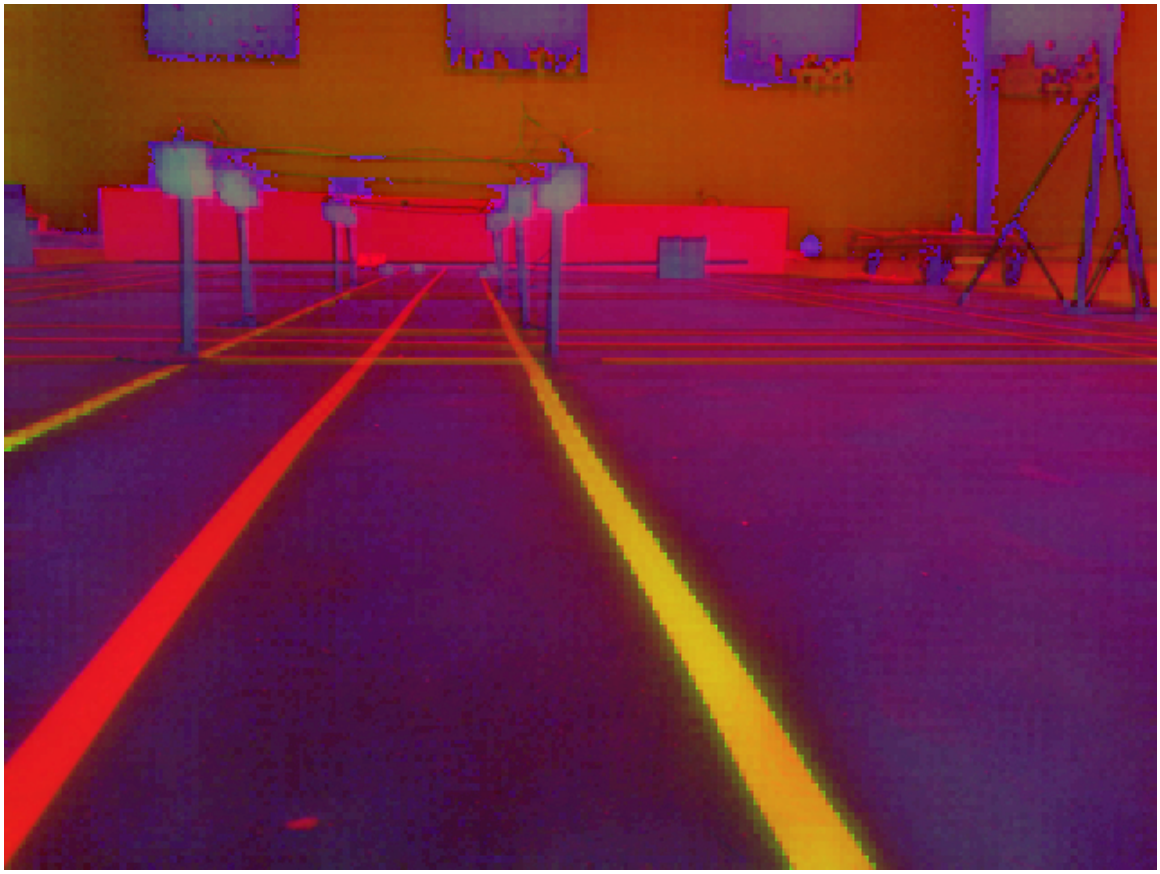

Figure 4.6: The lanes of the test-bed represented in HSV color space

value of the original image will be retained only if all the pixels under the kernel is one, else it will be made zero. This makes the pixels near the boundary to be discarded depending on the size of the kernel which decreases the size of the objects in the image. Erosion only works for binary image. While masking the image, it is mapped to binary format. The image after erosion is shown in the Figure 4.9 Even after the erosion the objects above the horizon are not removed from the image. To remove these a trapezoidal region of interest is selected and all the other pixels are made black. This would leave only the road to be shown in the image as depicted in Figure 4.10. The height of the trapezoid was determined by taking a point that is less than that of the infinite point for the camera. This frame after erosion is sent to the edge detection algorithm, the canny edge detection was chosen for

Figure 4.7: The HSV image is masked for pixels with high Value and is represented as a binary image

this. The canny edge detection algorithm first calculates the gradients based on the intensity of the pixels in horizontal direction G_x and vertical direction G_y. This gives out two images one with vertical gradient and one with horizontal gradient. From this, the edge gradient and the angle of each pixel is calculated as given below. The gradient direction will be perpendicular to the edges in the image.

$$G_{edge} = \sqrt{G_x^2 + G_y^2}$$

$$Angle(\theta) = \tan^{-1} \frac{G_x}{G_y}$$

Figure 4.8: Image after smoothening using Gaussian Blur with 5X5 kernel

After getting the direction and the magnitude of the gradient, the image is scanned again to remove unwanted pixels which are not attributing to an edge. To achieve this, every pixel is checked if it is a local maximum in its neighborhood in the direction of the gradient. If it is not the local maximum, it is suppressed to zero. This process eliminated unwanted noise in the image showing up as a edge. The algorithm also thresholds the magnitude of the gradient which gives a control on choosing how sharp of an edge is required. The resulting edge image will be binary image with thin edges as shown in Figure 4.11.

Using the edge image, the lines in the image are computed using Hough transforms(Duda and Hart (1972)) to extract the lanes. The Hough transform is a popular technique to detect shapes. A line can be represented by the liner equation $y_i = mx_i + c$, it can also be rep-

Figure 4.9: Image after eroding with kernel size of 11x11

resented as $c = -mx_i + y_i$. In this mc-plane for a given point x_i, y_i there can be multiple lines drawn with different slopes and intercepts. Consider a line segment, for different x_i and y_i on the line segment if all possible intercepts and slopes are calculated for the given points, there would be one intercept that is common for all the three points. The line can be represent with this slope and intercept. But for a vertical line the slope is infinite. To more efficiently represent the line, it is converted into polar coordinate system in which the line is described as $\rho = x_i cos\theta + y_I sin\theta$. Each point x_i, y_i in the xy-plane gives a sinusoid in the $\rho\theta$ plane. Assuming there are M co-linear points lying in the line, $\rho = x_i cos\theta + y_I sin\theta$. These points will give M sinosoids that would intersect at ρ_i, θ_j in the polar plane. This ρ and θ represents the line. This process is computationally very intense so a voting method

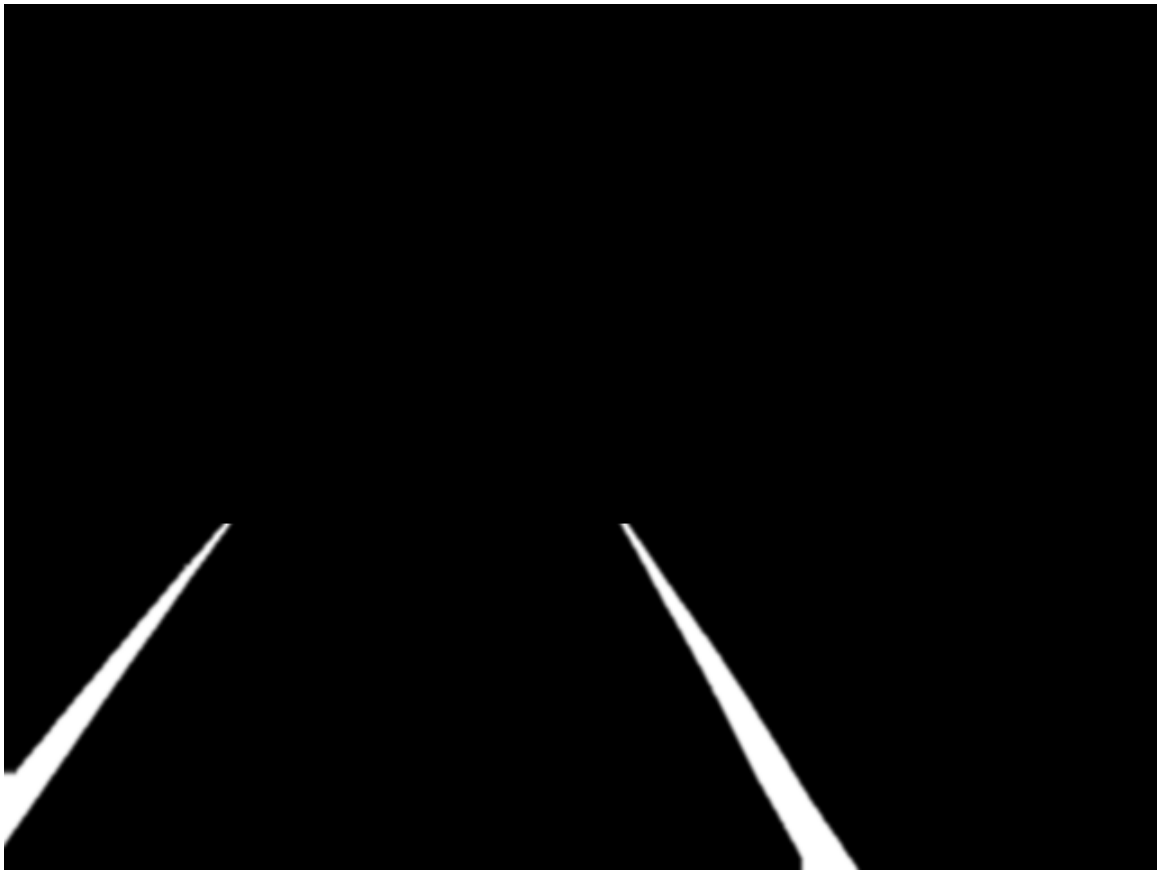

Figure 4.10: Trapezoidal region of interest which removes objects that are above half of the infinite point

is used to determine the line segment, this process is called probabilistic Hough transform (Kiryati *et al.* (1991)). The probabilistic Hough transform doesn't take all the points on the line into consideration but selects few points randomly and computes the extremes of the line segment. The lane with the Hough lines is shown in Figure 4.12

The resultant lines are separated as left lane and right lane based on the slope of the line. This is seen in Figure 4.12. The average between the left lane and the right lane is calculated to get the lane centre, the lane centre is plotted in Figure 4.13. An error is computed between the centre of the lane and the centre of the frame captured by the camera. This value represents the error between the robot and the lane, as the camera is placed on the centre point of the robot with respect to x axis.

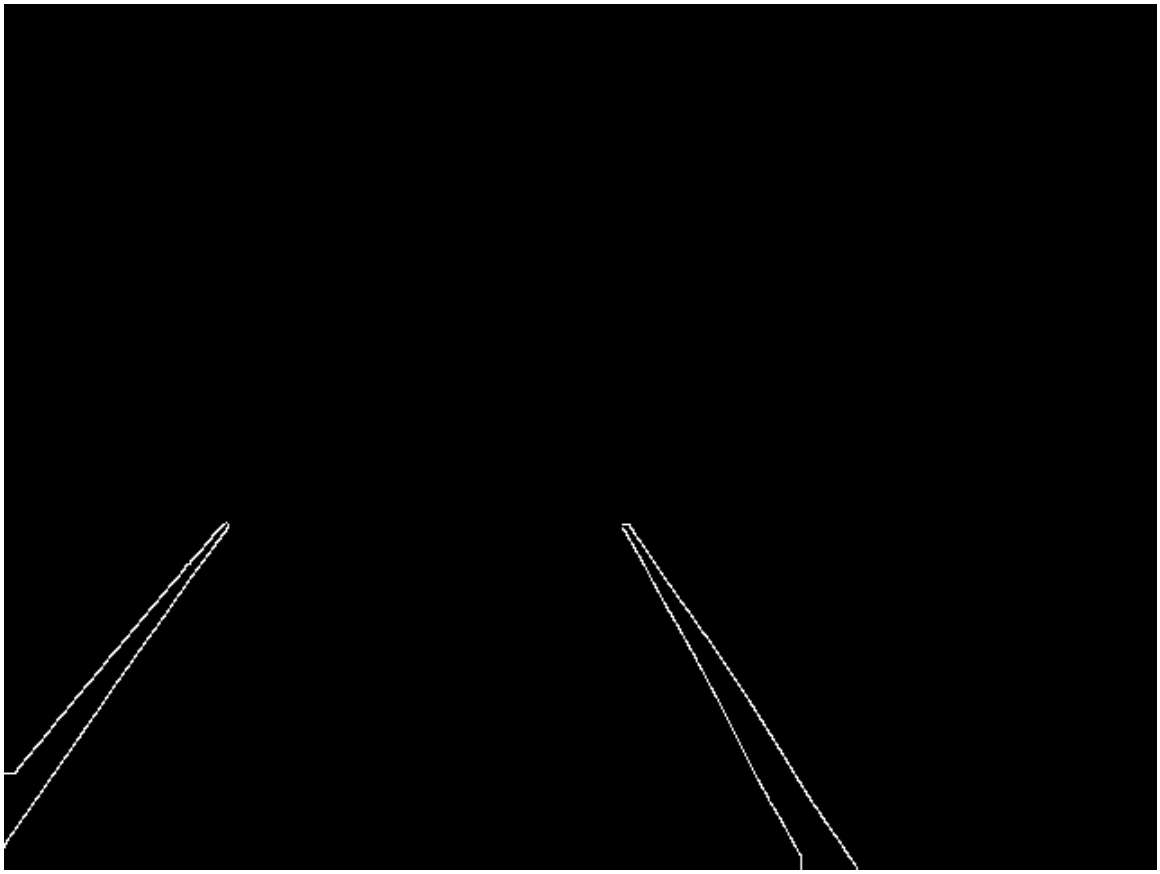

Figure 4.11: The Canny edge with a threshold band between 100 and 200 would result in extracting the edges of the lane

Another approach was experimented to compare the result with the afore mentioned algorithm. The frame that is captured is converted into a binary image i.e. a black and white image, this image is sent to a Homography algorithm that converts the image to birds eye view with the help of the transformation matrix. Homography maps points in one image to the corresponding points in another image, this mapping is performed by the homography matrix which is 3X3 matrix, this is briefed below

Let the a set of points taken from the first image be X_1, Y_1 and X_2, Y_2 taken from the second image. The mapping of the point from the second image to the first is done by multiplying X_2, Y_2 with the transformation matrix.

Figure 4.12: Probabilistic Hough transforms detecting the lines on the lane with minimum length of 10 pixels and maximum line gap of 20 pixels

$$\begin{bmatrix} X_1 \\ Y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} X_2 \\ Y_2 \\ 1 \end{bmatrix} \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

But in the case under study there is only one image and the perspective transformation matrix is not available. To solve this two lines parallel to the lanes are drawn and the points of intersection of these lines with the horizontal line drawn at a point less than the infinity point is taken as edge points of one image and the points for the second image is taken as the intersection of the two vertical lines drawn from the bottom of the image to the horizontal line drawn at a point less than the infinity point. This forms a set of four points these are

Figure 4.13: The centre of the two lanes is estimated by averaging the left and the right lane pixel positions

used to compute the Homography transformation matrix.The transformation matrix is used to map the points from the actual lane image to the birds eye view.

The birds eye image is used to compute the sum of pixel values of each columns, this sum gives a vector which has a length equal to the width of the image. This vector would have two regions of high values which corresponds to the start of the lane in the x direction. This vector is used to separate the left lane from the right lane. From this starting point, a sliding window is used to scan through a region wider than start of the left and right lane. The points inside the sliding window is fit with a second order polynomial which is used to estimate the curvature and the position of the left and the right lanes. The curvature is used to re-position the sliding window for the next segment. Similarly the entire image

is scanned to estimate the polynomial that represents the entire left and the right lane. With the help of these polynomials the center of the lane is calculated with which the error with the image center is computed. This method was found to be more accurate but computationally slow as the algorithm is trying to fit a cure to a given number of points which runs for multiple passes.

The error computed is sent to the control algorithm that dictates the motors to re-position the robot to align itself with the centre of the lane. The control algorithm is a PID controller that takes in error values and would compute the angular velocity of the robot. This angular velocity is used to compute the individual wheel velocities $v_r and v_l$ using a Robot model as shown below. This is a kinematic model that computes the position and heading of the robot in the Cartesian space given the velocities of the wheels.

$$\dot{x} = \frac{R}{2}(v_r + v_l)cos\theta$$

$$\dot{y} = \frac{R}{2}(v_r + v_l)sin\theta$$

$$\dot{\theta} = \frac{R}{L}(v_r - v_l)$$

where,

L is the axle length of the chassis

R is the wheel radius

x, y position of the robot in the Cartesian space

θ heading of the robot in the Cartesian space

Though this model is very efficient for commanding the robot to move in desired directions, designing a controller with this model is very cumbersome. So a unicycle model is used to design a controller for the robot. The unicycle model was considered

$$\dot{x} = vcos\theta$$

$$\dot{y} = vsin\theta$$

$$\dot{\theta} = \omega$$

where, ω is the angular velocity of the robot and v is the linear velocity of the robot

But this model failed to give individual wheel velocities for a differential drive robot so this model is merged with the kinematic model to get a differential drive kinematic model which was used to command the wheels.

$$v_r = \frac{2v + \omega L}{2R}$$

$$v_l = \frac{2v - \omega L}{2R}$$

The error message is sent from the raspberry pi computer to the Teensy computer which computes the individual wheel velocities. The PID loop runs at a speed of 48 Khz and the feed back data from the image processing segment is at the rate of 0.002 Khz. This delay is modeled in the feed back loop.

4.2.3 Traffic Light Detection

Traffic lights have been playing a major role in regulating the flow and control of the traffic in an intersection in the urban traffic system since the dawn of the age of automobile. It has mostly been a trivial task for the humans to see and detect the vivid red and green lights and respond accordingly, but when this comes to an autonomous car there are many factors into consideration, the car should be avoiding detecting false positives and false negative as any of these would result in a traffic light skipping which may endanger the vehicles around. The autonomous car should first recognize the traffic light structure and

Figure 4.14: Image after masking for the green color in the frame

then detect the state of the traffic light. We are using classic image processing techniques to recognize the traffic light and compare it with a deep neural network algorithm.

The frame captured by the Raspberry Pi camera is pipe-lined to traffic light detection algorithms after lane detection algorithm. The received image is converted to HSV space and red and green color is masked out in the image using experimented HSV values. This masking is shown in Figure 4.14 and Figure 4.15. The brightness of the traffic light causes pixel saturation (Hasinoff (2014)) and blooming, this makes it very difficult to separate the colors using the HSV mask. Saturation is a type of distortion when the image is limited to some maximum value or charge interfering with the measurement of bright regions of the scene and blooming is when these excess charges spill over to the neighbouring pixels

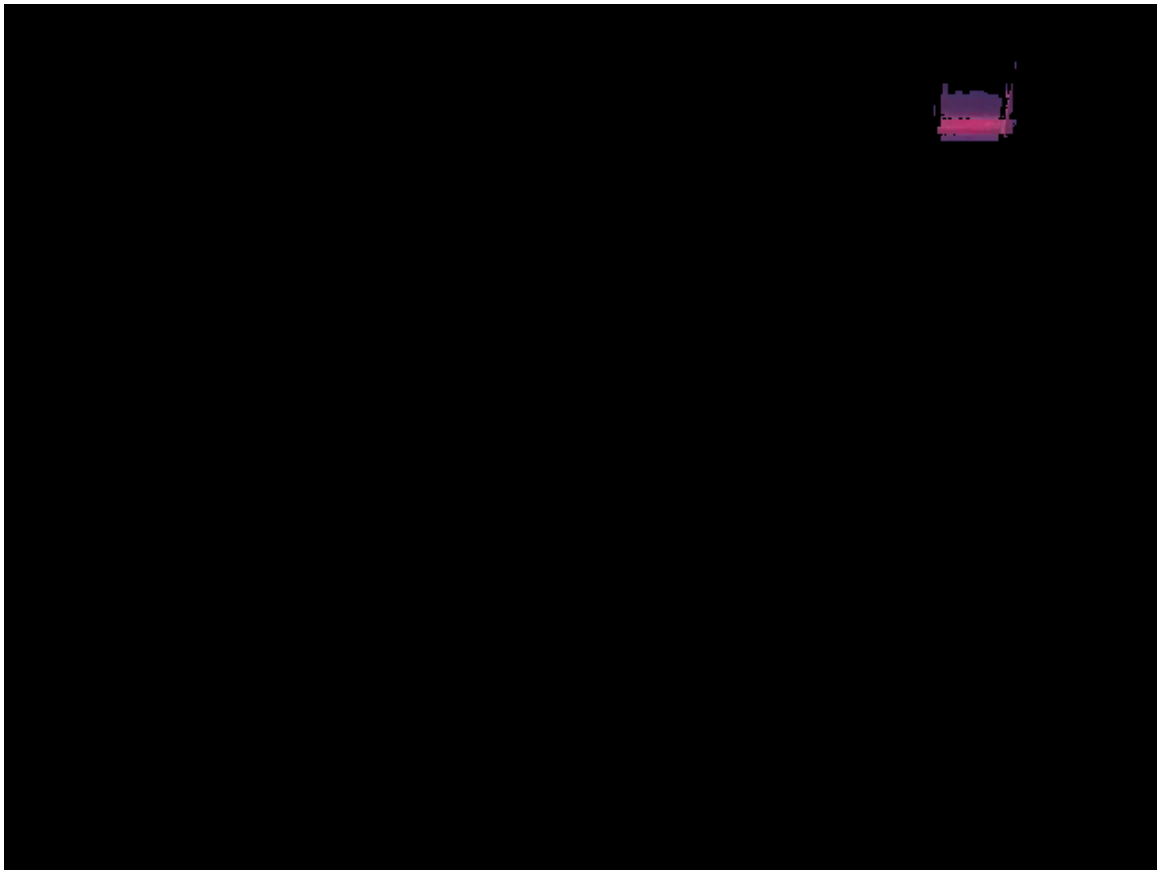

Figure 4.15: Image after masking for the red color in the frame

saturating them which would other wise not be saturated. When the overexposure is very high the pixels become saturated in all the color channels, there are many methods to correct this over exposure, one of the common method used is Dynamic ranging as proposed in (Reinhard *et al.* (2010)). This process captures multiple images in different exposure levels of the same subject and would render an image from all these images. Due to constrains in the processing speed and the real time requirement this process was not feasible to be implemented. A mechanical design solution was brought out where the light would be diffused before it leaves the structure. A diffusing structure was designed by studying the HSV data for different intensity of diffusion.

After masking the entire frame for the desired color, a region of interest is selected

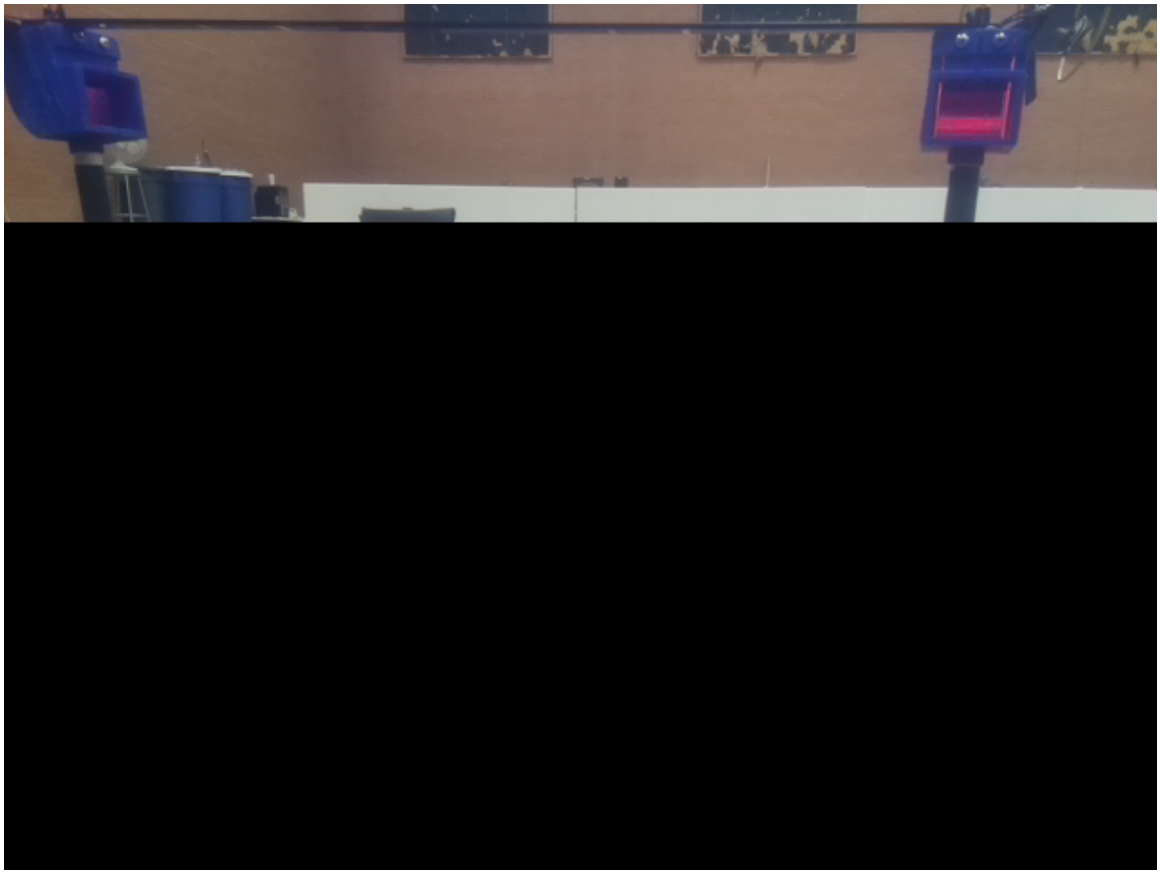

Figure 4.16: Selecting the region of interest for extracting the traffic lights

which separates the top of the frame, this would leave only the traffic structure visible and any other red or green in the image would be removed. This is shown in Figure 4.16. The image is used to find any counters using the Topological structure analysis by border following algorithm (Suzuki *et al.* (1985)). This would give the boundary points of all the shapes detected in the image. A rectangle is fit to these points to extract the dimension of the traffic light. The height and the width of the bounding rectangle is used to estimate the distance of robot from the traffic light as shown in Figure 4.17 and Figure 4.18. This distance is acquired by getting the pixels per centimeter value, after acquiring the distance from the traffic light the robot is made to respond to the traffic light only if it close enough to the traffic light provided it is not violating any other operating conditions. Based on what

Figure 4.17: Bounding box around red traffic light, the height and the width of the rectangle is used to estimate the distance of the traffic light

color the robot is detecting the corresponding command is sent to the Teensy controller to perform the required action. If the robot is encountering a Green signal, it has four choice of directions of equal probabilities to choose the trajectory. If the robot chooses to turn it will do so by ailing itself in the center of the lane and turning ninety degrees to the desired direction. While performing this action the robot will operate in open loop condition.

A neural network was used for comparing the performance results of the computer vision approach. The inception(Szegedy *et al.* (2015)) network architecture was selected for comparing and transferring. The learning process was used to re-train the last layer of the inception of inception model for detecting traffic lights. The system was trained on GTX 960m GPU with CUDA support. A locally created dataset was used with 100

Figure 4.18: Bounding box around green traffic light. The text on the video is used for post analysis

positive images for green and red each and 100 negative images were used to retrain the model. This gave a very good prediction with an accuracy greater than 99 percentage. But this process of classification took approximately 3 seconds in a GTX 960m GPU to give a result when an image is passed through the model so this was not experimented on the raspberry pi computer considering the high computational requirements.

Another functionality for the traffic light detection was added, where if robot detects red light but stops past the stop line it would reverse to align itself with the stop sign. This is to compensate for the fact that there is no yellow traffic light condition in the test bed, as mixture of red and green makes yellow and the camera is not sensitive enough to distinguish between shades of red and green with yellow.

4.2.4 Robot Detection

Another important factor in road traffic is the requirement to detect other vehicles on the road. This would ensure there are no crashes and the robots are following the rules of the road. To facilitate this, a light ring is added to each robot which is used as a medium of communication between the robots. This light ring is mimicking the lights on the rear of a car. It exhibits four states, where one state is nominal operation where the robot follows lane, one state exhibits the robots inability to detect any lanes, one state indicates the direction of turn Figure 4.19 and the last state represents rogue mode of the robot where the robot will not follow any traffic rules. The robot has a probability of 0.1 to get into rogue mode while it is operational. In rogue mode the robot doesn't respect traffic lights and would not detect other robots in the test bed. It would also randomly choose to increase its speed between 30 to 70 percent of the current operational speed.

The light ring used was a Neo pixel RGB array which can display multiple colors. The neopixels receive data with a fixed frequency of 800 Khz where each bits require 1.25 microseconds and each pixels have 24 bits which makes the data streaming to a speed of 30 microseconds. After the streaming the data for the last pixel the transmission should be held at least for 50 microseconds for the colors to latch. This reduces the speed of the loop and introduces a delay in the system. Hence this delay should also be considered for the feed back loop.

The image of the current frame is also pipe-lined to the robot detection algorithm. The robot detection algorithm first segments the region where the light ring can be expected. After segmenting the image, the image is converted into HSV color space and it is masked for red and green as shown in Figure 4.20. The red and green colors are used to indicate the sate of nominal operation. The size of the box bounding the detected colors is used to estimate the distance of the robot and the rate of change of size is used to detect the velocity

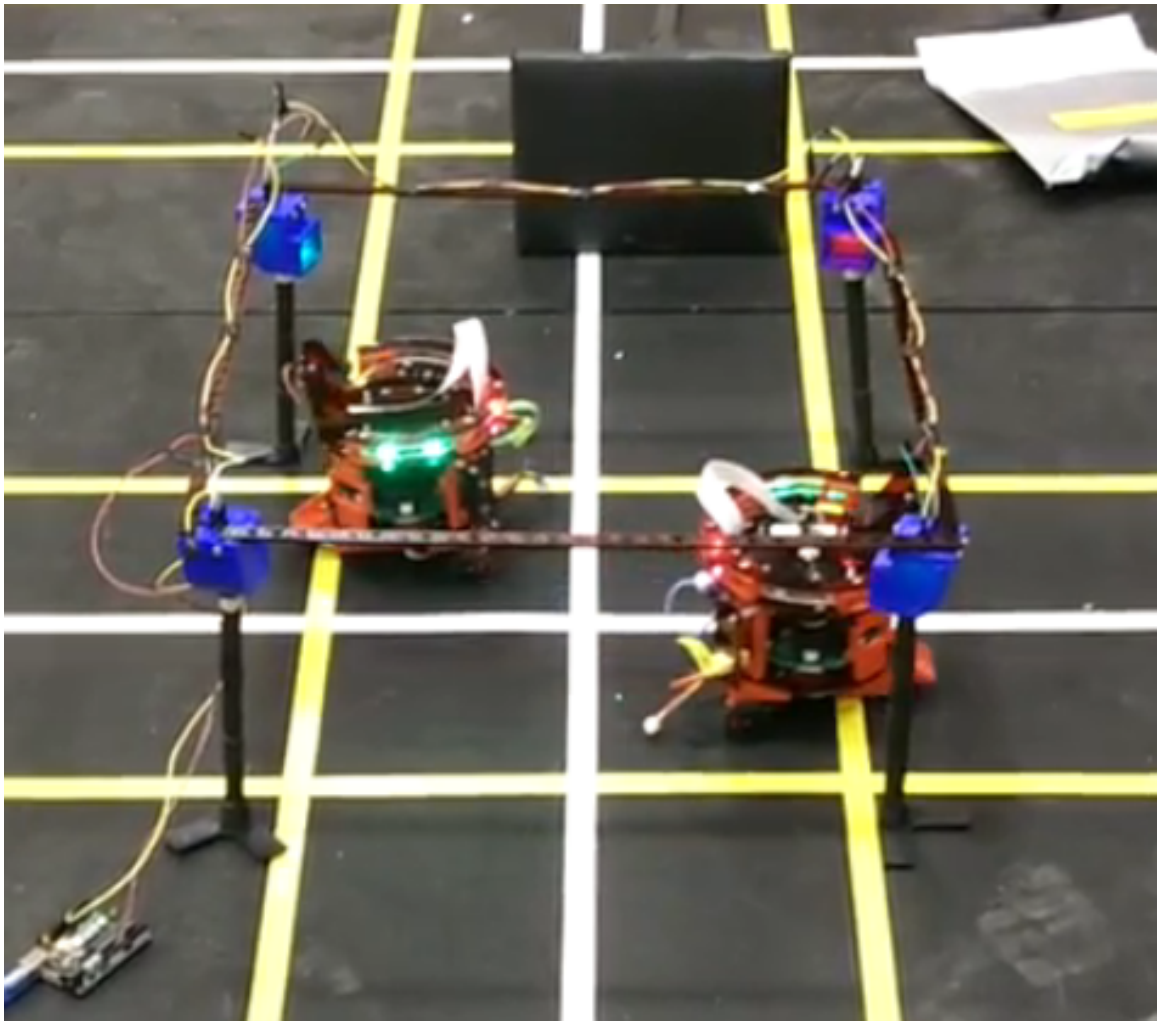

Figure 4.19: The Robots indicating left and right turns using the indicator LED

at which the robot ahead is travelling as shown in Figure 4.21.

4.2.5 Manual Control of the car

The test bed is trying to emulate the urban conditions of the current traffic scenario which dictates the requirement to have the cars be controlled manually. To enable this need, the test bed was equipped with a router which was enabling communication between the robots and a command station. All the robots were connected to the same network and every robot was assigned with a unique static IP address. The command station was also

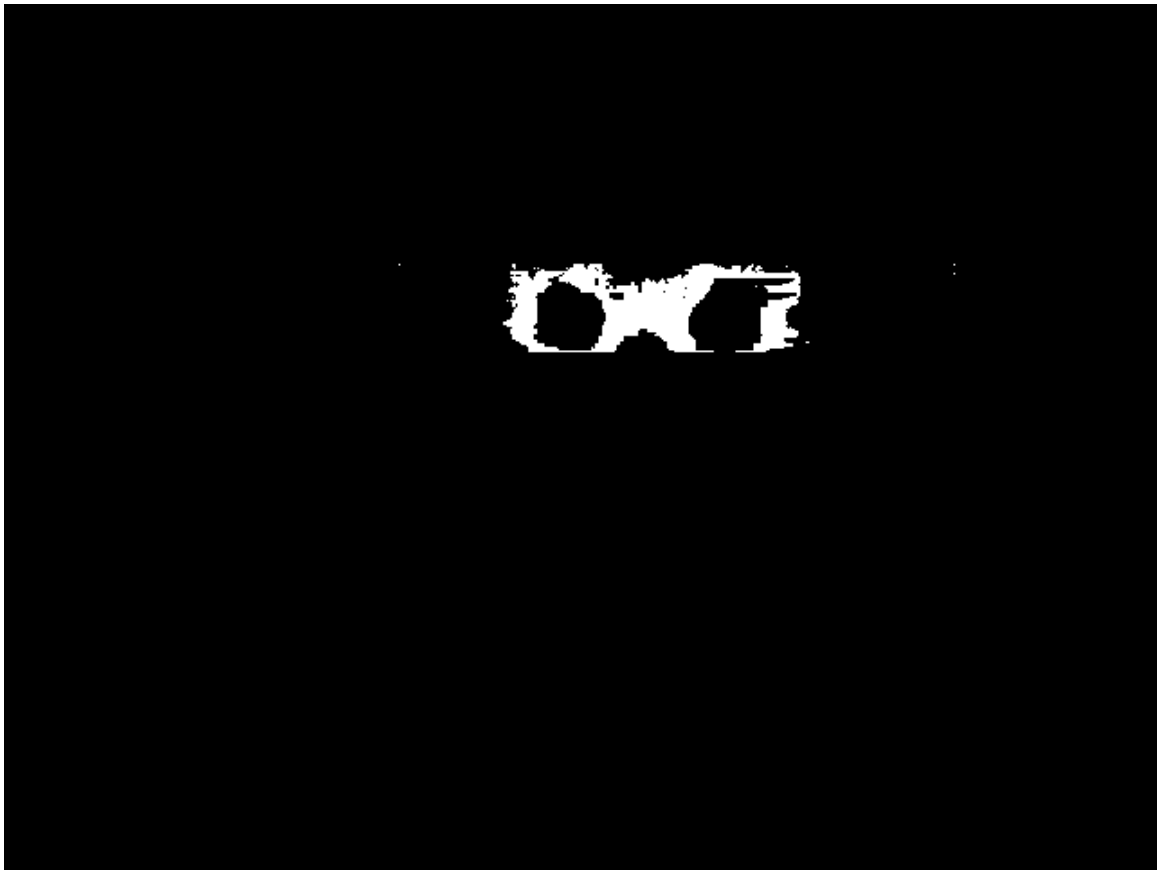

Figure 4.20: Masking the LED ring using the HSV value of the LED

connected to the same network. Every robot would create a socket for communicating to the command station, if the command station wants to communicate to a particular agent, it will connect to the socket using the knowledge of the IP and the port created by the robot. The station would send the required message and would exit the connection and would move to the next robot. The manual controller from the command station should visually see the movements of the robots in the test bed to control it. An experiment was carried where the robots would relay the video while they are being controlled, in this experiment it was found that the band width of the network is not wide enough to receive the video and send control commands for multiple robots.

Figure 4.21: Bounding Box around the LED of the robot ahead which is used to estimate the distance of the robot in front

Chapter 5

EXPERIMENTAL RESULTS

In this chapter, the results of different experiments are shown. There were two main experiments conducted - one was on a data set of road videos and another was on the test bed. The first experiment was on the road recordings which was to assert the lane tracking algorithms performance on a real road scenario and the second was where all the algorithms were tested on the test bed shown in the Figure 5.3.

The lane tracking algorithm was tested on a data set of video recordings of roads provided by Cognitive Technologies. The algorithm was able to estimate the position of the lanes on a clear bright day when the lanes were very clear and visible for the human eyes. This is shown in Figure 5.1. With minor tweaking of the parameters of the traffic light detection algorithm it was able to detect the the traffic lights on a real world condition.

The experiments with the test bed provided consistent results with the lane tracking algorithm, the traffic light detection and the robot detection algorithm. Different lighting conditions were experimented where the robots even performed in a condition similar to night conditions and the robots were attached with a flash light mimicking a head light of a car.

An experiment was orchestrated which was intended on comparing the results of human driving and autonomous driving on the test bed. The manual control of the robot was exploited to perform this experiment. A study group of six students were asked to drive around the test bed for five minutes sticking to the rules of the test bed, i.e. following lanes, stopping for traffic light and re-flowing back into the test bed. In this experiment, they were given a game pad to control the robots and allowed to move around the test bed for getting better perspectives. Their performance was recorded with the on-board camera which was

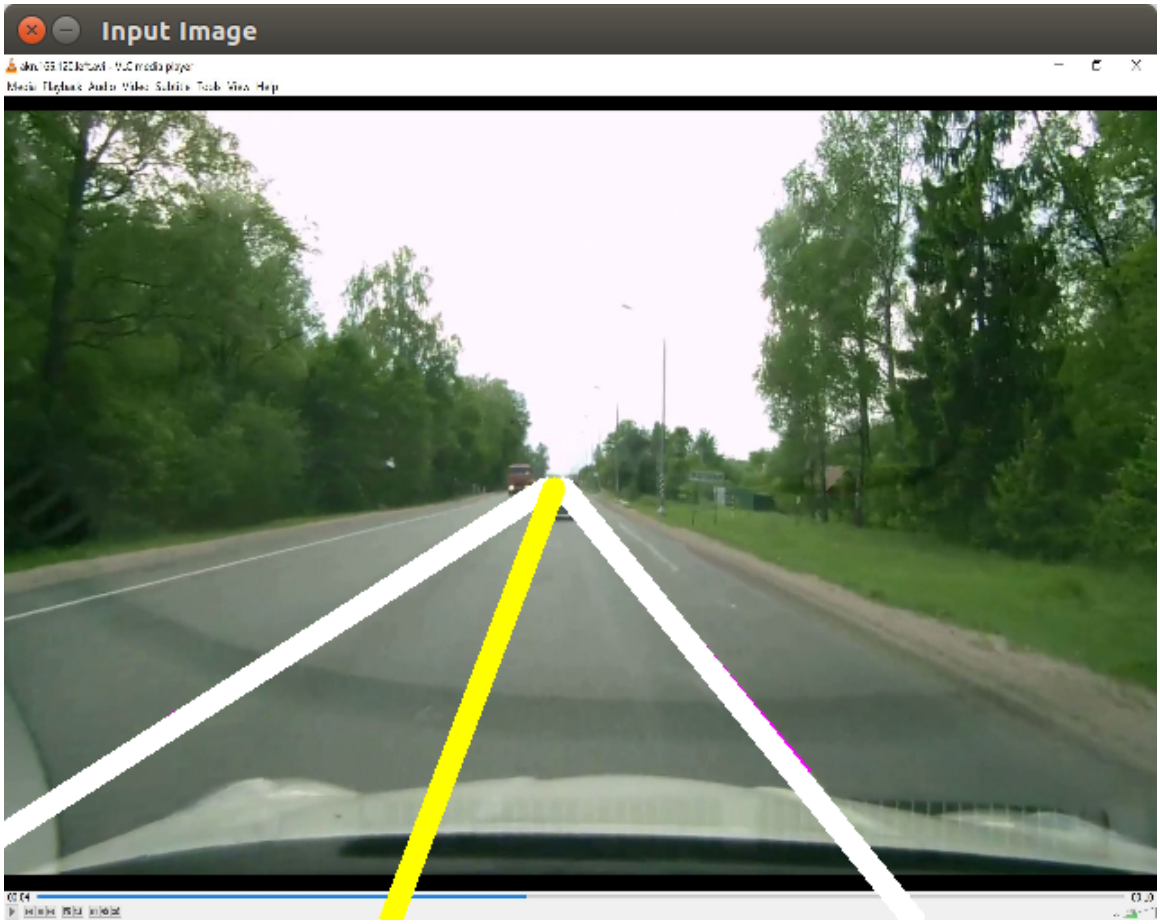

Figure 5.1: Lane Detection in a Real traffic video. The yellow line is estimated center of the lane

computing an error parameter with the center of the lane and this was being saved in a text file. The same experiment was conducted with the robot in the autonomous mode. The robots were made to run in the test bed for five minutes by self flowing back into the test bed when it detects the barrier at the end of the test bed. In this experiment the robots were operating in autonomous mode by detecting lanes, traffic lights and other robots on the road. The error between the center of the lane and the center of the robot are plotted in Figure 5.2. On an average the error was around 5 pixel length, which can be translated to 8 millimeters. The robots had huge errors immediately after it re flows in to the test bed, this is due to the fact that the reflow algorithm is operating on open loop which accumulates the

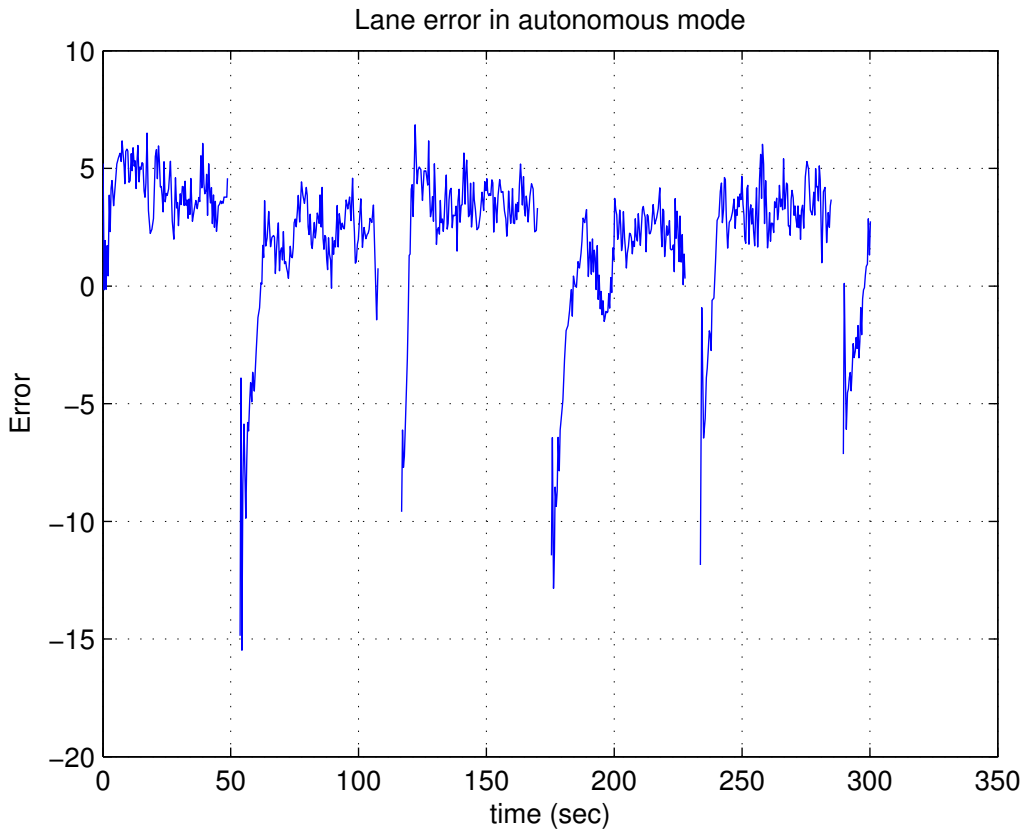

Figure 5.2: Lane error represented in no of pixels while driving in autonomous mode

error. This proves a stable lane tracking process.

When comparing the results of the human driving in the test bed and the autonomous mode driving in the test. The autonomous mode was performing better. This can be attributed to the fact that the control of the motors was done in a closed loop feed back system with a controller. Where as for the humans they had to observe and control the robot using a joystick. The micro movements of the joystick was found difficult to keep control over which were causing minute oscillations on the drive during manual mode. The autonomous mode was needing external assistance few times when it was going off the lane while operating in open loop to re flow back into the test bed. The human driver was able to drive better even in low lit conditions but the autonomous mode was not very efficient in

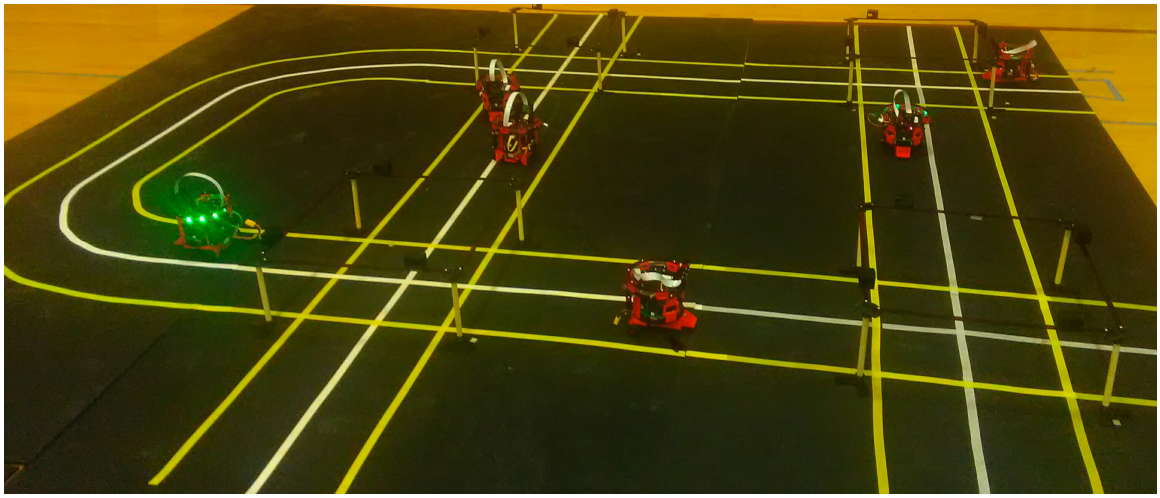

Figure 5.3: The test bed with two intersections and a curved lane

low lit condition mainly when the color of the lane markings were faded.

The battery life was also put to test for manual mode of operation and autonomous mode of operation. For the autonomous mode of operation the robot had to do multiple function, where as for the manual mode it had to receive commands from the control station and send the video back to the command station. This made the battery last longer for the manual mode.

Chapter 6

CONCLUSION

In this thesis, a self driving car test bed was presented in which miniature robots were used to mimic a self driving car. This miniature robot was able to emulate the operations of a self driving car by performing lane detection, traffic light classification, detection of other robots in the test bed and collision avoidance using 6 IR sensors. And these robots are scalable with dimensions and are easy to build so the number of vehicles in the test bed can be increased easily. The algorithms developed for the test bed were tested on videos of real traffic scenarios and were shown to work with minor parametric tuning. In addition to the robot operating in autonomous mode, it can be controlled using a joystick with the vehicles point of view video being relayed wirelessly to the humans control station. The autonomous driving mode was made to run for 5 minutes and it was found that the average lane error was less than 8 mm.

6.1 Future Directions

There are several future avenues for research. They include fusion with contemporary deep-learning architectures for exploiting the complementarity of both paradigms. Replacing the processing computer with a computer with higher processing power would enable the robot to do the image processing algorithms faster and the maximum speed of the robot can be increased. This will also pave way for use of real time machine learning algorithm.

To design of an indoor GPS using over head cameras can aid in better path planning and navigation. Some preliminary work has been done, where the server attached to the camera can read the QR code on top of each robot and would estimate the position and heading of

the robot. This position and heading data can be used to locate the robot in the test bed and the robot can perform much more informed actions.

Use of steering wheel, pedals and a VR headset for manual control can immerse the experience of the user and much more human based experiments can be carried out in the test bed. Inclusion of multiple re configurable traffic signs can be used to represent the traffic conditions in a better manner. Implementation of optimization algorithms on the intersection can be used to schedule the vehicles at an intersections and increase the through put of the intersection.

REFERENCES

Bertozzi, M. and A. Broggi, "Gold: A parallel real-time stereo vision system for generic obstacle and lane detection", IEEE transactions on image processing **7**, 1, 62–81 (1998).

Boudette, N. E., "Autopilot cited in death of chinese tesla driver", http://tiny.cc/7y83sy (2016).

Dosovitskiy, A., G. Ros, F. Codevilla, A. López and V. Koltun, "Carla: An open urban driving simulator", arXiv preprint arXiv:1711.03938 (2017).

Duda, R. O. and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures", Communications of the ACM **15**, 1, 11–15 (1972).

Dwoskin, E., "Mercedes-benz's self-driving big-rig proves that autonomous vehicles are coming sooner than we think", URL http://www.chicagotribune.com/bluesky/technology/ct-uber-self-driving-cars-pittsburgh-20160906-story.html (2016).

Felton, R., "The man who tested the first driverless car in 1925 had a bizarre feud with harry houdini", URL http://tiny.cc/3x83sy (2017).

Fisher, R., S. Perkins, A. Walker and E. Wolfart, "Gaussian smoothing", Hypermedia Image Processing Reference (2003).

Fleming, C., "Tesla car mangled in fatal crash was on autopilot and speeding, ntsb says", URL http://www.latimes.com/business/autos/la-fi-hy-autopilot-photo-20160726-snap-story.html (2016).

Friedland, B., *Control system design: an introduction to state-space methods* (Courier Corporation, 2012).

Hasinoff, S. W., "Saturation (imaging)", in "Computer Vision", pp. 699–701 (Springer, 2014).

Kala, R., *On-road intelligent vehicles : motion planning for intelligent transportation systems* (2016).

Kiryati, N., Y. Eldar and A. M. Bruckstein, "A probabilistic hough transform", Pattern recognition **24**, 4, 303–316 (1991).

Leighty, R. D., "Darpa alv (autonomous land vehicle) summary", Tech. rep., ARMY ENGINEER TOPOGRAPHIC LABS FORT BELVOIR VA (1986).

Logan, B., "Mercedes-benz's self-driving big-rig proves that autonomous vehicles are coming sooner than we think", URL http://www.businessinsider.com/mercedes-self-driving-big-rig-on-a-public-highway-2015-10 (2015).

Lohor, S., "A lesson of tesla crashes? computer vision can't do it all yet", URL https://www.nytimes.com/2016/09/20/science/computer-vision-tesla-driverless-cars.html (2016).

Majumder, A., "The pinhole camera", URL https://www.ics.uci.edu/~majumder/vispercep/cameracalib.pdf (2010).

Olson, E., "Apriltag: A robust and flexible visual fiducial system", in "Robotics and Automation (ICRA), 2011 IEEE International Conference on", pp. 3400–3407 (IEEE, 2011).

Paull, L., J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research", in "Robotics and Automation (ICRA), 2017 IEEE International Conference on", pp. 1497–1504 (IEEE, 2017).

Reese, H., "Tesla's autopilot: The smart person's guide", URL https://www.techrepublic.com/article/teslas-autopilot-the-smart-persons-guide/ (2017).

Reinhard, E., W. Heidrich, P. Debevec, S. Pattanaik, G. Ward and K. Myszkowski, *High dynamic range imaging: acquisition, display, and image-based lighting* (Morgan Kaufmann, 2010).

Schwarz, M. W., W. B. Cowan and J. C. Beatty, "An experimental comparison of rgb, yiq, lab, hsv, and opponent color models", ACM Trans. Graph. **6**, 2, 123–158, URL http://doi.acm.org/10.1145/31336.31338 (1987).

Stager, A., L. Bhan, A. Malikopoulos and L. Zhao, "A scaled smart city for experimental validation of connected and automated vehicles", arXiv preprint arXiv:1710.11408 (2017).

Suzuki, S. *et al.*, "Topological structural analysis of digitized binary images by border following", Computer vision, graphics, and image processing **30**, 1, 32–46 (1985).

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, J.-H. Rick Chang *et al.*, "Going deeper with convolutions", in "The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", (2015).

Thrun, S., M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge", Journal of field Robotics **23**, 9, 661–692 (2006).

Wilson, S., R. Gameros, M. Sheely, M. Lin, K. Dover, R. Gevorkyan, M. Haberland, A. Bertozzi and S. Berman, "Pheeno, a versatile swarm robotic research and education platform", IEEE Robotics and Automation Letters **1**, 2, 884–891 (2016).

Works, H. I., "Brief description", Transformation (1999).

Ziegler, J. G. and N. B. Nichols, "Optimum settings for automatic controllers", trans. ASME **64**, 11 (1942).