END-TO-END DRIVE BY-WIRE PID LATERAL CONTROL OF AN AUTONOMOUS VEHICLE USING CONVOLUTIONAL NEURAL NETWORKS

A Thesis

by

AKASH BASKARAN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Shankar P. Bhattacharyya
Co-Chair of Committee,	Alireza Talebpour
Committee Members,	Aniruddha Datta
	Dileep Kalathil
Head of Department,	Miroslav M. Begovic

December 2019

Major Subject: Electrical Engineering

ABSTRACT

Autonomous Vehicles has been a topic which has been highly researched in recent times. The number of vehicles with Autonomous capabilities, specifically Advanced Driver Assistance Systems (ADAS) has been increasing in the last few years. All passenger vehicles which come out in the market today are equipped with the basic ADAS features. These features help reduce the work done by the driver and at the same time help in reducing the risk of accidents on the roads. Though a lot of focus has been given on these vehicles, not much attention has been given to vehicles which were released a few years ago without these features. Another major issue which has been plaguing the development of Autonomous vehicles is seamless switching between manual and autonomous driving modes at variable speeds. The vehicles which are manufactured at present require the driver to travel at a constant speed on cruise control (typically at speeds above 30 miles per hour), only after which ADAS features such as steering control and lane assist can kick in.

In this work, we propose a method to implement ADAS features in vehicles which lack such features. We also try to simultaneously solve the problem of seamless switching between human and autonomous driving modes, specifically autonomous steering control at variable speeds. In this work, steering control of vehicles using voltage spoofing(can be extended to the throttle and braking modules), development of PID controller for the subsystems, and implementation of End-to-End driving to enable autonomous driving at variable speeds have been discussed. The controller parameters have been fixed by searching the stabilizing set and by implementing transfer learning, the aforementioned problems have been tackled.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Shankar P.Bhattacharyya and my co advisor Dr.Alireza Talebpour for their continuous motivation and guidance for this research. They consistently allowed this thesis to be my own work, but steered me in the right direction whenever they thought I needed it.I would also like to thank the members of my committee Dr.Aniruddha Datta and Dr. Dileep Kalathil for their patience and support in overcoming numerous obstacles I have been facing throughout my research.

Furthermore, I would like to thank Mohammad Reza for his valuable inputs throughout the research. I would like to thank my parents for believing in me and motivating me through out. Finally I would like to thank Sushmitha, Malliga, Niranjan, Adithyaa, Amrita, Anand, Karthik and Ramanathan for their continuous encouragement and support through out the course of my master's without which I wouldn't have been able to finish this work.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Robert M. Kennedy ’26 Professor II Shankar P. Bhattacharyya[Advisor], J.W. Runyon, Jr. ’35 Professor II Aniruddha Datta and Assistant Professor Dileep Kalathil of the Department of Electrical & Computer Engineering and Professor Alireza Talebpour[co-advisor] of the Zachary Department of Civil Engineering. I would like to thank Dr.Alireza Talebpour for providing the research vehicle to collect data and test my work.

All work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a Departmental Scholarship from The Electrical and Computer Engineering department of Texas A&M University. This study was supported by funding provided by Dr.Alireza Talebpour.

NOMENCLATURE

DARPA	Defense Advanced Research Projects Agency
ADAS	Advanced Driver Assistance Systems
CAN	Control Area Network
ROS	Robot Operating System
ECU	Engine Control Unit
PWM	Pulse Width Modulation
CNN	Convolutional Neural Network
OBD	On Board Diagnostic
PID	Proportional Integral Derivative
FPS	Frames Per Second
MPH	Miles Per Hour

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Motivation

One of the main objectives of Science and Technology has always been to improve the comfort and safety of human lifestyle, keeping in mind the requirements of a sustainable future . A study published by World Health Organization [1] in 2018 shows that there are about 1.35 million deaths per year due to accidents on the roads. Moreover it has also been estimated by the study that every year between 20 and 50 million people suffer non-fatal injuries out of which many have resulted in a form of disability. These translate to about 3700 deaths every day and about 100 injuries every minute due to road related accidents. These road related accidents have been on the rise in the past decade. This was one of the primary reasons for the development of Autonomous Vehicles, as bringing in automated vehicles would reduce the scope for error and hence reduce the margin of accidents. With the reinvention of neural networks, more and more research has been done in the field of Autonomous Vehicles. Though a lot of companies have been working on fully driverless cars, at present these models are either in a development phase or primary testing phase and are not at a stage of mass production. But that is not the case with Advanced Driver Assistance Systems (ADAS). Almost every vehicle which is being brought to the roads are equipped with ADAS features. Ranging from basic cruise control, to adaptive cruise control, from lane keeping to parking assist these features are numerous. These ADAS features help reduced the workload of the driver and hence help improve the concentration of the driver on the roads and thus reduce the risk of accidents. But in today's roadways, there are millions of vehicles which were launched a few years earlier that do not have such basic driver assistance systems.

Removing these vehicles from the roads increases the safety of the roads but then these vehicles occupy a major portion of today's road population. Replacing these vehicles involve a lot of money and natural resources and hence is not a sustainable solution. One way is to retro fit these vehicles in a way that brings in advanced driver assistance systems. In this work we try to develop a system

which can be used to retrofit vehicles without ADAS features and equip them with such features.

In the current scenario, to enable ADAS features such as steering assist and lane keeping it is required that the driver operate the vehicle on cruise control/adaptive cruise control(based on the vehicle make and model). So the driver first engages cruise control (which typically can be engaged at speeds over 30 MPH) and once that is done,steering control systems can be engaged. Since activation of cruise control is a necessity for activation of steering assist features, steering assist and other autonomy features aren't typically available at lower speeds and primarily are used only on highways. In our work, we try to use neural networks to tackle this issue as well such that, lateral autonomy (steering control) can be achieved at varying speeds [2].

1.2 Existing Methods

Over the years it has been customary to split autonomous vehicle control into clusters of subtasks and threads. These subtasks can vary from computationally facile tasks such as Lane Detection [3] [4] [5] [6], simple obstacle detection algorithms [7] to more complex tasks such as Simultaneous Localization and Mapping [8]. Apart from SLAM and obstacle avoidance, path planning [9] [10] and control logic [11] [12] are some other subtasks which the system has to work on. With the passage of time, some algorithms such as You Only Look Once (YOLO) [13] revolutionised some of the earlier tasks with the usage of covolutional Neural Networks [14].

But this cluster of tasks rely mainly on detection, extraction and segmentation [15] of features from the sensor data, typically cameras, radar's and LiDAR's. The features which are typically detected and used for navigation are human understandable/human specified features such as lane markings, curb markings and so on. The major drawback of this approach is that, since the features are defined manually, they may not be the most optimal solution from a machine's perspective. Sometimes this may lead to accumulation of errors over time and this accumulation can have detrimental effects.

Due to the increased impact of Neural networks, the area of mimicking human driver behaviour started coming to the forefront. Bojarski et al [16] in their work done at NVIDIA propose an end-to-end model which tries to mimic the driver behaviour by detecting features which the model

considers to be of more priority. Since the features are machine defined, the scope of human error in feature detection and definition start to go down. Also since the features are machine defined, they tend to be more optimized.

One main reason for the success of Neural networks was due to increase in the training data available with passage of time. The works of Bojarski et al work by analysing human driving behaviour in various situations on data collected over six months. Due to the presence of a vast amount of data, the accuracy and the efficiency of their system is really high. But due to the absence of access to such vast data, it is not possible to implement end-to-end driving for a system in a completely different environment. Due to the lack of training samples, there is a big chance that, when trained with a small amount of data overfitting is possible.

To prevent this, Transfer Learning [17] is a feasible solution. Using transfer learning, the initial weights for the convolutional layers can be transferred from popular works such as VGG16 [18], Inception-net [19]. The neural network is now retrained with a different image set and different outputs. Due to weight initialization, it has been observed that faster convergence with lower amount of data has been achieved.

From a control systems perspective, the transfer function of the vehicle is obtained by analysing the system response to a unit step input using logarithmic decrements. It has been observed and noted that PID controllers have been found to be a simple but an effective approach for control of steering throttle and the braking modules. We try to find the stabilizing set to find the controller parameters K_p , K_i, and K_d [20]. A more detailed explanation on the current state of the art on various topics have been provided in later sections of this work.

1.3 Layout of the work

The rest of this work is divided into 6 different chapters with each of them containing various sections and subsections to give a complete idea of the work being done. The next chapter discusses the problem statement and redefines it to give us a concise and clear picture. Furthermore it divides the problem statement into smaller tasks which need to be completed to get the whole puzzle solved.

Chapter 3 introduces us to the concept of drive by wire and its implementation. It describes in detail a method of communication which is used in the automotive industry (Control Area Network-CAN). It also explains about CAN sniffers which allow third party users to access the CAN Bus of the vehicle and also the CAN sniffer which is used in this work. It gives insight into components of an automobile such as torque sensor, throttle measurement sensor and the Electronic Control Unit which are subsequently used to control the vehicle using a joystick. This section also throws light on the vehicle which has been used for trial and testing of this whole system. It also gives an insight into the physical wiring of the test system to give the reader an idea about how to replicate these in any given vehicle. Subsequently ROS, a middleware used for time synchronous communication between various hardware and programs is explained. Finally the chapter is concluded by explaining the conventions of steering angle used in this work, converting CAN messages from binary values to meaningful steering values and on publishing the converted steering angles to corresponding topics on a ROS node.

The next chapter of this work deals with the software configuration and packages which are needed to get the modules working. This also gives an idea about drivers, various packages and languages which are needed. It has to be noted that every chapter which deals with the usage of specific software, also gives an overview of the packages involved.

The fifth chapter of this work starts with an overview of linear control theory. The reader is familiarized with popular jargon in the field of control systems before diving into the specifics related to the problem statement in the second section of this chapter. In the succeeding section, calculation of the transfer function of the steering system based on the method of logarithmic decrements has been explained in detail. After this various control strategies have been represented.This work then moves onto the concept of PID controllers and its parameters. The penultimate section of this chapter deals with tuning of parameters for PID controllers and finding the stabilizing gains for the system.

The sixth chapter of this work gives an overview of the data collection and data pre-processing techniques which were implemented in this work. In the succeeding section, the system configu-

ration and setup used for data collection has been outlined. With the setup defined, the following section explains about data storage and retrieval using ROS. It gives an overview about a specific form of storage called as ROSbag which helps the user to replay recorded data in real time of various nodes and ros topics which were published. The fourth section of this work goes onto explain about the camera being used in this work. The succeeding section of this work explains about frame rates and how various sensors have varied frame rates and solution to tackle such an issue. The sixth section of this chapter details about ROS playback and converting ROS messages into images and csv files which can be used for training a neural network model which will be discussed in future chapters. The final section deals with pre-processing which is done on the saved images to give way to easier computations when a neural network model is trained.

The seventh chapter, starts by describing Neural Networks and their invention. It goes onto explain about the resurgence of Neural Networks and various applications it has been used for over the years. The first section of the work goes on in detail about Convolutional Neural Networks and how they are being used widely for various purposes. This section also throws light onto various kinds of terms which are associated with Deep learning. The succeeding section introduces the reader to the concept of end-to-end driving approach and it describes the works of Bojarski et al. The subsequent section explains the drawbacks of this work and introduces the concept of Transfer Learning and explains about some famous works which are used in transfer learning to get the reader familiarized with this concept. The fourth section of this work explains about the network architecture involved in the hybrid approach we have used. The fifth section explains about the training strategy, different optimisers and normalisers used to achieve faster convergence and reduction of error.

The penultimate chapter of this thesis, goes on to explain about the various experiments conducted and the results obtained. It gives a detailed description about the accuracy of the neural network. Further the results of the work by Bojarski et al is compared with our results and the shortcomings and advantages are discussed.

The final chapter of this work offers some concluding remarks on the system and how a solution

to the problem statement discussed earlier has been achieved. This section also describes about various ideas and improvements which can be made to this work.

2. PROBLEM STATEMENT

As mentioned earlier, the end goal of this work can be classified into two parts. Firstly to develop a system using which we can retrofit cars which lack Advanced Driver Assistance features to have basic driver assistance systems. Secondly to develop a system in which seamless transition between autonomous control mode and human control mode is possible. More specifically to enable autonomous steering control even when the vehicle is not on cruise control(when the vehicle is not at a constant speed).

The tasks have been subdivided into multiple subtasks to give a perspective on how to proceed with them. The problem statement can be briefly classified into three parts. First, to come up with a system to make the vehicle controllable from a computer. Second to develop a control system for the steering and throttle modules and finally to develop a perception system which can map the required steering angles at various times.

2.1 Drive by Wire

To achieve a drive by wire system, it is required to communicate with the Electronic Control Unit of the vehicle. Almost all vehicles which were introduced in the last 2 decades use a popular method of communication protocol called the Control Area Network (more detail about CAN has been provided in the chapter about Drive by wire vehicles). Though recent vehicles allow the control of the steering and throttle modules using CAN messages, that is not the case with older vehicles. Using voltage spoofing on the steering and throttle modules, Drive by wire can be established. The final step would be to make the vehicle controllable using the analog sticks on a joystick.

2.2 Control Logic

Using the CAN Bus, the current steering angle and the speed of the vehicle can be obtained. Since Drive by wire is done, another important task would be to develop a controller which takes in the required steering angle/vehicle speed as input and the current steering angle/vehicle speed

as the feedback to the system.

2.3 Perception

Another important task is to predict the steering angles required by the vehicle to stay on the road/follow the path determined. Since we are planning to opt for an end-to-end approach there wont be a need for a path planning module. The neural network model which will be trained can handle it for us. For the neural network training, data is an important aspect. Hence data collection by driving the vehicle and recording driver behaviour is another important aspect

These tasks when put together should help us achieve the overall objective. Each of these tasks have been explained in succeeding chapters.

3. DRIVE BY WIRE

Over the course of time, Drive by wire refers to the usage of electrical/electro-mechanical signals to carry out functions in an automobile which were earlier carried out through mechanical linkages. In the last 2 decades there has been a shift in the automotive industry to move from fully mechanical systems to drive by wire systems. Reduction of wear and tear of mechanical components, quick system response due to electrical signals rather than mechanical actuations, and ease of use for the driver (in cases such as power steering) are few of the main reasons for an industry switch towards drive by wire systems. In recent times, driver by wire has been used in a slightly different sense. In this work we follow the latest convention and we define a drive by wire system as any system using which an automobile can be controlled using a computer by any user on a remote or central system.

3.1 Control Area Network

A Control Area Network(CAN) bus is a robust vehicle bus designed to allow micro controllers and devices to communicate with each other in applications with a host computer. This protocol was initially introduced at the Society of Automotive Engineers conference in Michigan in 1986 by BOSCH [21]. Over the years, CAN bus protocol [22] has been adopted by all automobile manufacturers as a standard method of communication due to the ease of use, reduction of bulky wiring systems and improved system responses. International Organization for Standardization in 1993 adopted CAN Bus protocol as standard communication method for automobiles (ISO 11898) [23].

In simpler terms, the usage of CAN bus in a automobile can be compared to that of the central nervous system in a human body. Every sensor and module of the automobile sends messages onto the central bus and the desired receptors of the message read from the central bus. The details of the implementation of CAN Bus is out of the scope of this work and it is sufficient to know about the existence of such a mode of communication.

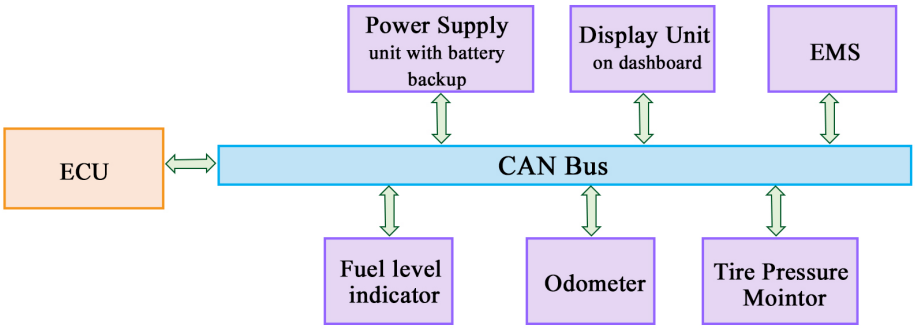

Figure 3.1: CAN Bus functioning.

3.1.1 CAN Sniffers

Most Automobile manufacturers tend to keep the software for accessing the CAN messages proprietary. This is done to ensure safety of the vehicle and reduce tampering of the system. But they do provide hardware access to the CAN bus through a port commonly known as OBD II. The On-Board Diagonostic (OBD) port is used by vehicle manufacturers and service center's to go through the CAN bus of the vehicle to detect fault modes in various components of the vehicle. Using this port, it is possible for a regular user to gain access to the CAN messages of a vehicles with access to specialised hardware.

This specialised piece of hardware which allows the user to read through the CAN bus of a vehicle is called as a CAN sniffer. With a proper software setup, all the messages which are being sent through the CAN be accessed by our CAN sniffer. One important piece of information is that, these CAN sniffers also have the ability (depends on the individual make of instrument) to write messages to the CAN bus, instructing different modules to perform different functions. The ability to accept CAN message writing depends on the vehicle and most vehicles released earlier do not allow it. Hence in most vehicles which we are considering for this work, it is feasible to read from the CAN bus but not write to it.

KVASER CAN Leaf HS V2 is one of the many CAN sniffers available antd it used in our work to read from the CAN Bus and publish all messages as ROS topics (more details about ROS topics

Figure 3.2: CAN sniffer - KVASER Leaf Light HS V2. Image reference: www.kvaser.com

are specified in section 3.4). By filtering the CAN messages which correspond to the steering, and converting binary data to decimal, we are able to read the current steering angle. The KVASER Leaf Light V2 is connected to the car using a DB9-OBD II connector and is connected to the computer using a USB cable.

3.2 Vehicle Sensors

Automobile manufactures equip vehicles with a wide variety of sensors to ease the use of a vehicle and also to increase the safety of the vehicle. Starting from simple sensors which check for seat belts, tire pressure sensor to complex sensors such as Radar which are used for adaptive cruise control applications these sensors have a major part to play. All these sensors use the CAN bus as a central mode of communication with the Electronic control Unit of the vehicle.In this section we will be taking a look at four major sensors/modules in any automobile, namely:

- Torque Sensor
- Electronic Power Steering
- Throttle Position Sensor
- Electronic Control Unit

3.2.1 Torque Sensor and EPS

In older vehicles it was common to have mechanical linkages from the steering to the wheels. But with time, these mechanical linkages were replaced by hydraulic or Electrical systems which made it easier for turning the steering wheel. This feature was called as Power steering. With the introduction of CAN messages and other electronic components into vehicles, hydraulic power steering was replaced by completely electronic systems called as the Electronic Power Steering(EPS).

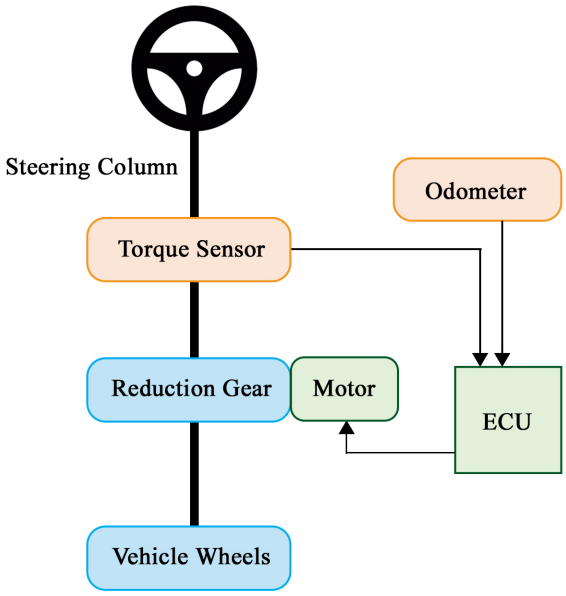

Figure 3.3: Torque Sensor & EPS setup

Figure 3.3 describes the system setup in all present day vehicles. The steering of the vehicle is connected to a torque sensor which measures the torque provided by the driver on the steering. This measured torque is converted to an electric signal and is sent to the Electronic Control Unit of the vehicle. The ECU in-turn sends in another electric signal to power the electric motor which is connected to the steering column of the vehicle. Hence a small torque on the steering wheel by the driver results in the electric motor assisting the driver with the turn.

3.2.2 Throttle Position Sensor

A throttle position sensor, also referred to as TPS, is a sensor which is used to monitor the air intake of an engine. The sensor is typically located on the spindle to directly monitor the position of the throttle. But in modern systems TPS has been replaced by non contact solutions such as magnetoresistive sensors which measure the position of the throttle and send in an electronic signal to the ECU of the vehicle. Based on the input signal from the TPS, the ECU controls the opening and closing of valves in the Engine to power the vehicle accordingly using an Engine control Unit.

3.2.3 Electronic Control Unit

If the CAN bus is considered equivalent to the central nervous system of a human body, the electronic control unit is the brain of an automobile. It is responsible for transmitting and receiving control signals to and from various modules of a vehicle. The two main functions of the ECU which are important to us is that, it is responsible for receiving the torque sensor measurements and providing the EPS motor with a suitable control signal proportional to it. It is also responsible for receiving the throttle position and sending in corresponding signals to the Engine control unit to work accordingly. The functioning of ECU is out of the scope of this work, and hence we just consider it as a black box which responds to signals from the torque and throttle position sensors.

3.3 Test Vehicle

The whole system has been developed on a 2016 version of Kia Soul. This car uses CAN bus based mode of communication. As specified earlier, since this is a slightly older vehicle, the user doesn't have the permission to write messages on the CAN bus using a CAN sniffer. Moreover analog signals are used to send in voltages from the torque sensor and throttle measurement sensor to the electronic control unit of the vehicle. The specifics of the signals from the torque sensor and the throttle position sensor to the ECU are discussed in section 3.5.

Figure 3.4: Test Vehicle - Kia Soul

3.4 Robot Operating System

The Robot Operating System (ROS) is a robotics middleware with flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Although it is not an operating system, it provides hardware abstraction, low level device control, message passing between process and package management. In our work, ROS is mainly used for synchronous communication between various hardware and software modules. In ROS data can be published and subscribed through hubs called nodes. These nodes can either publish to a topic (a cluster of messages under a single name) or subscribe from a topic. All sensors publish data and all processes which need to work with sensor data subscribe to these published topics.

These subscribers and publishers can be modified to publish/subscribe to data at different rates based on definition. Moreover ROS allows for interaction between multiple program languages. A module running python can communicate with a module running C++ and transfer information. This is specifically useful when deep learning (mostly in python) and sensors(mostly drivers in

C++) are being used together. All the features make ROS an important hub for machine based communication in all autonomous vehicle based applications.

3.5 Drive by wire implementation

As specified earlier, Drive by-wire in this work refers to the ability to control a vehicle from a computer. The first objective of this work is to control the steering of the test vehicle using the joy stick. Figure 3.5 gives the connection circuit from the Torque sensor to the ECU of the vehicle.

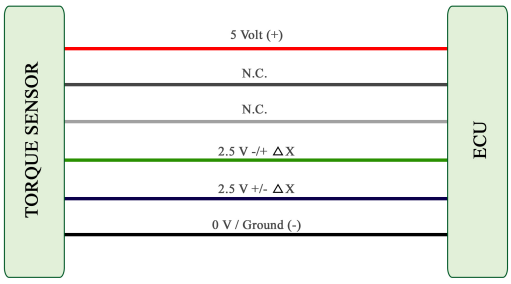

Figure 3.5: ECU - Torque sensor wiring diagram

The red and black wires in figure 3.5 are the +ve and -ve wires having a potential difference of +5 v in between them. The blue and green wires, have a potential difference of 2.5 V with respect to the ground, while the steering is at an angle of 0°. When a torque is applied in the clockwise direction, it leads to a change ΔX in voltage and the green and red wires carry a potential of 2.5+ΔX and 2.5-ΔX respectively. The end point being the sum of the potentials in both the wires is always 5 V. The other two wires are not needed for our application and hence are classified as Don't Cares(N.C.)

In a similar way the Figure 3.6 gives the connection circuit from the throttle measurement sensor to the ECU of the vehicle.

The red, black and the 2 don't care wires have the same functions as in the previous figure. The green wire, has a voltage of 2.5 V when the throttle is not engaged. Once the throttle is engaged,

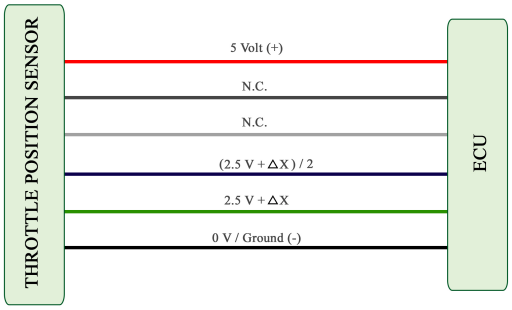

Figure 3.6: ECU - Throttle position sensor wiring diagram

this value can go as high as 4.5V based on the extent of engagement. The blue wire always carries half the potential carried by the green wire in the case of throttle position sensor on this vehicle. It has to be noted that these sensor values can vary from vehicle to vehicle and the values specified are particular to a Kia Soul 2016.

3.5.1 Hardware setup

Now by spoofing voltages into the wires which go to the ECU it is possible that we make the ECU believe the steering and throttle are rotated and engaged respectively. Based on experimentation it has been found that, the inputs given to the ECU need to be analog signals and not digital. So using a pulse width modulation from a microcontroller is one way to solve this issue. But providing the voltages using the digital pins of a microcontroller can either break the torque sensor or can lead to erratic steering behaviour due to its low resolution. Arduino Due, is a microcontroller which has been equipped with 2 onboard DAC pins with a resolution of 12 bits. Due to the increased resolution, the PWM generated by the pins are very closely aligned to an analog signal.

The pins 12 and 13 of an Arduino Due are the DAC pins/PWM pins with higher resolution. Hence we use each of them for connecting it to the wires. Figure 3.7 depicts the circuit diagram to spoof voltages to control the steering of a vehicle. For the throttle module, the wiring diagram is similar to the green wire connected to pin 12 and blue to pin 13 and is depicted by figure 3.8.

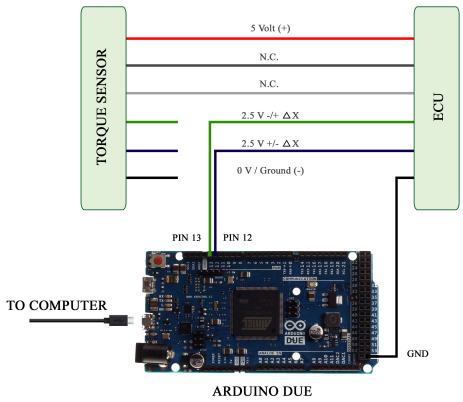

Figure 3.7: Steering: Voltage Spoofing wiring diagram

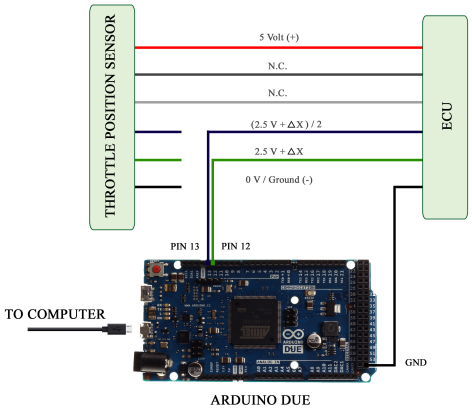

Figure 3.8: Throttle: Voltage Spoofing wiring diagram

3.5.2 Joystick Control

A Logitech F310 gamepad is used as the joystick for controlling the vehicle. The position of the analog sticks of the joy stick are published as messages on ROS. The published messages can then be subscribed to in ROS, they are mapped to corresponding analog values and is sent to the Arduino to be outputed as a voltage signal [1].

The code is written in such a way that, the left analog stick of the joy stick corresponds to the steering and RT button of the joystick corresponds to the throttle position required.

[1]More information and code for implementation can be found online from my github repository `https://github.com/Akashbaskaran/JoystickCarControl`

3.6 Conventions

A clockwise rotation of the steering wheel is considered a +ve angle and an anticlock wise rotation is considered as a -ve angle. The steering angles can vary from -500 degrees (approximately 2 anticlock wise rotations of the steering wheel) to +520 degrees (approximately 2 complete rotations of the steering wheel). Moreover the steering wheel to tire ratio is 15.7:1 on the Kia soul. That is, a rotation of 15.7° on the steering wheel corresponds to a rotation of 1° on the wheel of the vehicle.

3.6.1 ROS Nodes

Using the Kvaser leaf light, the CAN messages can be read from the vehicle. The launch function of the CAN bus reader publishes all the CAN messages read by the sniffer onto various topics depending on the frame ID. The frame number corresponding to the steering angle is 688. Each of the Kvaser leaf lights have a hardware id and they can be specifed in the launch file of the code. Messages read from the frame number 688 are published onto a topic. The stream corresponding to steering angle consists of 2 bits of data each of which are published onto two separate topics, namely /can_st and /can_st1[2]. Let bit2 be the most significant bit and bit1 be the least significant bit.

3.6.2 Steering angle conversion

Another subscriber subscribes to these two topics and converts them into meaningful steering angles. If the values of bit2 is less than 21, the steering angle is calculated by:

$$\frac{-(bit2 * 255 + bit1)}{10} \tag{3.1}$$

If bit2 is greater than 21, the steering angle can be calculated by:

$$\frac{(255 - bit2) * 255 + (255 - bit1)}{10} \tag{3.2}$$

[2]More specifications about reading steering bits from KVASER can be found on my GitHub repository `https://github.com/Akashbaskaran/kvaser_interface`

These calculated steering angles are then published onto the topic /can_str under the attribute P[3]. Now any part of our code which needs access to current steering angle of the vehicle can subscribe to this node to get the real time steering position data. This will be useful in chapters 5 and 6.

[3]More specifications about the published steering angles can be found on my GitHub repository `https://github.com/Akashbaskaran/learning_joy/blob/master/src/new.cpp`

4. SOFTWARE SETUP

Since the system requires multiple hardware and software collaborations, it is important to adhere to the same versions to replicate this work.

4.1 Work station

The NVIDIA Drive PX2 has been used as the On-board computer but it doesnt have the computational capability to train neural networks. It can be used only for the purpose of deployment. The training has been carried out on a desktop computer running Ubuntu 16.04 LTS. The desktop has about 132Gb of RAM and 1 TB of SSD storage space. Moreover the system is equipped with 3 NVIDIA GPU's each with a computational capability of 10Gb. The presence of multiple GPU's and high RAM availability allows faster convergence of Neural networks due to parallel processing.

4.2 ROS

The system used for data collection runs ROS kinetic Kame. ROS kinetic is a long term support version of ROS released primarily for Ubuntu 16.04. Using ROS Melodic Morenia or ROS Indigo Igloo should not cause much of an issue as most of the packages are nearly the same and extend to both. It is advisable to not use version of ROS before Indigo due to incompatibility and a complete change in the workspaces.

4.3 Drivers

Drivers are softwares written for errorless communication through packets between the hardware and the computer. Two main hardwares being used are the KVASER Leaf light V2 and the Pointgrey blackfly camera. This section specifies the verison of driver used for both of them.

4.3.1 KVASER Leaf light

As specified earlier, KVASER leaf light V2 has been used as a CAN sniffer. The driver for ubuntu systems can be downloaded form the KVASER website [1]. Once the latest drivers are installed and KVASER ROS bridge (discussed previously) setup, CAN messages can be parsed through in a terminal.

4.3.2 Camera

A FLIR (previously pointgrey) blackfly camera is used for the perception system. The drivers required for the camera can be downloaded from the FLIR website[2].

4.4 Python

Python is an interpreted, high-level, general-purpose programming language. Python is primarily used for handling the deep learning side of things. ROS is used as the middleware. Python 3.5.2 has been used as the primary python distribution on the desktop. The onboard computer uses both python 3.5.2 (for running neural network models) and python 2.7 for running ROS applications. Anaconda has been used as a package management tool on the desktop computer.

4.4.1 CUDA

CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. Both the desktop and onboard computers use 410.73 version of the NVIDIA graphics driver. The corresponding CUDA version is CUDA 10.0 and hence that is used. Cudnn 7.6.2 has been used and it can be downloaded from the NVIDIA developer website[3]. It is important to know that various versions of CUDA and Cudnn can be used but it is important to make sure all of them are compatible with each other, failing which computations will be using the CPU rather than the GPU.

[1] https://www.kvaser.com/linux-drivers-and-sdk/
[2] https://www.flir.com/support/products/blackfly-usb3#Overview
[3] https://developer.nvidia.com/cudnn

4.4.2 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano [4]. It was developed with a focus on enabling fast experimentation. In this work, we use keras with a tensorflow backend. We use keras 2.2.2 running over a tensorflow base of 1.10.0. It is recommended to use the same configuration for optimum results. Moreover the tensorflow being used is the gpu verion of tensorflow.

4.4.3 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products[5]. The desktop uses Open CV version 3.4.2 for its application. The onboard computer is equipped with both OpenCV2 and OpenCV3. OpenCv2 is used in conjuncture with python 2.7(for ROS applications) and OpenCv3 is used with python 3.5.2.

4.4.4 Other Packages

4.4.4.1 numpy

Numpy is a package used for facilitating scientific computing in python. Numpy 1.15.2 is used through out this work, but the version of numpy shouldn't affect the functionality.

4.4.4.2 scikit learn

Scikit learn is a simple tool in python mainly used for data mining and data analysis purposes. The version of scikit learn doesnt inhibit the functionalities. We use scikitlearn 0.20.0 in this work.

4.4.4.3 pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Pandas 0.23.4 is used in

[4] `https://keras.io/`
[5] `https://opencv.org/about/`

this work.

4.5 Cmake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice. The suite of CMake tools were created by Kitware in response to the need for a powerful, cross-platform build environment for open-source projects such as ITK and VTK [6]. Through out this work, we mainly stick to using catkin_make.

[6] https://cmake.org/

5. CONTROL SYSTEM

Given that joy stick control of the steering and throttle has been established, the next step is to try to control the steering/throttle to defined steering angle/speed. In this work, we will be focusing on the working of steering module and the same can be extended to the throttle module as well.

5.1 Basics of Control systems

In control systems, any system is represented by its transfer function. A transfer function is a mathematical function which theoretically models the device's output for each possible input. In modern control systems, any system with a controller can be described as show in figure 5.1.

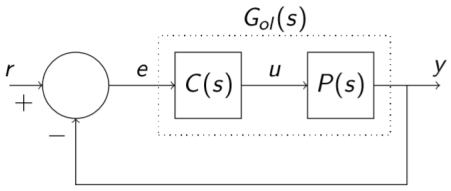

Figure 5.1: A typical feedback control system

Let P(s) be the transfer function of the plant and let C(s) be the transfer function of the controller. Moreover, the plant transfer function can be written as

$$\frac{n_p(s)}{d_p(s)}$$

and the controller transfer function can be written as

$$\frac{n_c(s)}{d_c(s)}.$$

Let us consider a unity feedback.The closed loop transfer function of the system is denoted by:

$$\frac{n_p(s)n_c(s)}{d_p(s)d_c(s) + n_p(s)n_c(s)} \tag{5.1}$$

Therefore the closed loop characteristic equation can be written as

$$d_{cl}(s) = d_p(s)d_c(s) + n_p(s)n_c(s) \tag{5.2}$$

The roots of $d_{cl}(s) = 0$ are the closed loop characteristic roots and a controller C (s) stabilizes a plant P (s) if and only if the roots lie in the Left Half plane, or $d_{cl}(s)$ is Hurwitz [20].

5.2 Logarithmic Decrements

Logarithmic decrements is a method which is used to find the damping ratio of an under-damped system. In this method, the system is given a unit step input and the response of the system is measured. Based on the plot of the response of the system, the logarithmic decrement δ can be calculated as:

$$\delta = \frac{1}{n} ln \frac{x(t)}{x(t + nT)} \tag{5.3}$$

where n is the overshoot of the peak which is n nodes away from the 1st over shoot. x(t) and x(t+nT) are the corresponding over shoots at time t and t+nT from the steady state value. The logarithmic decrements can be used to find the damping ratio which can then be used to calculate the transfer function of the system.

A unit step input corresponding to $\pi/2$ in steering value was provided as an input to the system and the steering response was recorded. Figure 5.2 is the response of the steering to a unit step input. Based on equation 5.3, the logarithmic decrement can be calculated as:

$$\delta = ln \frac{(224 - 165)}{5} = 2.468 \tag{5.4}$$

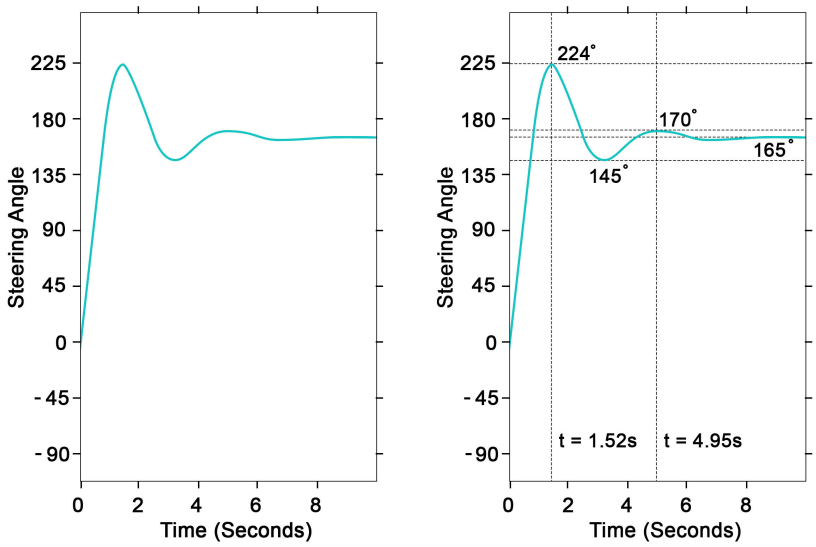

Figure 5.2: Steering response to a unit step input

5.3 Calculation of Transfer Function

The damping ratio ζ of the system can be found by:

$$\zeta = \frac{1}{\sqrt{1 + (\frac{2\pi}{\delta})^2}} \tag{5.5}$$

where δ is the logarithmic decrement of the system calculated from equation 5.3. Substituting the value of δ from equation 5.4, we get:

$$\zeta = \frac{1}{\sqrt{1 + (\frac{2\pi}{2.468})^2}} = 0.33647 \tag{5.6}$$

The damping ration can then be used to find the natural frequency ω_n of the system from its damped frequency ω_d.

$$\omega_d = \frac{2\pi}{T} \tag{5.7}$$

$$\omega_n = \frac{\omega_d}{\sqrt{1 - \zeta^2}} \tag{5.8}$$

where T is the period of the waveform.For the given system, from the response plot, the damped frequency is found to be around, 1.83 radian per second and hence the natural frequency can be found to be around 1.965 radian per second.

Making an assumption that the system is a second order linear time invariant system, the transfer function, P(s) of the system can be calculated as:

$$P(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \tag{5.9}$$

where ω_n is the natural frequency and ζ is the damping ratio of the system.

Therefore the transfer function of the system under consideration can be simplified as:

$$P(s) = \frac{3.862}{s^2 + 1.4334 * s + 3.8626} \tag{5.10}$$

A unit step input was provided to the transfer function in MATLAB and the following response was obtained. It can be noted that this is very similar to the response from the vehicle and hence this proves the model assumed is valid.

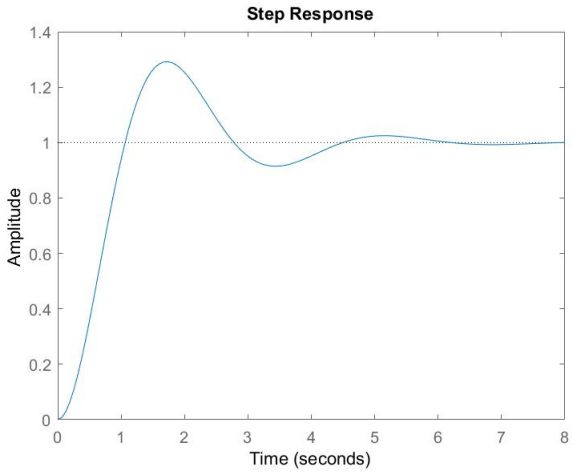

Figure 5.3: A portion of the training tracks

5.4 Proportional Integral Derivative Controller

To stabilise a given transfer function, lot of control strategies and methods can be adopted. The controller should have low settling time and as well as low overshoot. Many control strategies can be used to tackle this problem. They include, Proportional Integral Derviative (PID) controller [20], fuzzy controllers, h-infinity controllers and lag-lead compensators to name a few. Due to the simplicity of the design and the ease of development of PID controllers, they have been used in this work.

For a PID controller, the transfer function can be written as:

$$C(s) = \frac{K_d s^2 + K_p s + K_i}{s} \tag{5.11}$$

The characteristic equation can be calculated by:

$$N_p(s)N_c(s) + D_p(s)D_c(s) \tag{5.12}$$

The characteristic equation for this system can be written as:

$$= (s^2 + 1.4334s + 3.8626)(s) + (3.862)(K_d s^2 + K_p s + K_i) \tag{5.13}$$

$$= s^3 + (1.4334 + 3.8626K_d)s^2 + (3.8626(1 + K_p)s) + 3.8626K_i \tag{5.14}$$

5.4.1 Stability

s^3	1	3.8626(1+K_p)
s^2	(1.4334 + 3.8626K_d)	3.8626K_i
s^1	b1	0
s^0	3.8626K_i	0

Table 5.1: Routh Table for the given characteristic equation.

$$b1 = \frac{(1.4334 + 3.8626K_d)(3.8626)(1 + K_p)}{1.4334 + 3.8626K_d} \tag{5.15}$$

Using the Routh criterion, the stability of the system can be investigated to give us a primary idea about Kp, Ki and Kd. From the Routh table, using the Routh Hurwitz criterion for it to be stable, there should be no sign change in the first column and hence:

$$K_i > 0 \tag{5.16}$$

$$1.4334 + 3.8626K_d > 0 \tag{5.17}$$

and

$$b1 > 0 \implies \frac{(1.4334 + 3.8626K_d)(3.8626)(1 + K_p)}{1.4334 + 3.8626K_d} > 0 \tag{5.18}$$

Equation 5.17 can be simplified as:

$$Kd > \frac{-1.4334}{3.8626} = -0.3710 \tag{5.19}$$

Equation 5.18 can be simplifies as:

$$(1.4334 + 3.8626k_d)(3.8626 + 3.8626K_p) > 0 \tag{5.20}$$

$$5.536 + 5.536K_p + 14.919K_d + 14.919K_pK_d > 0 \tag{5.21}$$

$$K_p + 2.69K_d + 2.69K_PK_d > -1 \tag{5.22}$$

5.5 PID controller parameters

For the given plant transfer function,

$$N_p = 3.862 \qquad D_p = s^2 + 1.4334s + 3.862$$

Moreover the values of N_{pe}, N_{po}, D_{pe}, D_{po} can be written as:

$$N_{pe} = 3.862 \qquad N_{po} = 0$$

$$D_{pe} = s^2 + 3.862 \qquad D_{po} = 1.4334$$

Boundary induced by a drop in degree

This is the set of all PID control gains that will lead to a drop in degree. From equation 5.14, we can observe that there are no coefficients for the term with the highest power. Hence, there is no boundary corresponding to this case.

Boundary due to root at zero

This is the set of all K_p, K_i, k_d for which the closed loop polynomial has a root at zero. From equation 5.14, we can infer that $K_i = 0$.

Boundary induced by a purely imaginary root:

This is the set of all K_p, K_i, k_d for which the closed loop characteristic polynomial has a root at $j\omega$. This is equivalent to:

$$\Delta(jw) = jwD_{pe}(-\omega^2) - \omega^2 D_{po}(-\omega^2) + (N_{pe}(-\omega^2) + j\omega N_{po}(-\omega^2))(K_i - K_d\omega^2 + j\omega k_p) \quad (5.23)$$

Now substituting the values in equation 5.23,

$$j\omega(s^2 + 3.862) - \omega^2(1.4334) + (3.862 + j\omega(0))(K_i - K_d\omega^2 + j\omega K_p) \quad (5.24)$$

Let us denote $K_i - K_d\omega^2$ as α. Splitting the real and imaginary parts from equation 5.23 and equating them to zero, we get:

$$-\omega^2 + 3.862 + 3.862K_p = 0 \quad (5.25)$$

$$-1.4334\omega^2 + 3.862\alpha = 0 \tag{5.26}$$

$$\begin{pmatrix} 0 & 3.862 \\ 3.862 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ K_p \end{pmatrix} = \begin{pmatrix} \omega^2 - 3.862 \\ 1.4334\omega^2 \end{pmatrix} \tag{5.27}$$

$$\begin{pmatrix} \alpha \\ K_p \end{pmatrix} = \frac{1}{-(3.862)^2} \begin{pmatrix} 0 & -3.862 \\ -3.862 & 0 \end{pmatrix} \begin{pmatrix} \omega^2 - 3.862 \\ 1.4334\omega^2 \end{pmatrix} \tag{5.28}$$

and simplifying the equations, we get:

$$\alpha = 0.3711\omega^2 \tag{5.29}$$

$$K_p = \frac{1}{3.862}(\omega^2 - 3.862) \tag{5.30}$$

From equation 5.30, we get

$$\omega^2 - 3.862 - 3.862K_p = 0 \tag{5.31}$$

Let us assume $\omega^2 = \lambda$.

Critical Values:

K_{p0} can be found by setting P(0;K_{p0})=0. That is, when λ = 0,

$$K_p = -1; \tag{5.32}$$

Other critical values of K_p may be found by computing the breakpoints which can be found by calculating, $\frac{dp}{d\lambda}(\lambda, K_p)$. But in this case, it is not possible. Hence, there exists one critical value of K_p at -1. The critical values of K_p split the real line into two intervals ($-\infty$,-1) and (-1,∞).From these two intervals, a sample can be chosen and for each of these chosen K_p we can computer the boundaries of signature invariant regions. From each of the two intervals, we pick $K_p \epsilon \{-2, 2\}$ as the sample. Also all the coefficients of the characteristic polynomial should have the same sign

and hence one of the following linear programs namely LP1 and LP2 must be feasible for every set of stabilizing controller gains.

LP1:

$$1.4334 + 3.862K_d > 0, 3.862K_p + 3.862 > 0, 3.862K_i > 0 \tag{5.33}$$

LP2:

$$1.4334 + 3.862K_d < 0, 3.862K_p + 3.862 < 0, 3.862K_i < 0 \tag{5.34}$$

Now let us consider the case when K_p=-2 and from before $\lambda = \omega^2$.

$$\lambda - 3.862 - 3.862(-2) = 0 \implies \lambda = 3.862 \tag{5.35}$$

$$\alpha = \frac{1.4334}{3.862} * \lambda \implies \alpha = 1.4334 \tag{5.36}$$

Hence,

$$K_i - K_d(\lambda) = 1.4334 \implies K_i - 3.862K_d = 1.4334 \tag{5.37}$$

Combining with the results from equation 5.33 and 5.34, they can be partitioned as follows:

$$1.4334 + 3.862K_d < 0, 3.862K_p + 3.862 < 0, 3.862K_i < 0, K_i - 3.862K_d < 1.4334 \tag{5.38}$$

$$1.4334 + 3.862K_d < 0, 3.862K_p + 3.862 < 0, 3.862K_i < 0, K_i - 3.862K_d > 1.4334 \tag{5.39}$$

$$1.4334 + 3.862K_d > 0, 3.862K_p + 3.862 > 0, 3.862K_i > 0, K_i - 3.862K_d < 1.4334 \tag{5.40}$$

$$1.4334 + 3.862K_d > 0, 3.862K_p + 3.862 > 0, 3.862K_i > 0, K_i - 3.862K_d > 1.4334 \tag{5.41}$$

All the conditions must be satisfied for any of the partitions to be true. Given that K_p = -2, 5.40 and 5.41 are invalid.

Figure 5.4 depicts the plots for equation 5.38 and 5.39. A point from the shaded region is substituted in the characteristic equation and the Routh Hurwitz criterion is checked. Based on Routh Hurwitz criterion, all the gains in the region are unstable.

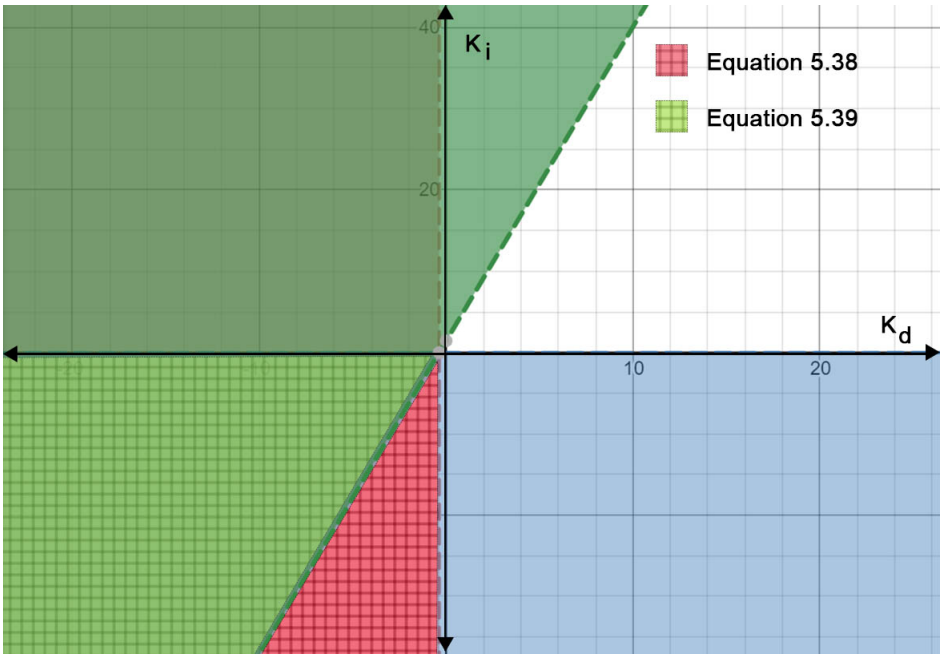

Figure 5.4: Graph for K_p = -2

Hence the interval (-∞,-1) doesn't have valid stabilizing gains for the controller.

Now let us consider the case when K_p=2 and as before $\lambda = \omega^2$. By substitution,

$$\lambda = 4.2955, \alpha = 1.4334 \tag{5.42}$$

Hence,

$$K_i - 2K_d = 4.2955 \tag{5.43}$$

Combining with the results from equation 5.33 ,5.34 and $K_p = 2$, they can be partitioned as follows:

$$1.4334 + 3.862K_d > 0, 3.862K_p + 3.862 > 0, 3.862K_i > 0, K_i - 2K_d < 4.2955 \tag{5.44}$$

$$1.4334 + 3.862K_d > 0, 3.862K_p + 3.862 > 0, 3.862K_i > 0, K_i - 2K_d > 4.2955 \tag{5.45}$$

Figure 5.5 depicts the plots for equation 5.44 and 5.45. A point from the shaded region is

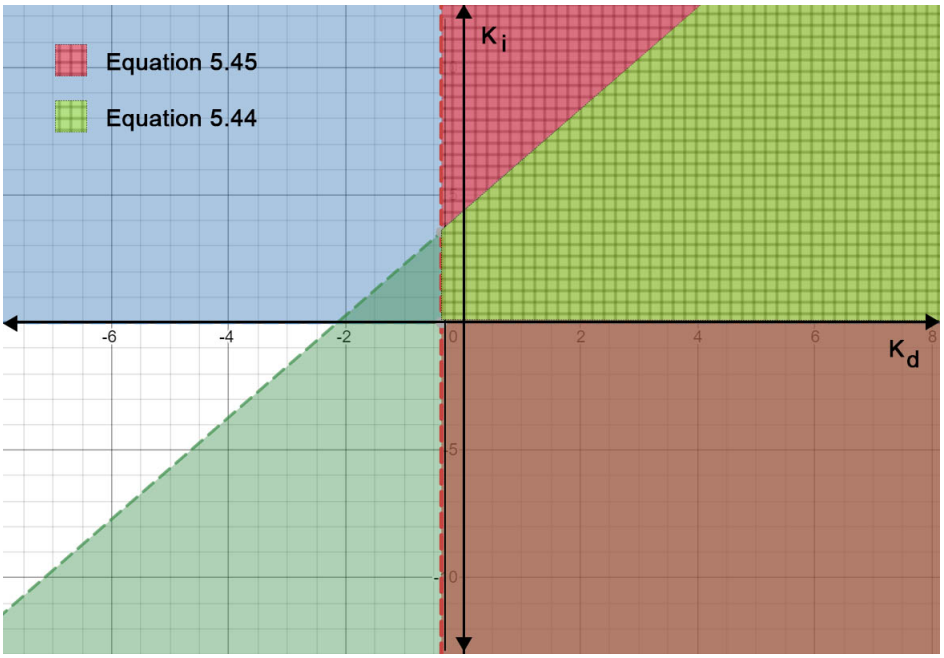

Figure 5.5: Graph for $K_p = 2$

substituted in the characteristic equation and the Routh Hurwitz criterion is checked. Based on this, the gains in the region denoted by equations 5.44 and 5.45 are valid. Hence this alternate interval has valid gains. A value of K_p, K_i and K_d from this interval stabilises the system. A value of controller gain was chosen from this set based on finer tuning.

6. DATA COLLECTION AND PRE-PROCESSING

6.1 Overview

An important aspect of neural networks is the data used to train the model. For the end-to-end approach data has been collected by a human driver by driving around the tracks in the Rellis campus of Texas A&M University. The vehicle is driven at slow speeds. The speed is around 10 miles per hour. Since the throttle is controlled by a human driver, the speed is not constant and can be higher at a few instances and lower at some other instances. On an average, the idea is to keep the vehicle moving at around constant speed. The tracks of Rellis consist of various stretches of straight roads and curved roads. The curves are a good mix of wide turns as well as some sharp turns with low radius of curvature. Figure 6.1 depicts a small portion of one of the test tracks used to collect data.

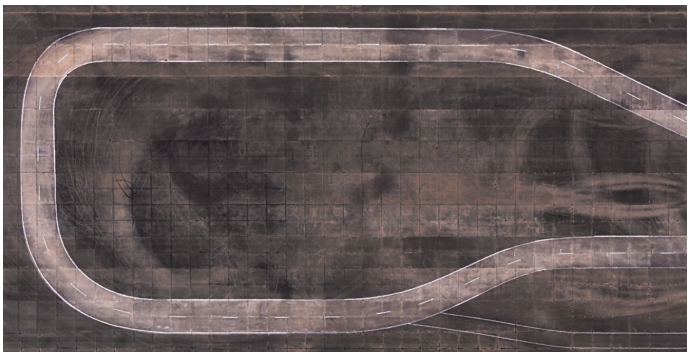

Figure 6.1: A portion of the training tracks

6.2 System Configuration

The pointgrey camera was mounted to the center of the vehicle just beneath the read view mirror. Figure 6.3 is a sample image obtained from the camera while recording training data.At the current resolution, the camera publishes data at about 7 frames per second.

The steering angle is read from the CAN bus of the vehicle and is published onto a topic.

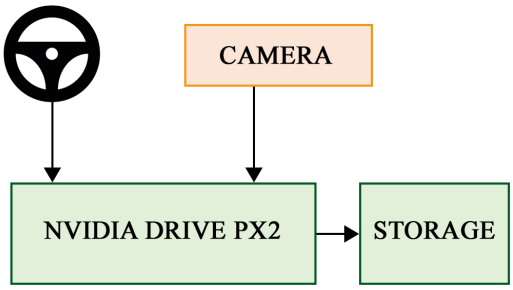

Figure 6.2: System setup

Figure 6.3: Sample output from camera

6.3 Rosbag files

ROS provides an inbuilt file format called as bags which can be used to store and play back ROS message data. These files are stored using an .bag extension. Moreover ROS also provides a variety of tools for play back, visualization and synchronization of these data files.

6.3.1 Data storage

The point of interest for this work is the current steering angle and also the rgb raw images from the camera. By using the rosbag record command, followed by the topics to record, the ROS stores all the messages published into bag files. To record topics published on topics /synchronized_can and /synchronized_image_raw, the following syntax can be used.

```
rosbag record /synchronized_can /synchronized_image_raw
```

6.3.2 Data Retrieval

The stored rosbags can be played back at any time to publish messages at the rate they were published while recording them. Rosbags also allow the user to play these bagfiles at various speeds. A bag file called 2019_10_10_09_00_00.bag can be played by using the rosbag play command.

6.4 Frame Rates and frame synchronization

6.4.1 Sensor frame rate

One specific issue which is present is frame rates for sensors which are not similar. In our case, the camera has a frame rate of about 7fps and the CAN messages are published at a rate of about 11fps. To find out which steering angle corresponds to which image, it is important to match the time stamps by some method and hence find matching pieces of data. Due to the difference in frame rates it is impossible to match the frames based on exact time stamps.

6.4.2 ROS Message Filters

ROS provides the user with a variety of tools to match frame rates specifically for issues such as this. According to the official ROS website, message_filters is a utility library for use with roscpp and rospy. It collects commonly used message "filtering" algorithms into a common space. A message filter is defined as something which a message arrives into and may or may not be spit back out of at a later point in time [1].

One such message filter is the time synchronizer. This is used to synchronize messages which are published onto different topics. The synchronizer achieves this using the time stamp of the topics published. There are multiple types of synchronizers which can be used, but for our application approximate time policy has been used.

[1] http://wiki.ros.org/message_filters

6.4.2.1 *Approximate Time policy*

In approximate time policy, ROS tries to match messages which are closer to each other based on their time stamp and publishes them at a new frequency.

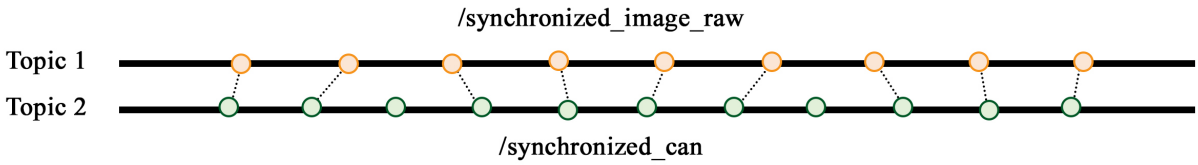

Figure 6.4: ROS approximate time sync

Figure 6.4 gives a good overview about approximate time policy. In the figure, the time moves ahead from left to right. Each line of the figure corresponds to a topic and each dot corresponds to a message. The dotted lines represent messages which are synchronized together. The algorithm behind the working of approximate time policy is beyond the scope of the work but more information on it can be found here [2].

The camera messages and CAN bus messages are time synchronized and are published onto new topics /synchronized_can and /synchronized_image_raw respectively[3]. /synchronized_can and /synchronized_image_raw contain all the information which is needed for training our neural network model and hence it is enough if both these topics are recorded.

6.5 ROS Playback

The recorded rosbag's have the images and steering angles published as ros messages on ros topics. To train the neural network, we will have to convert them as image files and a csv file respectively.

[2] `http://wiki.ros.org/message_filters/ApproximateTime`

[3] More specifications about the time synchronizer and its implementation can be found on my GitHub repository `https://github.com/Akashbaskaran/sync_example`

6.5.1 Rosbag to CSV and Rosbag to image converter

The conversion of rosmessages to CSV files is carried out using a simple ROS node which subscribes to the topic and writes all the messages to a CSV file.

A python script is used to convert ROS messages into jpeg images. The script subscribes to the /synchronized_image_raw topic and converts the images published into rgb8 format using OpenCV libraries and stores them in a given directory[4].

6.6 Data Pre-processing

To reduce the time taken for training and deployment of the neural network, the data obtained from the camera is pre-processed before being fed to the neural network. It can be seen from figure 6.5 that the hood of the car occupies a minor portion of the image being recorded by the camera. Another portion of the image obtained contains the sky. Hence the image is cropped at around 15% from the bottom and at about 40% from the top to keep image size small.

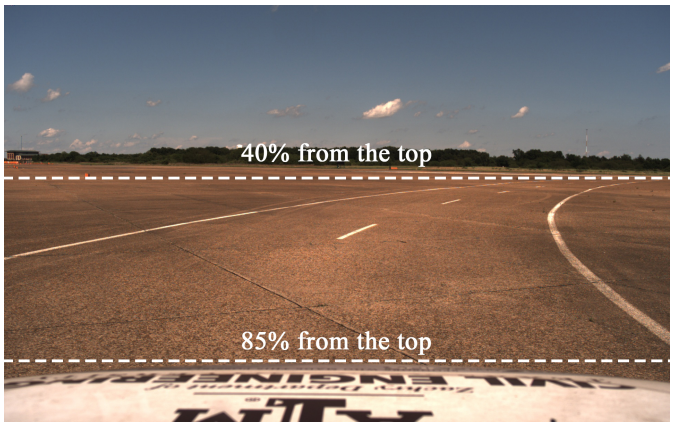

Figure 6.5: Image frame

It has to be noted that, this is an optional step used to reduce the computations required and hence the time taken and can be ignored. The cropped image is then reshaped to (200,66) using CV

[4]The code for rosbag to image converter can be found on my GitHub repository `https://github.com/Akashbaskaran/rosbag_to_img`

Figure 6.6: Image frame

libraries which is then fed to the neural network in the next step. Figure 6.6 is the image obtained after cropping and resizing.

7. NEURAL NETWORKS

Neural Networks have been a major contributor in bridging the gap between the capabilities of man and machine. Neural Networks try to replicate a human brain and hence try to accomplish tasks which don't have a sole defined mathematical definition. Neural networks, like the central nervous system of the human body is made up of building blocks called neurons. The neurons receive input from other interconnected neurons, processes them and sends out an output signal which serve as an input to succeeding neurons. The idea of Neural Network has been present since the early 1990's. But the resurgence of neural network can be attributed to the presence of vast amounts of data in today's world. Neural networks are mainly used for clustering and classification purposes.They can be used to group un-grouped data, label unknown data, classify and predict the outcome of unknown processes. In other words, neural networks try to map the input to the output and hence find correlation between them. Effective and complex computations are carried out by networks in which multiple neuron layers are stacked one above the other. These structured networks with multiple neuron layers stacked are called as deep neural networks. In our work, we try to mimic driver behaviour by using deep neural networks, specifically convolutional neural networks.

7.1 Convolutional Neural Networks

Convolutional Neural network (CNN) is a field of deep learning which is mainly applied to analyzing visual imagery. CNN's use a specialised mathematical operation called as convolution. CNN's use convolutions in place of matrix multiplication in atleast one of the hidden layers to carry out the required tasks. Most CNN's are used to detect useful features from the input data (mostly image frames). CNN's are used for a variety of reasons ranging from simple tasks such as identifying images which contain certain objects to hand writing recognition and even in natural language processing tasks.

A typical convolutional layer consists of hundreds of neurons which are interconnected with

each other and each of these perceptrons/neurons have an activation function which defines the output of every perceptrons. There are a variety of activation functions which are generally used such as sigmoid activation function, Tanh, ReLU, Leaky ReLU, ELU and so on.

7.2 End-to-End driving

Typical autonomous driving tasks involve working on various sub aspects of driving such as path planning, localization, object detection, feature recognition, sign detection to name a few. Tasks such as Object detection, feature recognition and sign detection require features to be manually picked based on which the algorithms can decide on the course and the outcome. These features lead not only to infusion of errors in various stages, they also may not be the most optimal feature from a machines perspectives. With various Original Equipment Manufacturers working on development of machine maps (maps which are incomprehensible to a human but perfect sense to a machine), it is imperative to look at solutions to autonomous driving which don't depend on Human input based feature detection. End-to-End driving [16] by Bojarski et al is a revolutionary method by which using a neural network a driver's behaviour is cloned. The training data would be comprised of driving data by a human driver across terrains and scenarios on various road ways and paths. Based on such a data set, Bojarski et al train a neural network which will then try to predict the steering angle based on the input from a camera. More details about the network architecture of an end-to-end based neural network model has been described in section 7.4.

7.3 Transfer Learning

Though from a research perspective lot of neural networks have been proven to be successful, from an implementation point of view, a major detractor has been the lack/limited availability of training data. Transfer learning is a method which has been popularly used where, a model which was trained to perform a completely different task can be reconfigured to perform a completely different new task. Transfer learning implementation can be mainly divided into two tasks.

- Developed Model Method
- Pre-Trained Model Method

7.3.0.1 Developed Model Method

A modelling problem which has a huge data set is initially chosen. It has to be noted that, the input,output and the concepts/features learnt during mapping/model development have some kind of relationship in between them. Next a deep learning model is developed for this problem. The developed model is now used as an initialization for the second model. Either a part or layers or the full model can be used for the required task. After a bit of tuning, the model can be adapted to work with the problem statement.

7.3.0.2 Pre-Trained Model Method

In this approach, an already existing pre-trained model is chosen. This model can be one of the popular models which have been released as open source in the past. The pre-trained model can be used as a initialization point for solving the required task. The model again can be used as whole or just a part of it can be used for the required task.

7.3.1 Drawbacks of End-to-End learning

Though the end-to-end approach appears to be a revolutionary idea, from an implementation point of view for a research institute, the lack of training data is a huge roadblock. The developers at NVIDIA (Bojarski et al) had access to about 6 months of training data collected over various terrains and scenarios. The data collected has not been opened up for public access and hence it is impossible to replicate the same network in reality. With such limited amount of data which is available it is difficult to make the model converge. Hence transfer learning is adapted to improve the convergence of the network with a small training data set. We use transfer learning to tackle this issue. The next section gives a brief description about some famous models which can be used for the purpose of transfer learning.

7.3.2 Famous models Deep learning models

7.3.2.1 Alex Net

Alexnet [14] is a CNN model submitted for ImageNet Large Scale Visual Recognition Competition(ILSVRC) 2012. This was developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. Alex net went on to win the ILSVRC 2012 on the image net data set, outperforming the runner up by more than 15%. Alex net set the platform for using CNN's for image recognition tasks. It had over 60 million parameters.

7.3.2.2 InceptionNet

The InceptionNet [24], popularly know as googlenet was a neural network model developed by the people at google, and it was a model which went on to win ILSVRC-2014. It consists of over 22 layers and around 4 million neurons. Using batch normalization, image distortions, RMS prop and several small convolutions to drastically reduce the number of parameters.

7.3.2.3 VGG-16

VGG-16 [25] is a CNN model submitted for ImageNet Large Scale Visual Recognition Competition(ILSVRC) 2014. This method was proposed by Simoyan and Zisserman from Oxford University. This was one of the best models in the Imagenet data set, which consists of over 14 million images belonging to over 1000 classes. The model achieved over 92.7% and improved upon the foundations laid by Alexnet. In this work, the large filters which are a specific characterization of Alex net are replaced by small constant size filters. One notable specification about VGG16 was that, it took over a week to train even when using Nvidia GPU's.

7.4 Network Architecture

Figure 7.1 describes the network architecture of the neural network proposed by Bojarski et al in their work. The RGB image is first reduced in size to a 200x66 pixel image. The image is then normalized, and the three normalized planes(each plane corresponding to one channel of the image) is convolved with a 5x5 kernel to create a convolutinal feature map of 24 layers, with a

size of 31x98 pixels. This convolutional map is convolved with two more 5x5 kernels to produce a convolutional feature map having 48 layers and size 5x22 pixels. The so obtained convolutional feature map, is now convoluted with 2 layers of 3x3 kernels which proceeds to yield a 64 layer 1X18 feature map. The 64 layers are flattened onto a single layer consisting of 1164 neurons. This layer is followed by 3 more fully connected layers which reduce the number of neurons from 1164 to 100 to 50 to 10 neurons. The 10 neurons are then pooled to give a single output which predicts the steering angle required.

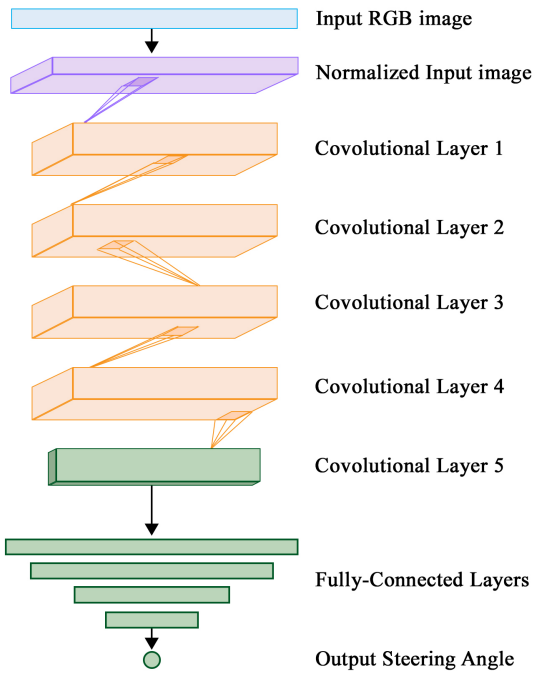

Figure 7.1: Network Architecture of the end-to-end network by Bojarski et al

Due to the constraint placed on the amount of Training data available, to implement transfer learning, we use VGG16. VGG16 is used to initialize the weights of the 5 convolutional layers present. Once the weights are initialized, the network is retrained with these pre-initialized weights on the new inputs collected. All the layers of the neural network in this work use Exponential Linear Unit as the activation function.

7.5 Training Strategy

The model was trained on data collected at different times and weather conditions to prevent over fitting. Moreover the data fed into the network is continuously shuffled for each epoch and also for each batch. The drop rate is an important parameter to maintain the convergence of the network and a drop rate of 0,2 has been used. ADAM optimizer has been used to aid the learning process. ADAM optimizer uses moving averages to allow larger and smaller step sizes as and when required. Moreover, 80% of the data has been used for training the neural network and about 20% of the data has been used for validation. A separate data set has been used for testing purposes. The results of the neural network testing phase have been recorded in the following chapter.

7.6 ROS pipeline

To predict the steering angles from the image observed by the camera, a pipeline in ROS was developed. A ROS subscriber subscribes to the /camera/image_raw topic and sends each of the individual images to the neural network model saved. The neural network model then predicts a steering angle. The steering angle predicted is now published on /predicted_steering_angle. To complete the pipeline and to merge the control and prediction results, the published topic is now subscribed to by the controller node. The controller node is also subscribed to the node which publishes real time steering angle. Based on the current steering angle and the desired steering angle, the controller drives the steering towards the required angle.

8. EXPERIMENTATION AND RESULTS

As specified earlier, data for testing was collected in either tracks unseen by the vehicle or in scenarios which were completely different to that while data was collected. In the works of Bojarski et al, an error is said to be an instance when a manual intervention is needed to correct the trajectory of the vehicle to stay on track. In their works, accuracy is defined by:

$$Accuracy = \frac{N_m * t_1}{T_d} \tag{8.1}$$

where N_m is the number of interventions made by the human driver to correct the course of the vehicle. t_1 is the average time taken by the driver to correct the course of the vehicle. T_d is the total driving time for which the tests have been conducted. Moreover t_1 is assumed to be around 6 seconds. But this metric cant be used as an empirical metric and especially in simulation scenarios. A new metric has been defined as the benchmark for prediction accuracy in this work. Since during data collection, the vehicle was moving at a speed of about 10MPH, a good alternative is to consider a deviation of upto 15° as a working solution and any value of steering angle predicted beyond this threshold is considered a fault. Moreover it has to be noted that the human driver driving data may not be the ideal way to steer the turn. Hence considering all of these factors, an angle of 15° is considered a good approximation.

8.1 Test Case 1 - Constant Speed

In this experiment the vehicle was driven around at a constant speed of around 10 miles per hour on an a track unknown to the neural network but making sure the features on the track remained the same in both cases. The constant speed was maintained by the human driver and hence can be assumed to be approximate. Figure 8.1 is a sample image from the test data set collected.

The data set collected was then tested with the neural network model trained.Figure 8.2 depicts a few cases from the test data. For fig.8.2(a) the model predicts a steering angle of 0.9882° whereas the human driver had taken a path with a steering angle of 0.400°. In fig.8.2(b) the model prediction

Figure 8.1: A sample image from data collected for Test Case 1

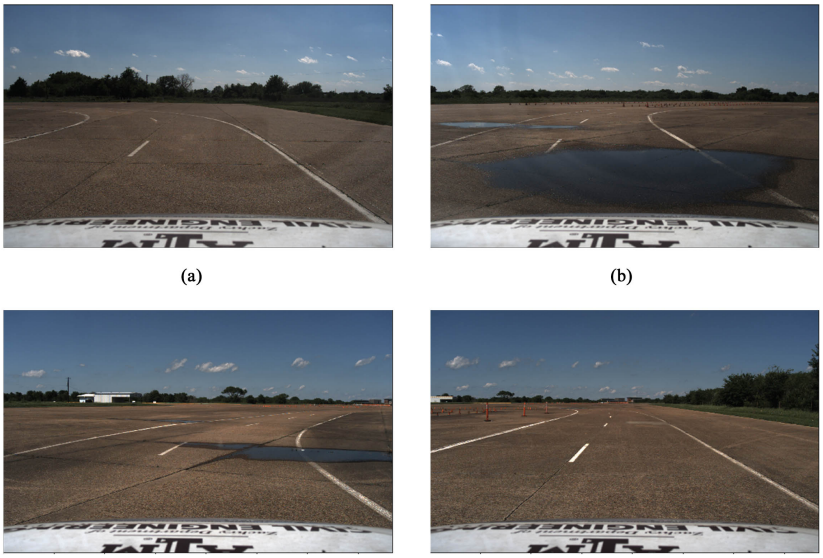

Figure 8.2: Sample image predictions from test data

was -50.3976° whereas the steering angle with a human driver was observed to be -49.0°. In fig.8.2(c) the model predicts 15.9641° whereas the human in the loop takes a path with a steering angle 16.10°. In fig.8.2(d) the neural network model had predicted an angle of -37.0185° whereas the ground truth was -38.09°.

Based on the bench mark defined above, an accuracy of about 94.6 % was achieved. Moreover due to the human in the loop, a better understanding can be achieved by looking at the plot of the human driven angles vs the machine predicted angles. Figure 8.3 depicts the plot of the predicted

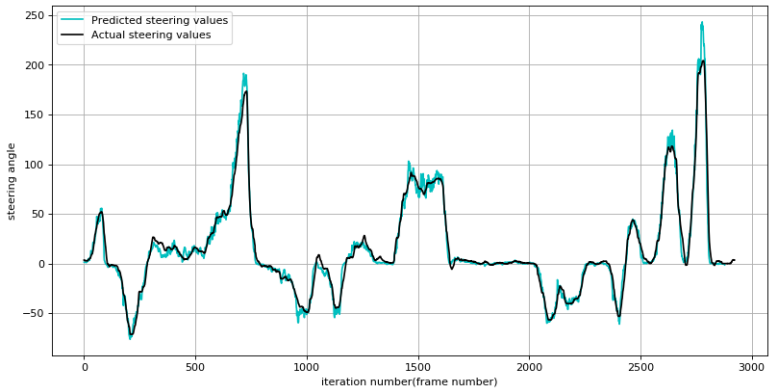

Figure 8.3: Predicted steering angle vs actual steering angle

steering angle for an image vs the actual steering angle with a human in the loop. It can be observed from the plots that both the predicted and actual steering values are nearly the same.

8.2 Test Case 2 - Varying Speed

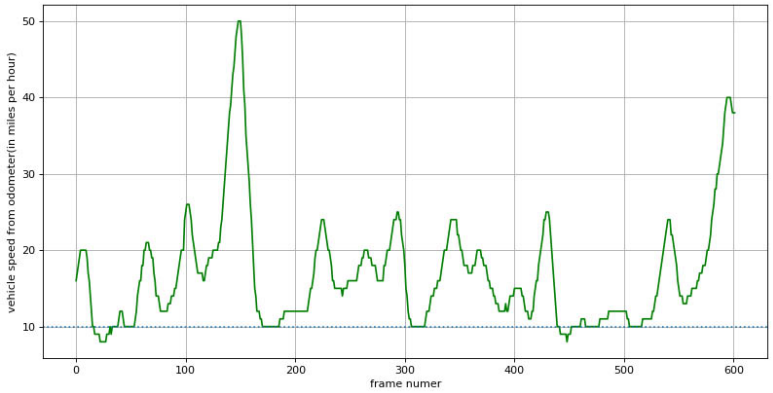

Figure 8.4: Speed profile of the vehicle

In this experiment the vehicle was driven around at varying speeds. The speeds varied from as slow as 8 miles per hour to as high as 50 miles per hour. Figure 8.4 describes the profile of the vehicle speed travelling in the specific track. Figure 8.5 provides the frequency of speeds across the course of the track. The speed profile has a mode of 11.6 miles per hour. The model was trained

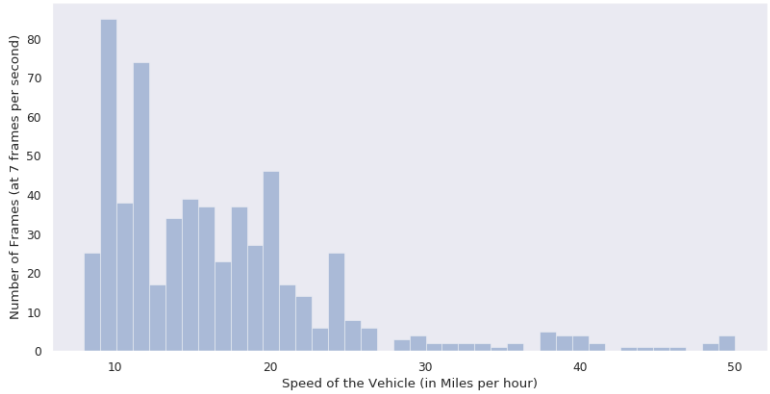

Figure 8.5: Frequency of speeds of the vehicle

with data collected at nearly constant speed of around 10 miles per hour.

Figure 8.6 depicts a few cases from the test data. For fig.8.6(a) the model predicts a steering angle of -5.47° whereas the human driver had taken a path with a steering angle of -8.5°. In fig.8.6(b) the model prediction was -158.3212° whereas the steering angle with a human driver was observed to be -142.100°. In fig.8.6(c) the model predicts -4.179° whereas the human in the loop takes a path with a steering angle -0.2°. In fig.8.6(d) the neural network model had predicted an angle of -71.991° whereas the ground truth was -73.80°.

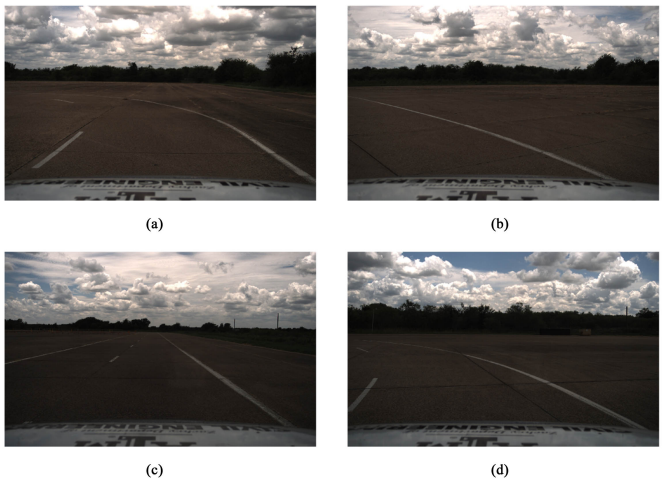

Figure 8.6: Sample image predictions from test data

Following the bench mark defined earlier, an accuracy of about 84.9 % was achieved. Moreover the average error and standard deviation was calculated to get a clear picture of the steering angle predictions. For the trained model, with varying speeds recorded, a mean steering error of 7.8352° was obtained with a standard deviation of 7.48°. As with the previous case, due to the human in the loop, a better understanding can be achieved by looking at the plot of the human driven angles vs the machine predicted angles. Figure 8.7 and 8.8 depict the plot of the predicted steering angle for an image vs the actual steering angle with a human in the loop when the vehicle was driven at varying speeds. It can be observed from the plots that the predicted and actual steering values are nearly the same. Since the speed of the vehicle was never constant through out the track, the reduction in accuracy can be accounted for. More insights into the results obtained are discussed in the succeeding chapter.

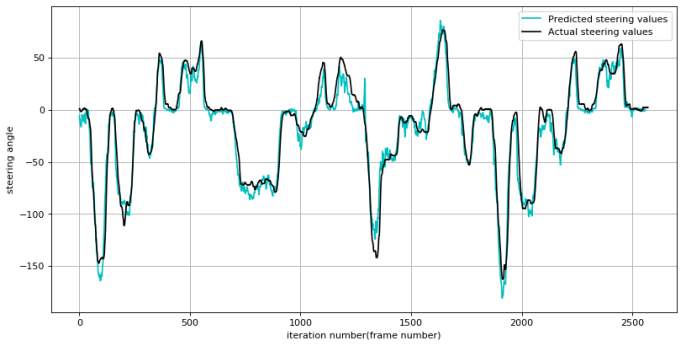

Figure 8.7: Predicted steering angle vs actual steering angle

8.3 Test Case 3- Night Driving

To validate the model trained, we introduce a scenario for which the model is untrained for. The model is tested on a datset collected at night with the headlights of the vehicle turned on. It has to be noted that, though some parts of the data was collected in the evenings(at around 4:00 - 5:00 pm) the training data set had no image which consisted of images collected at night with vehicle headlights turned on. Figure 8.9 depicts a sample image from the testing data set used. In

this test case, the vehicle was driven at around a constant speed of about 10 miles per hour in an unknown track, but with similar features as the original training track.

Figure 8.8: A sample image from data collected at night

The data set collected was then tested with the neural network model trained.Figure 8.2 depicts a few cases from the test data. For fig.8.2(a) the model predicts a steering angle of 3.0138° whereas the human driver had taken a path with a steering angle of 0.80°. In fig.8.2(b) the model prediction was -17.8624° whereas the steering angle with a human driver was observed to be -14.10°. In fig.8.2(c) the model predicts 38.35° whereas the human in the loop takes a path with a steering angle 32.2°. In fig.8.2(d) the neural network model had predicted an angle of -24.86° whereas the ground truth was -18.399°.

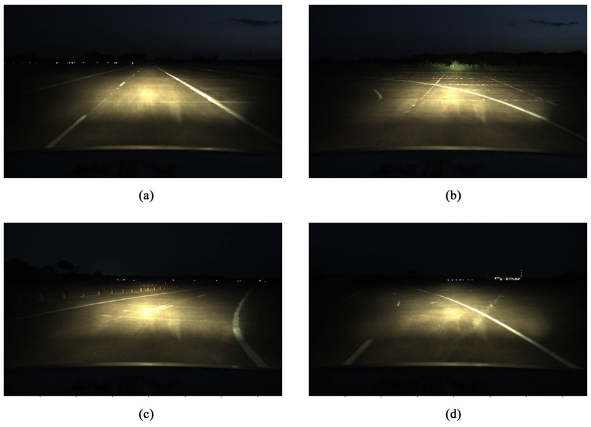

Figure 8.9: Sample image predictions from test data

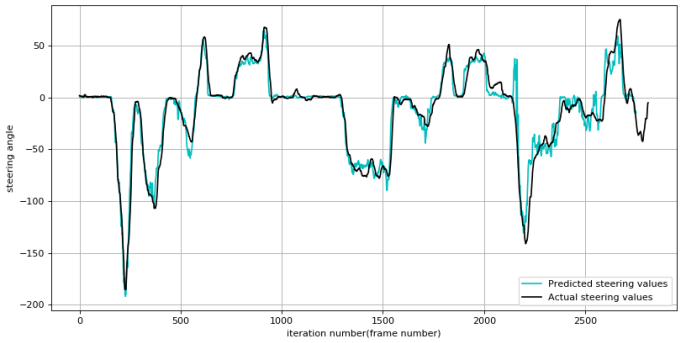

Figure 8.10: Predicted steering angle vs actual steering angle

Following the bench mark defined earlier, an accuracy of about 88.9 % was achieved. Moreover the average error and standard deviation was calculated to get a clear picture of the steering angle predictions. For the trained model, with varying speeds recorded, a mean steering error of 6.6412° was obtained with a standard deviation of 8.28°. As with the earlier cases, the presence of a human in the loop can introduce error into the measure and a better picture of the accuracy can be achieved by looking at the plot of human driven angles vs the machine predicted angles. Figure 8.10 depicts the plot of the predicted steering angle for an image vs the actual steering angle with a human in the loop when the vehicle was driven at varying speeds. More insights into the results obtained are discussed in the succeeding chapter.

9.1 Observation

9.1.1 Test Case 1 - Constant Speed

The trained neural network model was tested on data collected at driving the vehicle at 10 miles per hour. Since the testing data is a lot similar to the training data used, getting a good accuracy is important if the model has been trained properly. Overfitted or mistrained models can lead to a poor accuracy while testing. In test case 1, an accuracy of about 95% has been achieved. This implies, the features which are present in the scene have been currently chosen by the model and moreover the network has been properly trained. Figure 9.1 provides a zoomed in view of figure 8.3. It can be noted that, though the predicted angle line closely follows the ground truth line, there are jitters at lots of instances. A possible solution for this issue would be to include a linefitting alogirthm such as RANSAC [26] to smoothen out the output in the pipeline before the control procedure. This will present a smoother user experience while the vehicle in motion. Moreover the works of Bojarski et al, have used data with lots of features which was collected over a period of over 6 months. In the absence of such huge amounts of data and features, this can be considered as an acceptable solution.

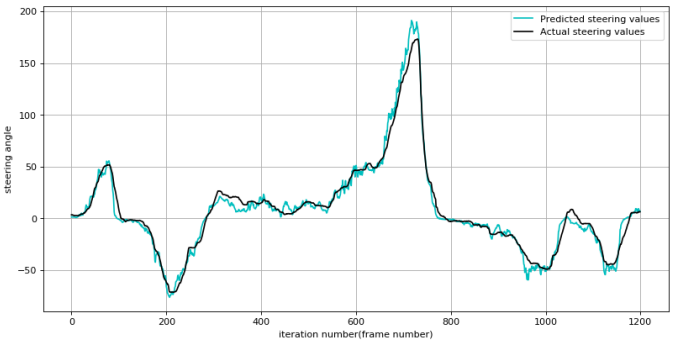

Figure 9.1: Zoomed in view of predicted vs ground truth values

9.1.2 Test Case 2 - Variable Speed

In test case2, the trained neural network, was tested with data collected at various speeds. Moreover the data was unknown by the network. The data was collected in a way in which a human driver would drive on empty roads with no speed limits. In stretches where it was possible to handle the vehicle safely, the speed was increased to the maximum and in turns the speed was reduced to control the vehicle to keep it inside the lanes. This task of predicting the steering angle at variable speeds is something which the model is not accustomed to. The premise for this experiment is that, at higher speeds, the frames will be more intermittent and if during any of the frames the model prediction is a bit further from the ground truth, it will compensate in the next frame. Based on this premise, the experiment was carried out.

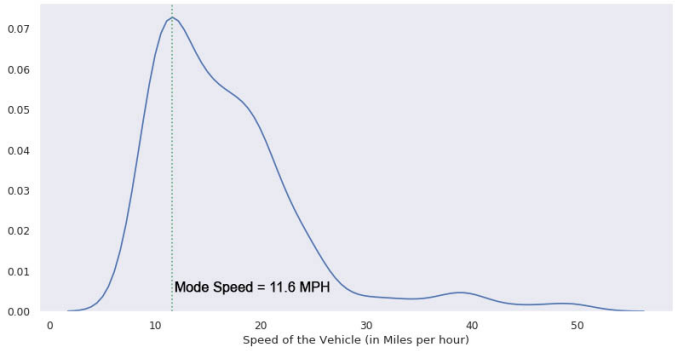

Figure 9.2: A line fitted speed distribution of the vehicle

With a mode speed of 11.6 miles per hour and a top speed of about 50 miles per hour, the model has been tested in extreme cases. Looking at figure 8.4, it can be observed that the speed of the vehicle is continuously changing. Due to the varying nature of the speeds, it is impossible to achieve accuracies like in the previous cases. An accuracy of about 85% was achieved with the varying speed data set. This implies that the model has been able to adapt to the varying speeds in a more than successful method. Figure 8.7 provides a more comprehensive understanding of the prediction. Figure 9.3 provides a zoomed in view of the predicted and actual values. It can

be observed that, compared to test case 1, in multiple occasions the predicted steering angles are a bit different to the driver behaviour. Moreover the presence of jitters is also increased. Using a polynomial filter like RANSAC can smoothen out the graph. With the implementation of such features, a smooth driver behaviour prediction can be obtained at any scenario, given that the speed of the vehicle is within controllable limits. Moreover, It can be noted that at higher speeds(at around 40 miles per hour), the driver behaviour is a bit deviated from the predicted behaviour. Hence the accuracy of 85% includes this behaviour as well. When the speeds given to the vehicle are controller, the accuracy will be even higher.

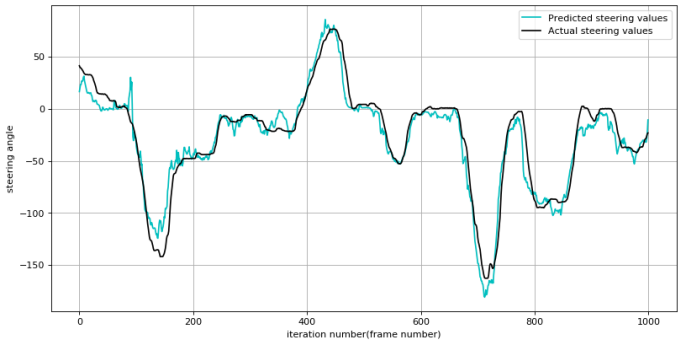

Figure 9.3: Zoomed in view of predicted vs ground truth values

9.1.3 Test Case 3 - Night data

Given that both test-cases achieve the requirement of this work, this test case was added to test the model behaviour at times for which it was not trained for. In real time applications, conditions are not predictable and the model can't be trained for every possible scenario and it should be able to adapt based on the situation. Given that the model had never been exposed to driving under headlights, it should give a really good accuracy if the features to be considered are properly defined. An accuracy of 89% implies that the features chosen by the model are satisfactory. Moreover this also hints at the robustness of the model. Moreover looking closely at Figure 8.10, we can notice a few anomalies. Figure 9.5 gives a zoomed in view around the anomaly observed in figure

8.10. On closer inspection, on the driving data corresponding to frames 920-970, it was observed that, in this time period, one of the lanes was invisible due to the turn made by the vehicle. Hence the deviation in prediction of steering angle can be justified.

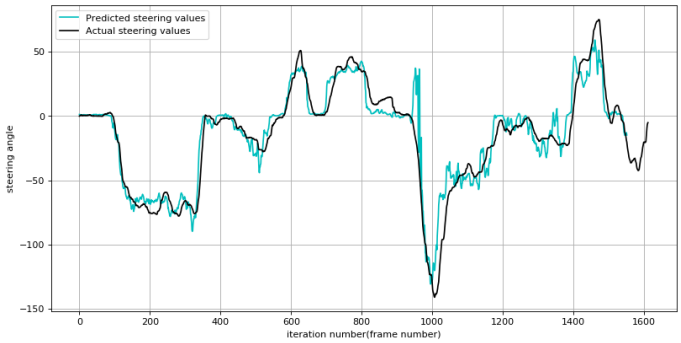

Figure 9.4: Anomaly noticed in predicted vs actual steering data

9.2 Conclusion and Future Works

The objective of this work was to develop a method by which vehicle which vehicles which lack advanced driver assistance can be retrofitted to enable ADAS features and also to achieve lateral autonomy at variable speeds. By using the approach of CAN sniffing and voltage spoofing, the task of drive by wire has been implemented. Any drive by wire vehicle can be equipped with ADAS features. By using an end-to-end driving approach, it has been proven that lateral control of the vehicle at various speeds can be achieved. Moreover by using transfer learning alongside, the amount of training data required and the time taken for convergence has been reduced multi-fold. Given that the test speed variations were highly randomized and irregular, we had achieved an accuracy of around 85%. In normal driving scenario, where a controller acceleration or deceleration is always present, the accuracy will tend to be much better. With the presence of increased training data and increased features, the accuracy of the prediction tends to increase further. Hence it can be concluded that the objective of this work has been achieved. In this work, the steering module was discussed in detail. The same work can be replicated to the throttle and braking systems as well.

With the implementation of such features in the market, it can help in reduction of road accidents. They not only contribute to the reduction of fatalities associated with roads but also help reduce the stress involved with driving.

Driving customs and discipline are starkly different in countries such as the United states and in countries like India. A solution based on object detection and tracking based on rule based behaviour might work in countries with strict driving rules, but might not be the best solution in other cases. In fact, it might even prove to be detrimental. In such cases, emulating driver behaviour using an approach such as end-to-end driving might prove to be a successful option .

REFERENCES

[1] W. H. Organization *et al.*, "Global status report on road safety 2018: Summary," tech. rep., World Health Organization, 2018.

[2] A. Baskaran, A. Talebpour, and S. Bhattacharyya, "End-to-end drive by-wire pid lateral control of an autonomous vehicle," in *Proceedings of the Future Technologies Conference*, pp. 365–376, Springer, 2019.

[3] J. He, H. Rong, J. Gong, and W. Huang, "A lane detection method for lane departure warning system," in *2010 International Conference on Optoelectronics and Image Processing*, vol. 1, pp. 28–31, IEEE, 2010.

[4] A. A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, "Real time lane detection for autonomous vehicles," in *2008 International Conference on Computer and Communication Engineering*, pp. 82–88, IEEE, 2008.

[5] M. Aly, "Real time detection of lane markers in urban streets," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 7–12, IEEE, 2008.

[6] M. Bertozzi and A. Broggi, "Gold: A parallel real-time stereo vision system for generic obstacle and lane detection," *IEEE transactions on image processing*, vol. 7, no. 1, pp. 62–81, 1998.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, IEEE, 1985.

[8] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Aaai/iaai*, vol. 593598, 2002.

[9] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[10] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[11] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[12] J.-H. Kim and J.-B. Song, "Control logic for an electric power steering system using assist motor," *Mechatronics*, vol. 12, no. 3, pp. 447–459, 2002.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[15] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

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

[17] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, IGI Global, 2010.

[18] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[19] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[20] S. P. Bhattacharyya, L. H. Keel, and A. Datta, *Linear control theory: structure, robustness, and optimization*. CRC press, 2009.

[21] CanInAutomation, "History of control area network technology." `https://www.can-cia.org/can-knowledge/can/can-history`.

[22] U. Kiencke, S. Dais, and M. Litschel, "Automotive serial controller area network," *SAE transactions*, pp. 823–828, 1986.

[23] I. Standard, "Iso 11898, 1993," *Road vehicles–interchange of digital information–Controller Area Network (CAN) for high-speed communication*, 1993.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[26] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.