



Department of Information Science & Engineering

2021-2022

A Project Report on

“Fully Autonomous Self Driving Vehicle Using Machine Learning and Artificial Intelligence”

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology In Information Science and Engineering

Submitted by

**Robbins Aby Manjamattom
18BTRIS037**

**Ruchi Tiwary
18BTRIS038**

**Sangeetha Arunraj
18BTRIS040**

**Palash Gupta
18BTRIS026**

Under the guidance of

Dr. Arun N
Assistant Professor
Department of Information Science & Engineering

Department of Information Science & Engineering

Jain Global campus
Kanakapura Taluk – 562112
Ramanagara District Karnataka,
India

CERTIFICATE

This is to certify that the project work titled “**FULLY AUTONOMOUS SELF DRIVING VEHICLE USING MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE**” is carried out by **Robbins Aby Manjamattom (18BTRIS036), Ruchi Tiwary (18BTRIS038), Sangeetha Arunraj (18BTRIS040), Palash Gupta (18BTRIS026)**, a Bonafede students of Bachelor of Technology at the Faculty of Engineering & Technology, Jain University (Deemed-to-be-University), Bangalore in partial fulfilment for the award of degree in Bachelor of Technology in Information Science & Engineering, during the year **2021-2022**.

Dr. Arun N

Assistant Professor
Dept. of ISE,
Faculty of Engineering &
Technology,
Jain University
Date:

Dr. Arun N

Head of the Department
Dept. of ISE,
Faculty of Engineering &
Technology,
Jain University
Date:

Dr. Hariprasad S A

Director,
Faculty of Engineering &
Technology,
Jain University
Date:

Name of the Examiner

Signature of Examiner

DECLARATION

We, **Robbins Aby Manjamattom (18BTRIS036), Ruchi Tiwary (18BTRIS038), Sangeetha Arunraj (18BTRIS040), Palash Gupta (18BTRIS026)**, are students of eighth semester B. Tech in **Information Science & Engineering**, at Faculty of Engineering & Technology, **Jain University (Deemed-to-be-university)**, hereby declare that the project titled **“Fully Autonomous Self Driving Vehicle Using Machine Learning and Artificial Intelligence”** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Information Science & Engineering** during the academic year 2018-2022. Further, the matter presented in the project has not been submitted previously by anybody for the award of any degree or any diploma to any other University, to the best of our knowledge and faith.

Project Members Signature of Examiner

1. Name: Robbins Aby Manjamattom
USN: 18BTRIS037
2. Name: Ruchi Tiwary
USN: 18BTRIS038
3. Name: Sangeetha Arun Raj
USN: 18BTRIS040
4. Name: Palash Gupta
USN: 18BTRIS026

Place: Bangalore

Date:

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to the Faculty of Engineering & Technology, Jain University for providing us with a great opportunity to pursue our bachelor's degree in this institution.

*In particular we would like to thank **Dr. Hariprasad S A, Director, Faculty of Engineering & Technology, Jain University** for his constant encouragement and expert advice.*

*It is a matter of immense pleasure to express our sincere thanks to **Dr. Arun N, in-charge Head of the department, Information Science & Engineering, Jain University,** for providing the right academic guidance that made our task possible.*

*We would like to thank our guide **Mr. Arun N, Head of the Dept , Dept of Information Science & Engineering, Jain University,** for sparing his valuable time to extend help in every step of our project work, which paved the way for smooth progress and the fruitful culmination of the project.*

*We would like to thank our Project Coordinator **Prof. Soumya K N** and all the staff members of Information Science & Engineering for their support.*

We are also grateful to our family and friends who provided us with every requirement throughout the course.

We would like to thank one and all who directly or indirectly helped us in completing the project work successfully.

Signature of Students

ABSTRACT

An autonomous car is a vehicle capable of sensing its environment and operating without human involvement. A human passenger is not required to take control of the vehicle at any time, nor is a human passenger required to be present in the vehicle at all. An autonomous car can go anywhere a traditional car goes and do everything that an experienced human driver does.

We want to build a Fully Autonomous Self Driving Car to implement autonomous driving ability to an electric All-Terrain Vehicle (ATV) by using Xbox Kinect camera, embedded systems, image processing technologies and self-built artificial model to deliver the decision instructions needed for the vehicle to drive autonomously. In this project, electric ATV is fitted with a windshield wiper motor to control steering and custom feedback loop to control acceleration. An Arduino with serial communication is used to communicate with the main processing unit which is also fitted in the ATV, this Arduino is responsible for converting digital signals (decisions from the processing unit) to electric signals (to directly interact with the motor which facilitates autonomous driving). The main parts of our implementation are detection model and control module. The main processing unit is also embedded with NVIDIA GTX1080 or higher to run the detection model. The control module is being run by a neural network model which is trained with human shared experience to make human like driving decisions to control the vehicle. The data is collected by driving the ATV on roads of various terrains. The dataset is collected in such a way that it picks every frame and it is segmented by the Nvidia model. The reason for this approach to achieve self-driving is because driving is a complicated task in which we may need to support the machine in a much better manner than just formulas and sometimes human intuition is the only way to tackle such complications, hence we train the neural network with direct human experience dataset.

Nvidia's semantic segmentation model returns every 10th frame array with classes per pixel. Using this, road edges are extracted from the arrays and road center line is calculated. Centre of the frame is used as center of the vehicle to calculate delta between vehicle center and road center. PID (Proportional Integral Derivative) controller is used to generate steering commands from the obtained delta. Any obstacles which could appear on the driveway are detected with Xbox Kinect Depth Camera and if objects have been detected a speed reduce command is initiated. Steering and speed commands are sent over the serial to Arduino as well and Arduino executes those commands.

TABLE OF CONTENTS

CHAPTER-1

1.1 Introduction

CHAPTER-2

2.1 Literature Survey

2.2 Limitation of our work

CHAPTER-3

3. Proposed Work

3.1 Problem Definition

3.2 Objectives

3.3 Methodology

3.4 Basic Theory

CHAPTER-4

4.1 Hardware and Software tools used

CHAPTER-5

5.1 Design and Implementation

CHAPTER-6

6.1 Results and Discussion

CHAPTER-7

7.1 Conclusion and Future Scope

7.2 References

LIST OF FIGURES

Fig. No	Description of the figure	Page No.
1	UML Diagram	15
2	Software Diagram	16
3	Output Filtering	19
4	Sample Image	20
5	Drive Model	21
6	Cuda logo	24
7	Open CV logo	25
8	PyTorch	25
9	Python logo	26
10	NumPy logo	26
11	Scikit-Learn logo	27
12	HDF logo	27
13	Scikit-Image logo	28
14	Pillow logo	28
15	Piexif logo	29
16	Tensor Board logo	30
17	Ninja-Ide logo	31
18	Pandas logo	31
19	Matplotlib logo	32
20	Pid Controller logo	32
21	CMake logo	33
22	Standard Firmata logo	34
23	Semantic Segmentation logo	35
24	CityScapes logo	35
25	Vehicle with electric drive train	35
26	Kinect	36
27	Wireless Router	36
28	Arduino Nano	36
29	Half bridge Driver	37
30	Wire	38
31	Implementations	39
32	Output Value Generated	48

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

Autonomous cars have the potential to improve road safety and provide millions of people with new transportation alternatives. Autonomous cars have huge potential to improve people's lives, whether they're used to assist people run errands, commute to work, or drop off kids at school. The goal of this project is to create an autonomous vehicle that can drive itself on genuine public roads. This works by leveraging powerful AI detection and learning technologies to train an AI model to anticipate steering and throttle reactions based on RGB-d camera picture.

A convolutional neural network (CNN) is trained to directly transfer raw pixels from a single front-facing camera to driving commands. This end-to-end strategy proved to be remarkably effective. The system learns to drive in traffic on local roads with or without lane markings, as well as on highways, with minimal human training data. It also works in locations where there is a lack of obvious visual direction, such as parking lots and gravel roads. With just the human steering angle as the training signal, the system automatically learns internal representations of the essential processing processes such as recognising valuable road elements.

The machine is supposed to learn on its own and improve accuracy over time, hence this project does not focus on writing each and every command to the machine. Our end-to-end approach optimises all processing steps simultaneously, as opposed to explicit breakdown of the issue, such as lane marker recognition, path planning, and control. We believe that this will lead to improved performance and smaller systems in the long run. Internal components will self- optimize to maximise total system performance rather on human-selected intermediate criteria, such as lane detection, resulting in improved performance. Such criteria are clearly chosen for simplicity of human understanding, but they do not ensure maximum system performance. Smaller networks are possible because the system learns to solve the problem with the minimal number of processing steps.

CHAPTER 2 LITERATURE SURVEY

2.1 Literature Survey

[1]. Title of paper: A brief introduction to opencv

Authors: I CULJAK, D ABRAM, T PRIBANIC, H DZAPO, M CIFRE

Advantages:

1. Quickly make a reader familiar with OpenCV (Open Source Computer Vision) basics without having to go through the lengthy reference manuals and books.
2. Extensive Use. Big companies like IBM, Google, Toyota and even startups like Zeitera and Applied Minds are using OpenCV for multifarious tasks.

Disadvantages:

1. It is mathematically complicated and computationally heavy.
2. For vehicles to be able to drive by themselves, they need to understand their surrounding world like human drivers, so they can navigate easily, but if the opencv is malfunctioned then it could create a big life threat situation. The purpose of this paper is to introduce and quickly make a reader familiar with OpenCV (Open Source Computer Vision) basics without having to go through the lengthy reference manuals and books. OpenCV is an open source library for image and video analysis, originally introduced more than decade ago by Intel. Since then, a number of programmers have contributed to the most recent library developments. The latest major change took place in 2009 (OpenCV 2) which includes main changes to the C++ interface. Nowadays the library has >2500 optimized algorithms. It is extensively used around the world, having >2.5M downloads and >40K people in the user group.

[2].Title of paper: Deployment of semantic segmentation network using tensor

Authors: Joohoon lee (nvidia), Chethan ningaraju (nvidia)

Advantages:

1. High performance neural network inference engine for production deployment
- 2.Deliver high-performance

3. Deploy faster, more responsive and memory efficient deep learning applications with INT8 and FP16 optimized precision support

Disadvantages:

1. Low latency inference demanded by real-time services

[3]. Title of paper: Using multi-scale attention for semantic segmentation

Authors: Joohoon lee (nvidia), Chethan ningaraju (nvidia)

Advantages:

1. Multi-scale inference is commonly used to improve the results of semantic segmentation.
2. Multiple images scales are passed through a network and then the results are combined with averaging or max pooling.

Disadvantages:

1. In addition to enabling faster training, this allows us to train with larger crop sizes which leads to greater model accuracy.
2. Multiple images scales are passed through a network and then the results are combined with averaging or max pooling. In this work, we present an attention-based approach to combining multi-scale predictions. We show that predictions at certain scales are better at resolving particular failures modes, and that the network learns to favor those scales for such cases in order to generate better predictions. Our attention mechanism is hierarchical, which enables it to be roughly 4x more memory efficient to train than other recent approaches. In addition to enabling faster training, this allows us to train with larger crop sizes which leads to greater model accuracy.

[4]. Title of paper: Opencv for computer vision application

Authors: M naveenkumar, A vadivel

Advantages:

1. Helps the computer to understand the content of an image.
2. Deliver high-performance.
2. It provides de-facto standard API for computer vision applications.

Disadvantages:

1. Lack of specialists

2.The aim of image processing is to help the computer to understand the content of an image. OpenCV is a library of programming functions mainly used for image processing. It provides de-facto standard API for computer vision applications.We can solve many real time problems using image processing applications.

[5]. Title of paper: Hardware-software stack for a rc car for testing autonomous driving algorithm

Authors: K shrivastava, M inukonda, S mitta

Disadvantages:

1. The challenges faced in the assembling different software tools and hardware platforms.

In this paper, we report our ongoing work on developing hardware and software support for a toy RC car. This toy car can be used as a platform for evaluating algorithms and accelerators for autonomous driving vehicles (ADVs). We describe different sensors and actuators used and interfacing of them with two processors, viz., Jetson Nano and Raspberry Pi. Where possible, we have used ROS nodes for interfacing. We discuss the advantages and limitations of different sensors and processors and issues related to their compatibility. We include both software (e.g., python code, linux commands) and hardware (e.g., pin configuration) information which will be useful for reproducing the experiments. This paper will be useful for robotics enthusiasts and researchers in the area of autonomous driving.

[6]. Title of paper: HSelf-driving cars – the human side working paper (2017)

Authors: P szikora, N madarász

Advantages:

1. Their appearance is designed to occupy less space on the road to avoid traffic jam.
2. Reduce the likelihood of accidents.

Disadvantages:

1. Drivers may see their driving skills degrade over time.
- 2.First, cognitive offloading can lead to forgetfulness or failure learn even basic operating procedures.

The usage and production of these cars has become a leading industry in almost every area of the world. The world's car stock exceeded 80 million after the Second World War, then more than 90 million in 1960. Five years later this number was 130 million, 291 million in 1980, 419 million in 1990, and 731 million in 2011.

According to the forecasts, it will reach two billion by 2020 [1].

Over the years and centuries, this industry has gone through enormous development, as the first vehicles were only powered by steam engine, then petrol and diesel came to public mind and currently it seems that the electric propulsion will be the future. Of course, with this development, faster and more useful vehicles can be produced, but in our accelerated world with more and more cars, unfortunately the numbers of accidents have increased.

[7]. Title of paper: Autonomous cars: past, present and future - a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology

Authors: K bimraw

Advantages:

1. High upfront cost. The technology will likely come with a high upfront cost for trucking companies to get started.

Disadvantages:

1. Improvement in fuel economy.

This paper can help one understand the trends in autonomous vehicle technology for the past, present, and future. We see a drastic change in autonomous vehicle technology since 1920s, when the first radio-controlled vehicles were designed. In the subsequent decades, we see autonomous electric cars powered by embedded circuits in the roads. By 1960s, autonomous cars having similar electronic guide systems came into picture. 1980s saw vision guided autonomous vehicles, which was a major milestone in technology and till date we use similar or modified forms of vision and radio guided technologies. Various semi- autonomous features introduced in modern cars such as lane keeping, automatic braking and adaptive cruise control are based on such systems.

[8]. Title of paper: Self-driving and driver relaxing vehicle (2016)

Authors: Q memon, M ahmed, S ali, ARmemon

Advantages:

1. High upfront cost. The technology will likely come with a high upfront cost for trucking companies to get started.

2. As the system takes over the control, the driver has a spare time to continue work or spend this time catching up with their loved ones without the having the fear about road safety.

Disadvantages:

1. Autonomous vehicles could be the next major target of the hackers as this vehicle continuously tracks and monitors details of the owner. This may lead to the possible collection of personal data.

In the modern era, the vehicles are focused to be automated to give human driver relaxed driving. In the field of automobile various aspects have been considered which makes a vehicle automated. Google, the biggest network has started working on the self-driving cars since 2010 and still developing new changes to give a whole new level to the automated vehicles. In this paper we have focused on two applications of an automated car, one in which two vehicles have same destination and one knows the route, where other don't. The following vehicle will follow the target (i.e., Front) vehicle automatically. The other application is automated driving during the heavy traffic jam, hence relaxing driver from continuously pushing brake, accelerator, or clutch. The idea described in this paper has been taken from the Google car, defining the one aspect here under consideration is making the destination dynamic. This can be done by a vehicle automatically following the destination of another vehicle. Since taking intelligent decisions in the traffic is also an issue for the automated vehicle so this aspect has been also under consideration in this paper.

[9]. Title of paper: A survey of autonomous driving: common practices and emerging technologies.

Authors: e yurtsever, j lambert, a carballo, k taked

Advantages:

1. 500% increase in lane capacity

Disadvantages:

1. More infrastructure. With more autonomous vehicles on the road, especially larger trucks, more infrastructure will likely be required Automated driving systems (ADSs) promise a safe, comfortable and efficient driving experience. However, fatalities involving vehicles equipped with ADSs are on the rise. The full potential of ADSs cannot be realized unless the robustness of state-of-the-art is improved further. This paper discusses unsolved problems and surveys the technical aspect of automated driving. Studies regarding present challenges, high-level system architectures, emerging methodologies and core functions including localization, mapping, perception.

[10]. Title of paper: Autonomous vehicles: the future of automobiles

Authors: MV rajasekhar, AK Jaiswal

Automated driving systems (ADSs) promise a safe, comfortable and efficient driving experience. However, fatalities involving vehicles equipped with ADSs are on the rise. The full potential of ADSs cannot be realized unless the robustness of state-of-the-art is improved further. This paper discusses unsolved problems and surveys the technical aspect of automated driving. Studies regarding present challenges, high-level system architectures, emerging methodologies and core functions including localization, mapping, perception, planning, and human machine interfaces, were thoroughly reviewed. Furthermore, many state-of-the-art algorithms were implemented and compared on our own platform in a real-world driving setting. The paper concludes with an overview of available datasets and tools for ADS development.

2.2 Limitations of the Current Work

1. Reduced Classes for Semantic Segmentation: Reducing the number of classes involved for semantic segmentation is required to optimize the computer vision of the vehicle.

2. Reduced Instruction Delivery: Instruction delivery to control the vehicle should be reduced to increase the computing speed within the system.

3. Cost Efficient: Self driving vehicles are often considered to be equipped with high technologies which are usually expensive. This norm must be broken by implementing the same efficient system at a cheaper cost.

➤ **Dataset Used:** A more diverse dataset needs to be collected to drive the vehicle around all terrain roads following human intuitions than predefined instructions.

CHAPTER 3: PROPOSED WORK

3.1 PROBLEM DEFINITION

The problem revolves around implementing a system which considers a lot of features, which would result in demand of high computational power. Which apparently points to high grade technology and that does not come inexpensive. In this project the system strictly sticks on the technologies that are inexpensive and methodologies which make computing power more efficient to perform a self-drive on a vehicle.

3.2 OBJECTIVE

- Use a semantic segmentation model which classifies the computer vision of the vehicle to minimal number of classes.
- Choosing the right dataset that facilitates semantic segmentation model.
- Finding the cheapest hardware possible to produce a cost-effective technology for the society.
- Finding ways to optimize the hardware and making successful connections.
- Build a working physical model of the vehicle.
- To establish self-driving capability on to the vehicle, a control model must be built so that it can map the segmented frames to the driving decisions which are basically the throttle value and the steering angle value.
- Train the control model using human driving to replicate human intuitions to the machine, so that the system works flawlessly even without lane markings or well-defined roads.
- Validation of the control model.
- The system should be capable to drive on all terrains even without proper lane markings and well-defined roads.
- The control model used to map segmented frames to driving decision values needs to be built highly robust so that the system can produce accuracy close enough to human intuitions.

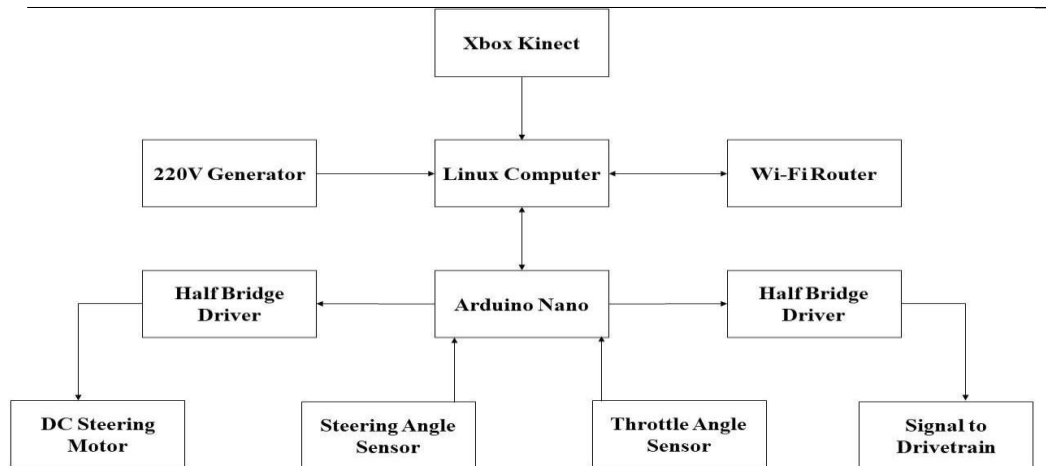
- The system should be equipped with hardware that comes in handy or inexpensive without compromising on the efficiency of the system.
- The system should have reduced number of sensors which leads to lower and optimized computational power.
- When an item is in its path, the vehicle must be able to halt. When there is a slower moving item in its path, the vehicle must also adjust its speed.
- The safety of both pedestrians and passengers must be prioritized.

3.3 METHODOLOGY

Fully Autonomous Self Driving Vehicle includes both hardware and software methodologies. Thereby both hardware and software are discussed below.

3.3.1 Hardware

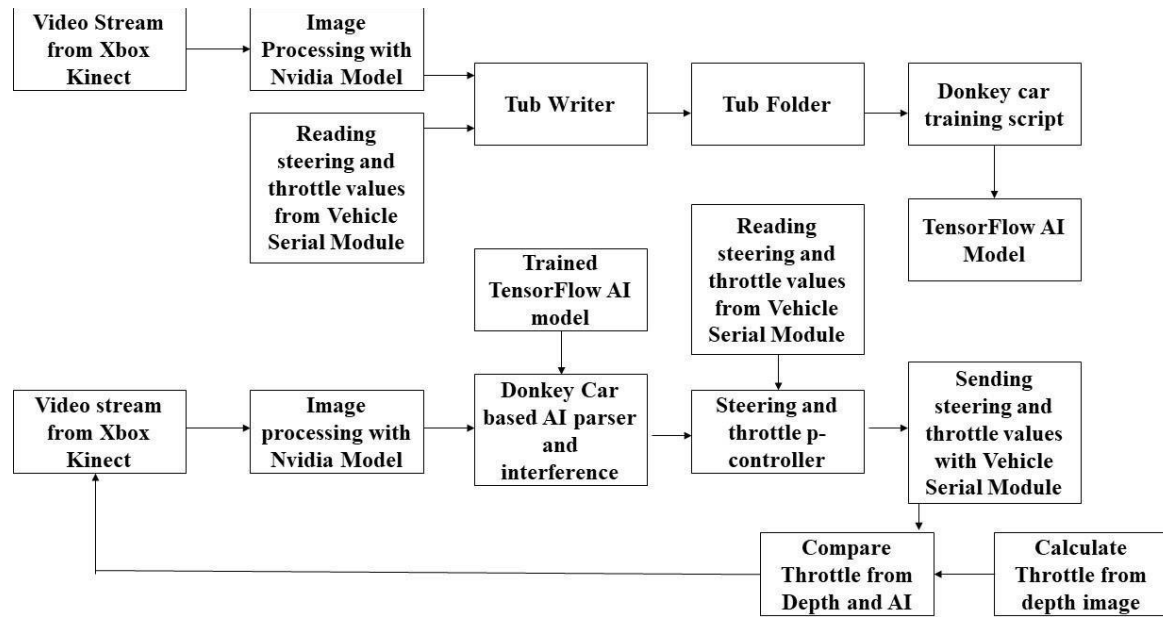
The system is equipped with inexpensive hardware components as possible. For enabling computer vision, we use Xbox Kinect, for computing the system uses a computer of Ryzen 7 or equivalent chip and a minimum graphics card of Nvidia GTX 1080 is used, Arduino Nano is used to mediate between the computer and the hardware components.



The frames captured by the Xbox Kinect is passed to the computer for semantic segmentation. The Nvidia's semantic segmentation model is used to segment every frame into basically 4 classes. They are road (white), vegetation or ground (blue), trees (green), unwanted things (black). Analyzing the segmented frames predictions are made using the control or driver model which are just two instructions; throttle value and steering angle value. These instructions are passed on to the motor's controllers and motors through Arduino Nano which pretends to be an extension of the computer and helps in mediating between computer and components. Steering and throttle sensors are used to collect their respective to train the driver model which the respective data and their respective frames. An optional generator is used to power the computer and Wi-Fi router is used to remotely debug the system.

3.3.1 Software

The software of the self-driving vehicle basically comprises of two Convolutional Neural Network Models. They are Nvidia's Semantic Segmentation Model and Control or Drive Model. Semantic segmentation model is responsible for making the computer understand what is more important and what should it pay attention to. Whereas the drive model is a CNN model which we built unlike the segmentation model which is a pretrained model trained with cityscapes dataset. The dataset is collected by driving the vehicle under all possible terrains to develop an extremely robust CNN model that learns from the combination of the segmented frames and the human intuitive decisions and tries to performs similar intuitive decisions while autonomous drive is performed in the vehicle.



Data acquisition and data exploration is performed by driving the kid's all-terrain vehicle through diverse terrains. The steering values, throttle values and the respective segmented frames are combined into a tub writer which is later parsed into the donkey car training algorithm to train the drive model with the help of TensorFlow. During prediction phase the Xbox Kinect picks the image frames and passes it to Nvidia model for image processing. Which is then later parsed by the donkey car AI parser and drive model to produce prediction and is passed on to the vehicle serial module. Finally, the values predicted by the AI and the calculated values from the depth camera of Xbox Kinect are compared and is fed it to the algorithm if mistake is made during prediction by the system.

3.4 BASIC THOERY

HARDWARE SYSTEM

3.4.1 Road and Object Detection

The Xbox Kinect v1 is used to detect roads. this sensor was chosen because it combines a 640p RGB camera with a 12m range IR matrix depth camera in a relatively inexpensive package. The sensor connects to the main computer through a USB wire. to obtain the input image, the sensor is affixed to the vehicle. Because the RGB and depth sensors lack optical image stabilization (OIS), a good suspension, or a sponge or something else to absorb vibrations and impacts, is required to provide a stable frame to the picture pipeline.

3.4.2 Steering System Sensor

A 10k potentiometer is positioned in line and in the middle of the steering column in the steering system. This potentiometer serves as a steering angle sensor and provides feedback to the program, allowing it to determine the current steering system angle. This is used to identify the steering angle so that it may be mapped to the control model and turned into a trained model. This basic point of termination (pot) is also used to calibrate and determine the actuator limitations.

3.4.3 Throttle System Sensor

We may utilize similar potentiometer-based arrangement as in the last section here. If it is coupled to the drivetrain/powertrain, that is input voltage from 0v to 5v, a 10k potentiometer may be added to the accelerator cable hub, or reverse engineer a PWM driver to retrieve the matching PWM values.

3.4.4 Throttle System

A PWM driver is employed in the throttle system (here I used a 20a dc 10-60 motor speed controller). The PWM input on this MOSFET-powered 7-speed controller spans from 0 to 5 volts. The Arduino- based embedded computer's PWM output is coupled to this input. after that, the Arduino may output a PWM signal to control the speed controller, which in turn controls the drive train.

3.4.5 Embedded Computer (Arduino Nano)

Two computers are installed in the self-driving car. The primary computer oversees detecting the road and translating it into throttle and steering orders. The second computer is an Arduino nano, which is an embedded computer. Because the primary computer has no direct hardware outputs, this configuration was chosen. It does have USB, as does the Arduino nano. As a result, the Arduino nano serves as a translator between the vehicle's hardware and the software on the main computer. The Arduino nano was chosen due of its tiny size and ease of use while yet having all the essential outputs.

3.4.6 Training Sensors

The vehicle must have data from two sensors to train the ai. These are the steering angle sensor and a throttle sensor, as described in the steering system section above. In training mode, the sensor must be able to move the car as intended while also reading the PWM value from an analogue pin on the Arduino. however, in autonomous mode, the Arduino must be able to communicate certain PWM values in order to replicate the usage of the throttle lever on the same wire. As a result, the throttle lever wire must be an input in

one mode and an output in the other. A manual switch has been added to ensure that the Arduino is not shorted, switching between an analogue input pin and a PWM output pin depending on the user's selection.

3.4.7 Main Computer (A Linux Machine with Nvidia GPU)

A powerful computer is required to process everything. A Jetson nano embedded computer is good, but it's not powerful enough for this purpose. As a result, a PC has been picked. This computer contains an i7 CPU and a GTX1650 graphics card for AI processing. Because of the faster speed and the vibrations and effects from the power supply system, the PC also has an SSD installed. On a rotating hard drive, these vibrations can cause read and write errors.

3.4.8 Wireless Connection

A wireless connection is required for debugging and managing the program on the PC. This is accomplished through the use of a vehicle mounted wireless router. The router is linked to the PC through an ethernet wire, and the user's laptop is connected to the router via Wi-Fi. The x2go screen sharing protocol, which uses SSH, allows the laptop to connect to the desktop computer. X2go outperforms VNC in terms of speed and reliability.

3.5 Software System

Three primary components of the autonomous vehicle's software are continually operating in parallel in distinct threads. Each component is described in detail below. In addition, the car must learn to drive in a learning mode. This style of learning/training is also detailed further down.

3.5.1 Road Detection and Control

The road detection and control thread are one of the most crucial components. This thread obtains the camera picture, runs it through a segmentation model for road identification, then runs it through an end-to-end drive model to create steering and throttle instructions, which it finally delivers to the control thread. Below is a more detailed explanation of each stage.

Camera Image: The first step is to obtain the Xbox Kinect camera picture. The library Freenect drivers and python wrapper are used to do this. To retrieve the picture in the commonly used OpenCV format, a simple `get_video()` method is utilized. The image is then passed through a Gamma colour filter to brighten it up a touch. This aids the AI's detection of roads.

The image is reduced to 512 by 256 pixels after the Gamma filter. This is due to the magnitude of the segmentation model's input.

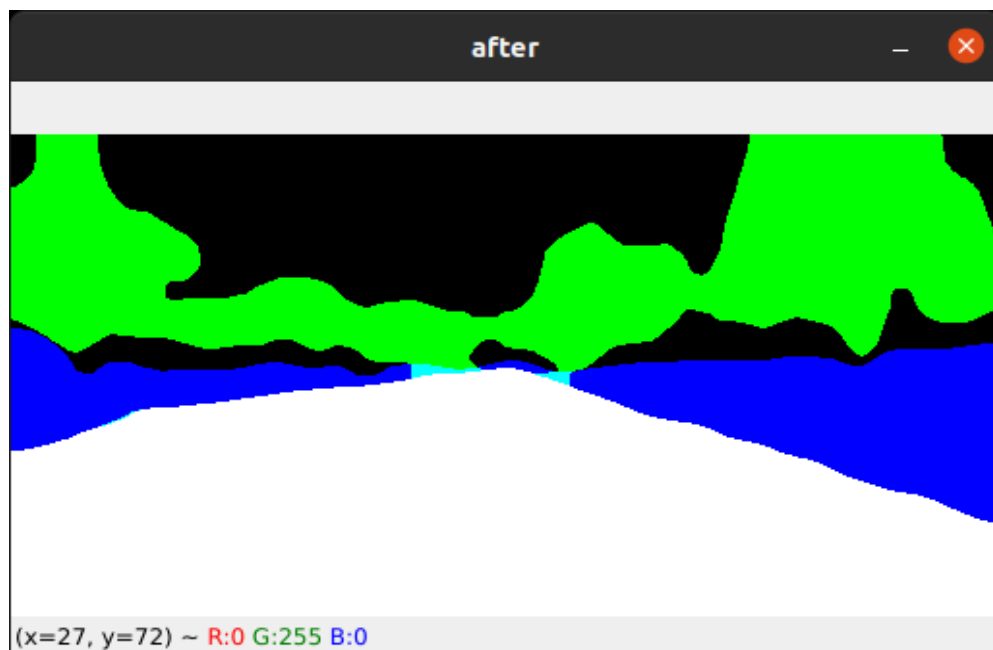
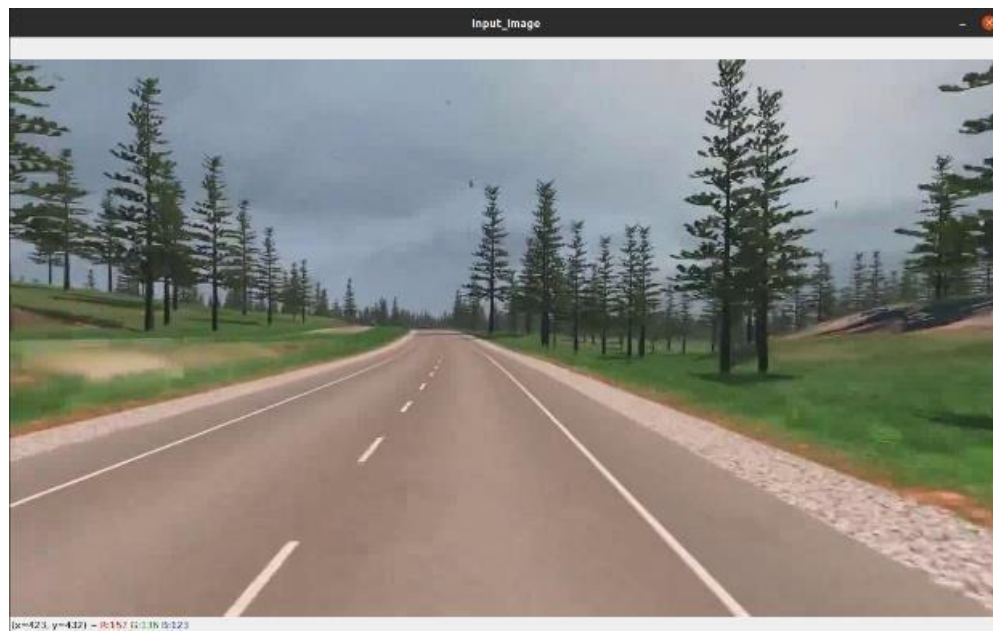
Segmentation Model: The scaled picture is sent into a segmentation model created by Nvidia. A Convolutional Neural Network (CNN) based on the Deeplabv3- Plus and WideResNet38 architecture is used in this model. Nvidia developed this model using the City Scapes dataset, which contains many roadways. As a result, the network can recognize roads and other types of objects effectively. The model also employs a novel approach for accelerating frame

segmentation by mixing frames for improved real-time performance, which is critical for autonomous vehicles. During testing, the model was able to achieve a frame rate of roughly 15 frames per second with a GPU usage of 99 percent the segmentation model is intended to draw attention to specific aspects of the whole image, including the road. This is done for the driving model to better understand which aspects are relevant and which are not. If the network is fed a regular camera picture, it will have a hard time determining which elements are essential and which are not. This is because the model is unaware of the importance of the road and, as a result, begins to focus on the surrounding landscape, such as the sky. It may, for example, begin to recognize that a particular pattern in the sky corresponds to a particular steering angle. This is obviously incorrect; thus, a scene segmentation is performed to aid the model. The output is filtered after segmentation, with significant classes highlighted and non-important improve by creating a new dataset that includes these sorts of roadways. There are also several quicker models available, but none of them have been pre-trained and are not available online. The choice was taken to utilize a pretrained model to save time and since the city scapes data collection is not publicly available. However, in a future trial, it may be worthwhile to consider creating a bespoke dataset. Classes deleted from the image. The next section explains how this works in detail. While the accuracy of the road detection is decent, it may be enhanced. While the car is being tested on roads in an agricultural region, the City Scapes dataset only contains roadways in metropolitan areas. The detection model's accuracy will

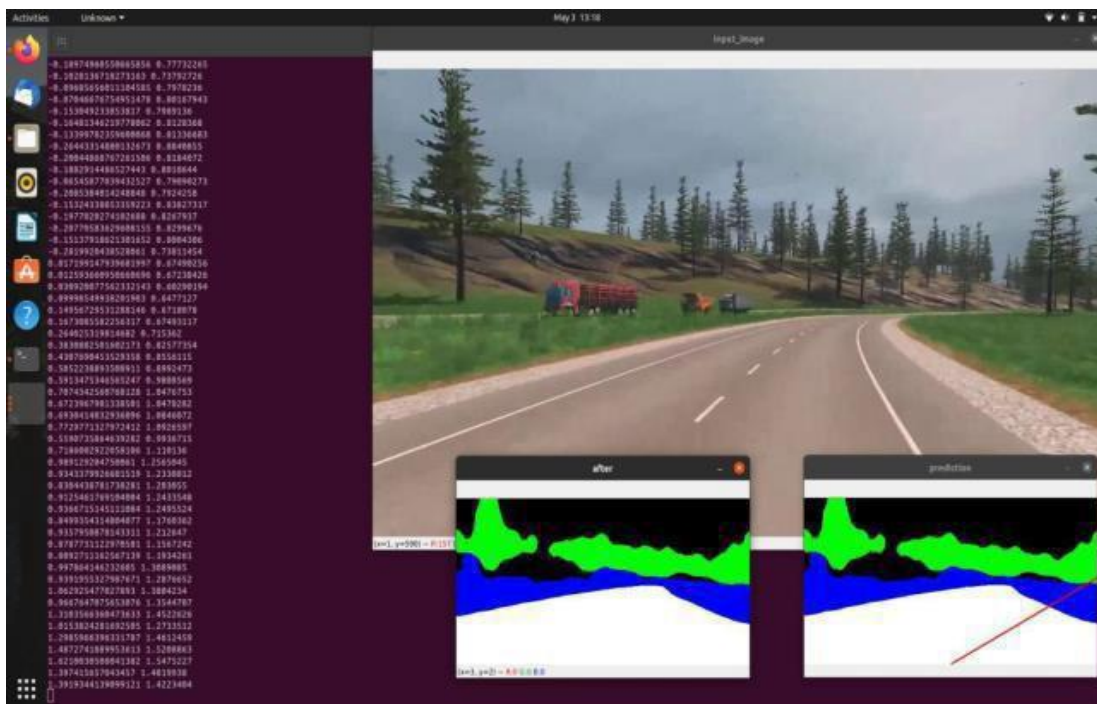
Output Filtering: The detection model's accuracy will improve by creating anew dataset that includes these sorts of roadways.



There are also several quicker models available, but none of them have been pre-trained and are not available online. The choice was taken to utilize a pretrained model to save time and since the City Scapes data collection is not publicly available. However, in a future trial, it may be worthwhile to consider creating a bespoke dataset. This is extremely excellent; however, the segmentation model's main objective is to filter out all non-essential classes and emphasize the most significant. Because a convolutional layer just receives an RGB value per pixel, highlighting a certain class is as simple as boosting its RGB value. This is why the road is white. The values for white colour in RGB are (255,255,255). This should optimally stimulate the neurons at these pixels, allowing the model to learn this feature quickly. Vegetation and ground are the other two essential groups. Because of the trees on the side of the road, the width of the road can be seen, and the ground is highlighted because it is what is on the side of the road. Both obtain a value of 255 for one RGB value and 0 for the other two. This provides the model with a very distinct contrast between the two classes, allowing it to learn edges more effectively. This is critical because the model must learn to recognize road boundaries in order to determine where to drive and where not to drive. All other classes that aren't significant are blacked out with RGB values (0,0,0). Because these pixels do not trigger neurons, the model will not attempt to learn characteristics from these classes. The detection in the sample image below is excellent, although this is not always the case. Sometimes there are potholes in the road, or there are other issues with the outcome. The output picture of the segmentation model is smoothed utilizing a morphological open approach to ensure that the drive model always receives a smooth image. To smooth down rough regions, a kernel of (15,15) is used to make the picture smaller and then bigger again. Although some information is lost because of this, it is unimportant because the drive model does not require per- pixel precision



Drive Model: There are two ai models in the autonomous car. The first divides the vision into pieces, while the second, known as the drive model, anticipates steering and throttle orders. the drive model is a CNN that is based on Nvidia's end-to-end driving paper. This model learns patterns between an input image and a set of throttle value and steering angle output using various convolutional and fully connected layers. once trained, the model can offer a matching throttle and steering value using only an image as input. It is able to do so because it is capable of learning complicated characteristics such as road boundaries, as demonstrated in the figure below. to perform at its optimum, the driving model need a comprehensive and well-balanced training program.



Vehicle Control: The drive ai sends the steering and throttle commands to the control thread. For steering, this is a number between -1 and 1, while for throttle, it is a value between 0 and 1. Every loop, the ai controller uses the Arduino to read the current throttle and steering angle. (The Arduino runs the standard Firmata code, which allows you to control the Arduino's outputs and inputs through USB using the standard fermata protocol in python.) The controller then compares the current read steering and throttle values to the drive model's most recent intended values. The difference between the current and intended values is determined, and the error is then supplied into a PID controller that is set to minimize the difference. The PID controller determines the necessary steering and throttle commands, which are then sent to the Arduino for execution. the controller thread operates at roughly 100hz, whilst the main thread runs at only 15hz. the controller will be able to better regulate

the steering and throttle because of this.

Obstacle Detection: The Xbox Kinect v1's depth camera is utilized for obstacle detection. A matrix of infrared dots is projected by this camera. The Kinect can tell you how far each dot in the matrix is apart. The distance between certain dots varies if an impediment is in front of the camera. This difference may be used by the Kinect to compute depth. The obstacle detection method operates by obtaining the Kinect's depth map at a rate of roughly 30 frames per second. The depth map is a NumPy array with values ranging from 0 to 2048 in each element. The 100000 lowest values are gathered, and the average of these lowest values is checked to see whether it exceeds a particular threshold, indicating the presence of an obstruction. If this is the case, the average is calculated and mapped to a range of 0 to 1, indicating the recommended throttle instruction. This proposed throttle command is compared to the Ai's suggested throttle command. After that, the control system chooses the lowest throttle command and sends it to it. The value of 100000 can be changed to identify smaller or bigger items better. In addition, the depth map must be trimmed such that the road is not identified as an object. One thing to keep in mind is that the Xbox Kinect v1 detects motion via infrared. In bright sunshine, this may create interference. If the Xbox Kinect is unreliable in intense sunlight, an alternative solution may be required.

3.5.2 Training

It is necessary to train the driving model for it to forecast the right steering and throttle values. This is accomplished with a method known as behavior cloning. To begin, I physically drive the vehicle over the road. I preserve the image, throttle, and steering values for each image frame. After a few hours of manual driving, a large dataset is created that demonstrates how to operate the car. The end-to-end driving model has now been trained to anticipate the values associated with the input picture. The Ai model has learnt to drive the automobile by studying how a person has driven it, and hence the human's behavior has been replicated.

It is critical that the data used by the AI model be of high quality and, most importantly, properly balanced for it to appropriately learn human behavior. This implies the AI has a lot of data on both steering left and right, as well as driving straight. A handful of training (next to centerline runs) runs with oscillations are required to ensure that both steering left and steering right are accurately represented. This implies that for both the left and right sides, the user must drive the car from the center to the side of the road and back to the center. This will also assist with edge detection since the driving model will learn where the road's edges are because the user will be moving to and from

the edge of the road. The oscillations also aid in learning how to fix a vehicle that is drifting off the road. If no oscillations were performed and the car was solely driven on the centerline of the road, the Ai would never learn to correct, making it to fail. It's crucial to execute only a few oscillations runs since the car will oscillate when driving autonomously if you don't. Below is an illustration of a well- balanced training set. This image depicts a training set for a nearly straight route. This is evident since in the training run, steering straight is done the most and there is a great balance between steering left and right.

3.5.3 Calibration

There are a few different calibrating tools available. These are the following: The steering and throttle positions may be calibrated with a tool. Because each vehicle is unique, this is quite significant. A user must be able to tune the throttle and steering ranges since they may differ. The actuator calibration test script, which asks the user to operate the steering wheel and throttle, may be used to do this. The application then records the control variables' minimum and maximum values. A tool for calibrating PID values by adjusting the steering and throttle to minimum and maximum values. After that, the user may alter the PID values to get the result they want.

CHAPTER 4: HARDWARE AND SOFTWARE TOOLS USED AND TOOLS DESCRIPTION

4.1 Tools Used

The build this system hardware and software tools are used, among which important ones are listed below.

4.1.1 Hardware Tools

1. Vehicle with electric drive train.
2. Powerful desktop pc/laptop with Nvidia graphics.
3. Mains power (gas) generator if you are using a pc. A laptop can run from its internal battery.
4. Xbox Kinect
5. Wi-Fi router for telemetry
6. Arduino Nano with USB cable
7. Half bridge driver BTS7960
8. Safety buttons
9. Large 10k potentiometer with metal shaft
10. A couple meters of wires.
11. Old ethernet cables work nicely
12. Welding machine and angle grinder
13. Basic tools

4.2 SOFTWARE TOOLS

• Nvidia	• Ninja
• Cuda	• Nose
• OpenCV	• Pandas
• PyTorch	• Thredded
• NumPy	• Time
• Python3	• Sys
• Sklearn	• Scipy
• H5py	• Matplotlib
• Scikit-image	• Simple-pid

• Pillow	• Argparse
• Piexif	• Freenect
• Cffi	• Cmake
• Tqdm	• Standard
• Dominate	• firmata
• Tensorboar Dx	• Libusb

4.2.1 Tool Description

Tool describes the entire list of hardware and software tools used to build the system.

➤ **CUDA**



CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) that allows software to use certain types of graphics processing units (GPUs) for general purpose processing, an approach called general-purpose computing on GPUs (GPGPU). CUDA is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. CUDA is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming. CUDA-powered GPUs also support programming frameworks such as OpenMP, Open ACC and OpenCL and HIP by compiling such code to CUDA. CUDA was created by Nvidia. When it was first introduced, the name was an acronym for Compute Unified Device Architecture,^[5] but Nvidia later dropped the common use of the acronym.

➤ **OPEN CV**



OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source Apache 2 License. Starting with 2011, OpenCV features GPU acceleration for real-time operations.

➤ **PyTorch**



PyTorch is an open source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Meta AI. It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface. A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers, PyTorch Lightning, and Catalyst. PyTorch provides two high-level features:

Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)

Deep neural networks built on a tape-based automatic differentiation system

➤ **Python 3.7**



Python is a simple language which syntax is similar language a c. It is easy to learn, effective programming language. It has efficient high-level information structures and easy and effective technique to object-oriented programming concepts. Python's fashionable syntax and dynamic typing, collectively with its interpreted nature, make it a high-level language for scripting and rapid application building across various platforms.

- Features of python 3.7
- Easier access to debuggers through a new breakpoint() built-in simple class creation using data classes
- Customized access to module attributes
- Improved support for type hinting
- Higher precision timing functions

This is our main backbone of our project. Using this language, we perform various operations on our datasets, create model, train and finally predict the user input news.

➤ **NumPy**



NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. NumPy is a Num FOCUS fiscally sponsored project.

➤ **Scikit-learn**



Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support- vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a Num FOCUS fiscally sponsored project.

➤ **HDF**



Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data. Originally developed at the National Centre for Supercomputing Applications, it is supported by The HDF Group, a non-profit corporation whose mission is to ensure continued development of HDF5 technologies and the continued accessibility of data

stored in HDF. In keeping with this goal, the HDF libraries and associated tools are available under a liberal, BSD-like license for general use. HDF is supported by many commercial and non-commercial software platforms and programming languages. The freely available HDF distribution consists of the library, command-line utilities, test suite source, Java interface, and the Java-based HDF Viewer (HDF View). The current version, HDF5, differs significantly in design and API from the major legacy version HDF4.

➤ **Scikit-image**



scikit-image is an open-source image processing library for the Python programming language. It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

➤ **Pillow**



Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7. Development of the original project, known as PIL, was discontinued in 2011. Subsequently, a successor project named Pillow forked the PIL repository and added Python 3.x support. This fork has been adopted as a replacement for the original PIL in Linux

distributions including Debian and Ubuntu (since 13.04).

➤ **Piexif**



Piexif simplifies interacting with EXIF data in Python. It includes the tools necessary for extracting, creating, manipulating, converting and writing EXIF data to JPEG, WebP and TIFF files.

➤ **CFFI**

C Foreign Function Interface for Python. Interact with almost any C code from Python, based on C-like declarations that you can often copy-paste from header files or documentation.

➤ **TQDM**

TQDM is a library in Python which is used for creating Progress Meters or Progress Bars. TQDM got its name from the Arabic name *taqaddum* which means ‘progress’. Implementing TQDM can be done effortlessly in our loops, functions or even Pandas. Progress bars are useful in Python because: One can see if the Kernel is still working.

Progress Bars are visually appealing to the eyes.

It gives Code Execution Time and Estimated Time for the code to complete which would help while working on huge datasets

➤ **Dominate**

Dominate is a Python library for creating and manipulating HTML

documents using an elegant DOM API. It allows you to write HTML using an elegant DOM API. It allows you to write HTML

➤ **Tensor Board**



Tensor Board provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs
- And much more

➤ **Nose**

Nose is a popular test automation framework in Python that extends unit test to make testing easier. The other advantages of using the Nose framework are the enablement of auto discovery of test cases and documentation collection.

➤ **NINJA-IDE**



NINJA-IDE is a cross-platform integrated development environment (IDE) designed to build Python applications. It provides tools to simplify Python software development and handles many kinds of situations thanks to its rich extensibility.

➤ **Pandas**



Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

➤ **Time**

This module provides various time-related functions. For related functionality, see also the datetime and calendar modules. Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this module call platform C library functions with the same name. It may sometimes be helpful to consult the platform documentation because the semantics of these functions varies among platforms.

➤ **Sys**

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

➤ **Argparse**

The argparse module makes it easy to write user-friendly command-line

interfaces. The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

➤ **SciPy**

SciPy in Python is an open-source library used for solving mathematical, scientific, engineering, and technical problems. It allows users to manipulate the data and visualize the data using a wide range of high-level Python commands. SciPy is built on the Python NumPy extension. SciPy is also pronounced as “Sigh Pi.”

➤ **Matplotlib**



Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter. Since then, it has an active development community and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012 and was further joined by Thomas Caswell. Matplotlib is a NumFOCUS fiscally sponsored project.

➤ **PID Controller**



A simple and easy to use PID controller in Python. If you want a PID controller without external dependencies that just works. The PID was

designed to be robust with help from Brett Beauregards guide.

➤ **Freenect**

The Python wrapper is written in Cython Ctypes. This is the Cython-based libfreenect Python wrappers. This provides async (e.g., using call-backs) and sync (e.g., simple function calls) interfaces to libfreenect. Most of the run loop is abstracted so that later upstream modifications will have minimal impact on your code; however, you are free to implement your own runloop and access all available functionality. The interface is designed to mimic the C library except where 1.) It is impractical or 2.) It would be "Un-Pythonic" to do so. See "Differences with C Library" for specific instances of this. The exposed interface supports both asynchronous (e.g., call-back) and synchronous (e.g., call and block) functions which is enabled by the C_Sync_Wrapper.

➤ **Random**

Python Random module is an in-built module of Python which is used to generate random numbers. These are pseudo-random numbers means these are not truly random. This module can be used to perform random actions such as generating random numbers, print random a value for a list or string, etc.

➤ **Pickle**

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “*Pickling*” is the process whereby a Python object hierarchy is converted into a byte stream, and “*unpickling*” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” 1 or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

➤ **Zlib**

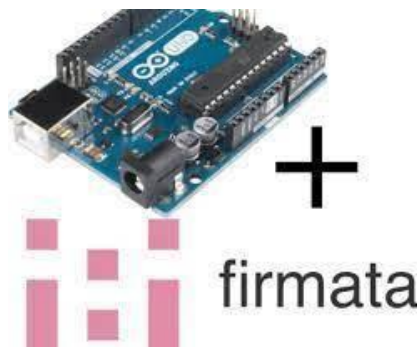
For applications that require data compression, the functions in this module allow compression and decompression, using the zlib library. There are known incompatibilities between the Python module and versions of the zlib library earlier than 1.1.3; 1.1.3 has a security vulnerability, so we recommend using or later. zlib’s functions have many options and often need to be used in a particular order.

➤ **CMake**



In software development, CMake is cross-platform free and open-source software for build automation, testing, packaging and installation of software by using a compiler-independent method.^[3] CMake is not a build system but rather it generates another system's build files. It supports directory hierarchies and applications that depend on multiple libraries. It is used in conjunction with native build environments such as Make, Qt Creator, Ninja, Android Studio, Apple's XCode, and Microsoft Visual Studio. It has minimal dependencies, requiring only a C++ compiler on its own build system. CMake is distributed as open-source software under permissive BSD-3- Clause license.

➤ **Standard Firmata**



Standard Firmata is a software library that allows Arduino devices to communicate with your computer using the Firmata protocol. There are numerous software packages that make use of Standard Firmata, including Processing Python pyFirmata, HyperStudio, HyperDuino, and many others.

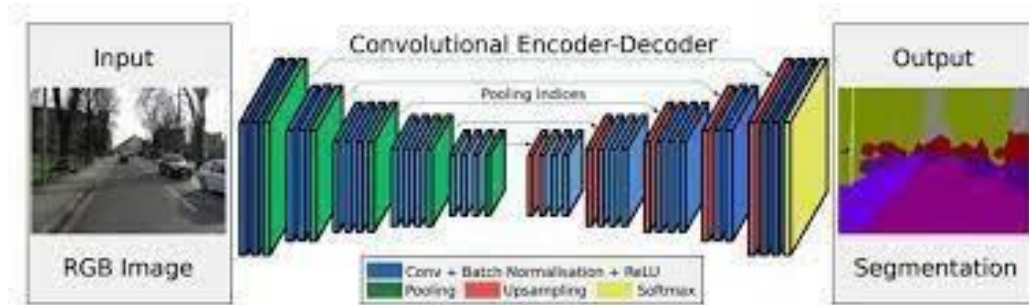
➤ **Libusb**

Python **libusb** module is a low-level binding for libusb C library. It is an effort to allow python programs full access to the API implemented and provided by the well-known *libusb* library.

➤ **Pyfirmata**

pyFirmata is a Python interface for the Firmata protocol. It is fully compatible with Firmata 2.1, and has some functionality of version 2.2. It runs on Python 2.7, 3.3 and 3.4.

➤ **Semantic Segmentation**



Semantic Segmentation follows three steps:

Classifying: Classifying a certain object in the image.

Localizing: Finding the object and drawing a bounding box around it.

Segmentation: Grouping the pixels in a localized image by creating a segmentation mask.

➤ **Cityscapes**



Cityscapes is a dataset consisting of diverse urban street scenes across 50 different cities at varying times of the year as well as ground truths for several vision tasks including semantic segmentation, instance level segmentation (TODO), and stereo pair disparity inference.

➤ **Vehicle with electric drive train**



In automotive engineering, the drivetrain is the group of components of a motor vehicle that deliver power to the drive wheels. This excludes the engine or motor that generates the power. In contrast, the powertrain is considered to include both the engine and/or motor(s) as well as the drivetrain. In marine applications, the drive shaft will drive a propeller rather than a drive axle, while the actual engine might be highly similar to an automotive engine.

➤ **Kinect**



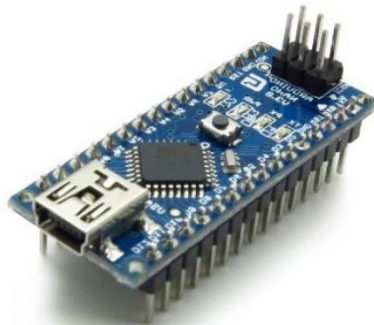
Kinect is a line of motion sensing input devices produced by Microsoft and first released in 2010. The devices generally contain RGB cameras, and infrared projectors and detectors that map depth through either structured light or time of flight calculations, which can in turn be used to perform real-time gesture recognition and body skeletal detection, among other capabilities. They also contain microphones that can be used for speech recognition and voice control.

➤ **Wireless router**



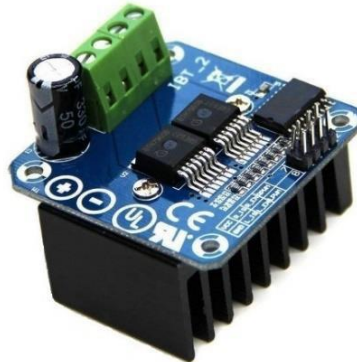
A wireless router is a device that performs the functions of a router and also includes the functions of a wireless access point. It is used to provide access to the Internet or a private computer network. Depending on the manufacturer and model, it can function in a wired local area network, in a wireless-only LAN, or in a mixed wired and wireless network.

➤ **Arduino Nano**



The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one.

➤ **Half bridge driver BTS7960**



The Double BTS7960 43A H-Bridge High-Power Stepper Motor Driver Module is; a fully integrated high current H bridge for motor drive applications using the BTS7960 high current half bridge. the BTS7960 is part of the NovalithICTM family. This containing one p-channel high side MOSFET and one n-channel low side MOSFET; with an integrated driver IC in one package. Due to the p-channel high side switch; the need for a charge pump is eliminated thus minimizing EMI.

➤ **Safety Button**



A laser safety button serves to interrupt the current supply manually in case of emergency. Safety buttons are mainly used in cases where there is no separate interlock interface available at the laser projector. As laser radiation can cause bad damages on eyes, skin and textiles it is important to take certain safety precautions

➤ **10k Potentiometer**



An adjustable potentiometer, 10K Potentiometer Module can open many interesting user interfaces. Turn the pot and the resistance changes. Connect VCC to an outer pin, GND to the other, and the center pin will have a voltage that varies from 0 to VCC depending on the rotation of the pot.

➤ **Jump Wire**



A jump wire is an electrical wire, or group of them in a cable, with a connector or pin at each end which is normally used to interconnect the components of a breadboard or other prototype or test circuit, internally or with other equipment or components, without soldering. Individual jump wires are fitted by inserting their "end connectors" into the slots provided in a breadboard, the header connector of a circuit board, or a piece of test equipment.

➤ **Wire**



A wire is a flexible strand of metal. Wire is commonly formed by drawing the metal through a hole in a die or draw plate. Wire gauges come in various standard sizes, as expressed in terms of a gauge number. Wires are used to bear mechanical loads, often in the form of wire rope. In electricity and telecommunications signals, a "wire" can refer to an electrical cable, which can contain a "solid core" of a single wire or

separate strands in stranded or braided forms. Usually cylindrical in geometry, wire can also be made in square, hexagonal, flattened rectangular, or other cross-sections, either for decorative purposes, or for technical purposes such as high-efficiency voice coils in loudspeakers. Edge-wound coil springs, such as the Slinky toy, are made of special flattened wire.

➤ **Ethernet**



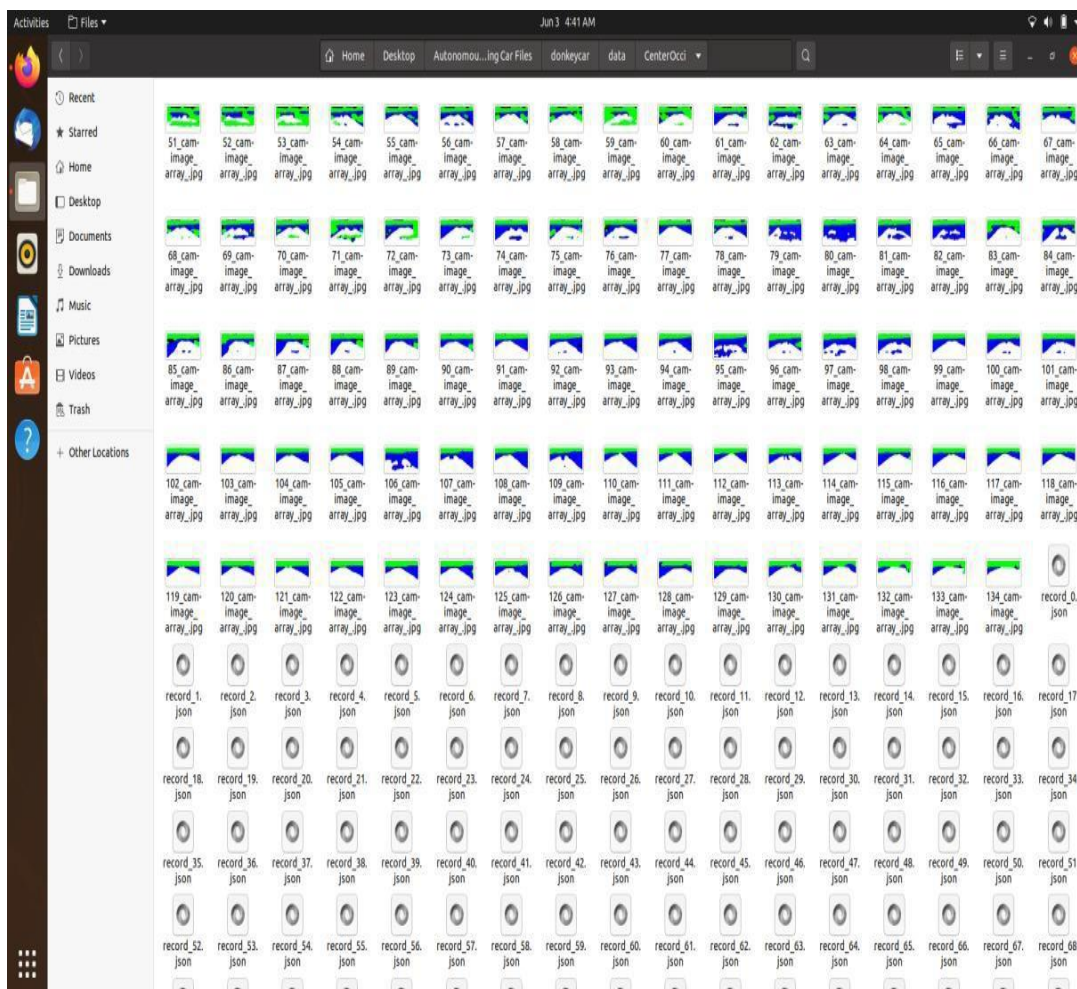
Ethernet is a family of wired computer networking technologies commonly used in local area networks (LAN), metropolitan area networks (MAN) and wide area networks (WAN). It was commercially introduced in 1980 and first standardized in 1983 as IEEE 802.3. Ethernet has since been refined to support higher bitrates, a greater number of nodes, and longer link distances, but retains much backward compatibility. Over time, Ethernet has largely replaced competing wired LAN technologies such as Token Ring, FDDI and ARCNET.

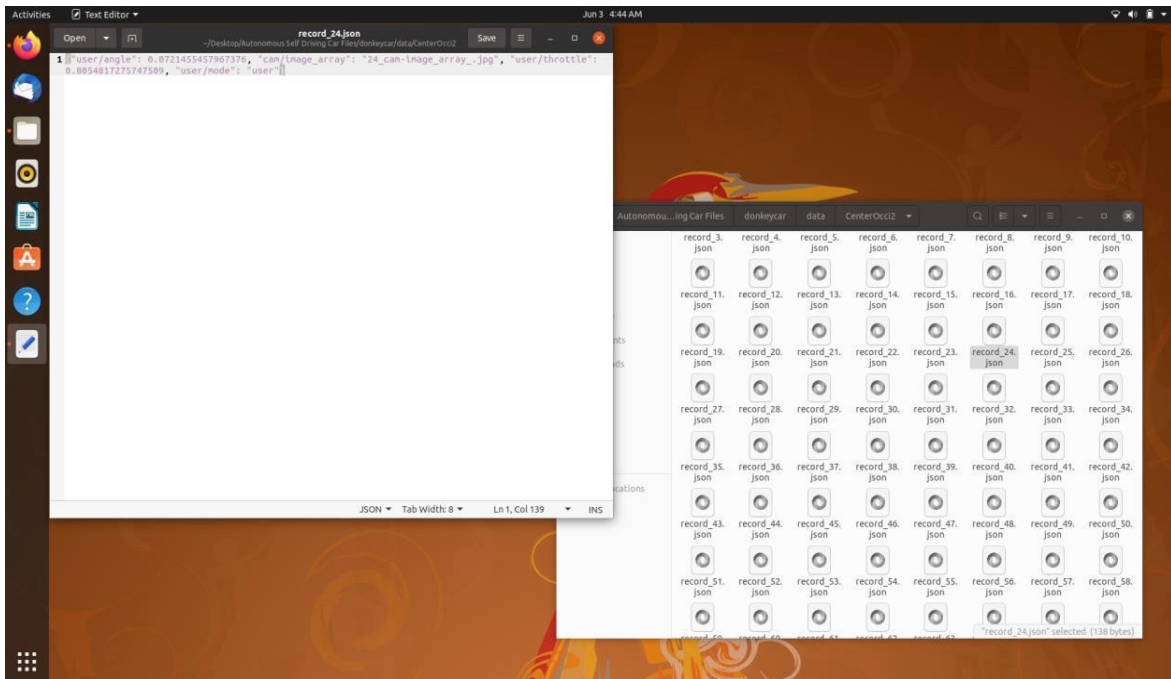
CHAPTER 5: DESIGN AND IMPLEMENTATION

5.1 Implementation

• Data Acquisition and Pre-processing

Self-driving vehicle system uses Cityscapes dataset to segment the images. For the drive model dataset is collected by driving the ATV on our own. We basically collect three types of data during data acquisition. They are segmented frame, throttle value and steering angle value for each respective segmented frame. The collected throttle value and steering angle is stored in JSON (JavaScript Object Notation) file, which is later synced by writing the frames and their respective values on to a tub.



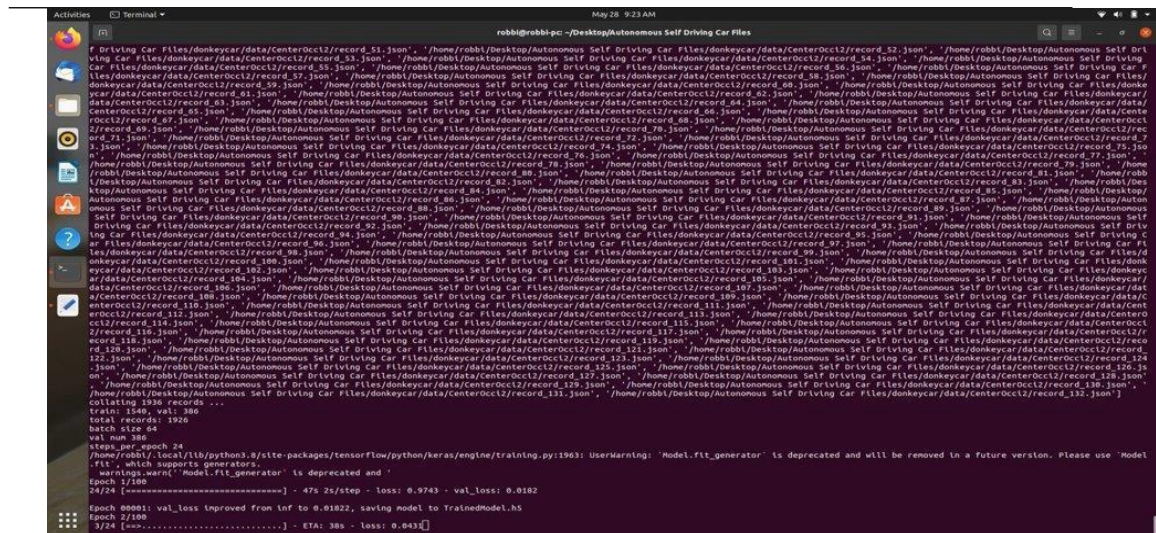


- **Data Exploration**

During data exploration the system should be able to extract the knowledge out of the data. We extract the width of the class named road and apply donkey car algorithm to train the drive model.

- **Training**

For training the drive model we collected around 1936 records or samples of data. Among which 1540 samples were chosen to be part of training the model and the remaining 386 samples were set for validating the trained model. The batch size is 64 and the steps per epoch is 24. While training we set the epoch value to 100 so that the training data was looped 100 times through the model resulting in decreased loss error.

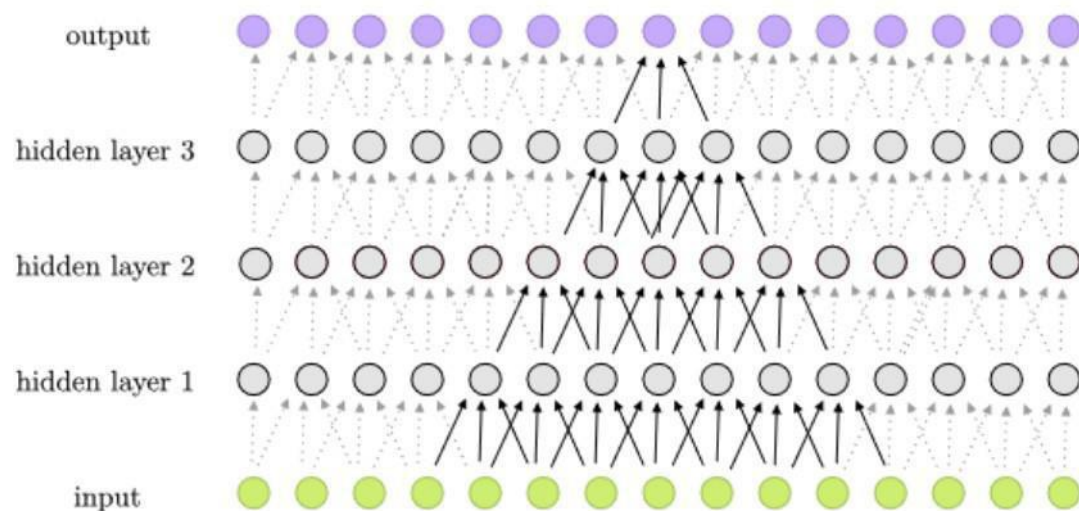


```
robbl@robbl-pc: ~/Desktop/Autonomous Self Driving Car Files
f Driving Car Files/donkeycar/data/centerOccl2/record_51.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_52.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_53.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_54.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_55.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_56.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_57.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_58.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_59.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_60.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_61.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_62.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_63.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_64.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_65.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_66.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_67.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_68.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_69.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_70.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_71.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_72.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_73.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_74.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_75.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_76.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_77.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_78.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_79.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_80.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_81.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_82.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_83.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_84.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_85.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_86.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_87.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_88.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_89.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_90.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_91.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_92.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_93.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_94.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_95.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_96.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_97.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_98.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_99.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_100.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_101.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_102.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_103.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_104.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_105.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_106.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_107.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_108.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_109.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_110.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_111.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_112.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_113.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_114.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_115.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_116.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_117.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_118.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_119.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_120.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_121.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_122.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_123.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_124.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_125.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_126.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_127.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_128.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_129.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_130.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_131.json', '/home/robbl/Desktop/Autonomous Self Driving Car Files/donkeycar/data/centerOccl2/record_132.json']
collating 1936 records ...
train: 1540, val: 396
total records: 1936
batch size: 64
val num 386
steps_per_epoch 24
/home/robbl/.local/lib/python3.8/site-packages/tensorflow/python/keras/engine/training.py:1963: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit' which supports generators.
warnings.warn('Model.fit_generator' is deprecated and
Epoch 1/100
24/24 [-----] - 47s 2s/step - loss: 0.9743 - val_loss: 0.9182
Epoch 2/100
9/24 [-----] - ETA: 38s - loss: 0.9434
```

5.2 Software Algorithm

This project is built using convolutional neural network and rectified linear activation function.

- Convolutional Neural Network (CNN)



A convolutional neural network (CNN) is a deep learning algorithm specifically designed to process image data. Convolutional neural networks are used in image recognition and processing.

The neural networks in a CNN are arranged similarly to the frontal lobe of the

human brain, a part of the brain responsible for processing visual stimuli.

The convolutional neural network consists of:

- A convolutional layer,
- A pooling layer,
- A fully connected input layer,
- A fully connected layer, and

Unlike multi-level perceptron, CN is sophisticated enough to apply filters that capture the spatial and temporal dependencies in an image. This way, CNN can provide higher accuracy, even with high-resolution images.

Convolutional neural networks stand out in their ability to learn by themselves. This algorithm works by assigning learnable weights and biases to an image to distinguish it from other images while retaining the features critical for achieving a good prediction.

Images are represented as a matrix of pixels with different planes. A grayscale image consists of one plane while an RGB image consists of three planes.

Now, in a convolutional neural network, there are multiple layers of artificial neurons, each with a mathematical function that calculates the sum of multiple inputs.

When an image is inputted in a CNN, the first layer extracts basic features of the image. These include the edges of the image. Once this layer extracts these basic features, the image moves to the next layer which detects more complex features such as combination edges.

As the image moves through multiple layers of the convolutional neural network, more complex features are detected. At the classification layer, the algorithm assigns classes in which an image is more likely to belong. This is where the neural network will assign an image as a person, cat, horse, etc.

The advantages of convolutional neural networks include:

They can detect important features without human supervision. It has the highest accuracy amongst image detection algorithms.

It is easy to understand and implement.

- **Rectified Linear Unit Activation Function (ReLU)**

The rectified linear activation function or ReLU is a non-linear function or piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.

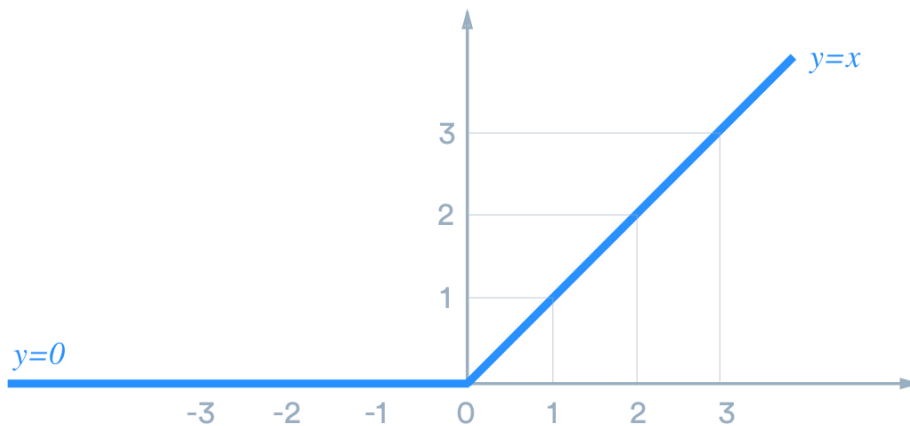
It is the most used activation function in neural networks, especially in Convolutional Neural Networks (CNNs) & Multilayer perceptron.

It is simple yet it is more effective than its predecessors like sigmoid or tanh.

Mathematically, it is expressed as:

$$f(x) = \max(0, x)$$

Graphically it is represented as,



ReLU is used in the hidden layers instead of Sigmoid or tanh as using sigmoid or tanh in the hidden layers leads to the infamous problem of "Vanishing Gradient". The "Vanishing Gradient" prevents the earlier layers from learning important information when the network is backpropagating. The sigmoid which is a logistic function is more preferable to be used in regression or binary classification related problems and that too only in the output layer, as the output of a sigmoid function ranges from 0 to 1. Also, Sigmoid and tanh saturate and have lesser sensitivity.

Some of the advantages of ReLU are:

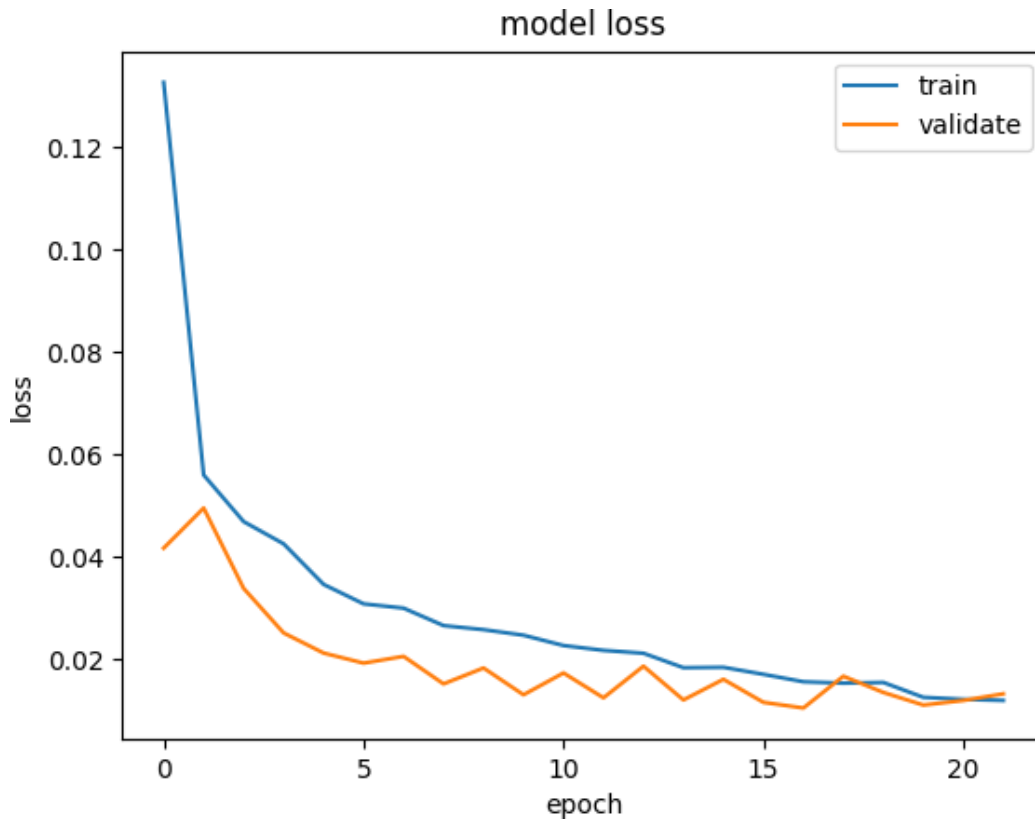
Simpler Computation: Derivative remains constant i.e. 1 for a positive input and thus reduces the time taken for the model to learn and in minimizing the errors.

5.3 Representational Sparsity: It is capable of outputting a true zero value.

Linearity: Linear activation functions are easier to optimize and allow for a smooth flow. So, it is best suited for supervised tasks on large sets of labelled data.

CHAPTER 6: RESULT AND DISCUSSION

We can observe the following from the above work -



Various measures, accuracy, specificity, sensitivity, and precision are derived from the above data set graph. So, with the help of the above measures a successful execution of fully autonomous self-driving vehicle has been done.

Rigorous training and testing of models in regular time interval with pre trained data models have been done to identify the probable fault that could occur and being rectified beforehand.

CHAPTER 7: CONCLUSIONS AND FUTURE SCOPE

7.1 CONCLUSION

Without manual breakdown into road or lane marker recognition, semantic abstraction, path planning, and control, we have empirically proved that CNNs can learn the whole job of lane and road following. To some extent, a tiny quantity of training data from a few hours of driving was enough to train the automobile to operate in a variety of settings.

From a limited training input, the CNN is able to acquire relevant road characteristics (steering alone). During training, the system learns to recognize the shape of a road, for example, without the use of explicit labelling. More work is needed to increase the network's resilience, establish techniques to validate the network's robustness, and improve network-internal processing step visualization.

More classes for segmentation and object detection can be added to the project to make it more advanced. to improve detection and sensing, more costly and high- quality hardware might be employed. To improve the model, more sensors and larger datasets might be employed. better microcontrollers and CPUs can boost performance dramatically. to acquire more exact or accurate outcomes, a very precise drivetrain and steering mechanism can be devised. as a result, a higher degree of automation and resilience may be attained. we also want to construct a self-drive physical model/vehicle soon.

7.2 REFERENCES

- [1] A brief introduction to OpenCV (2016 proceedings of the 35th international convention MIPRO)
Authors: I CULJAK, D ABRAM, T PRIBANIC, H DZAPO, M CIFREK
https://www.researchgate.net/publication/261424692_A_brief_introduction_to_OpenCV
- [2] Deployment of semantic segmentation network using Tensorrt Authors: Joohoon Lee (NVIDIA), Chethan Ningaraju (NVIDIA)
<https://on-demand.gputechconf.com/gtc-eu/2017/presentation/53021-joohoon-lee-deployment-of-semantic-segmentation-network-using-tensorrt.pdf>
- [3] Using multi-scale attention for semantic segmentation Authors: Andrew Tao, Karan Sapra, Bryan Catanzaro <https://arxiv.org/abs/2005.10821>
- [4] OpenCV for computer vision applications Authors: M Naveen Kumar, A Vadivel
https://www.researchgate.net/publication/301590571_OpenCV_for_Computer_Vision_Applications

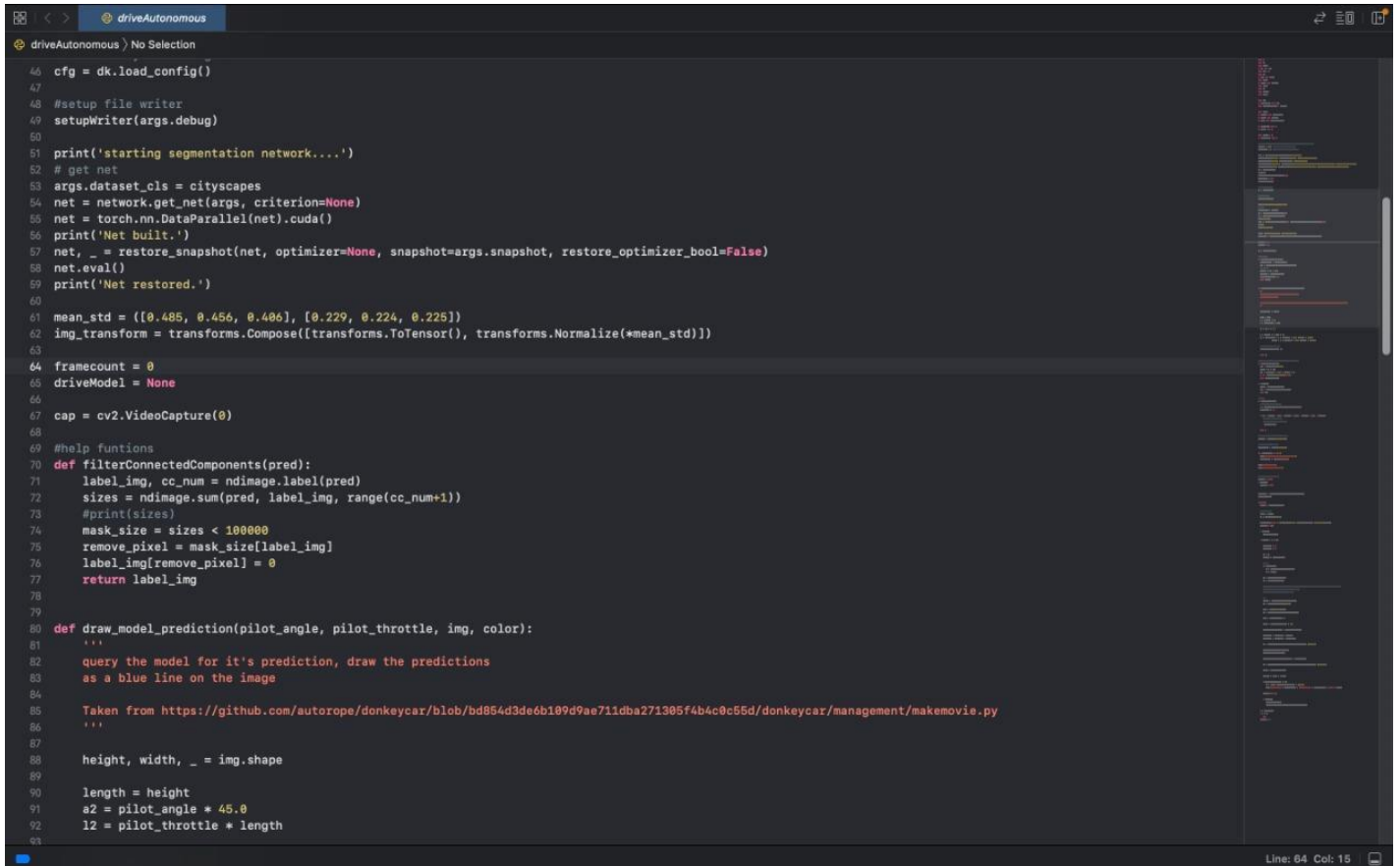
- [5] Hardware-software stack for a RC car for testing autonomous driving algorithms
Authors: K Shrivastava, M Inukonda, S Mittal
https://www.researchgate.net/publication/336411476_Hardware-Software_Stack_for_an_RC_car_for_testing_autonomous_driving_algorithms
- [6] Self-driving cars – the human side working paper (2017)
Authors: P Szikora, N Madarász
https://www.researchgate.net/publication/324095773_Self-driving_cars_-_The_human_side
- [7] Autonomous cars: past, present and future - a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology
Authors: K Bimbraw
https://www.researchgate.net/publication/283757446_autonomous_cars_past_present_and_future_-_a_review_of_the_developments_in_the_last_century_the_present_scenario_and_the_expected_future_of_autonomous_vehicle_technology
- [8] Self-driving and driver relaxing vehicle (2016)
Authors: Q Memon, M Ahmed, S Ali, AR Memon
<https://www.iosrjournals.org/iosr-jmce/papers/ncrime-2018/volume-3/7.%2047-56.pdf>
- [9] A survey of autonomous driving: common practices and emerging technologies
Authors: E Yurtsever, J Lambert, A Carballo, K Takeda <https://arxiv.org/abs/1906.05113>
- [10] Autonomous vehicles: the future of automobiles Authors: Mv Rajasekhar, Ak Jaiswal <https://ieeexplore.ieee.org/document/7386874>
- [11] Using Multi-Scale Attention for Semantic Segmentation | NVIDIA Developer Blog
[www.developers.nvidia.com/blog/UsingMulti-Scale Attention for Semantic](http://www.developers.nvidia.com/blog/UsingMulti-Scale%20Attention%20for%20Semantic)
- [12] Sematic-Segmentation toolkit documentation
[www.docs.nvidia.com/metropolis/TLT/tlt- userguide/text/semantic_segmentation/index.html](http://www.docs.nvidia.com/metropolis/TLT/tlt-userguide/text/semantic_segmentation/index.html)
- [13] Hierarchical Multi-Scale Attention for Semantic Segmentation Andrew Tao, Karan 21May2020]
<https://arxiv.org/abs/2005.10821>
- [14] Szalay, Z., Tettamanti, T., Esztergár-Kiss, D., Varga, I., & Bartolini, C. (2018).
- [15] Development of a test track for driverless cars: vehicle design, track configuration, and liability considerations. Periodica Polytechnica Transportation Engineering, 46(1), 29-35.
- [16] Lu, H., Bansal, G., & Kenney, J. (2019). U.S. Patent Application No. 15/921,404. Nilsson, J., Branstrom, M., Coelingh, E., & Fredriksson, J. (2017). Lane change maneuvers for automated vehicles. IEEE Transactions on Intelligent Transportation Systems, 18, 1087–1096.
- [17] UK Parliament. (2016). Vehicle technology and automation bill. Retrieved from https://publications.parliament.uk/pa/bills/cbill/2016-2017/0143/cbill_2016-20170143_en_2.html

- [18] Martinez, L. M., & Viegas, J. M. (2017). Assessing the impacts of deploying a shared self-driving urban mobility system: An agent-based model applied to the city of Lisbon, Portugal. *International Journal of Transportation Science and Technology*
- [19] Zhang, W., Guhathakurta, S., Fang, J., & Zhang, G. (2015). Exploring the impact of shared autonomous vehicles on urban parking demand: An agent-based simulation approach. *Sustainable Cities and Society*
- [1] Volvo. (2017). Autonomous driving. Retrieved from <https://www.volvocars.com/intl/about/our-innovationbrands/intellisafe/autonomous-driving>
- [2] Wadud, Z., MacKenzie, D., & Leiby, P. (2016). Help or hindrance? *Transportation Research Part A*, 86,
- [3] Snyder, R. (2016). Implications of autonomous vehicles: A planner's perspective. *Institute of Transportation Engineers Journal*, 86, 25–28.
- [4] Spyropoulou, I., Penttinen, M., Karlaftis, M., Vaa, T., & Golias, J. (2008). ITS solutions and accident risks: Prospective and limitations. *Transport Reviews*, 28, Piao, J., McDonald, M., Hounsell, N., Graindorge, M., Graindorge, T., & Malhene, N. (2016). Public views towards implementation of automated vehicles in urban areas. *Transportation Research* Proce Zohdy, I. A., & Rakha, H. A. (2016). Intersection management via vehicle connectivity: The intersection cooperative adaptive cruise control system concept. *Journal of Intelligent Transportation Systems*.

APPENDIX – I

SOURCE CODE

driveAutonomous.py



```
driveAutonomous > No Selection
46 cfg = dk.load_config()
47
48 #setup file writer
49 setupWriter(args.debug)
50
51 print('starting segmentation network...')
52 # get net
53 args.dataset_cls = cityscapes
54 net = network.get_net(args, criterion=None)
55 net = torch.nn.DataParallel(net).cuda()
56 print('Net built.')
57 net, _ = restore_snapshot(net, optimizer=None, snapshot=args.snapshot, restore_optimizer_bool=False)
58 net.eval()
59 print('Net restored.')
60
61 mean_std = ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
62 img_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(*mean_std)])
63
64 framecount = 0
65 driveModel = None
66
67 cap = cv2.VideoCapture(0)
68
69 #help funtions
70 def filterConnectedComponents(pred):
71     label_img, cc_num = ndimage.label(pred)
72     sizes = ndimage.sum(pred, label_img, range(cc_num+1))
73     #print(sizes)
74     mask_size = sizes < 100000
75     remove_pixel = mask_size[label_img]
76     label_img[remove_pixel] = 0
77     return label_img
78
79
80 def draw_model_prediction(pilot_angle, pilot_throttle, img, color):
81     '''
82     query the model for it's prediction, draw the predictions
83     as a blue line on the image
84
85     Taken from https://github.com/autorope/donkeycar/blob/bd854d3de6b109d9ae711dba271305f4b4c0c55d/donkeycar/management/makemovie.py
86     '''
87
88     height, width, _ = img.shape
89
90     length = height
91     a2 = pilot_angle * 45.0
92     l2 = pilot_throttle * length
93
```


<Fully Autonomous Self Driving Vehicle Using Machine Learning & Artificial Intelligence>

```
driveAutonomous ) No Selection
89
90     length = height
91     a2 = pilot_angle * 45.0
92     l2 = pilot_throttle * length
93
94     mid = width // 2 - 1
95
96     p2 = tuple((mid + 2, height - 1))
97     p22 = tuple((int(p2[0] + l2 * math.cos((a2 + 270.0) * (math.pi / 180.0))),
98                 int(p2[1] + l2 * math.sin((a2 + 270.0) * (math.pi / 180.0)))))
99
100    # user is green, pilot is blue
101    cv2.line(img, p2, p22, color, 2)
102
103    return img
104
105    #brighthen image because xbox kinect rgb cam is pretty shitty
106    def adjust_gamma(image, gamma):
107        image = cv2.resize(image, (512, 256))
108        invGamma = 1.0 / gamma
109        table = np.array([(i / 255.0) ** invGamma] * 255)
110        for i in np.arange(0, 256)].astype("uint8")
111        return cv2.LUT(image, table)
112
113    def get_video():
114        array, _ = freenect.sync_get_video()
115        array = cv2.cvtColor(array, cv2.COLOR_RGB2BGR)
116        return array
117
118    #load model
119    def loadModel(model_path):
120        #from donkeyCar file complete.py
121        kl = dk.utils.get_model_by_type(cfg.DEFAULT_MODEL_TYPE, cfg)
122        model_reload_cb = None
123
124        if '.h5' in model_path or '.uff' in model_path or '.tflite' in model_path or '.pkl' in model_path:
125            #when we have a .h5 extension
126            #load everything from the model file
127            kl.load(model_path)
128
129        return kl
130
131    #driveModel = loadModel('LinearRunFixed.h5')
132    driveModel = loadModel('TrainedModel.h5')
133
134    #connect to autonomous vehicle
135    connectionResult = connect('/dev/ttyUSB0')
136
```

```
driveAutonomous ) No Selection
1 import os
2 import sys
3 import argparse
4 from PIL import Image
5 import numpy as np
6 import cv2
7 from scipy import ndimage
8 import skimage
9 from skimage import morphology
10 import datetime
11 import json
12 import threading
13 import freenect
14
15 import torch
16 from torch.backends import cudnn
17 import torchvision.transforms as transforms
18
19 import network
20 from optimizer import restore_snapshot
21 from datasets import cityscapes
22 from config import assert_and_infer_cfg
23
24 from vehicleSerial import *
25 from control import *
26
27 import donkeycar as dk
28 from donkeycar.utils import *
29
30 #These offsets can be used when sensors where not placed correctly during training
31 steerOffset = -0.22; #is added to predicted steer angle
32 throttleOffset = 0; # is added to predicted throttle value
33
34 parser = argparse.ArgumentParser(description='Drive autonomous')
35 parser.add_argument('--debug', type=bool, default=False, help='enable debug messages')
36 parser.add_argument('--video', type=str, default='', help='path to video')
37 parser.add_argument('--snapshot', type=str, default='./pretrained_models/cityscapes_cv0_seresnext50_nosdcaug.pth', help='pre-trained checkpoint')
38 parser.add_argument('--arch', type=str, default='network.deepv3.DeepSRNX50V3PlusD_m1', help='network architecture used for inference')
39 args = parser.parse_args()
40 print(args)
41 assert_and_infer_cfg(args, train_mode=False)
42 cudnn.benchmark = False
43 torch.cuda.empty_cache()
44
45 #load donkeyCar config
46 cfg = dk.load_config()
47
48 #setup file writer
```

```
driveAutonomous > No Selection
183 #apply connected component method to find largest connected object. Use this to only train the road to the network
184 #smooth_pred = ndimage.gaussian_filter(pred,3.0)
185 #pred = filterConnectedComponents(smooth_pred)
186
187 #show
188 colorized = args.dataset_cls.colorize_mask(pred)
189 img = np.array(colorized.convert('RGB'))
190
191 kernel = np.ones((15,15),np.uint8)
192 img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
193
194 median = cv2.medianBlur(img, 17)
195
196 pred_img = median.astype(np.float32) / 255.0
197
198 steeringValue, throttleValue = driveModel.run(pred_img)
199
200 steeringValue = steeringValue + steerOffset
201 throttleValue = throttleValue + throttleOffset
202
203 img = draw_model_prediction(steeringValue, throttleValue, img, (0, 0, 255))
204
205 setDesiredSteeringAngle(steeringValue)
206 setDesiredThrottle(throttleValue)
207
208 currentSteeringValue, currentThrottleValue = getCurrentStatus()
209
210 img = draw_model_prediction(currentSteeringValue, currentThrottleValue, img, (0, 255, 255))
211
212 endTime = datetime.datetime.now()
213
214 elapsedTime = endTime - beginTime
215
216 if(elapsedTime.microseconds > 0.0):
217     fps = round(1 / (elapsedTime.microseconds * 10**-6),2)
218     print("steeringValue > " + str(steeringValue) + " throttleValue > " + str(throttleValue) + " fps: " + str(fps))
219
220 cv2.imshow("OUT", img)
221
222 if args.debug:
223     writeOutDebugImage(img)
224     writeAngleAndThrottle(steeringValue,throttleValue,framecount)
225
226 k = cv2.waitKey(1)
227 if k == 27:
228     break
229 framecount += 1
230
```

driveTrainData.py

```
driveTrainData > No Selection
50 # get data
51 mean_std = ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
52 img_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(*mean_std)])
53
54 #cap = cv2.VideoCapture(args.video)
55
56 framecount = 0
57
58 prevFrame = None
59
60
61 def draw_user_angle(pilot_angle, pilot_throttle, img):
62     """
63     query the model for it's prediction, draw the predictions
64     as a blue line on the image
65
66     Taken from https://github.com/autorope/donkeycar/blob/bd854d3de6b109d9ae711dba271385f4b4c8c55d/donkeycar/management/makemovie.py
67     """
68
69     height, width, _ = img.shape
70
71     length = height
72     a2 = pilot_angle * 45.0
73     l2 = pilot_throttle * length
74
75     mid = width // 2 - 1
76
77     p2 = tuple((mid + 2, height - 1))
78     p22 = tuple((int(p2[0] + l2 * math.cos((a2 + 270.0) * (math.pi / 180.0))),
79                 int(p2[1] + l2 * math.sin((a2 + 270.0) * (math.pi / 180.0)))))
80
81     # user is green, pilot is blue
82     cv2.line(img, p2, p22, (0, 0, 255), 2)
83
84     return img
85
86
87 def filterConnectedComponents(pred):
88     label_img, cc_num = ndimage.label(pred)
89     sizes = ndimage.sum(pred, label_img, range(cc_num+1))
90     #print(sizes)
91     mask_size = sizes < 50
92     remove_pixel = mask_size[label_img]
93     label_img[remove_pixel] = 0
94     return label_img
95
96 def map(x, in_min, in_max, out_min, out_max):
97     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
98
```

```
driveTrainData
driveTrainData No Selection
1 import os
2 import sys
3 import argparse
4 from PIL import Image
5 import numpy as np
6 import cv2
7 from scipy import ndimage
8 import skimage
9 from skimage import morphology
10 import datetime
11 import json
12 import freenect
13
14 import torch
15 from torch.backends import cudnn
16 import torchvision.transforms as transforms
17
18 import network
19 from optimizer import restore_snapshot
20 from datasets import cityscapes
21 from config import assert_and_infer_cfg
22
23 from vehicleSerial import *
24 from fileWriter import *
25
26 parser = argparse.ArgumentParser(description='drive training data')
27 parser.add_argument('--savevideo', type=str, default='', help='save incoming video')
28 parser.add_argument('--snapshot', type=str, default='./pretrained_models/cityscapes_cv0_seresnext50_nosdcaug.pth', help='pre-trained checkpoint')
29 parser.add_argument('--arch', type=str, default='network.deepv3.DeepSRNX50V3PlusD_m1', help='network architecture used for inference')
30 args = parser.parse_args()
31 print(args)
32 assert_and_infer_cfg(args, train_mode=False)
33 cudnn.benchmark = False
34 beginTime = datetime.datetime.now()
35 torch.cuda.empty_cache()
36
37 setupWriter(False)
38
39 print('starting segmentation network....')
40
41 # get net
42 args.dataset_cls = cityscapes
43 net = network.get_net(args, criterion=None)
44 net = torch.nn.DataParallel(net).cuda()
45 print('Net built.')
46 net, _ = restore_snapshot(net, optimizer=None, snapshot=args.snapshot, restore_optimizer_bool=False)
47 net.eval()
48 print('Net restored.')
```

```
driveTrainData
driveTrainData No Selection
96 def map(x, in_min, in_max, out_min, out_max):
97     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
98
99 def get_video():
100     array, _ = freenect.sync_get_video()
101     array = cv2.cvtColor(array, cv2.COLOR_RGB2BGR)
102     return array
103
104 #connect to autonomous vehicle
105 connectionResult = connect('/dev/ttyUSB0')
106
107 while connectionResult != "success":
108     print('arduino failed to connect but trying again...')
109     connectionResult = connect('/dev/ttyUSB0')
110
111 print("arduino connected!")
112
113 if args.savevideo:
114     out = cv2.VideoWriter("OcciRunBad" + ".avi", cv2.VideoWriter_fourcc('M','J','P','G'),10,(512,256))
115
116 cap = cv2.VideoCapture(0)
117
118 while True:
119     #img = get_video()
120     ret, img = cap.read()
121     if not ret:
122         break
123
124     img = cv2.resize(img,(512,256))
125     cv2.imshow("In", img)
126
127     if args.savevideo:
128         out.write(img)
129
130     if framecount % 5 == 0:
131         #reading steering and throttle value as fast as possible to make sure there is no delay
132         steeringValue, throttleValue = readValue()
133
134         if steeringValue != -1 or steeringValue != 1:
135             img2 = img
136             img_tensor = img_transform(img2)
137
138             # predict
139             with torch.no_grad():
140                 img2 = img_tensor.unsqueeze(0).cuda().cpu()
141                 pred = net(img2)
```

```
driveTrainData
driveTrainData No Selection
126 cv2.imshow('IN', img)
127
128 if args.savevideo:
129     out.write(img)
130
131 if framecount % 5 == 0:
132     #reading steering and throttle value as fast as possible to make sure there is no delay
133     steeringValue, throttleValue = readValue()
134
135     if steeringValue != -1 or steeringValue != 1:
136         img2 = img
137         img_tensor = img_transform(img2)
138
139         # predict
140         with torch.no_grad():
141             img2 = img_tensor.unsqueeze(0).cuda().cpu()
142             pred = net(img2)
143
144             pred = pred.cpu().numpy().squeeze()
145             pred = np.argmax(pred, axis=0)
146
147             colored = args.dataset_cls.colorize_mask(pred)
148             img = np.array(colored.convert('RGB'))
149             kernel = np.ones((15,15),np.uint8)
150             img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
151
152             median = cv2.medianBlur(img, 17)
153
154             #write data to disk for training
155             writeTrainData(steeringValue,throttleValue,median,int(framecount/5))
156
157             #endTime = datetime.datetime.now()
158
159             #elapsedTime = endTime - beginTime
160             #if(elapsedTime.microseconds > 8.0):
161             #    fps = round(1 / (elapsedTime.microseconds + 18*-6),2)
162             #    cv2.putText(img,"fps: " + str(fps),(10,90), cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,255),4,cv2.LINE_AA)
163
164             #print(str(steeringValue) + ", " + str(throttleValue))
165             median = draw_user_angle(steeringValue,throttleValue,median)
166
167             cv2.imshow("OUT", median)
168
169             k = cv2.waitKey(1)
170             if k == 27:
171                 break
172             framecount += 1
```

segmentationOnly.py

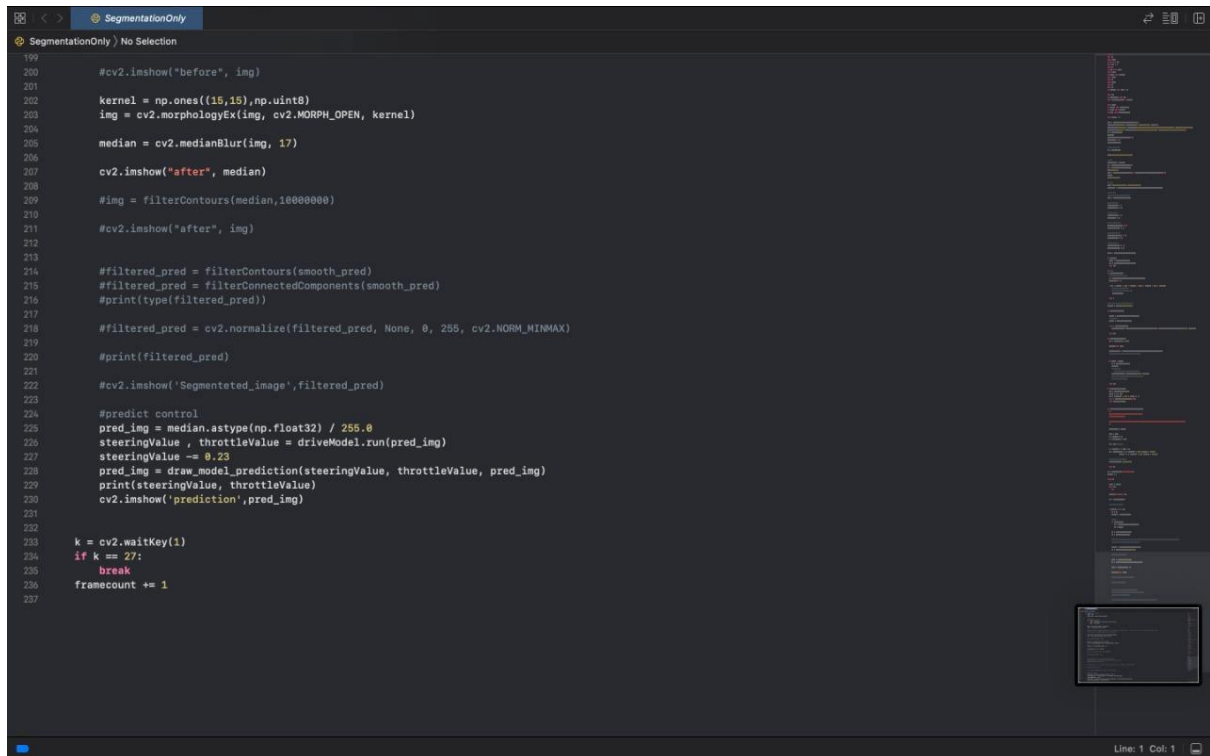
```
SegmentationOnly
SegmentationOnly No Selection
1 import os
2 import sys
3 import argparse
4 from PIL import Image
5 import numpy as np
6 import cv2
7 from scipy import ndimage
8 import skimage
9 from skimage import morphology
10 import datetime
11 import json
12 import freenect
13 import math
14 import time
15 from matplotlib import pyplot as plt
16
17 import torch
18 from torch.backends import cudnn
19 import torchvision.transforms as transforms
20
21 import network
22 from optimizer import restore_snapshot
23 from datasets import cityscapes
24 from config import assert_and_infer_cfg
25
26 import donkeycar as dk
27
28 parser = argparse.ArgumentParser(description='demo')
29 parser.add_argument('--video', type=str, default='', help='path to video', required=True)
30 parser.add_argument('--snapshot', type=str, default='./pretrained_models/cityscapes_cv0_seresnext50_nosdaug.pth', help='pre-trained checkpoint')
31 parser.add_argument('--arch', type=str, default='network.deeppv3.DeepSRNX50V3plus0_m1', help='network architecture used for inference')
32 args = parser.parse_args()
33 print(args)
34 assert_and_infer_cfg(args, train_mode=False)
35 cudnn.benchmark = False
36 torch.cuda.empty_cache()
37
38 #load donkeyCar config
39 cfg = dk.load_config()
40
41 print('starting segmentation network...')
42
43 # get net
44 args.dataset_cls = cityscapes
45 net = network.get_net(args, criterion=None)
46 net = torch.nn.DataParallel(net).cuda()
47 print('Net built.')
48 net._ = restore_snapshot(net, optimizer=None, snapshot=args.snapshot, restore_optimizer_bool=False)
```

```
SegmentationOnly
SegmentationOnly ) No Selection
48 net, _ = restore_snapshot(net, optimizer=None, snapshot=args.snapshot, restore_optimizer_bool=False)
49 net.eval()
50 print('Net restored.')
51
52 # get data
53 mean_std = ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
54 img_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(*mean_std)])
55
56 # Blob detector
57 # Setup SimpleBlobDetector parameters.
58 params = cv2.SimpleBlobDetector_Params()
59
60 # Change thresholds
61 params.minThreshold = 10
62 params.maxThreshold = 1000
63
64 # Filter by Area.
65 params.filterByArea = True
66 params.minArea = 0.01
67
68 # Filter by Circularity
69 params.filterByCircularity = False
70 params.minCircularity = 0.1
71
72 # Filter by Convexity
73 params.filterByConvexity = False
74 params.minConvexity = 0.87
75
76 # Filter by Inertia
77 params.filterByInertia = False
78 params.minInertiaRatio = 0.02
79
80 detector = cv2.SimpleBlobDetector_create(params)
81
82 def get_video():
83     array, _ = freenect.sync_get_video()
84     array = cv2.cvtColor(array, cv2.COLOR_RGB2BGR)
85     return array
86
87 #load model
88 def loadModel(model_path):
89     #from donkeyCar file complete.py
90     kl = dk.utils.get_model_by_type(cfg.DEFAULT_MODEL_TYPE, cfg)
91     model_reload_cb = None
92
93     if '.h5' in model_path or '.uff' in model_path or '.tflite' in model_path or '.pkl' in model_path:
94         #when we have a .h5 extension
95         #load everything from the model file
96         kl.load(model_path)
97
98     return kl
99
100 driveModel = loadModel('LinearRunFixed.h5')
101 driveModel = loadModel('TrainedModel.h5')
102
103 def filterBlobs(image, size):
104     imageGray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
105     # Detect blobs.
106     keypoints = detector.detect(imageGray)
107
108     for x in range(1, len(keypoints)):
109         image = cv2.circle(image, (np.int(keypoints[x].pt[0]), np.int(keypoints[x].pt[1])), radius=np.int(keypoints[x].size), color=(0, 255, 0), thickness=-1)
110
111     return image
112
113 def filterContours(image, size):
114     edged = cv2.Canny(image, 175, 200)
115
116     cv2.imshow("canny", edged)
117
118     contours, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
119     #cv2.drawContours(image, contours, -1, (0, 0, 255), 100)
120
121     for contour in contours:
122         area = cv2.contourArea(contour)
123         print(area)
124         #if area < size:
125             #cv2.drawContours(image, [contour], 0, 255, -1)
126         cv2.fillPoly(image, pts=[contour], color=(0, 0, 255), maxLevel=2)
127         #cv2.drawContours(image, [approx], -1, (0, 0, 255), 3)
128         #hull = cv2.convexHull(cnt)
129
130     return image
131
132 def adjust_gamma(image, gamma):
133     image = cv2.resize(image, (612, 256))
134     invGamma = 1.0 / gamma
135     table = np.array([(i / 255.0) ** invGamma * 255
136                       for i in np.arange(0, 256)].astype('uint8'))
137     return cv2.LUT(image, table)
```

```
SegmentationOnly
SegmentationOnly ) No Selection
93 if '.h5' in model_path or '.uff' in model_path or '.tflite' in model_path or '.pkl' in model_path:
94     #when we have a .h5 extension
95     #load everything from the model file
96     kl.load(model_path)
97
98     return kl
99
100 driveModel = loadModel('LinearRunFixed.h5')
101 driveModel = loadModel('TrainedModel.h5')
102
103 def filterBlobs(image, size):
104     imageGray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
105     # Detect blobs.
106     keypoints = detector.detect(imageGray)
107
108     for x in range(1, len(keypoints)):
109         image = cv2.circle(image, (np.int(keypoints[x].pt[0]), np.int(keypoints[x].pt[1])), radius=np.int(keypoints[x].size), color=(0, 255, 0), thickness=-1)
110
111     return image
112
113 def filterContours(image, size):
114     edged = cv2.Canny(image, 175, 200)
115
116     cv2.imshow("canny", edged)
117
118     contours, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
119     #cv2.drawContours(image, contours, -1, (0, 0, 255), 100)
120
121     for contour in contours:
122         area = cv2.contourArea(contour)
123         print(area)
124         #if area < size:
125             #cv2.drawContours(image, [contour], 0, 255, -1)
126         cv2.fillPoly(image, pts=[contour], color=(0, 0, 255), maxLevel=2)
127         #cv2.drawContours(image, [approx], -1, (0, 0, 255), 3)
128         #hull = cv2.convexHull(cnt)
129
130     return image
131
132 def adjust_gamma(image, gamma):
133     image = cv2.resize(image, (612, 256))
134     invGamma = 1.0 / gamma
135     table = np.array([(i / 255.0) ** invGamma * 255
136                       for i in np.arange(0, 256)].astype('uint8'))
137     return cv2.LUT(image, table)
```

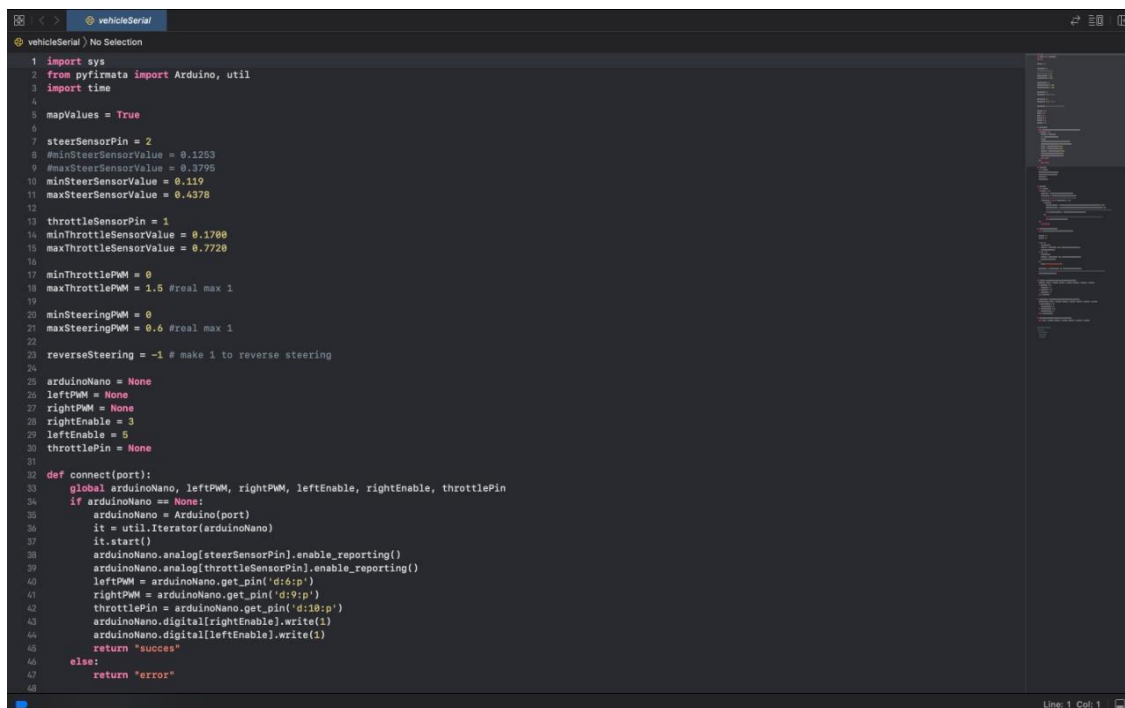
```
SegmentationOnly ) No Selection
def draw_model_prediction(pilot_angle, pilot_throttle, img):
    """
    query the model for it's prediction, draw the predictions
    as a blue line on the image
    Taken from https://github.com/autorope/donkeycar/blob/bd854d3de6b109d9ae711dba271905f4b4c0c55d/donkeycar/management/makemovie.py
    """
    height, width, _ = img.shape
    length = height
    a2 = pilot_angle * 45.0
    l2 = pilot_throttle * length
    mid = width // 2 - 1
    p2 = tuple((mid + 2, height - 1))
    p22 = tuple((int(p2[0] + l2 * math.cos((a2 + 270.0) * (math.pi / 180.0))),
    int(p2[1] + l2 * math.sin((a2 + 270.0) * (math.pi / 180.0)))))
    # user is green, pilot is blue
    cv2.line(img, p2, p22, (0, 0, 255), 2)
    return img
cap = cv2.VideoCapture("videoplayback2.avi")
framecount = 0
while True:
    ret, img = cap.read()
    if not ret:
        break
    cv2.imshow("Input_image", img)
    img = adjust_gamma(img,1)
    #cv2.imshow("gamma", img)
    if framecount % 1 == 0:
        img2 = img
        img_tensor = img_transform(img2)
        # predict
        with torch.no_grad():
            img2 = img_tensor.unsqueeze(0).cuda().cpu()
            pred = net(img2)
            pred = pred.cpu().numpy().squeeze()
            pred = np.argmax(pred, axis=0)
            #apply connected component method to find largest connected object. Use this to only train the road to the network
            smooth_pred = ndimage.gaussian_filter(pred,3.0)
            colorized = args.dataset_cls.colorize_mask(pred)
            img = np.array(colorized.convert('RGB'))
            #cv2.imshow("before", img)
            kernel = np.ones((15,15),np.uint8)
            img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
            median = cv2.medianBlur(img, 17)
            cv2.imshow("after", median)
            #img = filterContours(median,10000000)
            #cv2.imshow("after", img)
            #filtered_pred = filterContours(smooth_pred)
            #filtered_pred = filterConnectedComponents(smooth_pred)
            #print(type(filtered_pred))
            #filtered_pred = cv2.normalize(filtered_pred, None, 0, 255, cv2.NORM_MINMAX)
            #print(filtered_pred)
            #cv2.imshow('Segmentated_image',filtered_pred)
            #predict control
            pred_img = median.astype(np.float32) / 255.0
            steeringValue, throttleValue = driveModel.run(pred_img)
            steeringValue -= 0.23
            pred_img = draw_model_prediction(steeringValue, throttleValue, pred_img)
            print(steeringValue, throttleValue)
```

```
SegmentationOnly ) No Selection
if framecount % 1 == 0:
    img2 = img
    img_tensor = img_transform(img2)
    # predict
    with torch.no_grad():
        img2 = img_tensor.unsqueeze(0).cuda().cpu()
        pred = net(img2)
        pred = pred.cpu().numpy().squeeze()
        pred = np.argmax(pred, axis=0)
        #apply connected component method to find largest connected object. Use this to only train the road to the network
        smooth_pred = ndimage.gaussian_filter(pred,3.0)
        colorized = args.dataset_cls.colorize_mask(pred)
        img = np.array(colorized.convert('RGB'))
        #cv2.imshow("before", img)
        kernel = np.ones((15,15),np.uint8)
        img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
        median = cv2.medianBlur(img, 17)
        cv2.imshow("after", median)
        #img = filterContours(median,10000000)
        #cv2.imshow("after", img)
        #filtered_pred = filterContours(smooth_pred)
        #filtered_pred = filterConnectedComponents(smooth_pred)
        #print(type(filtered_pred))
        #filtered_pred = cv2.normalize(filtered_pred, None, 0, 255, cv2.NORM_MINMAX)
        #print(filtered_pred)
        #cv2.imshow('Segmentated_image',filtered_pred)
        #predict control
        pred_img = median.astype(np.float32) / 255.0
        steeringValue, throttleValue = driveModel.run(pred_img)
        steeringValue -= 0.23
        pred_img = draw_model_prediction(steeringValue, throttleValue, pred_img)
        print(steeringValue, throttleValue)
```

```
199
200 #cv2.imshow("before", img)
201
202 kernel = np.ones((15,15),np.uint8)
203 img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
204
205 median = cv2.medianBlur(img, 17)
206
207 cv2.imshow("after", median)
208
209 #img = filterContours(median,10000000)
210
211 #cv2.imshow("after", img)
212
213
214 #filtered_pred = filterContours(smooth_pred)
215 #filtered_pred = filterConnectedComponents(smooth_pred)
216 #print(type(filtered_pred))
217
218 #filtered_pred = cv2.normalize(filtered_pred, None, 0, 255, cv2.NORM_MINMAX)
219
220 #print(filtered_pred)
221
222 #cv2.imshow('Segmenteted_image',filtered_pred)
223
224 #predict control
225 pred_img = median.astype(np.float32) / 255.0
226 steeringValue , throttleValue = driveModel.run(pred_img)
227 steeringValue -= 0.23
228 pred_img = draw_model_prediction(steeringValue, throttleValue, pred_img)
229 print(steeringValue, throttleValue)
230 cv2.imshow('prediction',pred_img)
231
232
233 k = cv2.waitKey(1)
234 if k == 27:
235     break
236 framecount += 1
237
```

vehicleSerial.py



```
1 import sys
2 from pyfirmata import Arduino, util
3 import time
4
5 mapValues = True
6
7 steerSensorPin = 2
8 #minSteerSensorValue = 0.1253
9 #maxSteerSensorValue = 0.3796
10 minSteerSensorValue = 0.119
11 maxSteerSensorValue = 0.4378
12
13 throttleSensorPin = 1
14 minThrottleSensorValue = 0.1700
15 maxThrottleSensorValue = 0.7720
16
17 minThrottlePWM = 0
18 maxThrottlePWM = 1.5 #real max 1
19
20 minSteeringPWM = 0
21 maxSteeringPWM = 0.6 #real max 1
22
23 reverseSteering = -1 # make 1 to reverse steering
24
25 arduinoNano = None
26 leftPWM = None
27 rightPWM = None
28 rightEnable = 3
29 leftEnable = 5
30 throttlePin = None
31
32 def connect(port):
33     global arduinoNano, leftPWM, rightPWM, leftEnable, rightEnable, throttlePin
34     if arduinoNano == None:
35         arduinoNano = Arduino(port)
36         it = util.Iterator(arduinoNano)
37         it.start()
38         arduinoNano.analog[steerSensorPin].enable_reporting()
39         arduinoNano.analog[throttleSensorPin].enable_reporting()
40         leftPWM = arduinoNano.get_pin('d:6:p')
41         rightPWM = arduinoNano.get_pin('d:9:p')
42         throttlePin = arduinoNano.get_pin('d:10:p')
43         arduinoNano.digital[rightEnable].write(1)
44         arduinoNano.digital[leftEnable].write(1)
45         return "success"
46     else:
47         return "error"
48
```

```
48
49 def disconnect():
50     global arduinoNano
51     board.digital[rightEnable].write(0)
52     board.digital[leftEnable].write(0)
53     writeValue(0,0)
54     arduinoNano.exit()
55
56
57 def readValue():
58     global arduinoNano
59     if arduinoNano != None:
60         steerPosition = arduinoNano.analog[steerSensorPin].read()
61         throttlePosition = arduinoNano.analog[throttleSensorPin].read()
62         #TODO translate value from min max to -1 1 and 0 1 for throttle
63         if(steerPosition != None and throttlePosition != None):
64             if(mapValues):
65                 mappedSteeringPosition = mapSteerPisitition(steerPosition, minSteerSensorValue, maxSteerSensorValue, -1, 1)
66                 mappedThrottlePosition = mapThrottle(throttlePosition, minThrottleSensorValue, maxThrottleSensorValue, 0, 1)
67                 #print("readed steeringposition > " + str(mappedSteeringPosition) + " readed throttle " + str(mappedThrottlePosition))
68                 return((mappedSteeringPosition * reverseSteering,mappedThrottlePosition))
69             else:
70                 #print("readed steeringposition > " + str(steerPosition) + " readed throttle " + str(throttlePosition))
71                 return((steerPosition,throttlePosition))
72         else:
73             return("error")
74
75 def writeValue(left,right,throttle):
76     global leftPWM, rightPWM, leftEnable, rightEnable, throttlePin
77
78     mappedLeft = 0
79     mappedRight = 0
80
81     if left > 0:
82         rightPWM.write(0)
83         mappedLeft = round(map(left, -1, 1, minSteeringPWM, maxSteeringPWM),3)
84         leftPWM.write(mappedLeft)
85     elif right > 0:
86         leftPWM.write(0)
87         mappedRight = round(map(right, -1, 1, minSteeringPWM, maxSteeringPWM),3)
88         rightPWM.write(mappedRight)
89     else:
90         print("left right commando incorrect")
91
92     mappedThrottle = round(map(throttle, 0, 1, minThrottlePWM, maxThrottlePWM),3)
93     #print("writing values left> " + str(mappedLeft) + " right> " + str(mappedRight) + " throttle> " + str(mappedThrottle))
94     throttlePin.write(mappedThrottle)
95
```

```
96
97 def mapThrottle (value, fromSource, toSource, fromTarget, toTarget):
98     mappedThottle = (value - fromSource) / (toSource - fromSource) * (toTarget - fromTarget) + fromTarget
99     if mappedThottle > 1:
100         mappedThottle = 1
101     elif mappedThottle < 0:
102         mappedThottle = 0
103     return mappedThottle
104
105 def mapSteerPisitition (value, fromSource, toSource, fromTarget, toTarget):
106     mappedSteerPisitition = (value - fromSource) / (toSource - fromSource) * (toTarget - fromTarget) + fromTarget
107     if mappedSteerPisitition > 1:
108         mappedSteerPisitition = 1
109     elif mappedSteerPisitition < -1:
110         mappedSteerPisitition = -1
111     return mappedSteerPisitition
112
113 def map(value, fromSource, toSource, fromTarget, toTarget):
114     return (value - fromSource) / (toSource - fromSource) * (toTarget - fromTarget) + fromTarget
115
116
117 #connect('/dev/ttyUSB0')
118 #while(True):
119     #writeValue(1,0)
120     #time.sleep(10)
121     #disconnect()
122
```