A MAJOR PROJECT REPORT ON

**“Autopilot Simulation for Efficient Development of Self Driving Cars”**

In partial fulfillment of the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**

Submitted by

**D.Pranitha Jsvs Jogendra Kapgate Manish Yadav**

**17911A1208 17911A1218 17911A1216**

**Under the Esteemed Guidance of**

**Mrs. Laxmi Soni**

# **Assoc. Professor**



**DEPARTMENT OF INFORMATION TECHNOLOGY**

# **VIDYA JYOTHI INSTITUTE OF TECHNOLOGY**

(Accredited by NBA, Approved by AICTE, Affiliated to JNTU Hyderabad) Aziz Nagar Gate, C.B.Post, Chilkur Road, Hyderabad – 500075

2020 - 2021

**VIDYA JYOTHI INSTITUTE OF TECHNOLOGY**

(Accredited by NBA, Approved by AICTE, Affiliated to JNTU Hyderabad)

Aziz Nagar Gate, C.B.Post, Chilkur Road, Hyderabad - 500075

## DEPARTMENT OF INFORMATION TECHNOLOGY



**CERTIFICATE**

This is to certify that the Project Report on “**Autopilot Simulation For Efficient Development Of Self Driving Cars ”** is a bonafide work by Jsvs **Jogendra Kapgate (17911A1218), D.Pranitha (17911A1208), and Manish Yadav(17911A1216)** in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in **“INFORMATION TECHNOLOGY”** JNTU Hyderabad during the year **2020 - 2021.**

**Project Guide Head of the department**

Mrs. Laxmi Soni Mr. B. Srinivasulu

Assistant professor Professor

**External Examiner**

**VIDYA JYOTHI INSTITUTE OF TECHNOLOGY**

(Accredited by NBA, Approved by AICTE, Affiliated to JNTU Hyderabad)

Aziz Nagar Gate, C.B.Post, Chilkur Road, Hyderabad - 500075

### 2020 - 2021



## DECLARATION

We, **Jsvs Jogendra Kapgate (17911A1218), D.Pranitha(17911A1208), Manish Yadav(17911A1216)**hereby declare that Project Report entitled “**Autopilot Simulation For Efficient Development Of Self Driving Cars**”, is submitted in the partial fulfillment of the requirement for the award of **Bachelor of Technology** in Information Technology to **VidyaJyothi Institute of Technology**, affiliated to JNTU - Hyderabad, is an authentic work and has not been submitted to any other university or institute for the degree.

**D.Pranitha(17911A1208)**

**Manish Yadav (17911A1216)**

**Jsvs Jogendra Kapgate(17911A1218)**

**ACKNOWLEDGEMENT**

It is a great pleasure to express our deepest sense of gratitude and indebtedness to our

internal guide **Mrs.Laxmi Soni,** Associate Professor, Department of IT, VJIT, for having been a source of constant inspiration, precious guidance and generous assistance during the project work. We deem it as a privilege to have worked under his guidance. Without his close monitoring and valuable suggestions this work wouldn’t have taken this shape. We feel that this help is un-substitutable and unforgettable.

We wish to express our sincere thanks to **Dr. P. Venugopal Reddy,** Director VJIT, for providing the college facilities for the completion of the project. We are profoundly thankful to **Mr.B.Srinivasulu,** Professor and Head of Department of IT, for his cooperation and encouragement. Finally, we thank all the faculty members, supporting staff of IT Department and friends for their kind co-operation and valuable help for completing the project.

TEAM MEMBERS

**D.Pranitha(17911A1208)**

**Manish Yadav (17911A1216)**

**Jsvs Jogendra Kapgate(17911A1218)**

**INDEX**

**Table of Contents Page No.**

**ABSTRACT i**

1. **INTRODUCTION 2**

**1.1 Motivation 3**

**1.2 Existing System 3**

**1.3 Proposed System 3**

**1.4 System Requirements 4**

**1.5 Algorithm 5**

**2. LITERATURE SURVEY 9**

**2.1 Methodology & Data Collection Procedure 11**

**2.2 Introduction of Python 12**

**2.3 History of Python 14**

**2.4 Python Features 14**

**2.5 20 Python libraries 16**

**2.6 Python class and objects 18**

**2.7 Python XML Parser 23**

**2.8 Python Web Frameworks 24**

**3. SYSTEM STUDY 26**

**3.1 Feasibility Study 26**

**3.2 Economical Feasibility 26**

**3.3 Technical Feasibility 26**

**3.4 Social Feasibility 27**

**4. SYSTEM DESIGN 28**

**4.1 UML Diagrams 28**

**4.1.1 Use Case Diagram 29**

**4.1.2 Class Diagram 30**

**4.1.3 Activity Diagram 31**

**4.1.4 Collaboration Diagram 32**

**4.1.5 Sequence Diagram 33**

**4.1.6 Deployment Diagram 34**

**4.1.7 Component Diagram 34**

**5. DESIGN METHODOLOGY OF AUTOPILOT SIMULATION 35**

**5.1 Overview of the DAVE-2 System 35**

**5.2 Data Collection 37**

**5.3 Network Architecture 38**

**5.4 Data Selection 39**

**5.5 Augmentation 39**

**5.6 Simulation 39**

**5.7 Software Development Life Cycle 42**

**5.8 SDLC Methodologies 43**

**6. TESTING AND RESULT ANALYSIS 46**

**6.1 Types of Tests 46**

**6.2 System Test 47**

**6.3 White Box Testing 47**

**6.4 Black Box Testing 48**

**6.5 Testing 48**

**6.6 Evaluation 48**

**6.7 Results 49**

**6.8 Outputs 50**

**7. APPENDIX 52**

**8. CONCLUSION AND FUTURE WORK 63**

**9. BIBLIOGRAPHY 64**

|  |  |  |
| --- | --- | --- |
|  | **LIST OF FIGURES** |  |
| **Figure 1.5.1&1.5.2** | **Convolution Operation** | **5** |
| **Figure 1.5.3** | **CNN Scan Images** | **6** |
| **Figure 1.5.4&1.5.5** | **CNN Architecture** | **7** |
| **Figure 1.5.6** | **Layers of CNN** | **8** |
| **Figure**  **2.1** | **Sixty five years of automotive baby steps** | **10** |
| **Figure 4.1.1** | **Use Case Diagram** | **29** |
| **Figure 4.1.2** | **Class Diagram** | **30** |
| **Figure 4.1.3** | **Activity Diagram** | **31** |
| **Figure 4.1.4** | **Collaboration Diagram** | **32** |
| **Figure 4.1.5** | **Sequence Diagram** | **33** |
| **Figure 4.1.6** | **Deployment Diagram** | **34** |
| **Figure 4.1.7** | **Component Diagram** | **34** |
| **Figure 5.1.1** | **High-level view of data collection system** | **35** |
| **Figure 5.1.2** | **Training the neural network** | **36** |
| **Figure 5.1.3** | **Trained network used for steering commands** | **37** |
| **Figure 5.3.1** | **CNN architecture** | **38** |
| **Figure 5.6.1** | **Block-diagram of simulator** | **40** |
| **Figure 5.6.2** | **Loss plot** | **41** |
| **Figure 5.7.1** | **SDLC** | **42** |
| **Figure 5.8.1** | **Spiral Model** | **44** |
| **Figure 6.7.1** | **How the CNN “sees” an unpaved road** | **49** |
| **Figure 6.8.1,6.8.2,**  **6.8.3&6.8.4** | **Autopilot Simulator** | **50** |

#### ABSTRACT

We trained a convolutional neural network (CNN) to map raw pixels from a single

Front-facing camera directly to steering commands. This end-to-end approach

Proved surprisingly powerful. With minimum training data from humans the system

Learns to drive in traffic on local roads with or without lane markings and on

Highways. It also operates in areas with unclear visual guidance such as in parking

Lots and on unpaved roads.

The system automatically learns internal representations of the necessary processing

Steps such as detecting useful road features with only the human steering angle

As the training signal. We never explicitly trained it to detect, for example, the outline

Of roads.

Compared to explicit decomposition of the problem, such as lane marking detection,

Path planning, and control, our end-to-end system optimizes all processing

Steps simultaneously. We argue that this will eventually lead to better performance

And smaller systems. Better performance will result because the internal

Components self-optimize to maximize overall system performance, instead of optimizing

Human-selected intermediate criteria, e. g., lane detection. Such criteria

Understandably are selected for ease of human interpretation which doesn’t automatically

Guarantee maximum system performance. Smaller networks are possible

Because the system learns to solve the problem with the minimal number of

Processing steps.

i

#### INTRODUCTION

CNNs have revolutionized pattern recognition. Prior to the widespread adoption of CNNs

most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

While CNNs with learned features have been in commercial use for over twenty years, their adoption has exploded in the last few years because of two recent developments. First, large, labelled data sets such as the Large Scale Visual Recognition Challenge (ILSVRC) have become available for training and validation. Second, CNN learning algorithms have been implemented on the massively parallel graphics processing units (GPUs) which tremendously accelerate learning and inference.

In this paper, we describe a CNN that goes beyond pattern recognition. It learns the entire processing pipeline needed to steer an automobile. The groundwork for this project was done over 10 years ago in a Defense Advanced Research Projects Agency (DARPA) seedling project known as DARPA Autonomous Vehicle (DAVE) in which a sub-scale radio control (RC) car drove through a junk-filled alley way. DAVE was trained on hours of human driving in similar, but not identical environments. The training data included video from two cameras coupled with left and right steering commands from a human operator.

In many ways, DAVE-2 was inspired by the pioneering work of Pomerleau who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. It demonstrated that an end-toend trained neural network can indeed steer a car on public roads. Our work differs in that 25 years of advances let us apply far more data and computational power to the task. In addition, our experience with CNNs lets us make use of this powerful technology. (ALVINN used a fully-connected network which is tiny by today’s standard.)

While DAVE demonstrated the potential of end-to-end learning, and indeed was used to justify starting the DARPA Learning Applied to Ground Robots (LAGR) program, DAVE’s performance was not sufficiently reliable to provide a full alternative to more modular approaches to off-road driving. DAVE’s mean distance between crashes was about 20 meters in complex environments. Nine months ago, a new effort was started at NVIDIA that sought to build on DAVE and create a robust system for driving on public roads. The primary motivation for this work is to avoid the need to recognize specific human-designated features, such as lane markings, guard rails, or other cars, and to avoid having to create a collection of “if, then, else” rules, based on observation of these features. This paper describes preliminary results of this new effort.

##### 1.1Motivation

Deep Learning is a very rampant field with so many applications coming out day by day. The application of autopilot simulation is extensive and significant, for example, the realization of human computer interaction. The development of autopilot simulation will help to test the trained model before its tested on road minimizing the chances of damage due to ill trained model and gives plenty of room to optimize the model.

###### 1.2 Existing System

Existing system involves the use of LiDAR and sonar system which can sense the environment around it using light and sound waves.

The main drawback of this system is that it can only detect the surrounding for a few meters.

The system is incapable of processing various road signs and lacks the support for computer vision. Lack of visual data makes it more challenging to train a model for such system.

###### 1.3 Proposed System

Proposed system involves computer aided vision of the vehicle which is much more efficient and safer. Capabilities of the vehicle can be improved constantly by collecting visual data and calibrating the vehicle according to the environment. Presence of visual data makes it much easier to train and improve the model. The system is capable of processing various road signs.

###### 1.4 System Requirements

1.4.1 Software Requirements

Operating System : Windows 7/8/10

Programming Language: Python 3

Machine Learning Platform: Tensorflow

Tools used : Notepad++

1.4.2 Hardware Requirements

Processor : Intel Core i5(Min.)

Hard Disk : 500 GB (Min.)

RAM : 4 GB (Min.)

* 1. **Algorithm**

**Convolutional Neural Network**

**Step1: convolutional operation**

The first building block in our plan of attack is convolution operation. In this step, we will touch on feature detectors, which basically serve as the neural network's filters. We will also discuss feature maps, learning the parameters of such maps, how patterns are detected, the layers of detection, and how the findings are mapped out.

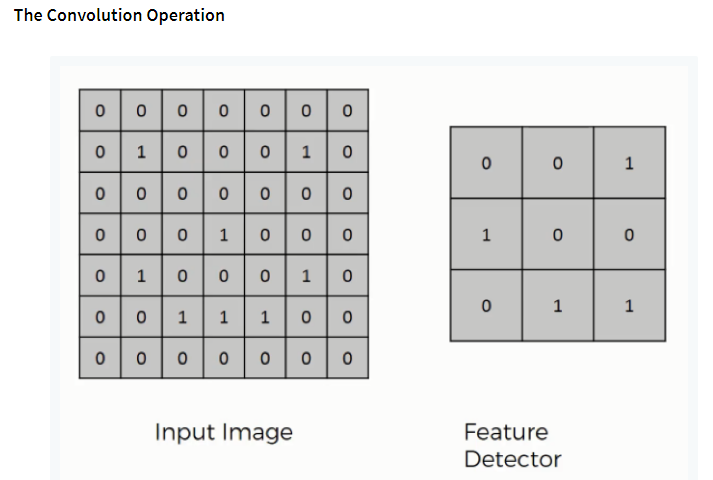


Figure 1.5.1 Convolution Operation

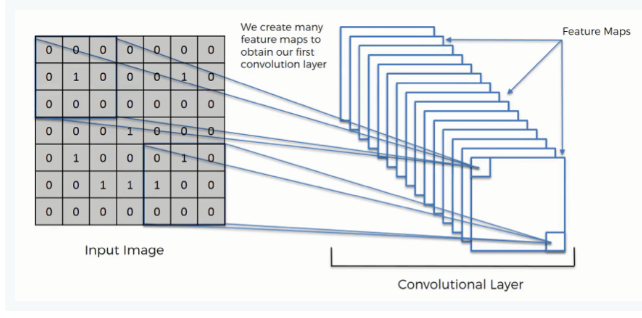


Figure 1.5.2 Convolution Operation

**Step (1b): ReLU Layer**

The second part of this step will involve the Rectified Linear Unit or Relook. We will cover Relook layers and explore how linearity functions in the context of Convolutional Neural Networks.

Not necessary for understanding CNN's, but there's no harm in a quick lesson to improve your skills.

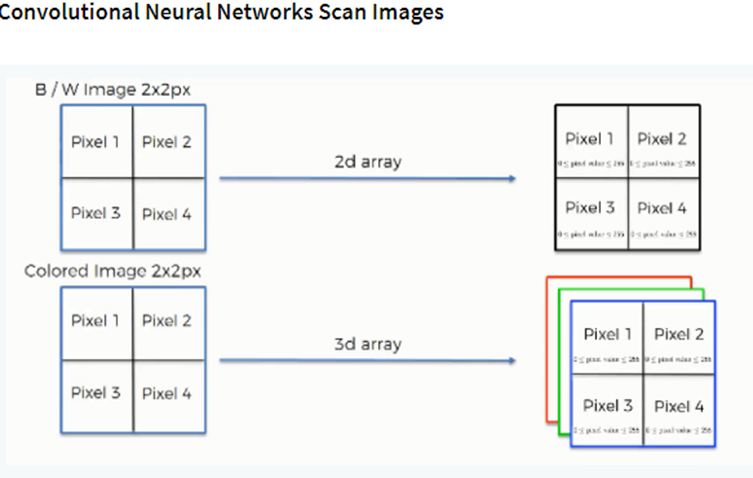


Figure 1.5.3 CNN Scan Images

**Step 2: Pooling Layer**

In this part, we'll cover pooling and will get to understand exactly how it generally works. Our nexus here, however, will be a specific type of pooling; max pooling. We'll cover various approaches, though, including mean (or sum) pooling. This part will end with a demonstration made using a visual interactive tool that will definitely sort the whole concept out for you.

**Step 3: Flattening**

This will be a brief breakdown of the flattening process and how we move from pooled to flattened layers when working with Convolutional Neural Networks.

**Step 4: Full Connection**

In this part, everything that we covered throughout the section will be merged together. By learning this, you'll get to envision a fuller picture of how Convolutional Neural Networks operate and how the "neurons" that are finally produced learn the classification of images.

**Summary**

In the end, we'll wrap everything up and give a quick recap of the concept covered in the section. If you feel like it will do you any benefit (and it probably will), you should check out the extra tutorial in which Soft ax and Cross-Entropy are covered. It's not mandatory for the course, but you will likely come across these concepts when working with Convolutional Neural Networks and it will do you a lot of good to be familiar with them.

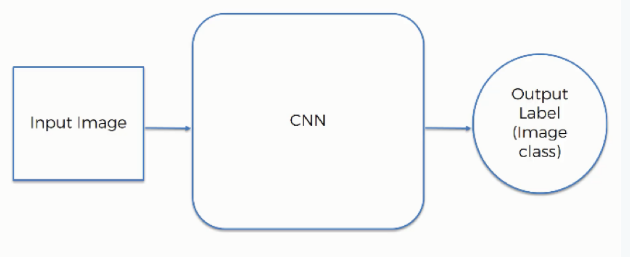


Figure 1.5.4 CNN Architecture

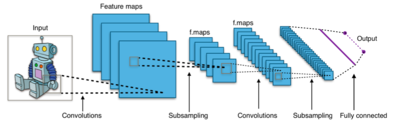


Figure 1.5.5. CNN Architecture

30, 2012. The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training.

AlexNet contained eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and the last three were fully connected layers. It used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid. The architecture is shown in figure 1.5.6.

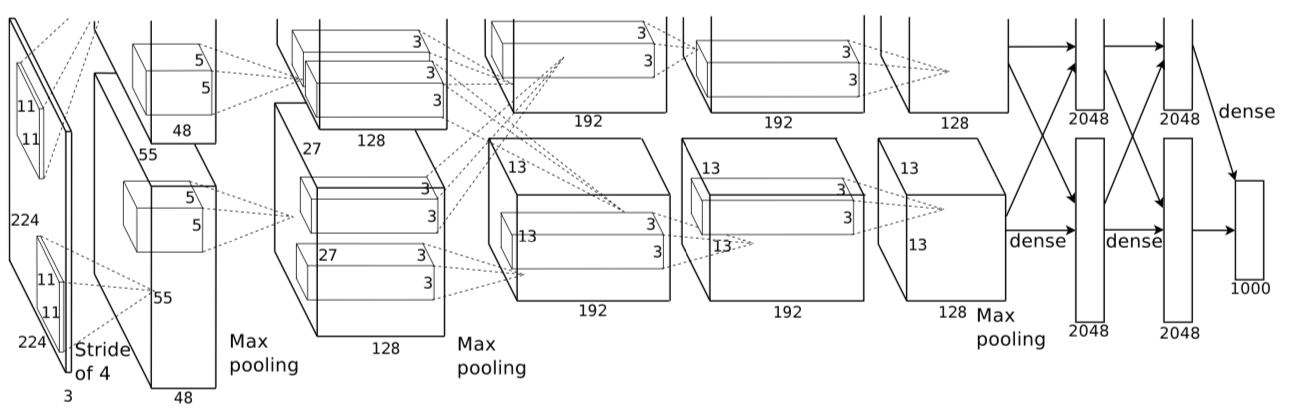


Figure 1.5.6 Layers of CNN

#### LITERATURE SURVEY

Autonomous Driving has been said to be the next big disruptive innovation in the years to come. Considered as being predominantly technology driven, it is supposed to have massive societal impact in all kinds of fields. In this section a brief overview on the technology and development will prove helpful to understand the need of customer acceptance on the topic that until now has been, as demonstrated in section 2, neglected. According to Marlon G. Boarnet (Ross, 2014, p. 90), a specialist in transportation and urban growth at the University of Southern California “Approximately every two generations, we rebuild the transportation infrastructure in our cities in ways that shape the vitality of neighborhoods; the settlement patterns in our cities and countryside; and our economy, society and culture” and as many believe, autonomous driving cars are this new big change everyone is talking about. Leading not only to high impact environmental benefits such as the improvement of fuel economy (Payre, et al. 2014; Luettel, et al., 2012) , through the optimization of highways (Luettel, et al., 2012; Le Vine, et al., 2015; Payre, et al., 2014; Hamish Jamson, et al., 2013; Merat, et al.; 2012), the reduction of required cars to only 15% of the current amount needed (Ross, 2014), and platoon driving that would save to 20-30% fuel consume (Weyer, et al., 2015), but also leading to societal aspects such as immense productivity gains while commuting (Le Vine, et al., 2015; Payre, et al., 2014; Hamish Jamson, et al., 2013), decline on the accident and death tolls considered as the eight highest death cause worldwide in 2013 (World Health Organization, 2013), stress reduction (RudinBrown & Parker, 2004a; Stanton & Young, 2005), and the decline of parking space to up to ¼ of the current capacity (Alessandrini, et al., 2015). It would also, according to a study by Morgan Stanley (2013) lead to an average 38 hours reduction of commuting time per individual per year as well as saving the US economy alone 1.3 trillion dollars per year, creating a shift on the possibilities and different applications, developing completely new markets, partnerships and possible business models (Bartl, 2015). That will change the society as we know it. All the benefits of course, and not only to mention the Technological difficulties do not come without a certain amount of challenges, complications and necessary changes in current systems in order to work. By now several States in the US have passed laws permitting autonomous cars testing on their roads (Walker S., 2014). The National Highway Traffic Safety Administration in the United States (2013) provides an official self-driving car classification dividing into No-Automation (Level 0), Function-specific Automation (Level 1), Combined Function Automation (Level 2), Limited Self-Driving Automation (Level 3) and Full Self-Driving Automation (Level 4). Europeans have also started modifying the Vienna Convention on Road Traffic and the Geneva Convention on Road Traffic (Reuters, 2014) in order to be able to adapt this new technology, but legal issues and doubt still arise as one of the main concerns of discussion. Some of the main issues surrounding the autonomous driving field found throughout the literature and the web are; test and standard set for critical event control, how to deal with the requirement for a ‘driver’, ownership and maintenance (Teare, 2014), civil and criminal liability, corporate manslaughter (Browning, 2014), insurance, data protection and privacy issues (Khan, et al., 2012). Having a closer look at the history of Autonomous Driving, as explained in the IEEE Spectrum (Ross, 2014) in Figure 1 it can be observed that the technological development and main milestones of the autonomous driving field started already a few decades ago. Leading to a vast analysis of some semi-autonomous features, development of present technologies and understanding on the future problematic while focusing in the near future in the connected car.

##### 

Figure 2.1 Sixty five years of automotive baby steps

##### 2.1 Methodology & Data Collection Procedure

The research conducted focused in a systematic keyword search in the topic section of literature databases, including EBSCO, Science Direct, Emerald and ISI web of knowledge. The search conducted included the specific terms, “Autonomous driving”, “Self driving car” and “Driverless car” either in the title, keywords or abstract and including only academic journals listed in the mentioned databases. Hence, the aim of this research was not to find all literature regarding Autonomous Driving that because of its size would lead to an enormous amount of results due to the extensive applications, testing and research in other fields (robotics, underwater vehicles, military, aeronautics, space vehicles, etc.), but to achieve an overview and overall classification to identify current gaps in the scientific literature body. Therefore taking into consideration only the literature publications relevant for roads, traffic, crossroads and studies related to commuting, transportation or production, and including all relevant publications found related to the automotive industry, as well as also considering papers in other topics that acknowledge that the application could be relevant for self-driving cars. The search yielded 483 papers, 154 of which were found in EBSCO, 122 in Science Direct, 8 in Emerald and 199 in ISI Web of Knowledge. Furthermore, a thorough analysis of abstracts, titles and journals was conducted in order to eliminate duplicates and results that were found in more than one database, leading to the elimination of 63 repetitions, some of which were even found 3 times, reflecting the thoroughness of the search, delivering a total amount of 420 publications. Finally a last filtering process eliminated publications not fulfilling scientific standards nor a peer-review process lead to the exclusion of 21 further publications delivering a final number of 399 Publications, which can be found in Appendix I. A set of analysis explained in the next sections were then conducted with the remaining 399 publications in order to understand of the current state of the literature, its focus and development over time.

**2.2 Introduction To Python**

**What is a Script?**

Up to this point, I have concentrated on the interactive programming capability of Python.  This is a very useful capability that allows you to type in a program and to have it executed immediately in an interactive mode

**Scripts are reusable**

Basically, a script is a text file containing the statements that comprise a Python program.  Once you have created the script, you can execute it over and over without having to retype it each time.

**Scripts are editable**

Perhaps, more importantly, you can make different versions of the script by modifying the statements from one file to the next using a text editor.  Then you can execute each of the individual versions.  In this way, it is easy to create different programs with a minimum amount of typing.

**You will need a text editor**

Just about any text editor will suffice for creating Python script files.

You can use Microsoft Notepad, Microsoft WordPad, Microsoft Word*,*or just about any word processor if you want to.

**Difference between a script and a program**

**Script:**

Scripts are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the end-user. Scripts are often interpreted from source code or byte code, whereas the applications they control are traditionally compiled to native machine code.

**Program:**

The program has an executable form that the computer can use directly to execute the instructions.

The same program in its human-readable source code form, from which executable programs are derived (e.g., compiled)

**Python**

What is Python? Chances you are asking yourself this. You may have found this book because you want to learn to program but don’t know anything about programming languages. Or you may have heard of programming languages like C, C++, C#, or Java and want to know what Python is and how it compares to “big name” languages. Hopefully I can explain it for you.

**Python concepts**

If you’re not interested in the hows and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it’s a great one to start programming with.

• Open source general-purpose language.

• Object Oriented, Procedural, Functional

• Easy to interface with C/ObjC/Java/Fortran

• Easy-is to interface with C++ (via SWIG)

• Great interactive environment

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

* **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* **Python is Interactive** − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

**2.3 History of Python**

Python was developed by Guido van Possum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Possum still holds a vital role in directing its progress.

**2.4 Python Features**

Python's features include −

**Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

**Easy-to-read** − Python code is more clearly defined and visible to the eyes.

**Easy-to-maintain** − Python's source code is fairly easy-to-maintain.

**A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

**Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

**Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

**Extendable** − you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

**Databases** − Python provides interfaces to all major commercial databases.

**GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

**Scalable** − Python provides a better structure and support for large programs than shell scripting.

**Variables**

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

**Standard Data Types**

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

* Numbers
* String
* List
* Tuple
* Dictionary

**Python Numbers**

Number data types store numeric values. Number objects are created when you assign a value to them

**Python Strings**

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

**Python Lists**

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator.

**Python Tuples**

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.

**Python Dictionary**

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

**2.5 20 Python libraries**

**1.** Requests. The most famous http library written by Kenneth remits. It’s a must have for every python developer.

**2.** Scrappy. If you are involved in web scraping then this is a must have library for you. After using this library you won’t use any other.

**3.** Python. A guy toolkit for python. I have primarily used it in place of tinder. You will really love it.

**4.** Pillow. A friendly fork of PIL (Python Imaging Library). It is more user friendly than PIL and is a must have for anyone who works with images.

**5.** SQLAlchemy. A database library. Many love it and many hate it. The choice is yours.

**6.** Beautiful Soup. I know it’s slow but this xml and html parsing library is very useful for beginners.

**7.** Twisted. The most important tool for any network application developer. It has a very beautiful ape and is used by a lot of famous python developers.

**8.** Numbly. How can we leave this very important library? It provides some advance math functionalities to python.

**9.** Skippy. When we talk about numbly then we have to talk about spicy. It is a library of algorithms and mathematical tools for python and has caused many scientists to switch from ruby to python.

**10.** Matplotlib. A numerical plotting library. It is very useful for any data scientist or any data analyser.

**11.** Pygmy. Which developer does not like to play games and develop them? This library will help you achieve your goal of 2d game development.

**12.** Piglet. A 3d animation and game creation engine. This is the engine in which the famous [python port](https://github.com/fogleman/Minecraft) of mine craft was made

**13.** Pit. A GUI toolkit for python. It is my second choice after python for developing GUI’s for my python scripts.

**14.** Pit. Another python GUI library. It is the same library in which the famous Bit torrent client is created.

**15.** Scaly. A packet sniffer and analyser for python made in python.

**16.** Pywin32. A python library which provides some useful methods and classes for interacting with windows.

**17.** Notch. Natural Language Toolkit – I realize most people won’t be using this one, but it’s generic enough. It is a very useful library if you want to manipulate strings. But its capacity is beyond that. Do check it out.

**18.** Nose. A testing framework for python. It is used by millions of python developers. It is a must have if you do test driven development.

**19.** Simply. Simply can do algebraic evaluation, differentiation, expansion, complex numbers, etc. It is contained in a pure Python distribution.

**20.** I Python. I just can’t stress enough how useful this tool is. It is a python prompt on steroids. It has completion, history, shell capabilities, and a lot more. Make sure that you take a look at it.

**Numpy**

Humpy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In numbly dimensions are called *axes*. The number of axes is *rank*.

• Offers Matlab-ish capabilities within Python

• Fast array operations

• 2D arrays, multi-D arrays, linear algebra etc.

**Matplotlib**

• High quality plotting library.

**2.6 Python class and objects**

These are the building blocks of OOP. Class creates a new object. This object can be anything, whether an abstract data concept or a model of a physical object, e.g. a chair. Each class has individual characteristics unique to that class, including variables and methods. Classes are very powerful and currently “the big thing” in most programming languages. Hence, there are several chapters dedicated to OOP later in the book.

The class is the most basic component of object-oriented programming. Previously, you learned how to use functions to make your program do something.

Now will move into the big, scary world of Object-Oriented Programming (OOP). To be honest, it took me several months to get a handle on objects.

When I first learned C and C++, I did great; functions just made sense for me.

Having messed around with BASIC in the early ’90s, I realized functions were just like subroutines so there wasn’t much new to learn.

However, when my C++ course started talking about objects, classes, and all the new features of OOP, my grades definitely suffered.

Once you learn OOP, you’ll realize that it’s actually a pretty powerful tool. Plus many Python libraries and APIs use classes, so you should at least be able to understand what the code is doing.

One thing to note about Python and OOP: it’s not mandatory to use objects in your code in a way that works best; maybe you don’t need to have a full-blown class with initialization code and methods to just return a calculation. With Python, you can get as technical as you want.

As you’ve already seen, Python can do just fine with functions. Unlike languages such as Java, you aren’t tied down to a single way of doing things; you can mix functions and classes as necessary in the same program. This lets you build the code

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

Here’s a brief list of Python OOP ideas:

• The class statement creates a class object and gives it a name. This creates a new namespace.

• Assignments within the class create class attributes. These attributes are accessed by qualifying the name using dot syntax: ClassName.Attribute.

• Class attributes export the state of an object and its associated behaviour. These attributes are shared by all instances of a class.

• Calling a class (just like a function) creates a new instance of the class.

This is where the multiple copies part comes in.

• Each instance gets ("inherits") the default class attributes and gets its own namespace. This prevents instance objects from overlapping and confusing the program.

• Using the term self identifies a particular instance, allowing for per-instance attributes. This allows items such as variables to be associated with a particular instance.

**Inheritance**

First off, classes allow you to modify a program without really making changes to it.

To elaborate, by sub classing a class, you can change the behavior of the program by simply adding new components to it rather than rewriting the existing components.

As we’ve seen, an instance of a class inherits the attributes of that class.

However, classes can also inherit attributes from other classes. Hence, a subclass inherits from a superclass allowing you to make a generic superclass that is specialized via subclasses.

The subclasses can override the logic in a superclass, allowing you to change the behavior of your classes without changing the superclass at all.

**Operator Overloads**

Operator overloading simply means that objects that you create from classes can respond to actions (operations) that are already defined within Python, such as addition, slicing, printing, etc.

Even though these actions can be implemented via class methods, using overloading ties the behaviour closer to Python’s object model and the object interfaces are more consistent to Python’s built-in objects, hence overloading is easier to learn and use.

User-made classes can override nearly all of Python’s built-in operation methods

**Exceptions**

I’ve talked about exceptions before but now I will talk about them in depth. Essentially, exceptions are events that modify program’s flow, either intentionally or due to errors.

They are special events that can occur due to an error, e.g. trying to open a file that doesn’t exist, or when the program reaches a marker, such as the completion of a loop.

Exceptions, by definition, don’t occur very often; hence, they are the "exception to the rule" and a special class has been created for them. Exceptions are everywhere in Python.

Virtually every module in the standard Python library uses them, and Python itself will raise them in a lot of different circumstances.

Here are just a few examples:

• Accessing a non−existent dictionary key will raise a Key Error exception.

• Searching a list for a non−existent value will raise a Value Error exception

. • Calling a non−existent method will raise an Attribute Error exception.

• Referencing a non−existent variable will raise a Name Error exception.

• Mixing data types without coercion will raise a Type Error exception.

One use of exceptions is to catch a fault and allow the program to continue working; we have seen this before when we talked about files.

This is the most common way to use exceptions. When programming with the Python command line interpreter, you don’t need to worry about catching exceptions.

Your program is usually short enough to not be hurt too much if an exception occurs.

Plus, having the exception occur at the command line is a quick and easy way to tell if your code logic has a problem.

However, if the same error occurred in your real program, it will fail and stop working. Exceptions can be created manually in the code by raising an exception.

It operates exactly as a system-caused exceptions, except that the programmer is doing it on purpose. This can be for a number of reasons. One of the benefits of using exceptions is that, by their nature, they don’t put any overhead on the code processing.

Because exceptions aren’t supposed to happen very often, they aren’t processed until they occur.

Exceptions can be thought of as a special form of the if/elf statements. You can realistically do the same thing with if blocks as you can with exceptions.

However, as already mentioned, exceptions aren’t processed until they occur; if blocks are processed all the time.

Proper use of exceptions can help the performance of your program.

The more infrequent the error might occur, the better off you are to use exceptions; using if blocks requires Python to always test extra conditions before continuing.

Exceptions also make code management easier: if your programming logic is mixed in with error-handling if statements, it can be difficult to read, modify, and debug your program.

**User-Defined Exceptions**

I won’t spend too much time talking about this, but Python does allow for a programmer to create his own exceptions.

You probably won’t have to do this very often but it’s nice to have the option when necessary.

However, before making your own exceptions, make sure there isn’t one of the built-in exceptions that will work for you.

They have been "tested by fire" over the years and not only work effectively, they have been optimized for performance and are bug-free.

Making your own exceptions involves object-oriented programming, which will be covered in the next chapter. To make a custom exception, the programmer determines which base exception to use as the class to inherit from, e.g. making an exception for negative numbers or one for imaginary numbers would probably fall under the Arithmetic Error exception class. To make a custom exception, simply inherit the base exception and define what it will do.

**Python modules**

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a *module*; definitions from a module can be *imported* into other modules or into the *main* module.

**Testing code**

As indicated above, code is usually developed in a file using an editor.

To test the code, import it into a Python session and try to run it.

Usually there is an error, so you go back to the file, make a correction, and test again.

This process is repeated until you are satisfied that the code works. T

His entire process is known as the development cycle.

There are two types of errors that you will encounter. Syntax errors occur when the form of some command is invalid.

This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

**Functions in Python**

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function.

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task.

To carry out that specific task, the function might or might not need multiple inputs. When the task is carved out, the function can or cannot return one or more values.

There are three types of functions in python:

Help (), min (), print ().

**Python Namespace**

Generally speaking, a **namespace** (sometimes also called a context) is a naming system for making names unique to avoid ambiguity. Everybody knows a name spacing system from daily life, i.e. the naming of people in first name and family name (surname).

An example is a network: each network device (workstation, server, printer,) needs a unique name and address. Yet another example is the directory structure of file systems.

The same file name can be used in different directories, the files can be uniquely accessed via the pathnames.

Many programming languages use namespaces or contexts for identifiers. An identifier defined in a namespace is associated with that namespace.

This way, the same identifier can be independently defined in multiple namespaces. (Like the same file names in different directories) Programming languages, which support namespaces, may have different rules that determine to which namespace an identifier belongs.

Namespaces in Python are implemented as Python dictionaries, this means it is a mapping from names (keys) to objects (values). The user doesn't have to know this to write a Python program and when using namespaces.

Some namespaces in Python:

* **global names** of a module
* **local names** in a function or method invocation
* **built-in names**: this namespace contains built-in functions (e.g. abs(), camp(), ...) and built-in exception names

**Garbage Collection**

Garbage Collector exposes the underlying memory management mechanism of Python, the automatic garbage collector. The module includes functions for controlling how the collector operates and to examine the objects known to the system, either pending collection or stuck in reference cycles and unable to be freed.

**2.7 Python XML Parser**

XML is a portable, open source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and/or developmental language.

What is XML? The Extensible Markup Language XML is a markup language much like HTML or SGML.

This is recommended by the World Wide Web Consortium and available as an open standard.

XML is extremely useful for keeping track of small to medium amounts of data without requiring a SQL-based backbone.

XML Parser Architectures and APIs the Python standard library provides a minimal but useful set of interfaces to work with XML.

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

Simple API for XML SAX: Here, you register call-backs for events of interest and then let the parser proceed through the document.

This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.

Document Object Model DOM API : This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical tree − based form to represent all the features of an XML document.

SAX obviously cannot process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.

SAX is read-only, while DOM allows changes to the XML file. Since these two different APIs literally complement each other, there is no reason why you cannot use them both for large projects.

**2.8 Python Web Frameworks**

A web framework is a code library that makes a developer's life easier when building reliable, scalable and maintainable web applications.

Why are web frameworks useful?

Web frameworks encapsulate what developers have learned over the past twenty years while programming sites and applications for the web. Frameworks make it easier to reuse code for common HTTP operations and to structure projects so other developers with knowledge of the framework can quickly build and maintain the application.

**Common web framework functionality**

Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include:

1. URL routing
2. HTML, XML, JSON, and other output format tinplating
3. Database manipulation
4. Security against Cross-site request forgery (CSRF) and other attacks
5. Session storage and retrieval

Not all web frameworks include code for all of the above functionality. Frameworks fall on the spectrum from executing a single use case to providing every known web framework feature to every developer. Some frameworks take the "batteries-included" approach where everything possible comes bundled with the framework while others have a minimal core package that is amenable to extensions provided by other packages.

**Comparing web frameworks**

There is also a repository called [compare-python-web-frameworks](https://github.com/mattmakai/compare-python-web-frameworks) where the same web application is being coded with varying Python web frameworks, tinplating engines and object.

**Web framework resources**

* When you are learning how to use one or more web frameworks it's helpful to have an idea of what the code under the covers is doing.
* Frameworks is a really well done short video that explains how to choose between web frameworks. The author has some particular opinions about what should be in a framework. For the most part I agree although I've found sessions and database ORMs to be a helpful part of a framework when done well.
* What is a web framework? Is an in-depth explanation of what web frameworks are and their relation to web servers?
* Jingo vs. Flash vs. Pyramid: Choosing a Python web framework contains background information and code comparisons for similar web applications built in these three big Python frameworks.
* This fascinating blog post takes a look at the code complexity of several Python web frameworks by providing visualizations based on their code bases.
* Python’s web frameworks benchmarks  is a test of the responsiveness of a framework with encoding an object to JSON and returning it as a response as well as retrieving data from the database and rendering it in a template. There were no conclusive results but the output is fun to read about nonetheless.
* What web frameworks do you use and why are they awesome? Is a language agnostic Reedit discussion on web frameworks? It's interesting to see what programmers in other languages like and dislike about their suite of web frameworks compared to the main Python frameworks.
* This user-voted question & answer site asked "What are the best general purpose Python web frameworks usable in production?” The votes aren't as important as the list of the many frameworks that are available to Python developers.

**3.** **SYSTEM STUDY**

**3.1 Feasibility Study**

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* Economical Feasibility
* Technical Feasibility
* Social Feasibility

**3.2 Economical Feasibility**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

**3.3 Technical Feasibility**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

##### 3.4 Social Feasibility

##### The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

##### 4. SYSTEM DESIGN

**4.1 UML Diagrams**

The Unified Modelling Language allows the software engineer to express an analysis model using the modelling notation that is governed by a set of syntactic semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

**User Model View**

This view represents the system from the users perspective. The analysis representation describes a usage scenario from the end-users perspective.

**Structural Model view**

In this model the data and functionality are arrived from inside the system. This model view models the static structures.

**Behavioural Model View**

It represents the dynamic of behavioural as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

**Implementation Model View**

In this the structural and behavioural as parts of the system are represented as they are to be built.

**Flow Chart:**

**4.1.1 Use Case Diagram**

A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system.

This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

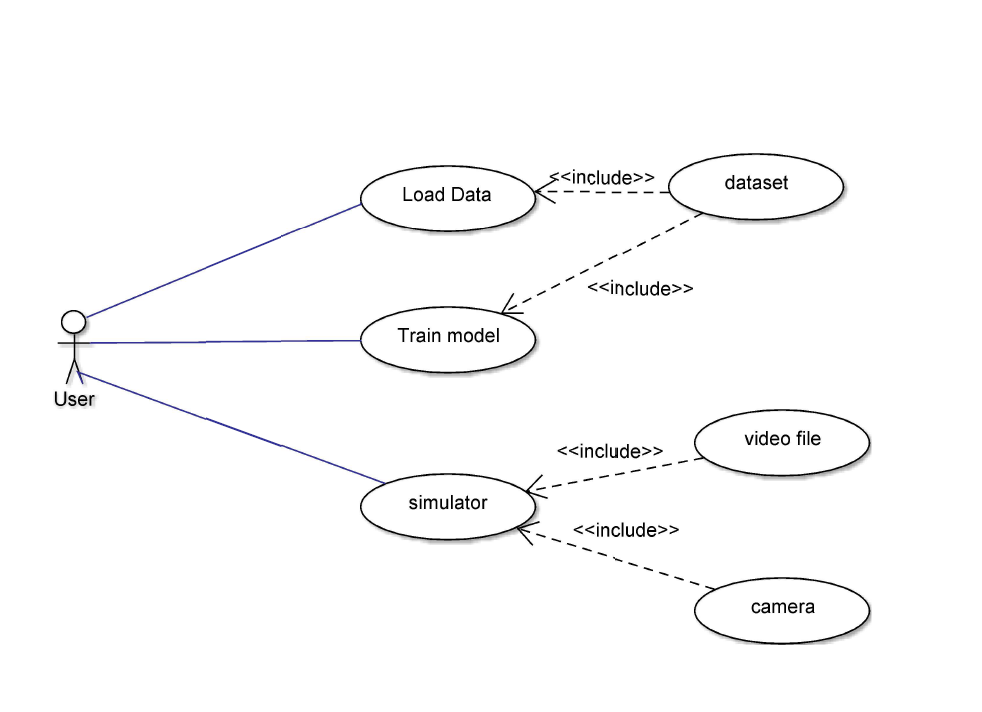


Figure 4.1.1 Use Case Diagram

* + 1. **Class Diagram**

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. A class with three sections, in the diagram, classes is represented with boxes which contain three parts:

The upper part holds the name of the class

The middle part contains the attributes of the class

The bottom part gives the methods or operations the class can take or undertake.

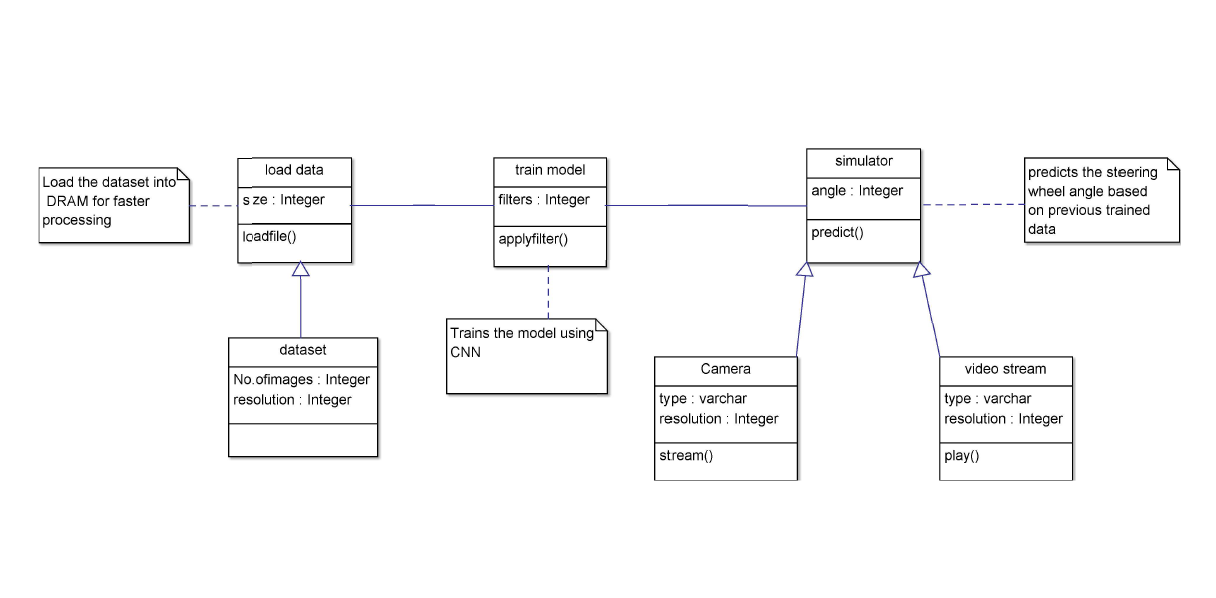
****

Figure 4.1.2 Class Diagram

**4.1.3 Activity Diagram**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

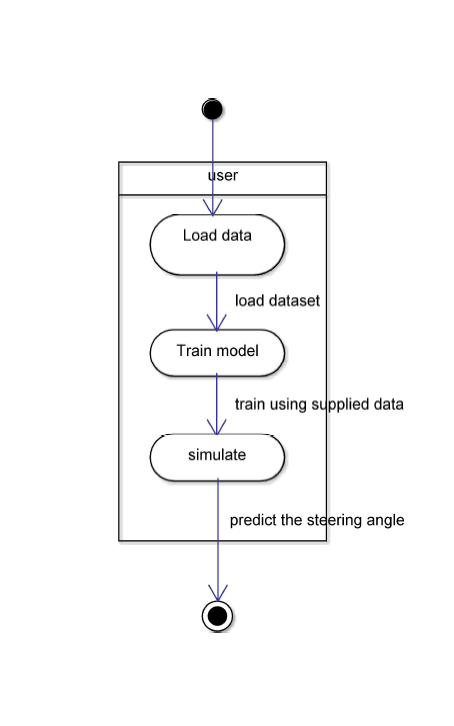


Figure 4.1.3 Activity Diagram

**4.1.4 Collaboration Diagram**

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language (UML). These diagrams can be used to portray the dynamic behaviour of a particular use case and define the role of each object.

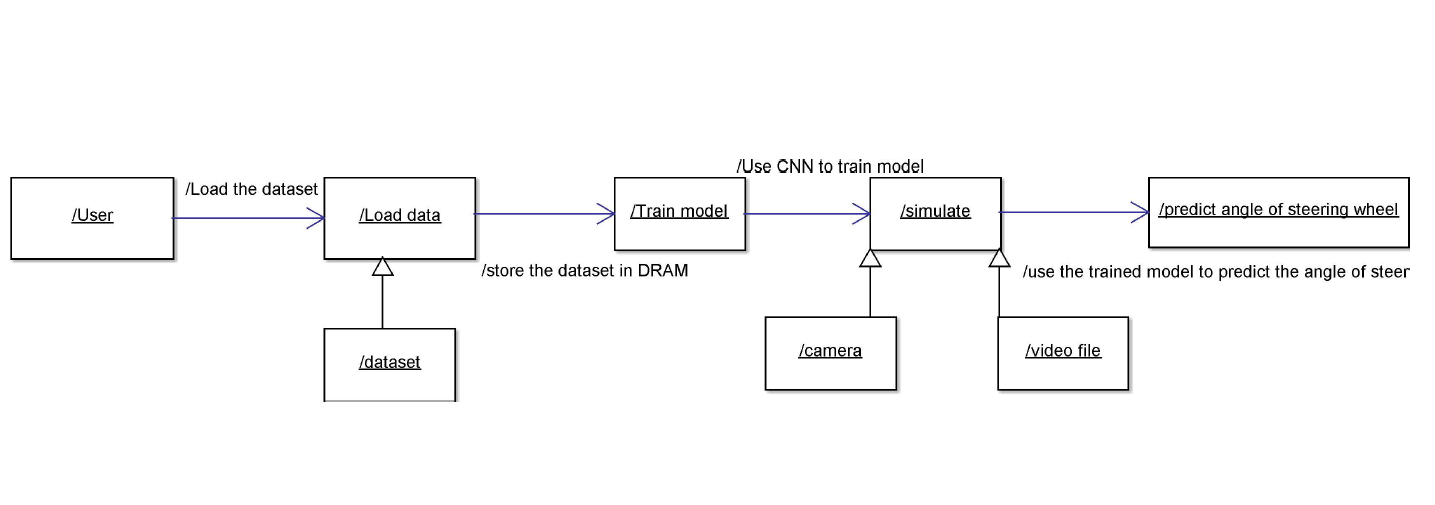


Figure 4.1.4 Collaboration Diagram

* + 1. **Sequence Diagram**

A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

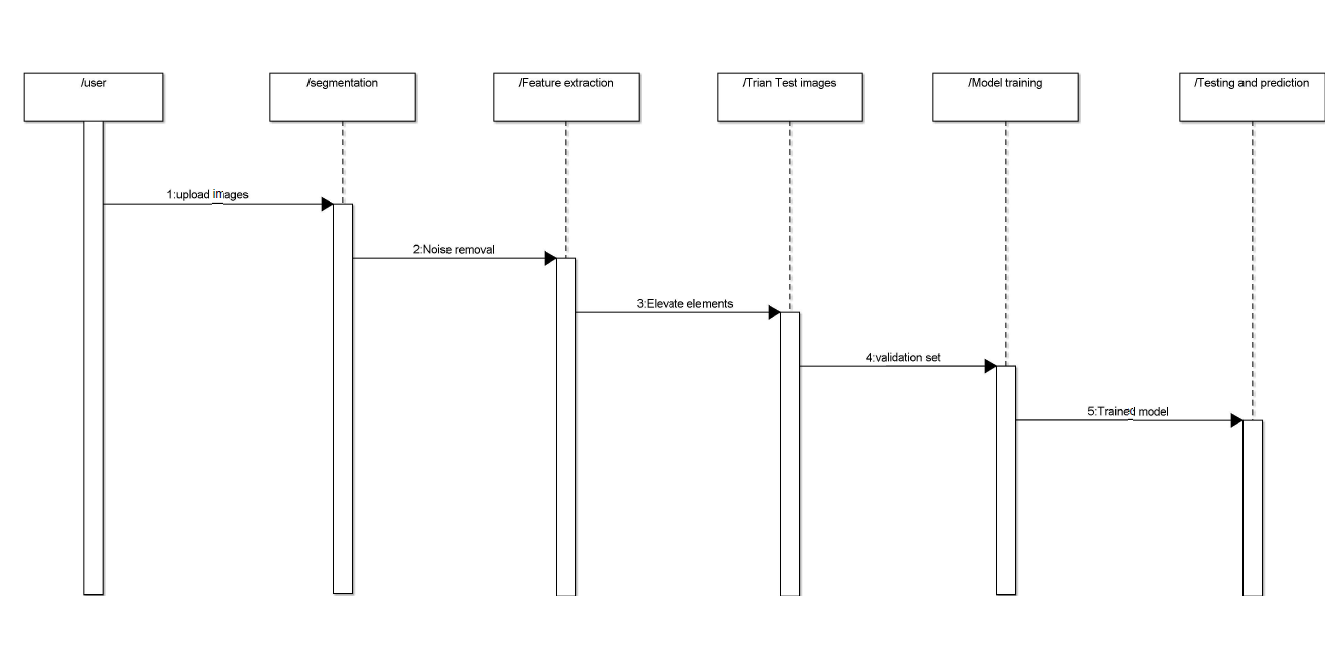
****

Figure 4.1.5 Sequence Diagram

**4.1.6 Deployment diagram**

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware’s used to deploy the application.

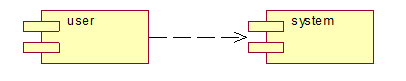
##### 

##### Figure 4.1.6 Deployment Diagram

**4.1.7 Component diagram**

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical **c**omponents in a system. Component diagrams are often drawn to

help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.



##### Figure 4.1.7 Component Diagram

##### 5. DESIGN METHODOLOGY OF AUTOPILOT SIMULATION

##### 5.1 Overview of the DAVE-2 System

Figure 4.1.1 shows a simplified block diagram of the collection system for training data for DAVE-2.

Three cameras are mounted behind the windshield of the data-acquisition car. Time-stamped video from the cameras is captured simultaneously with the steering angle applied by the human driver.

This steering command is obtained by tapping into the vehicle’s Controller Area Network (CAN) bus. In order to make our system independent of the car geometry, we represent the steering command as 1=r where r is the turning radius in meters. We use 1=r instead of r to prevent a singularity when driving straight (the turning radius for driving straight is infinity). 1=r smoothly transitions through zero from left turns (negative values) to right turns (positive values).

Training data contains single images sampled from the video, paired with the corresponding steering command (1=r). Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. Otherwise the car will slowly drift off the road. The training data is therefore augmented with additional images that show the car in different shifts from the centre of the lane and rotations from the direction of the road.

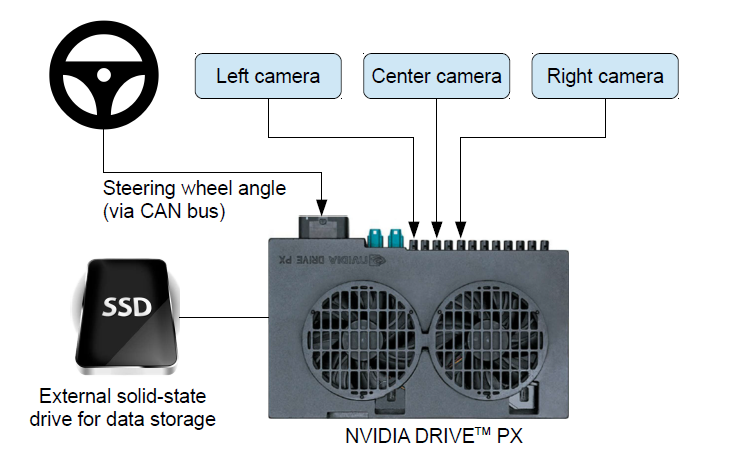


Figure 5.1.1 High-level view of the data collection system

Images for two specific off-centre shifts can be obtained from the left and the right camera. Additional shifts between the cameras and all rotations are simulated by viewpoint transformation of the image from the nearest camera. Precise viewpoint transformation requires 3D scene knowledge which we don’t have. We therefore approximate the transformation by assuming all points below the horizon are on flat ground and all points above the horizon are infinitely far away. This works fine for flat terrain but it introduces distortions for objects that stick above the ground, such as cars, poles, trees, and buildings. Fortunately these distortions don’t pose a big problem for network training.

The steering label for transformed images is adjusted to one that would steer the vehicle back to the desired location and orientation in two seconds.

A block diagram of our training system is shown in Figure 5.1.2. Images are fed into a CNN which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the CNN are adjusted to bring the CNN output closer to the desired output.

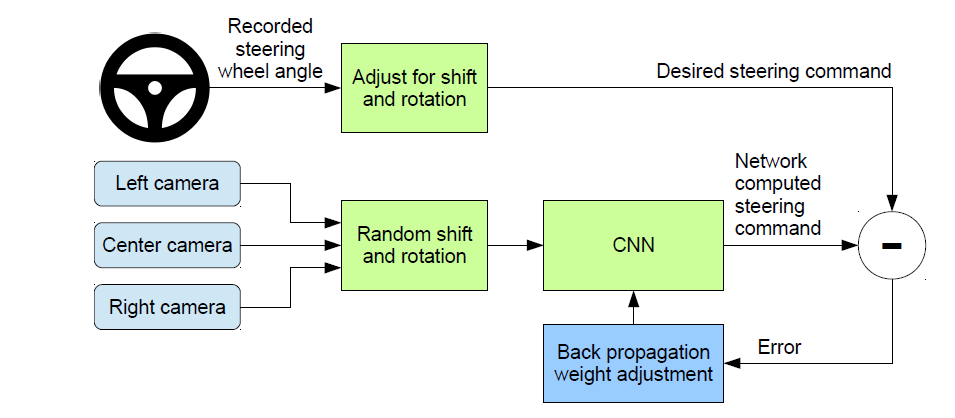


Figure 5.1.2 Training the neural network.

Once trained, the network can generate steering from the video images of a single centre camera.

This configuration is shown in Figure 5.1.3.

##### 

Figure 5.1.3 The trained network is used to generate steering commands from a single front-facing Centre camera.

##### 5.2 Data Collection

Training data was collected by driving on a wide variety of roads and in a diverse set of lighting and weather conditions. Most road data was collected in central New Jersey, although highway data was also collected from Illinois, Michigan, Pennsylvania, and New York. Other road types include two-lane roads (with and without lane markings), residential roads with parked cars, tunnels, and unpaved roads. Data was collected in clear, cloudy, foggy, snowy, and rainy weather, both day and night. In some instances, the sun was low in the sky, resulting in glare reflecting from the road surface and scattering from the windshield.

Data was acquired using either our drive-by-wire test vehicle, which is a 2016 Lincoln MKZ, or using a 2013 Ford Focus with cameras placed in similar positions to those in the Lincoln. The system has no dependencies on any particular vehicle make or model. Drivers were encouraged to maintain full attentiveness, but otherwise drive as they usually do. As of March 28, 2016, about 72 hours of driving data was collected.

##### 5.3 Network Architecture

We train the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off-centre and rotated images. Our network architecture is shown in Figure 5.3.1. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network.

The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2x2 stride and a 5x5 kernel and a non-strided convolution with a 3x3 kernel size in the last two convolutional layers. We follow the five convolutional layers with three fully connected layers leading to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller.

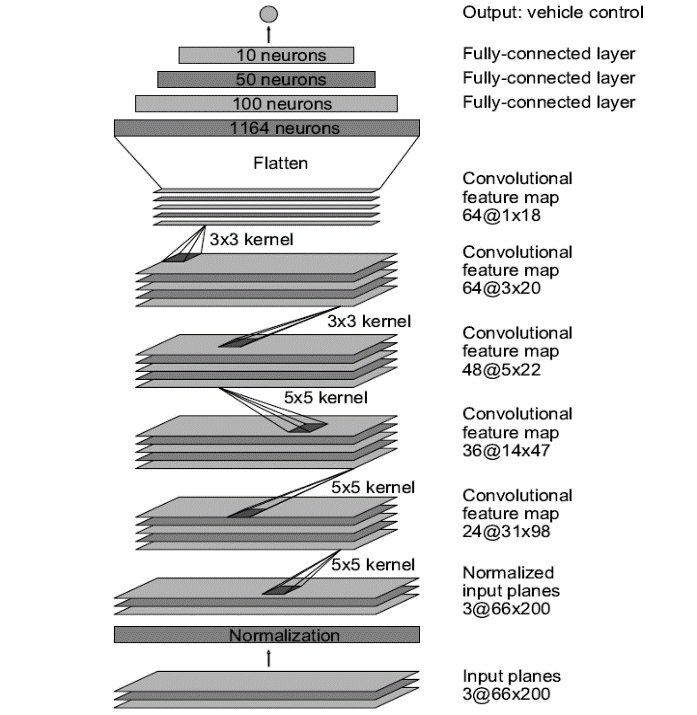


Figure 5.3.1 CNN architecture

##### 5.4 Data Selection

The first step to training a neural network is selecting the frames to use. Our collected data is

Labelled with road type, steering angle, and the driver’s activity (staying in a lane, switching lanes, turning, and so forth). To train a CNN to do lane following we only select data where the driver was staying in a lane and discard the rest. We then sample that video at 10 FPS. A higher sampling rate would result in including images that are highly similar and thus not provide much useful information. To remove a bias towards driving straight the training data includes a higher proportion of frames that represent road curves.

##### 5.5 Augmentation

After selecting the final set of frames we augment the data by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation. The magnitude of these perturbations is chosen randomly from a normal distribution. The distribution has zero mean, and the standard deviation is twice the standard deviation that we measured with human drivers. Artificially augmenting the data does add undesirable artefacts as the magnitude increases

5.6 Simulation

Before road-testing a trained CNN, we first evaluate the networks performance in simulation. A simplified block diagram of the simulation system is shown in Figure 3.5.1. The simulator takes pre-recorded videos from a forward-facing on-board camera on a human-driven

Data-collection vehicle and generates images that approximate what would appear if the CNN were, instead, steering the vehicle. These test videos are time-synchronized with recorded steering commands generated by the human driver.

Since human drivers might not be driving in the centre of the lane all the time, we manually calibrate the lane centre associated with each frame in the video used by the simulator. We call this position the “ground truth”. The simulator transforms the original images to account for departures from the ground truth. Note that this transformation also includes any discrepancy between the human driven path and the ground truth. The simulator accesses the recorded test video along with the synchronized steering commands that occurred when the video was captured. The simulator sends the first frame of the chosen test video,

Adjusted for any departures from the ground truth, to the input of the trained CNN. The CNN then returns a steering command for that frame. The CNN steering commands as well as the recorded human-driver commands are fed into the dynamic model of the vehicle to update the position and orientation of the simulated vehicle. The simulator then modifies the next frame in the test video so that the image appears as if the vehicle were at the position that resulted by following steering commands from the CNN. This new image is then fed to the CNN and the process repeats. The simulator records the off-centre distance (distance from the car to the lane centre), the yaw, and the distance travelled by the virtual car. When the off-centre distance exceeds one meter, a virtual human intervention is triggered, and the virtual vehicle position and orientation is reset to match the ground truth of the corresponding frame of the original test video.

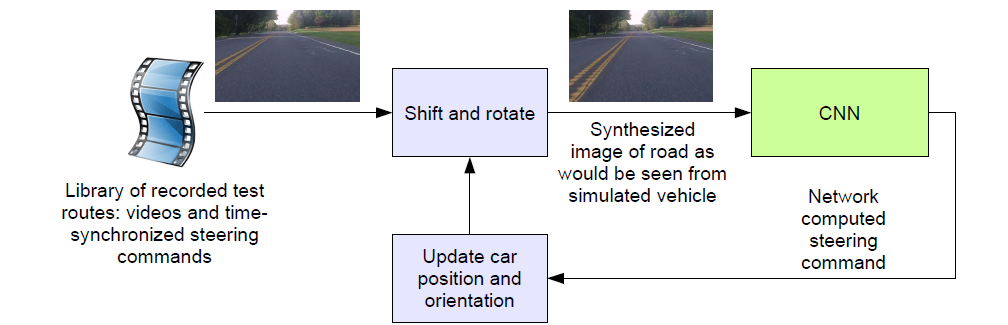


Figure 5.6.1 Block-diagram of the autopilot simulator.

Loss Plot

During an epoch, the loss function is calculated across every data item and it is guaranteed to give the quantitative loss measure at the given epoch. But plotting curve across iterations only gives the loss on a subset of the entire dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. It is a measure of the number of times all of the training vectors are used once to update the weights. For batch training all of the training samples pass through the learning algorithm simultaneously in one epoch before weights are updated.

Below figure 3.6.1 gives the loss plot of Epochs vs Loss (quantitative loss) across every predicted and target image.

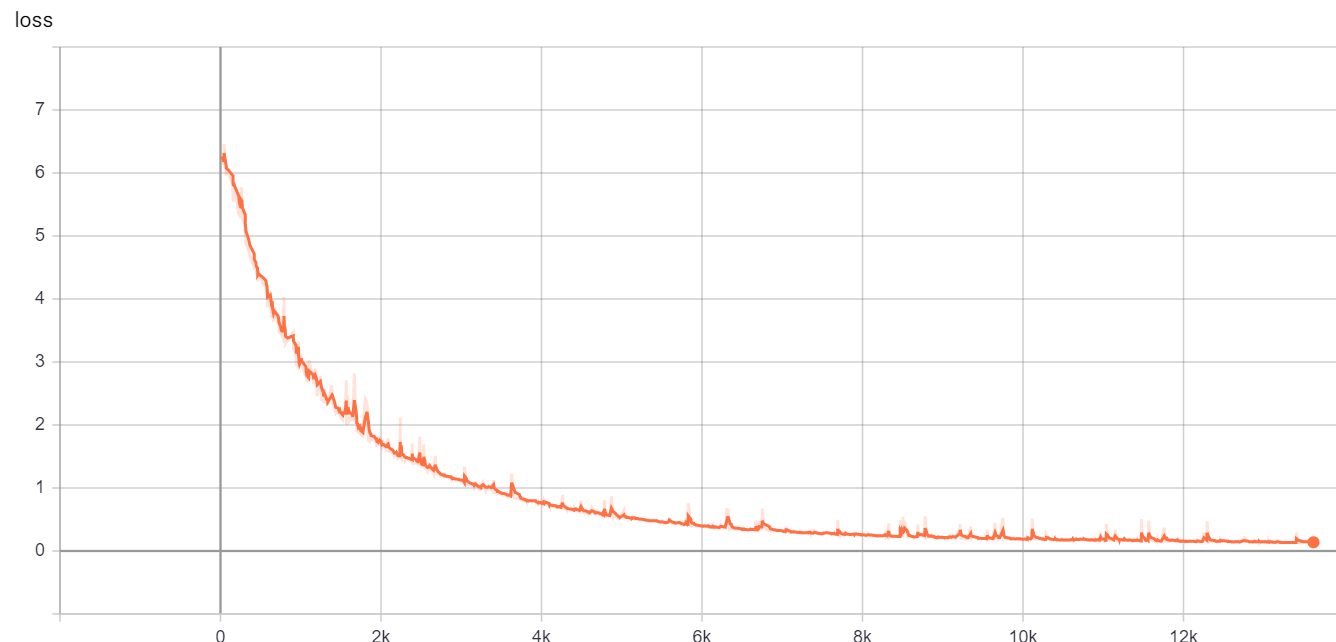


Figure 5.6.2 Loss Plot for Epochs and Loss across Predicted to target image in training

It is observed that as the epochs are increasing in training, the loss across the model is decreasing. So, the longer the training time or the higher the epoch number, the loss is greatly reduced and output prediction gets accurate.

**5.7 Software Development Life Cycle**

There is various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred

as "Software Development Process Models". Each process model follows a particular life cycle in order to ensure success in process of software development.



Figure 5.7.1 SDLC

**Requirements**

Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements. Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. This produces a nice big list of functionality that the system should provide, which describes functions the system should perform, business logic that processes data, what data is stored and used by the system, and how the user interface should work. The overall result is the system as a whole and how it performs, not how it is actually going to do it.

**Design**

The software system design is produced from the results of the requirements phase. Architects have the ball in their court during this phase and this is the phase in which their

focus lies. This is where the details on how the system will work is produced. Architecture, including hardware and software, communication, software design (UML is produced here) are all part of the deliverables of a design phase.

**Implementation**

Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle. For a developer, this is the main focus of the life cycle because this is where the code is produced. Implementation my overlap with both the design and testing phases. Many tools exists (CASE tools) to actually automate the production of code using information gathered and produced during the design phase.

**Testing**

During testing, the implementation is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. Unit tests and system/acceptance tests are done during this phase. Unit tests act on a specific component of the system, while system tests act on the system as a whole.

So in a nutshell, that is a very basic overview of the general software development life cycle model. Now let’s delve into some of the traditional and widely used variations.

**5.8 SDLC Methodologies**

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.



Figure 5.8.1 Spiral Model

**The steps for Spiral Model can be generalized as follows**

• The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.

• A preliminary design is created for the new system.

• A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.

• A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.

2. Defining the requirements of the second prototype.

3. Planning a designing the second prototype.

4. Constructing and testing the second prototype.

• At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.

• The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.

• The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.

• The final system is constructed, based on the refined prototype.

• The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

#### 6. TESTING AND RESULT ANALYSIS

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**6.1 Types of Tests**

**Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**6.2 System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**6.3 White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**6.4 Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

##### 6.5 Testing

We estimate what percentage of the time the network could drive the car (autonomy). The metric is determined by counting simulated human interventions (see Section 6). These interventions occur when the simulated vehicle departs from the centre line by more than one meter. We assume that in real life an actual intervention would require a total of six seconds: this is the time required for a human to retake control of the vehicle, re-centre it, and then restart the self-steering mode. We calculate the percentage autonomy by counting the number of interventions, multiplying by 6 seconds, dividing by the elapsed time of the simulated test, and then subtracting the result from 1:

Autonomy = (1 -(number of interventions) x6 seconds/elapsed time [seconds])x100

##### 6.6 Evaluation

Evaluating our networks is done by simulation. In simulation we have the networks provide steering commands in our simulator to an ensemble of pre-recorded test routes that correspond to about a total of three hours and 100 miles of driving in Monmouth County, NJ. The test data was taken in diverse lighting and weather conditions and includes highways, local roads, and residential streets.

###### 6.7 Results

The results in this experiment so far are encouraging. After trying our different combinations of hyper parameters, the simulator accesses the recorded test video along with the synchronized steering commands that occurred when the video was captured. The simulator sends the first frame of the chosen test video, adjusted for any departures from the ground truth, to the input of the trained CNN. The CNN then returns a steering command for that frame. The CNN steering commands as well as the recorded human-driver commands are fed into the dynamic model of the vehicle to update the position and orientation of the simulated vehicle. The simulator then modifies the next frame in the test video so that the image appears as if the vehicle were at the position that resulted by following steering commands from the CNN. This new image is then fed to the CNN and the process repeats.

The simulator records the off-centre distance (distance from the car to the lane centre), the yaw, and the distance travelled by the virtual car. When the off-centre distance exceeds one meter, a virtual human intervention is triggered, and the virtual vehicle position and orientation is reset to match the ground truth of the corresponding frame of the original test video.



Figure 6.7.1 How the CNN “sees” an unpaved road

###### 6.8 Outputs

The final outputs for Autopilot Simulation are quite good.

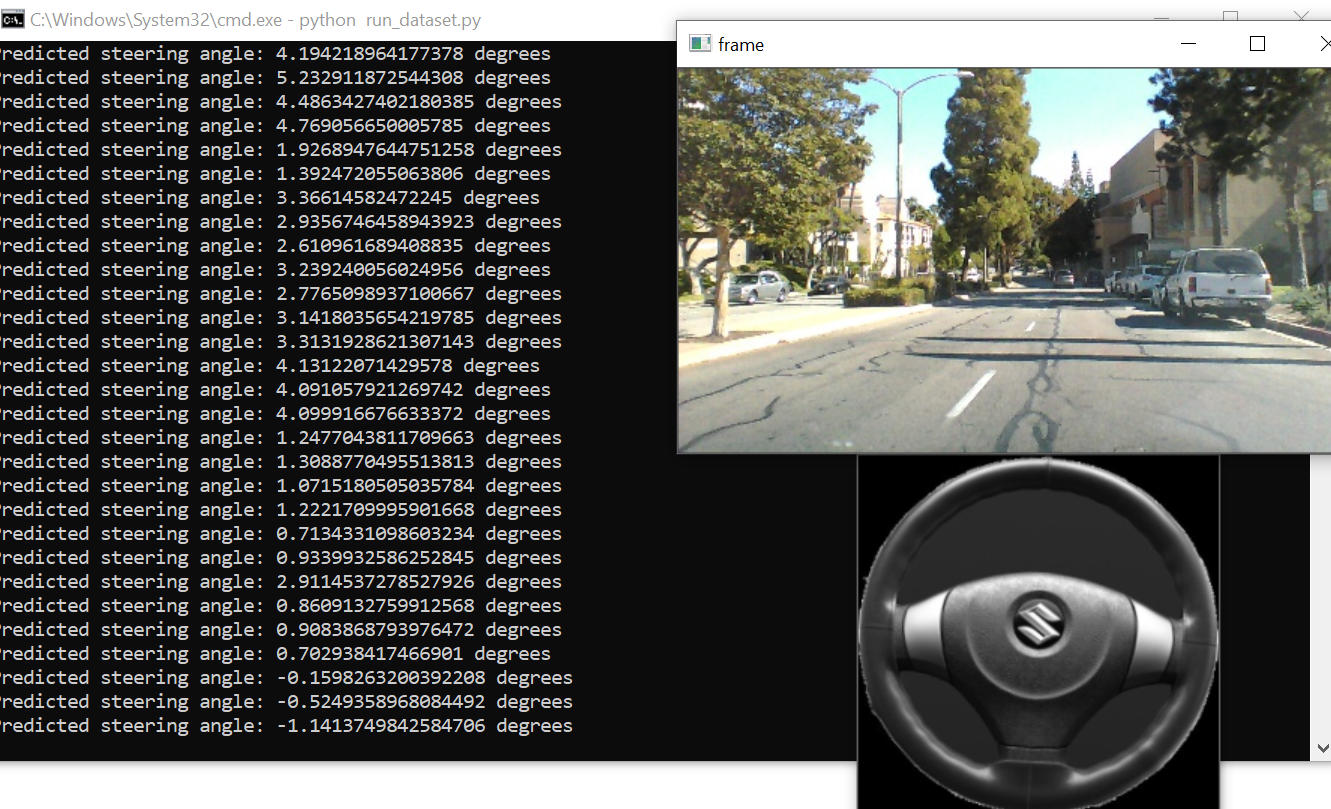


Figure 6.8.1 Autopilot Simulator running trained dataset

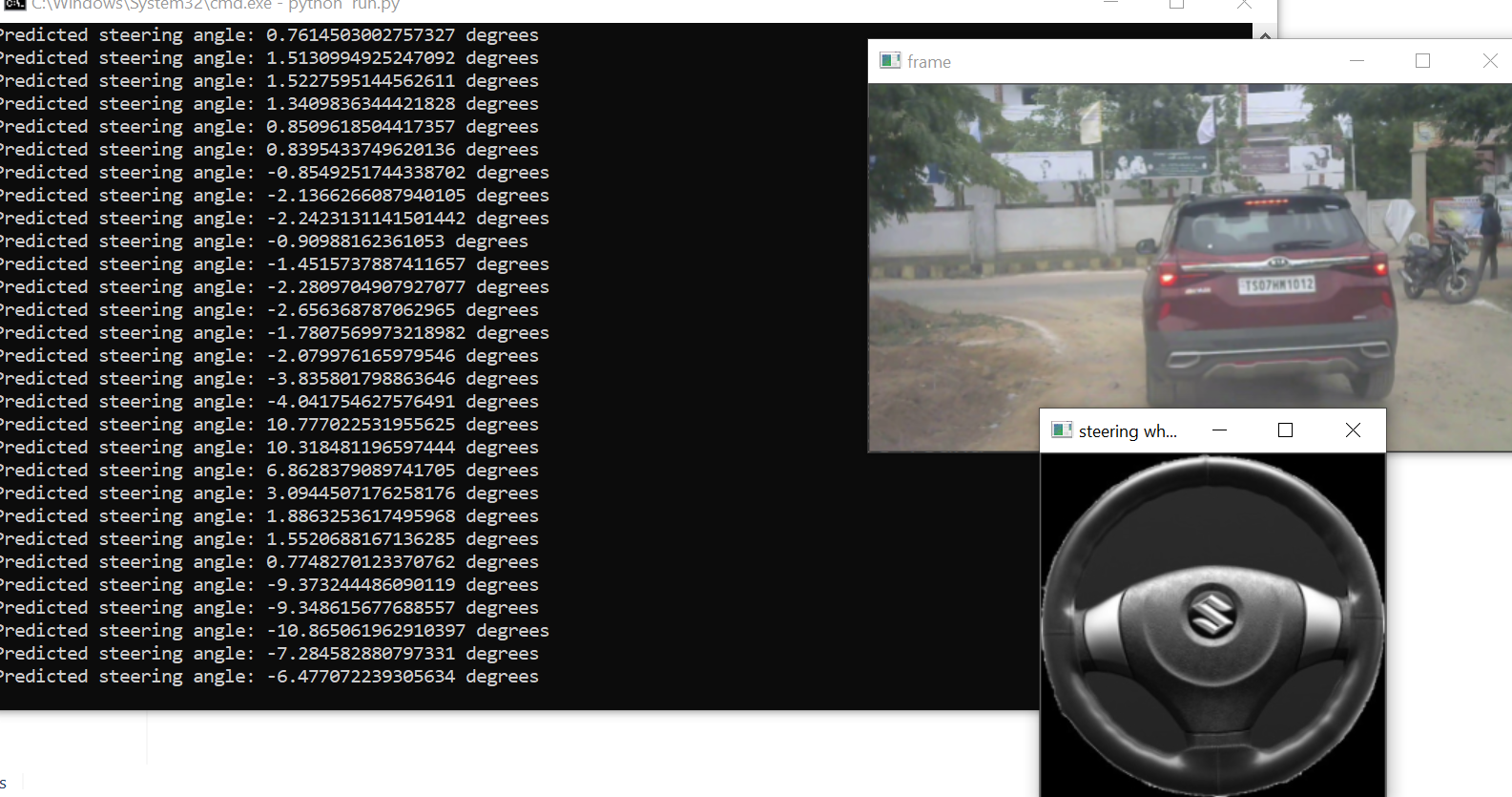


Figure 6.8.2 Autopilot Simulator running on test video

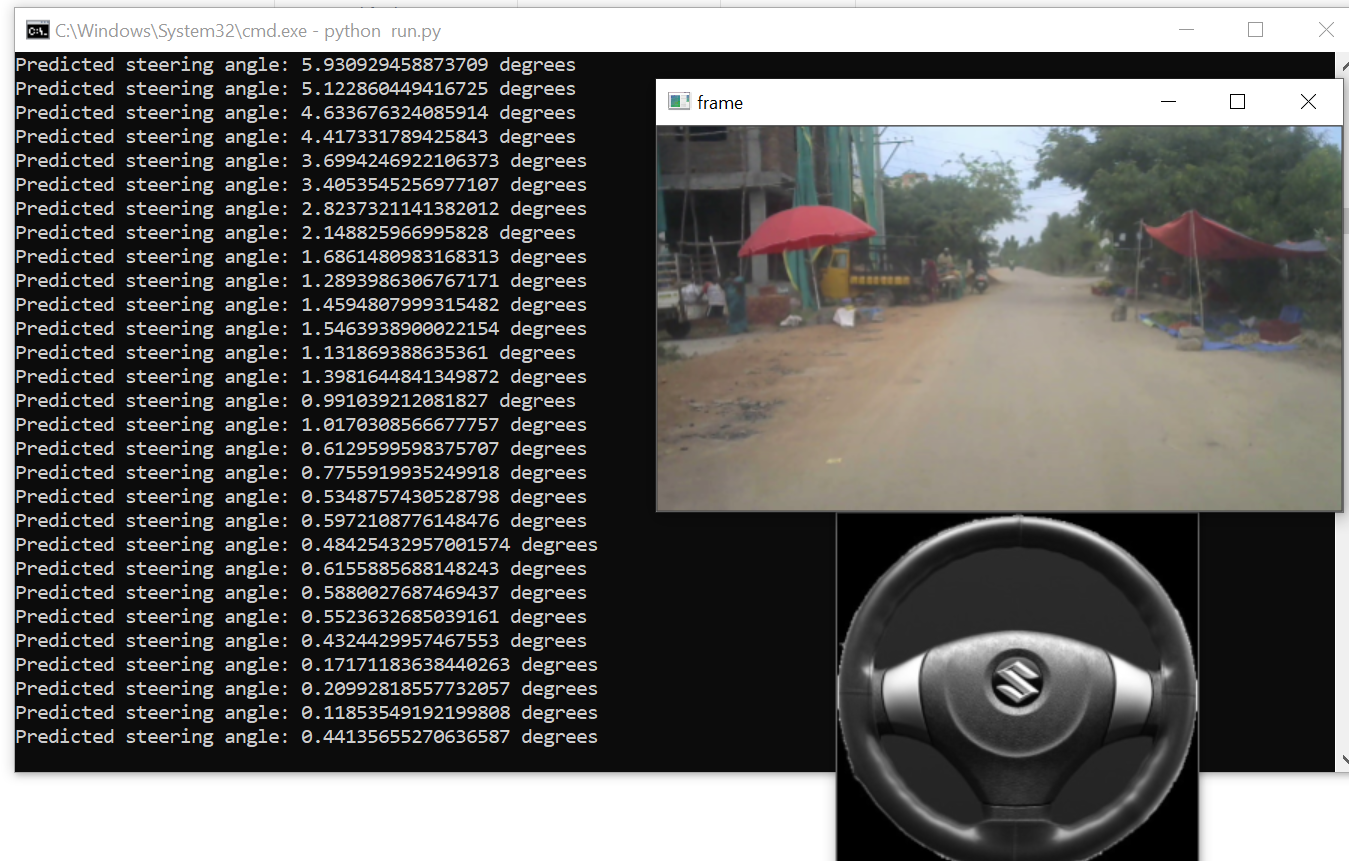


Figure 6.8.3 Autopilot Simulator running in real time

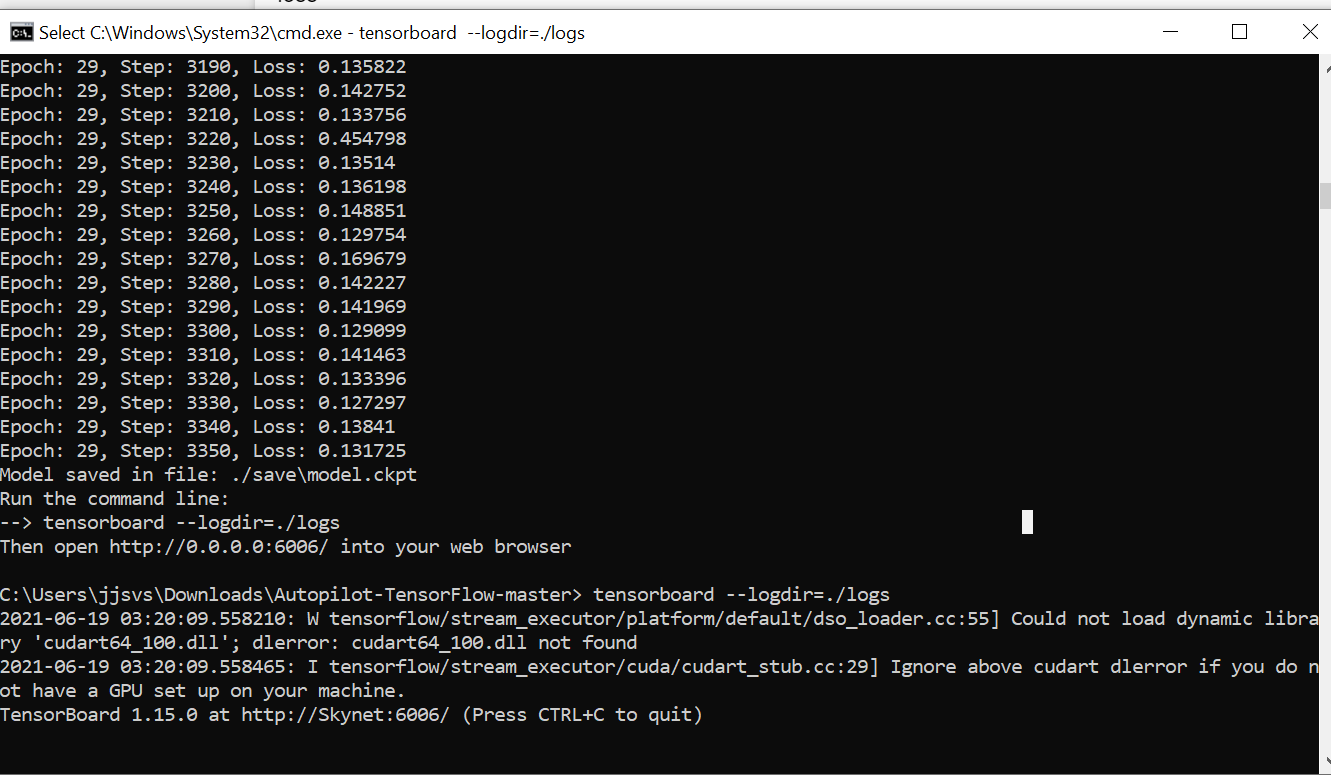
****

Figure 6.8.4 Autopilot Simulator being trained with dataset

**7. APPENDIX**

### #Implementing Autopilot Simiulator

import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

import scipy

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W, stride):

return tf.nn.conv2d(x,W,strides=[1,stride,stride,1], padding='VALID')

x = tf.placeholder(tf.float32, shape=[None, 66, 200, 3])

y\_ = tf.placeholder(tf.float32, shape=[None, 1])

x\_image = x

#first convolutional layer

W\_conv1 = weight\_variable([5, 5, 3, 24])

b\_conv1 = bias\_variable([24])

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1, 2) + b\_conv1)

#second convolutional layer

W\_conv2 = weight\_variable([5, 5, 24, 36])

b\_conv2 = bias\_variable([36])

h\_conv2 = tf.nn.relu(conv2d(h\_conv1, W\_conv2, 2) + b\_conv2)

#third convolutional layer

W\_conv3 = weight\_variable([5, 5, 36, 48])

b\_conv3 = bias\_variable([48])

h\_conv3 = tf.nn.relu(conv2d(h\_conv2, W\_conv3, 2) + b\_conv3)

#fourth convolutional layer

W\_conv4 = weight\_variable([3, 3, 48, 64])

b\_conv4 = bias\_variable([64])

h\_conv4 = tf.nn.relu(conv2d(h\_conv3, W\_conv4, 1) + b\_conv4)

#fifth convolutional layer

W\_conv5 = weight\_variable([3, 3, 64, 64])

b\_conv5 = bias\_variable([64])

h\_conv5 = tf.nn.relu(conv2d(h\_conv4, W\_conv5, 1) + b\_conv5)

#FCL 1

W\_fc1 = weight\_variable([1152, 1164])

b\_fc1 = bias\_variable([1164])

h\_conv5\_flat = tf.reshape(h\_conv5, [-1, 1152])

h\_fc1 = tf.nn.relu(tf.matmul(h\_conv5\_flat, W\_fc1) + b\_fc1)

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

#FCL 2

W\_fc2 = weight\_variable([1164, 100])

b\_fc2 = bias\_variable([100])

h\_fc2 = tf.nn.relu(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

h\_fc2\_drop = tf.nn.dropout(h\_fc2, keep\_prob)

#FCL 3

W\_fc3 = weight\_variable([100, 50])

b\_fc3 = bias\_variable([50])

h\_fc3 = tf.nn.relu(tf.matmul(h\_fc2\_drop, W\_fc3) + b\_fc3)

h\_fc3\_drop = tf.nn.dropout(h\_fc3, keep\_prob)

#FCL 3

W\_fc4 = weight\_variable([50, 10])

b\_fc4 = bias\_variable([10])

h\_fc4 = tf.nn.relu(tf.matmul(h\_fc3\_drop, W\_fc4) + b\_fc4)

h\_fc4\_drop = tf.nn.dropout(h\_fc4, keep\_prob)

#Output

W\_fc5 = weight\_variable([10, 1])

b\_fc5 = bias\_variable([1])

y = tf.multiply(tf.atan(tf.matmul(h\_fc4\_drop,W\_fc5)+b\_fc5), 2)

**#Download and prepare the dataset**

import cv2

import random

import numpy as np

xs = []

ys = []

#points to the end of the last batch

train\_batch\_pointer = 0

val\_batch\_pointer = 0

#read data.txt

with open("driving\_dataset/data.txt") as f:

for line in f:

xs.append("driving\_dataset/" + line.split()[0])

#the paper by Nvidia uses the inverse of the turning radius,

#but steering wheel angle is proportional to the inverse of turning radius

#so the steering wheel angle in radians is used as the output

ys.append(float(line.split()[1]) \* 3.14159265 / 180)

#get number of images

num\_images = len(xs)

#shuffle list of images

c = list(zip(xs, ys))

random.shuffle(c)

xs, ys = zip(\*c)

train\_xs = xs[:int(len(xs) \* 0.8)]

train\_ys = ys[:int(len(xs) \* 0.8)]

val\_xs = xs[-int(len(xs) \* 0.2):]

val\_ys = ys[-int(len(xs) \* 0.2):]

num\_train\_images = len(train\_xs)

num\_val\_images = len(val\_xs)

def LoadTrainBatch(batch\_size):

global train\_batch\_pointer

x\_out = []

y\_out = []

for i in range(0, batch\_size):

x\_out.append(cv2.resize(cv2.imread(train\_xs[(train\_batch\_pointer + i) % num\_train\_images])[-150:], (200, 66)) / 255.0)

y\_out.append([train\_ys[(train\_batch\_pointer + i) % num\_train\_images]])

train\_batch\_pointer += batch\_size

return x\_out, y\_out

def LoadValBatch(batch\_size):

global val\_batch\_pointer

x\_out = []

y\_out = []

for i in range(0, batch\_size):

x\_out.append(cv2.resize(cv2.imread(val\_xs[(val\_batch\_pointer + i) % num\_val\_images])[-150:], (200, 66)) / 255.0)

y\_out.append([val\_ys[(val\_batch\_pointer + i) % num\_val\_images]])

val\_batch\_pointer += batch\_size

return x\_out, y\_out

#### #Training the Dataset

import os

import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

from tensorflow.core.protobuf import saver\_pb2

import driving\_data

import model

LOGDIR = './save'

sess = tf.InteractiveSession()

L2NormConst = 0.001

train\_vars = tf.trainable\_variables()

loss = tf.reduce\_mean(tf.square(tf.subtract(model.y\_, model.y))) + tf.add\_n([tf.nn.l2\_loss(v) for v in train\_vars]) \* L2NormConst

train\_step = tf.train.AdamOptimizer(1e-4).minimize(loss)

sess.run(tf.global\_variables\_initializer())

# create a summary to monitor cost tensor

tf.summary.scalar("loss", loss)

# merge all summaries into a single op

merged\_summary\_op = tf.summary.merge\_all()

saver = tf.train.Saver(write\_version = saver\_pb2.SaverDef.V2)

# op to write logs to Tensorboard

logs\_path = './logs'

summary\_writer=tf.summary.FileWriter(logs\_path, graph=tf.get\_default\_graph())

epochs = 30

batch\_size = 100

# train over the dataset about 30 times

for epoch in range(epochs):

for i in range(int(driving\_data.num\_images/batch\_size)):

xs, ys = driving\_data.LoadTrainBatch(batch\_size)

train\_step.run(feed\_dict={model.x: xs, model.y\_: ys, model.keep\_prob: 0.8})

if i % 10 == 0:

xs, ys = driving\_data.LoadValBatch(batch\_size)

loss\_value = loss.eval(feed\_dict={model.x:xs, model.y\_: ys, model.keep\_prob: 1.0})

print("Epoch: %d, Step: %d, Loss: %g" % (epoch, epoch \* batch\_size + i, loss\_value))

# write logs at every iteration

summary=merged\_summary\_op.eval(feed\_dict={model.x:xs, model.y\_: ys, model.keep\_prob: 1.0})

summary\_writer.add\_summary(summary,epoch\*driving\_data.num\_images/batch\_size + i)

if i % batch\_size == 0:

if not os.path.exists(LOGDIR):

os.makedirs(LOGDIR)

checkpoint\_path=os.path.join(LOGDIR, "model.ckpt")

filename = saver.save(sess, checkpoint\_path)

print("Model saved in file: %s" % filename)

print("Run the command line:\n"\"--> tensorboard --logdir=./logs"\"\nThen open <http://0.0.0.0:6006/> into your web browser")

**#Running Simulator for Trained Dataset**

import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

import model

import cv2

from subprocess import call

import os

#check if on windows OS

windows = False

if os.name == 'nt':

windows = True

sess = tf.InteractiveSession()

saver = tf.train.Saver()

saver.restore(sess, "save/model.ckpt")

img = cv2.imread('save/steering\_wheel\_image.jpg',0)

rows,cols = img.shape

smoothed\_angle = 0

i = 0

while(cv2.waitKey(10) != ord('q')):

full\_image = cv2.imread("driving\_dataset/" + str(i) + ".jpg")

image = cv2.resize(full\_image[-150:], (200, 66)) / 255.0

degrees = model.y.eval(feed\_dict={model.x: [image], model.keep\_prob: 1.0})[0][0] \* 180.0 / 3.14159265

if not windows:

call("clear")

print("Predicted steering angle: " + str(degrees) + " degrees")

cv2.imshow("frame", full\_image)

#make smooth angle transitions by turning the steering wheel based on the difference of the current angle

smoothed\_angle+=0.2 \* pow(abs((degrees - smoothed\_angle)), 2.0/3.0)\*(degrees-smoothed\_angle)/abs(degrees- smoothed\_angle)

M=cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed\_angle,1)

dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow("steering wheel", dst)

i += 1

cv2.destroyAllWindows()

**#Running Simulator for Video file**

import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

import model

import cv2

from subprocess import call

import os

import imutils

#check if on windows OS

windows = False

if os.name == 'nt':

windows = True

sess = tf.InteractiveSession()

saver = tf.train.Saver()

saver.restore(sess, "save/model.ckpt")

img = cv2.imread('save/steering\_wheel\_image.jpg',0)

rows,cols = img.shape

smoothed\_angle = 0

cap = cv2.VideoCapture('save/run.mp4')

while(cv2.waitKey(10) != ord('q')):

ret, frame = cap.read()

frame = imutils.resize(frame, width=455)

frame = imutils.resize(frame, height=256)

image = cv2.resize(frame, (200, 66)) / 255.0

degrees=model.y.eval(feed\_dict={model.x:[image], model.keep\_prob: 1.0})[0][0] \* 180 / 3.14159265

if not windows:

call("clear")

print("Predicted steering angle:"+str(degrees)+" degrees")

cv2.imshow('frame', frame)

#make smooth angle transitions by turning the steering wheel based on the difference of the current angle

#and the predicted angle

smoothed\_angle+=0.2\*pow(abs((degrees-smoothed\_angle)), 2.0/3.0)\*(degrees-smoothed\_angle)/abs(degrees-smoothed\_angle)

M=cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed\_angle,1)

dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow("steering wheel", dst)

cap.release()

cv2.destroyAllWindows()

**#Running Simulator in Real time**

import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

import model

import cv2

from subprocess import call

import os

import imutils

#check if on windows OS

windows = False

if os.name == 'nt':

windows = True

sess = tf.InteractiveSession()

saver = tf.train.Saver()

saver.restore(sess, "save/model.ckpt")

img = cv2.imread('save/steering\_wheel\_image.jpg',0)

rows,cols = img.shape

smoothed\_angle = 0

cap = cv2.VideoCapture(0)

while(cv2.waitKey(10) != ord('q')):

ret, frame = cap.read()

frame = imutils.resize(frame, width=455)

frame = imutils.resize(frame, height=256)

image = cv2.resize(frame, (200, 66)) / 255.0

degrees=model.y.eval(feed\_dict={model.x:[image], model.keep\_prob: 1.0})[0][0] \* 180 / 3.14159265

if not windows:

call("clear")

print("Predicted steering angle:"+str(degrees)+" degrees")

cv2.imshow('frame', frame)

#make smooth angle transitions by turning the steering wheel based on the difference of the current angle

smoothed\_angle+=0.2\*pow(abs((degrees-smoothed\_angle)), 2.0/3.0)\*(degrees-smoothed\_angle)/abs(degrees- smoothed\_angle)

M=cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed\_angle,1)

dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow("steering wheel", dst)

cap.release()

cv2.destroyAllWindows()

#### 8. CONCLUSION AND FUTURE WORK

We have empirically demonstrated that CNNs are able to learn the entire task of lane and road

following without manual decomposition into road or lane marking detection, semantic abstraction,

path planning, and control. A small amount of training data from less than a hundred hours of driving

was sufficient to train the car to operate in diverse conditions, on highways, local and residential

roads in sunny, cloudy, and rainy conditions. The CNN is able to learn meaningful road features

from a very sparse training signal (steering alone). The system learns for example to detect the outline of a road without the need of explicit labels during training. More work is needed to improve the robustness of the network, to find methods to verify the robustness, and to improve visualization of the network-internal processing steps.

##### Future Work

A lot of modifications can be made to improve this model like -

* Using a **larger** dataset.
* Changing the model architecture. (Adding Batch Normalization Layer, Dropouts etc.)
* Doing more **hyper parameter tuning** (learning rate, batch size, number of layers, number of units, dropout rate, batch normalisation etc.).
* Keep researching on this topic and optimizing the solution even further.

## BIBLIOGRAPHY

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation

applied to handwritten zip code recognition. Neural Computation, 1(4):541–551, Winter 1989.

URL: http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with

deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and

K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages

1097–1105. Curran Associates, Inc., 2012. URL: http://papers.nips.cc/paper/

4824-imagenet-classification-with-deep-convolutional-neural-networks.

pdf.

[3] L. D. Jackel, D. Sharman, Stenard C. E., Strom B. I., , and D Zuckert. Optical character recognition for

self-service banking. AT&T Technical Journal, 74(1):16–24, 1995.

[4] Large scale visual recognition challenge (ILSVRC). URL: http://www.image-net.org/

challenges/LSVRC/.

[5] Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004.

Final technical report. URL: http://net-scale.com/doc/net-scale-dave-report.pdf.

[6] Dean A. Pomerleau. ALVINN, an autonomous land vehicle in a neural network. Technical report,

Carnegie Mellon University, 1989. URL: http://repository.cmu.edu/cgi/viewcontent.

cgi?article=2874&context=compsci.

[7] Wikipedia.org. DARPA LAGR program. http://en.wikipedia.org/wiki/DARPA\_LAGR\_

Program.

[8] Danwei Wang and Feng Qi. Trajectory planning for a four-wheel-steering vehicle. In Proceedings of

the 2001 IEEE International Conference on Robotics & Automation, May 21–26 2001. URL: http:

//www.ntu.edu.sg/home/edwwang/confpapers/wdwicar01.pdf.

[9] DAVE 2 driving a lincoln. URL: https://drive.google.com/open?id=

0B9raQzOpizn1TkRIa241ZnBEcjQ.