Cairo University
Faculty of Computers and Information
Department of Computer Sciences

Simulated Self-Driving Car

Supervised by
Dr. Hanaa Bayoumy

Implemented by

20166033	**Mohamed Essam Abd-El Hamid**
20166020	**Ammar Ashraf Khedr Mousa**
20156011	**Mazen Amr Ahmed**

Graduation Project
Academic Year 2018-2019
Final Documentation

ACKNOWLEDGMENT

We would like to offer our most sincere thanks and gratitude to Dr Hanaa Bayoumy for her help and guidance throughout the project and her continuous support and help for making this work possible.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

YOLO	You only look once
CNN	convolutional neural network
OPENCV	Open computer vision

Chapter One

1 Introduction

1.1 Problem Definition

The Self-Driving Car has a very rich domain. There are a number of factors that affect the driving process such as the speed of the car, obeying the traffic rules, the position of other vehicles on the road, the speed at which those vehicles are operating and many more. The system is not fully aware of all the factors that affect its operation and many a times it is necessary to respond quickly to avoid any untoward incident. It is because of this reason that the domain is one of the most challenging and intriguing in the field of Artificial Intelligence.

1.2 Motivation

- **Reduction in Accidents** It is estimated that over 33000 people are killed every year in the US alone due to vehicular crashes and over a million people worldwide. Ninety percent of the accidents are due to human error. The Self-Driving Car can reduce the accidents occurring due to human error.

- **Environment-Friendly** The Self-Driving Car can save 10-15% on fuel by operating at optimal speeds thereby saving fuel

- **Increased Roadway Capacity** The cars can maintain a fixed distance between each other thereby increasing the road capacity. An increased road capacity results in more convenient travel and a reduction in congestion.

- **Increased Productivity** It is estimated that in the United States people who drive cars spent around 157 hours per year driving. The task of driving being taken over by the car, productive work can be done during the travel time such as replying to emails, preparing for a meeting, preparing slides for presentation or even take some well deserved nap.

1.3 Project Objective

The benefits of fully autonomous vehicles can go way beyond removing the need of a human driver. Transportation services such as Uber will start using self-driving cars instead of human drivers and might become a cheaper and better alternative for end consumers than owning a car. This will represent a shift in the way cities are planned, as fewer parking places will be needed, and most importantly, in a smart city with most of its vehicles being connected and autonomous, traffic optimization will be able to be heavily applied by coordinating movement.

We will apply computer vision and deep learning techniques to create an automated vehicle that can detect lane lines and objects, predict steering angles and take decisions depending on the traffic lights.

1.4 Gantt chart

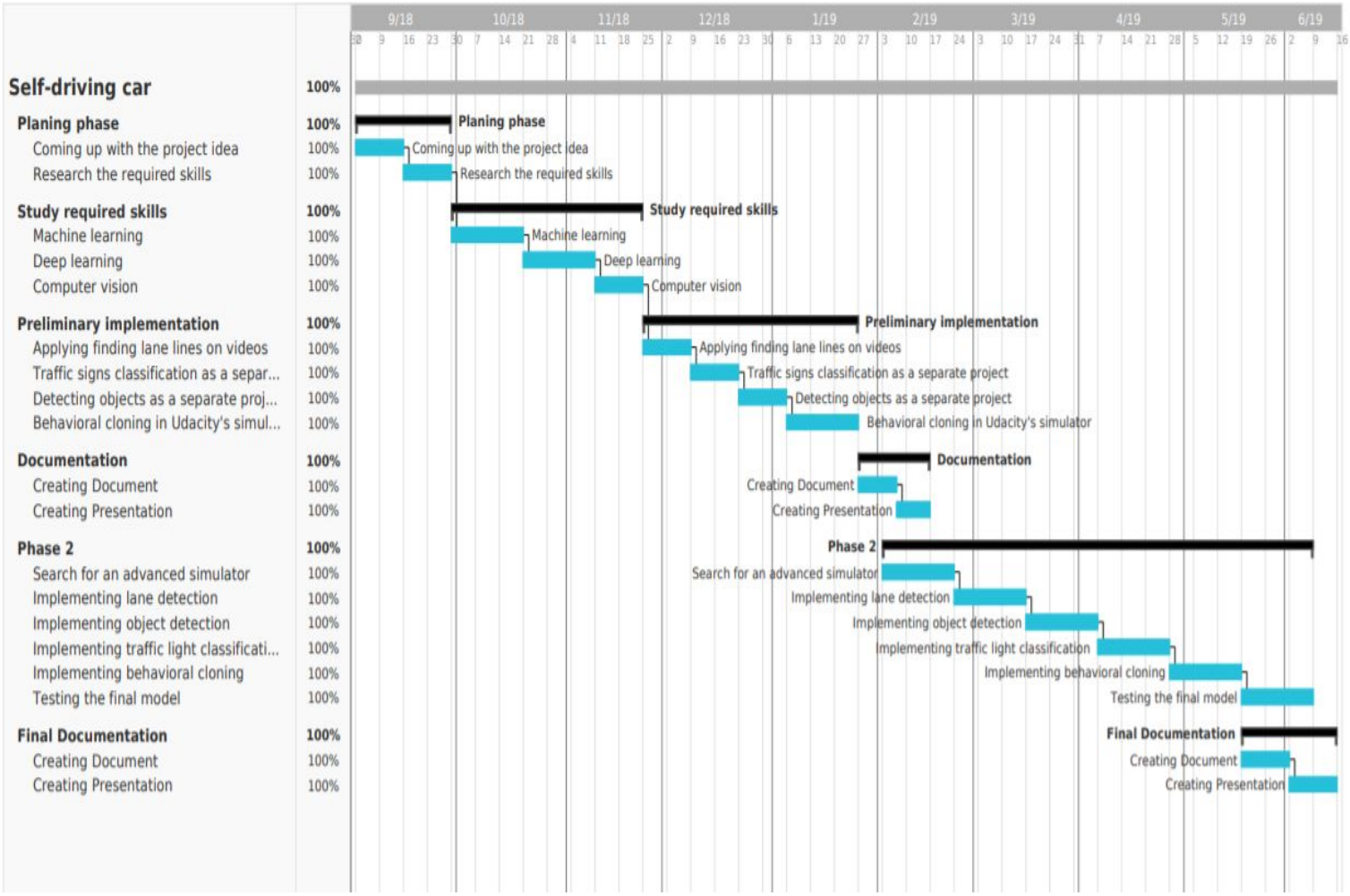

Figure1.1

1.5 Project development Methodology

1.5.1 deep learning

Deep learning is a subfield of machine learning, which is a subfield of AI, Deep learning is concerned with Neural Networks and their usage.

Neural Networks are an algorithms family in Machine Learning which is loosely inspired by biology, these networks consists of layers on top of layers of nonlinearities

that act as hierarchical feature extractors, they have shown great performance in Computer Vision and Natural Language Processing.

An algorithm is deep if the input is passed through several non-linearities before being output. Most modern learning algorithms (including decision trees and SVMs and naive bayes) are "shallow".

1.5.2 computer vision

Humans use their eyes and their brains to see and visually sense the world around them. Computer vision is the science that aims to give a similar, if not better, capability to a machine or computer.

Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding.

Recorded Video Hours are increasing rapidly due to increased use of surveillance cameras or personal cameras. Here comes the use for Computer Vision to analyze these long hours that will need huge number of human workers to be done and will not be that accurate. From knowing what actions are happening in the video frames to describing it with captions.

The applications of computer vision are numerous and include:

- agriculture
- augmented reality
- autonomous vehicles
- biometrics
- character recognition
- forensics
- industrial quality inspection
- face recognition

1.5.3 object detection

Object detection is one of the classical problems in computer vision:

Recognize what the objects are inside a given image and also where they are in the image.

Detection is a more complex problem than classification, which can also recognize objects but doesn't tell you exactly where the object is located in the image — and it won't work for images that contain more than one object.

Object detection is actually a two-part process, image classification and then image localization. Image classification is determining what the objects in the image are, like a car or a person, while image localization is providing the specific location of these objects, as seen by the bounding boxes above.

To perform image classification, a convolutional neural network is trained to recognize various objects, like traffic lights and pedestrians. A convolutional neural network performs convolution operations on images in order to classify them.

1.6 Tools

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

1.7 Report Organization

- Chapter II : Related work including papers and systems related to self driving cars

- Chapter III : Project specification
  - Dataset
  - Tools
  - Algorithms
  - System inputs and outputs
  - Obstacles

- Chapter IV: System design how each component in the system works independently using algorithms and how they work interdependently

- Chapter V: Implementation and testing
  System running and samples of the applied test cases.

Chapter Two

2 Related Work

2.1 End to End Learning for Self-Driving Cars

This paper was published in 2016 by NVIDIA to demonstrate that the behavioral cloning approach in self-driving cars is applicable and has extremely high accuracy on unexplored and unseen roads. The paper talks mainly about their DAVE-2 project that was carried out in its initial stage in a simulator. The testing of the model was done on a track that is completely different from the training track. The performance of the car was astounding so the team decided to reproduce this experiment on a real car in a real-world track. The steps were similar to the previous attempt. The performance wasn't perfect, but the system did drive the car for 98% of the time leaving the human just 2% of the driving to do.

2.2 Case-Based Prediction of Teen Driver Behavior and Skill

This paper on Case Based Prediction of Teen Driver Behavior and Skill uses case based techniques to model vehicle control behavior of teens. The paper employed a pure case based reasoning approach to predict the behavior of the driver. The problem of predicting driver behavior is the regression task of predicting the values of steer, throttle and brake for a given situation based on previous experiences of cases.

2.3 Understanding and Modeling the HumanDriver

This paper discusses a driver model which models the human driver. The paper describes the sensory limitations, steering control calculation, speed control, path planning and prediction capability of the driver model. Our project accounts for the sensory limitations using the random event generators. The prediction of the possible next state(in case of an exceptional event) and the corresponding action it should perform is handled by the case based reasoning system. The controller component of the system uses the retrieved case to manipulate the speed of the car.

2.4 A review of near-collision driver behavior models

This paper describes a number of near-collision driver behavior models such as avoidance by braking, avoidance by steering and a combination of both. The paper describes the application of these models depending on the situation. For example, if there is a scenario of head-on collision, then the best approach might be to brake as well as steer towards the left or right. This not only reduces the impact of collision(if it occurs) but by moving away from line of impact, the chances of collision are reduced.

2.5 Autonomous Driving in Urban Environments: Boss and the Urban Challenge

This paper describes Boss, an autonomous vehicle that uses on-board sensors and a three layered planning system to drive in urban environments. The first layer decides the route to take, the second layer decides the behavior of the car such as when to change lanes and the third layer performs actions to avoid obstacles. This is quite similar to the approach followed by our system.

2.6 Detection, Prediction, and Avoidance of Dynamic Obstacles in Urban Environments

This paper discusses the techniques to predict the future location and speed of a dynamic object in an urban environment so that handling of exceptional events involving the dynamic object can be performed. Our system performs this future prediction based on the similarity of the retrieved case. The retrieved case produces a plan which allows the system to determine the action to be taken to handle the event.

Chapter Three

3 PROJECT SPECIFICATIONS

3.1 DATASET

Building a model in any machine learning system requires a dataset that can be used to train and evaluate the model, to start with our process we needed to collect our dataset from manually driving around in GTA V.

Each row in our dataset contains:

- The captured frame
- Whether the car is throttling or not
- The steering angle
- Speed of the car
- Direction the car is going

3.1.1 Data collection:

- The training data was collected by driving a car in gta V in clear weather conditions and specific lighting conditions
- Data was acquired using a specific kind of vehicle
- Initially, this data is generated from a human agent (human driver) then the data is modified and used to autonomously drive the vehicle
- The image, speed, throttle, steering, and direction are saved in a file with .h5 format

3.1.2 Data balancing:

A balanced dataset is the one that contains equal or almost equal number of samples for each label class. To balance our data we started by dividing the data into bins and using fixed amount of frames from each bin which provides us more balanced data.

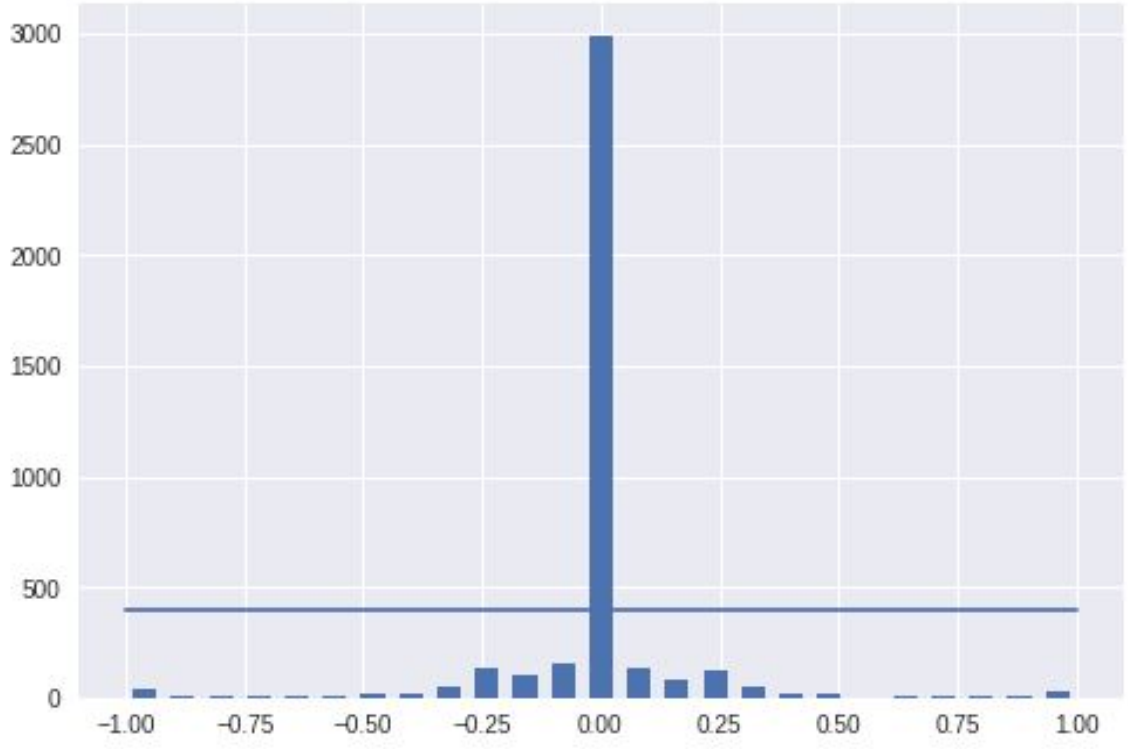

figure 3.1

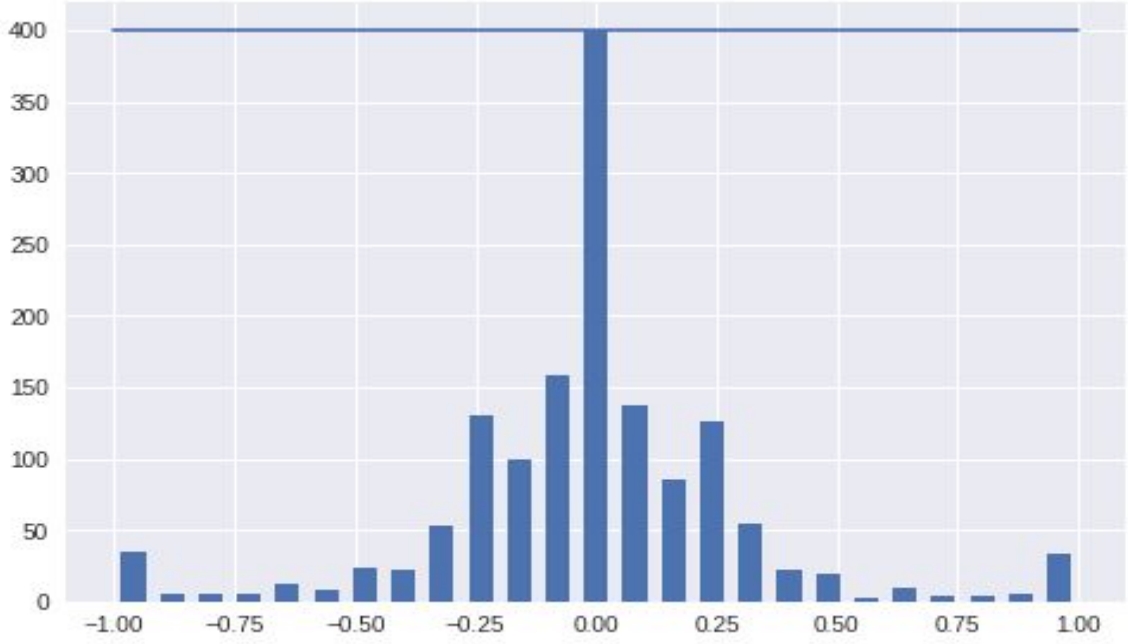

figure 3.2

3.1.3 Data 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.

3.2 TOOLS:

3.2.1 Tensorflow

TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks.

3.2.2 Keras

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

3.2.3 Darknet

Darknet is a framework to train neural networks, it is open source and written in C/CUDA and serves as the basis for YOLO.

Darknet is used as the framework for training YOLO, meaning it sets the architecture of the network.

3.2.4 Darkflow

DarkFlow is a network builder adapted from Darknet, it allows building TensorFlow networks from cfg. files and loading pre trained weights. We will use it to run YOLO.

3.3 ALGORITHMS AND TECHNIQUES

3.3.1 Lane finding

Identifying lanes of the road is very common task that human driver performs. This is important to keep the vehicle in the constraints of the lane. This is also very critical task for an autonomous vehicle to perform. And very simple Lane Detection pipeline is possible with simple Computer Vision techniques.

3.3.1.1 Lane Detection Pipeline:

1. Convert original image to grayscale.
2. Darkened the grayscale image (this help in reducing contrast from discoloured regions of road)
3. Convert original image to HLS colour space.
4. Isolate yellow from HLS to get yellow mask. (for yellow lane markings)
5. Isolate white from HLS to get white mask. (for white lane markings)
6. Apply slight Gaussian Blur.
7. Apply canny Edge Detector (adjust the thresholds — trial and error) to get edges.
8. Define Region of Interest. This helps in weeding out unwanted edges detected by canny edge detector.
9. Retrieve Hough lines.
10. Consolidate and extrapolate the Hough lines and draw them on original image.

3.3.1.2 hough line transform

- The Hough Line Transform is a transform used to detect straight lines.
- To apply the Transform, first an edge detection preprocessing is desirable.

How does it work?

a line in the image space can be expressed with two variables. For example:

- In the Cartesian coordinate system: Parameters: (m,b).
- In the Polar coordinate system: Parameters: (r,θ)

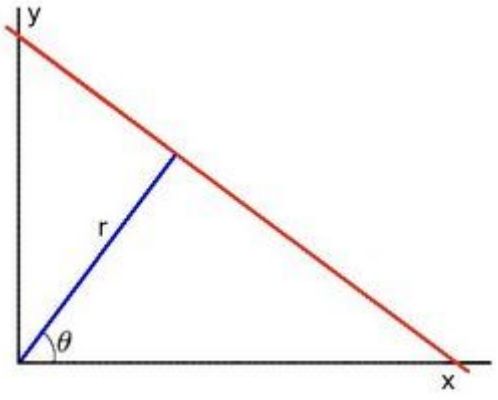

figure 3.3

For Hough Transforms, we will express lines in the *Polar system*. Hence, a line equation can be written as:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

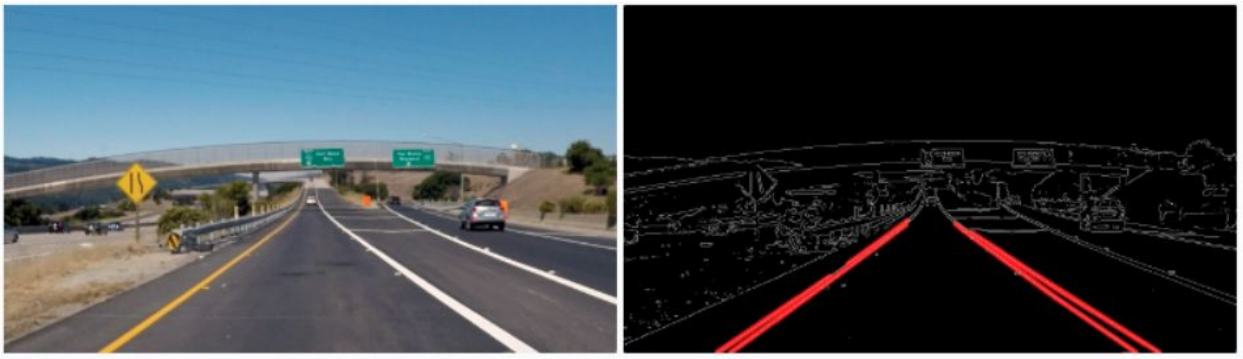

figure 3.4

3.3.1.2.1 Canny edges

the Canny algorithm aims to satisfy three main criteria:

- Low error rate: Meaning a good detection of only existent edges.
- Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.
- Minimal response: Only one detector response per edge.

Steps:

1- Filter out any noise. The Gaussian filter is used for this purpose. An example of a Gaussian kernel of size=5 that might be used is shown below:

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2- Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:

a- Apply a pair of convolution masks in x and y directions:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

b- Find the gradient strength and direction with:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan(\frac{G_y}{G_x})$$

The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)

3- *Non-maximum* suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.

4-*Hysteresis*: The final step. Canny does use two thresholds (upper and lower):

1. If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge
2. If a pixel gradient value is below the *lower* threshold, then it is rejected.
3. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

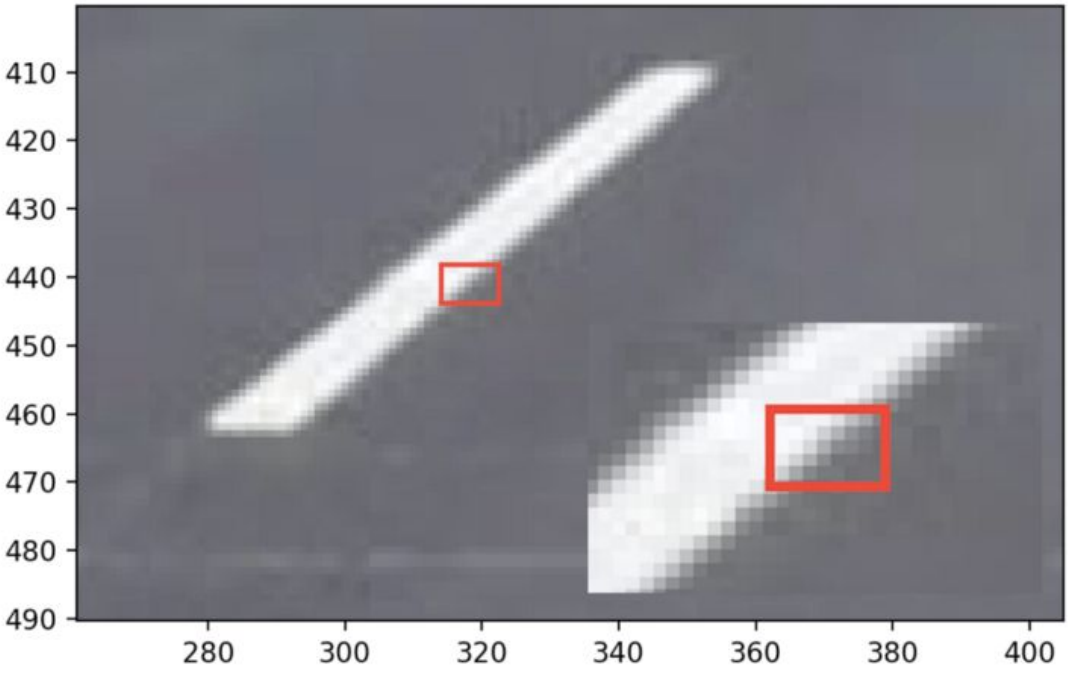

figure 3.5

3.3.1.2.1 Segmenting lane area

We will handcraft a triangular mask to segment the lane area and discard the irrelevant areas in the frame to increase the effectiveness of our later stages

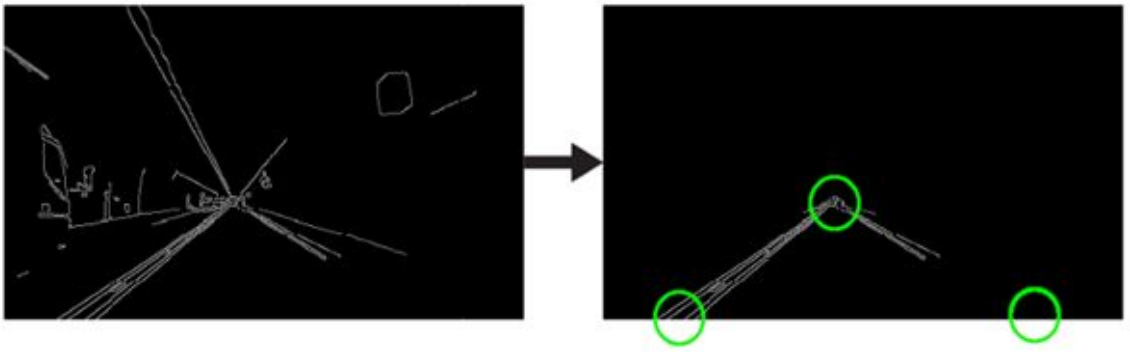

Figure 3.6

3.3.1.2.1 Hough Space

In the Cartesian coordinate system, we can represent a straight line as y = mx + b by plotting y against x. However, we can also represent this line as a single point in Hough space by plotting b against m. For example, a line with the equation y = 2x + 1 may be represented as (2, 1) in Hough space.

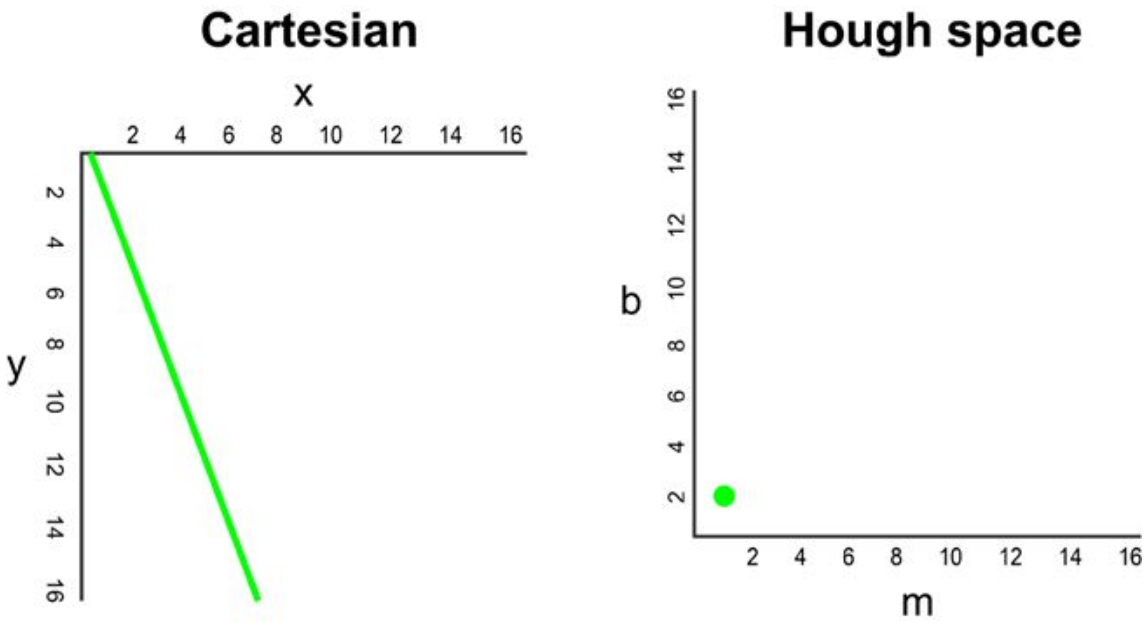

Figure 3.7

Now, what if instead of a line, we had to plot a point in the Cartesian coordinate system. There are many possible lines which can pass through this point, each line with different values for parameters m and b. For example, a point at (2, 12) can be passed by y = 2x + 8, y = 3x + 6, y = 4x + 4, y = 5x + 2, y = 6x, and so on. These possible lines can be plotted in Hough space as (2, 8), (3, 6), (4, 4), (5, 2), (6, 0). Notice that this produces a line of m against b coordinates in Hough space.

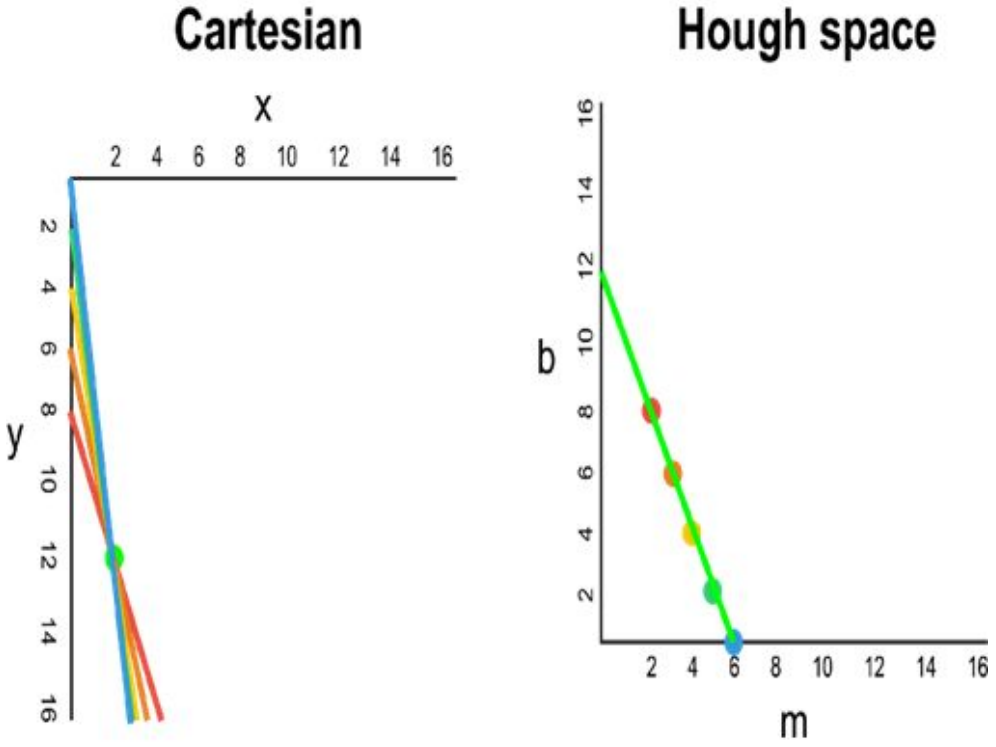

Figure 3.8

Whenever we see a series of points in a Cartesian coordinate system and know that these points are connected by some line, we can find the equation of that line by first plotting each point in the Cartesian coordinate system to the corresponding line in Hough space, then finding the point of intersection in Hough space. The point of intersection in Hough space represents the m and b values that pass consistently through all of the points in the series.

3.3.2 Object detection

Object detection is the craft of detecting instances of a certain class, like animals, humans and many more in an image or video.

3.3.2.1 YOLO

You only look once (YOLO) is a state-of-the-art, real-time object detection system. On a Titan X it processes images at 40-90 FPS and has a mAP on VOC 2007 of 78.6% and a mAP of 48.1% on COCO test-dev.

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
Old YOLO	VOC 2007+2012	2007	63.4	40.19 Bn	45		link
SSD300	VOC 2007+2012	2007	74.3	-	46		link
SSD500	VOC 2007+2012	2007	76.8	-	19		link
YOLOv2	VOC 2007+2012	2007	76.8	34.90 Bn	67	cfg	weights
YOLOv2 544x544	VOC 2007+2012	2007	78.6	59.68 Bn	40	cfg	weights
Tiny YOLO	VOC 2007+2012	2007	57.1	6.97 Bn	207	cfg	weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200	cfg	weights

Figure 3.9

3.3.2.1.1 How YOLO solves exhaustive search in object detection

CNNs can usually only classify images with a single object that take up a sizable portion of it. To solve this problem, we can use sliding windows!
As we slide the window over the image, we take the resulting image patch and run it through the convolutional neural network to see if it corresponds to any possible object. If it's just an image of the road or the sky, it would be a false prediction. If it's an image of a car or a person, it would return as a true prediction.

But what if there's an object a lot larger or smaller than the window size? It wouldn't be detected! So, we'll have to use multiple window sizes and slide them over the image. Since this can be very computationally expensive and take lots of time

3.3.2.1.2 How YOLO works

YOLO takes a completely different approach. It's not a traditional classifier that is repurposed to be an object detector. YOLO actually looks at the image just once. YOLO divides up the image into a grid of 13 by 13 cells:

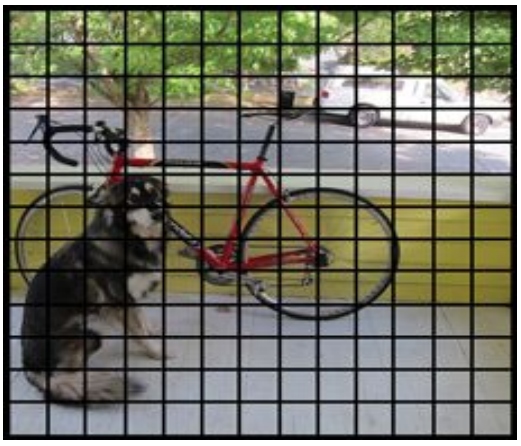

figure 3.10

Each of these cells is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses an object.
YOLO also outputs a confidence score that tells us how certain it is that the predicted bounding box actually encloses some object. This score doesn't say anything about what kind of object is in the box, just if the shape of the box is any good.
The predicted bounding boxes may look something like the following (the higher the confidence score, the fatter the box is drawn):

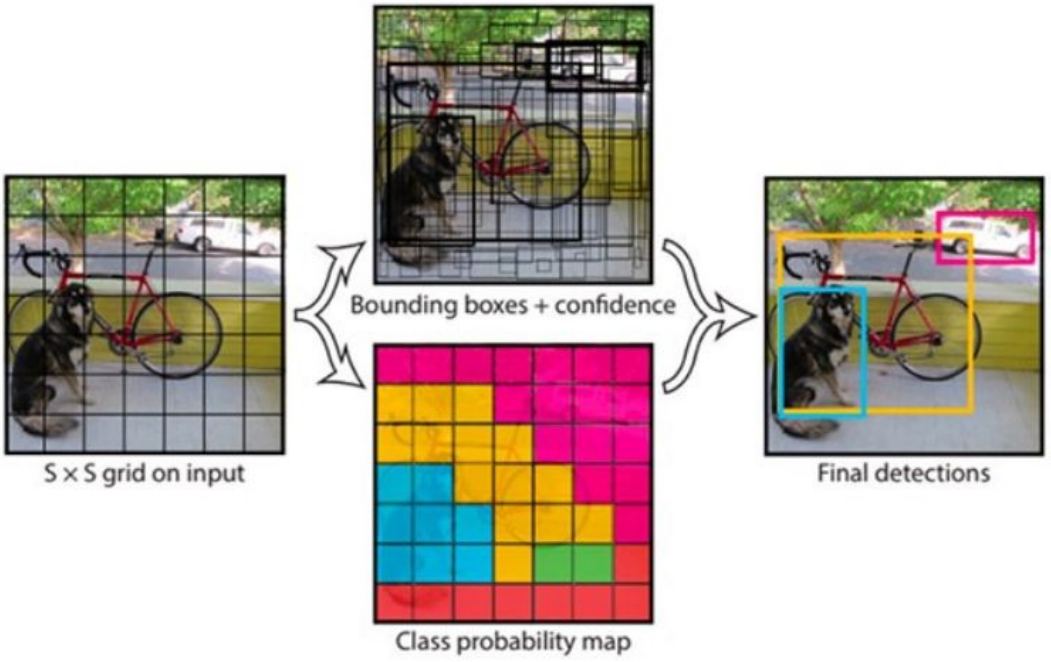

figure 3.11

For each bounding box, the cell also predicts a class. This works just like a classifier: it gives a probability distribution over all the possible classes. The version of YOLO we're using is trained on the PASCAL VOC dataset, which can detect many different classes including cars, persons, traffic lights, etc.

The confidence score for the bounding box and the class prediction are combined into one final score that tells us the probability that this bounding box contains a specific type of object. For example, the big fat yellow box on the left is 85% sure it contains the object "dog":

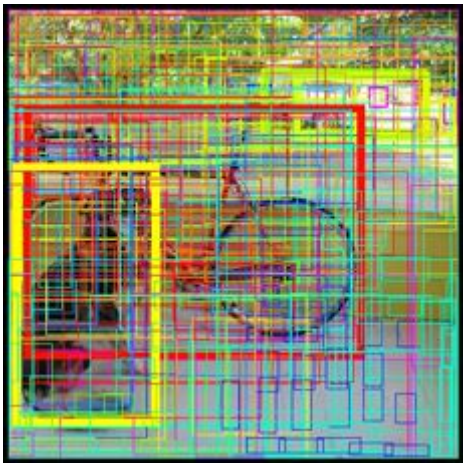

figure 3.12

3.3.2.1.3 YOLO network architecture

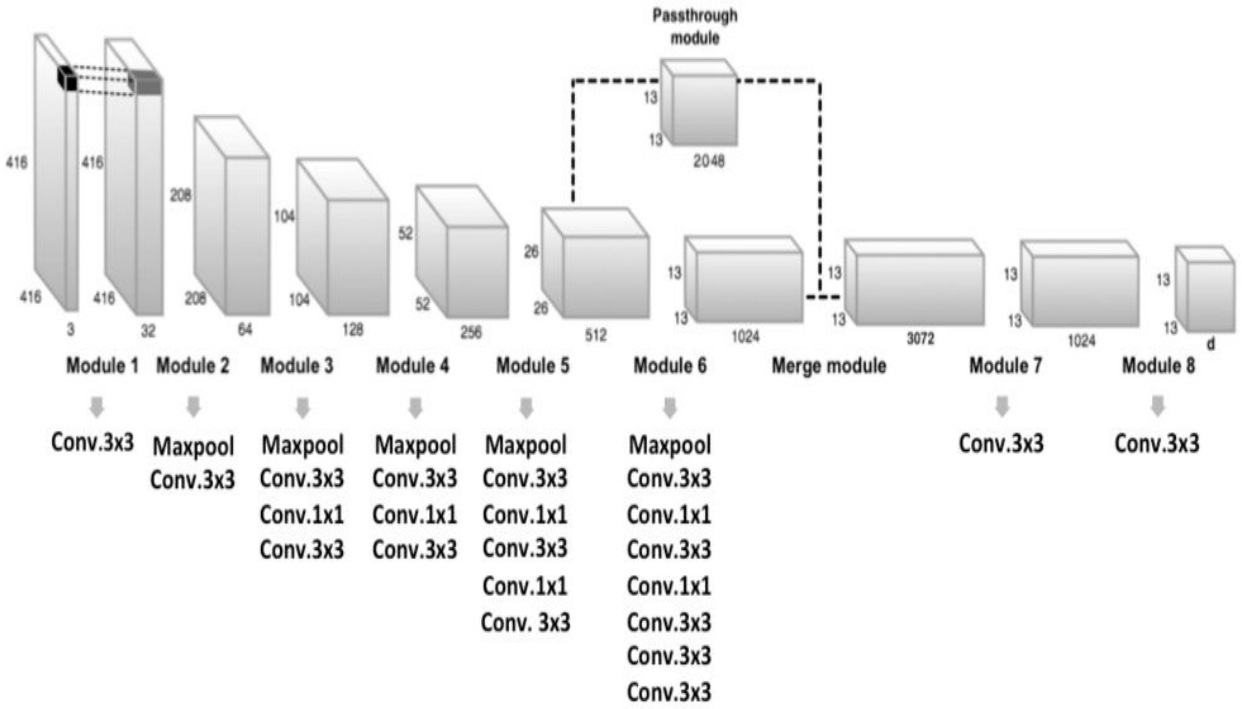

figure 3.13

Loss function

YOLO predicts multiple bounding boxes per grid cell. To compute the loss for the true positive, we only want one of them to be **responsible** for the object. For this purpose, we select the one with the highest IoU (intersection over union) with the ground truth. This strategy leads to specialization among the bounding box predictions. Each prediction gets better at predicting certain sizes and aspect ratios.

YOLO uses sum-squared error between the predictions and the ground truth to calculate loss. The loss function is composed of:

- The **classification loss**.
- The **localization loss** (errors between the predicted boundary box and the ground truth).
- The **confidence loss** (the objectness of the box).

Classification loss

If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

where

$\mathbb{1}_i^{obj} = 1$ if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

Localization loss

The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.

Confidence loss

If an object is detected in the box, the confidence loss (measuring the objectness of the box) is:

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

where

$\hat{C}_i$ is the box confidence score of the box j in cell i.

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

Most boxes do not contain any objects. This causes a class imbalance problem, i.e. we train the model to detect background more frequently than detecting objects. To remedy this, we weight this loss down by a factor λnoobj (default: 0.5).

Final loss

The final loss adds localization, confidence and classification losses together.

3.3.3 Behavioral cloning

Behavioral cloning is a method by which human sub cognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. It can also be used for training.

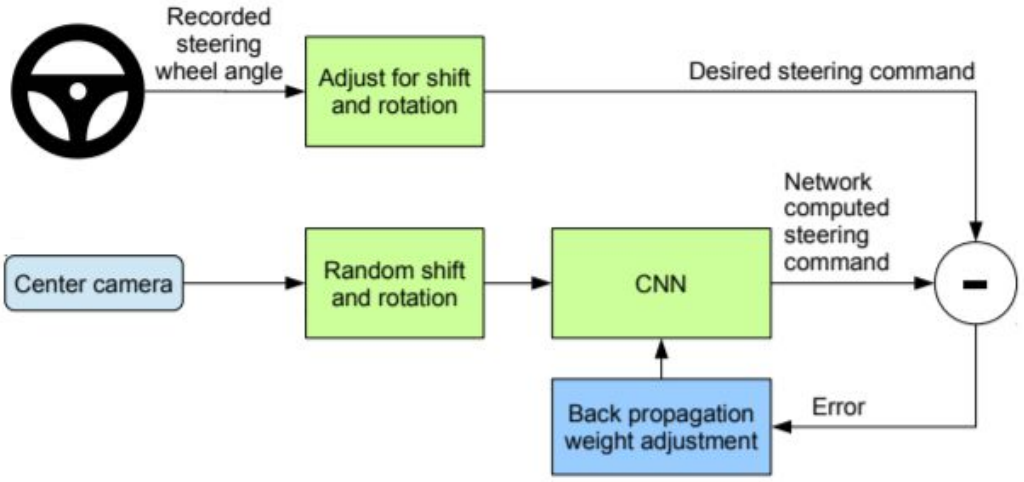

figure 3.14

3.3.3.1 CNN Architecture

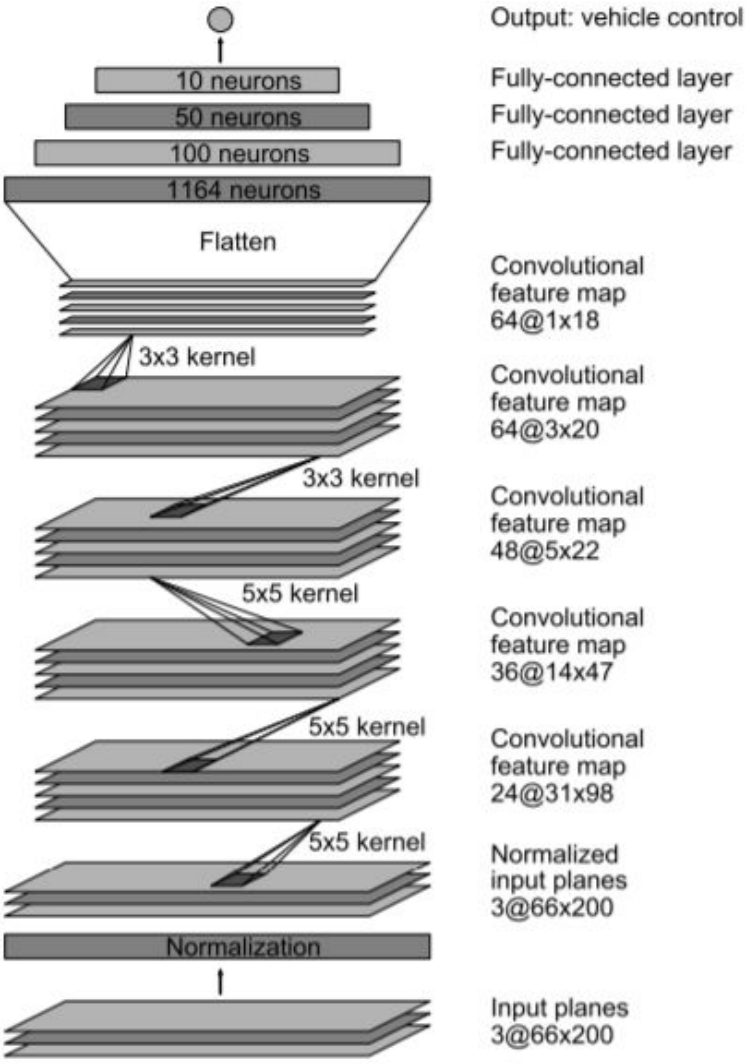

figure 3.15

3.4 SYSTEM'S INPUTS/OUTPUTS

The input to the system will be the state of the current environment as images while the output of the system will be the action that it should take in this state.

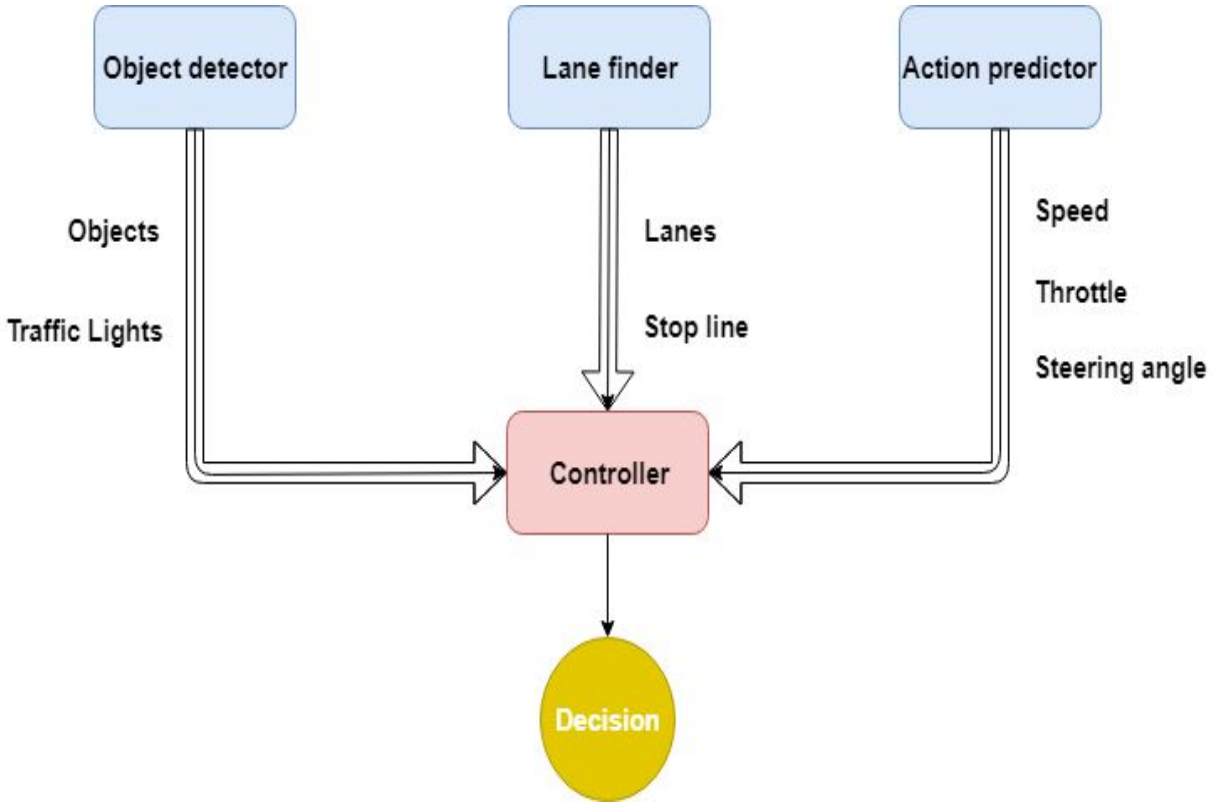

Figure 3.16

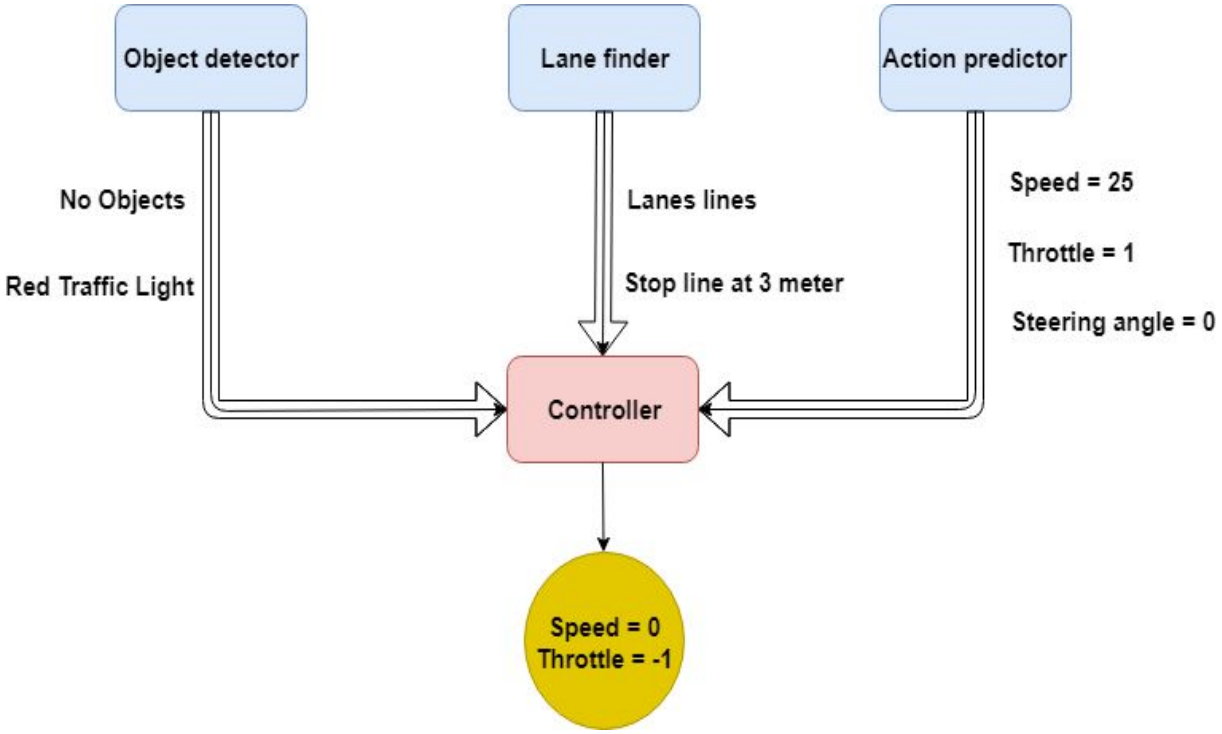

Figure 3.17

3.5 OBSTACLES

3.5.1 computational constraints

Our computers lack the computational power to train our behavioral cloning model on a variety of environmental conditions such as different weather conditions, different times of the day, and from the perspective of different vehicles.

The way we overcome this obstacle was by training our model on a specific environment state having a clear weather and constant time of noon.

3.6 FUNCTIONAL REQUIREMENTS

ID **Requirement**

RQ1 The system shall automatically a car from the starting location to the ending location

RQ2 The system shall devise a route for the car to drive

RQ3 The system shall use a MAP of simulator to find current location

RQ4 The system shall assume constant speed

RQ5 The system shall identify traffic lights

RQ6 The system shall identify the color of traffic lights

RQ7 The system shall stop the car if the traffic light is red

RQ8 The system shall identify road objects and obstacles

RQ9 The system shall classify road objects and obstacles

RQ10 The system shall calculate distance between road objects and the car

RQ11 The system shall stop the car when the distance between it and the upcoming object is small

RQ12 The system shall identify lane lines

RQ13 The system shall enforce car movement between lane lines

RQ 14 The system shall use the trained model to turn right

RQ15 The system shall use the trained model to go forward

RQ16 The system shall use the trained model to go backward

RQ17 The system shall self-update by training

RQ18 The system shall be easily accessible

RQ19 When the car is turned off the system shall shut down

3.7 Non-Functional requirement

ID	Requirement
RQ1	The system shall have fast response time
RQ2	The system shall have scalability
RQ3	The system shall be reliable
RQ4	The system shall be serviceable
RQ5	The system data shall have proper integrity
RQ6	The system shall have excellent usability
RQ7	The system shall have interoperability

3.8 Product Functions

- Autonomous motion
- Object detection
- Obstacle avoidance
- Follows traffic signs

3.9 SCENARIOS

3.9.1 Scenario: Encountering a red traffic light

Description

The car encounters a red traffic light in front of it on the road. The car makes the speed equals to zero and throttle equals to 0 and it doesn't exceed the stop line, then resumes its motion if the traffic light becomes green.

Functional Response

The object detection component detects the traffic light and its color is red, sends this information to the controller. The controller doesn't follow the predictions of the trained model and takes the decision of stopping the car. Once the traffic light is green, trained model predictions is followed and the car resumes its motion.

3.9.2 Scenario: There is a car in front of our car

Description

The car encounters a car in front of it on the road and distance between them is too small. The car makes the speed equals to zero, then resumes its motion if the distance between the two cars becomes safe and there is no risk.

Functional Response

The object detection component detects a car in front of our car and the distance between the two cars, sends this info to controller. The controller doesn't follow the predictions of trained model and take the decision of stopping the car. Once distance becomes safe, model predictions is followed and the car resumes its motion.

3.9.3 Scenario: The car reached destination

Description

The car reaches the preset destination on the map.

Functional Response

The car follows the map of simulator and when it reaches the point the user chose the car will stop.

3.9.4 Scenario: There is no obstacles in the road

Description

The car takes steering angle and drives to the preset location with appropriate speed

Functional Response

The car takes the predictions from the behavioral cloning trained model and detect lane lines by lane finding component, sends it to controller and the controller enforce car to stay in its lane accordingly

3.9.5 Scenario: Encountering a pedestrian crossing the street

Description

The car encounters a person crossing the street in front of it on the road. The car makes the speed equals to zero and throttle equals to 0.

Functional Response

The object detection component detects the person, sends this information to the controller. The controller doesn't follow the predictions of the trained model and takes the decision of stopping the car.

3.10 Assumption and Dependencies

3.10.1 Assumptions

- The basic assumption is that the roads are ideal with specific weather conditions.
- The time of driving is noon.

3.10.2 Dependencies

- Tensorflow 1.10.0
- Numpy
- Opencv-python
- pywin32
- h5py(For Data management)

- Keras
- Scikit-learn
- Cython
- CUDA 9
- CUDNN 9

3.11 Test cases

3.11.1 Test case 1

Test scenario: Another car is in front of our car
Expected result: Stopping before the car
Actual result: Car successfully stopped before the car
Pass/Fail: Pass
priority: High

3.11.2 Test case 2

Test scenario: When another car was in front of our car and moved away
Expected result: Car automatically resumes its course
Actual result: Successfully resumes its course
Pass/Fail: Pass
priority: Medium

3.11.3 Test case 3

Test scenario: A red traffic light is ahead
Expected result: Successfully stopping before the stop line when the traffic light ahead is red
Actual result: As expected
Pass/Fail: Pass
Priority: High

3.11.4 Test case 4

Test scenario: Traffic light turns green
Expected result: Car automatically resume when the traffic light turns green
Actual result: As expected
Pass/Fail: Pass
Priority: Medium

3.11.5 Test case 5

Test scenario: Check the car at night
Expected result: Car runs successfully
Actual result: Car crashes when the time of driving is not noon
Pass/Fail: Fail
Priority: Low

3.11.6 Test case 6

Test scenario: A car is passing by in another lane
Expected result: The car doesn't stop when a different car is in next lane
Actual result: As expected
Pass/Fail: Pass
Priority: High

3.11.7 Test case 7

Test scenario: Checking the car when a different car color was used
Expected result: Car runs successfully
Actual result: The car crashed when a different car color was used
Pass/Fail: Fail
Priority: Medium

3.11.8 Test case 8

Test scenario: A pedestrian is crossing the road
Expected result: Car stops when a pedestrian is crossing the road
Actual result: As expected
Pass/Fail: Pass
Priority: High

Chapter Four

4 System Design

To achieve the fully autonomous driving mode we depend on 3 main components, 2 of which feed the controller with the surrounding environment while the 3rd component predicts the steering angle the car should have so that the controller decides the right course of action to take.

4.1 Object detector

While the car is moving the object detector component sends the controller the information of the location of objects surrounding the vehicle, their class, and the distance from the vehicle. Also it sends the controller the nearest traffic light and its colour.

Figure 4.1

4.2 LANE DETECTOR

This component detects the lane lines the vehicle is moving between and also if there is a stop line before any traffic light and its relative position to the car.

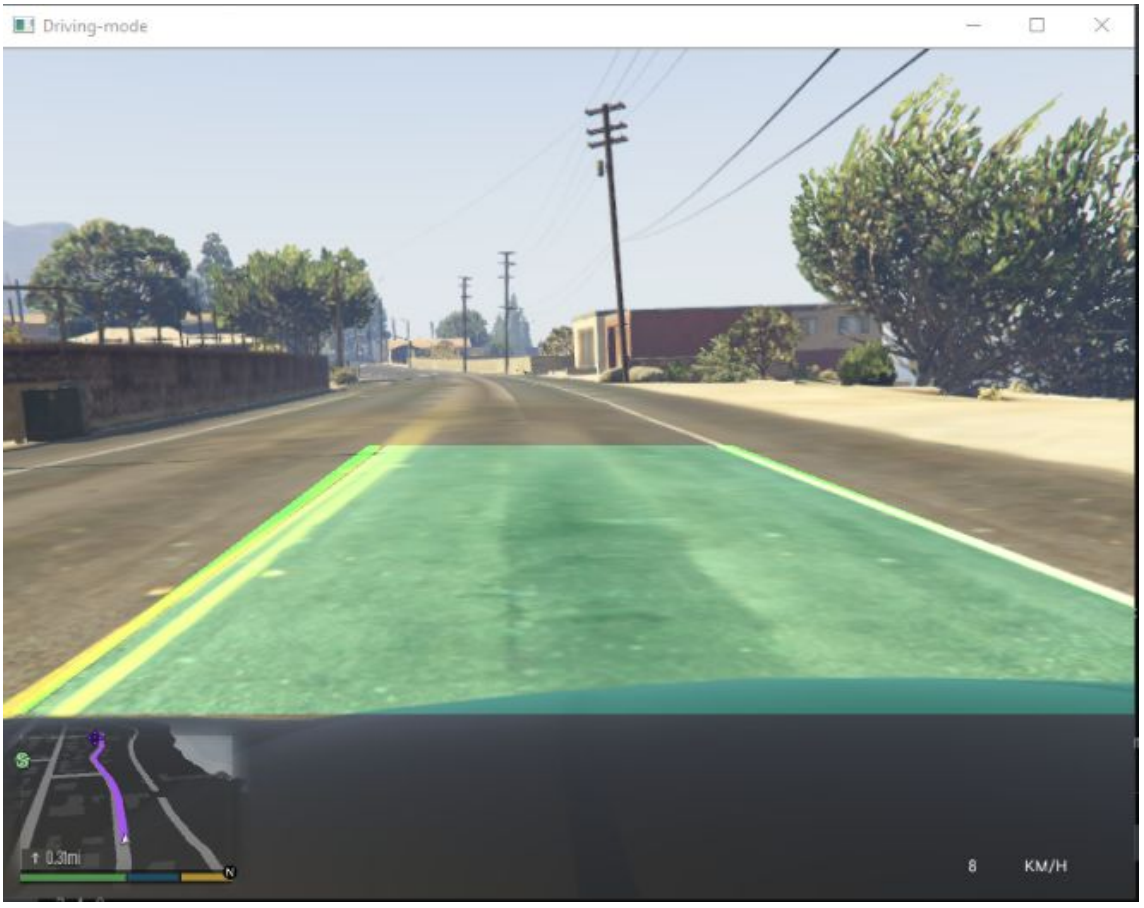

Figure 4.2

Figure 4.3

4.3 PREDICTOR (TRAINED MODEL)

The trained model's purpose is to determine the steering angle that the car should have in this instance of time.

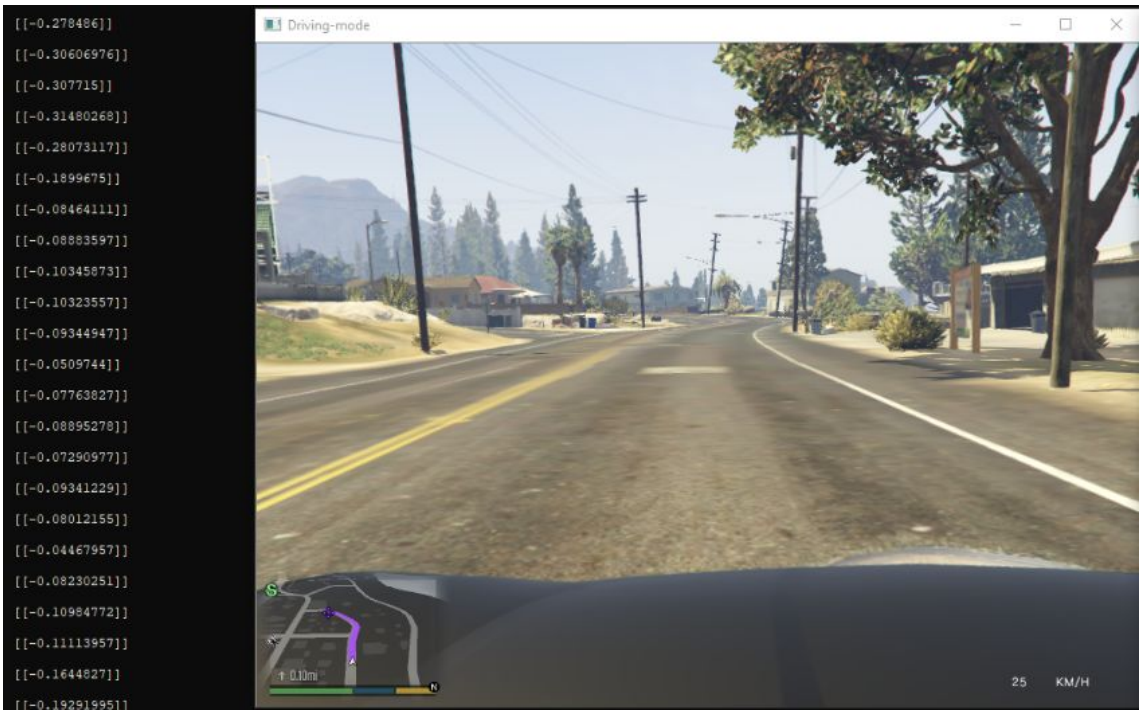

Figure 4.4

4.4 Driver (controller)

The controller uses the input from each component to decide the action to be taken in the current instance of time. For example in the case of having a red traffic light ahead, the controller gets that information from the object detector while the lane detector feeds the controller with the position of the stop line and its relative distance so the car can stop before the stop line.

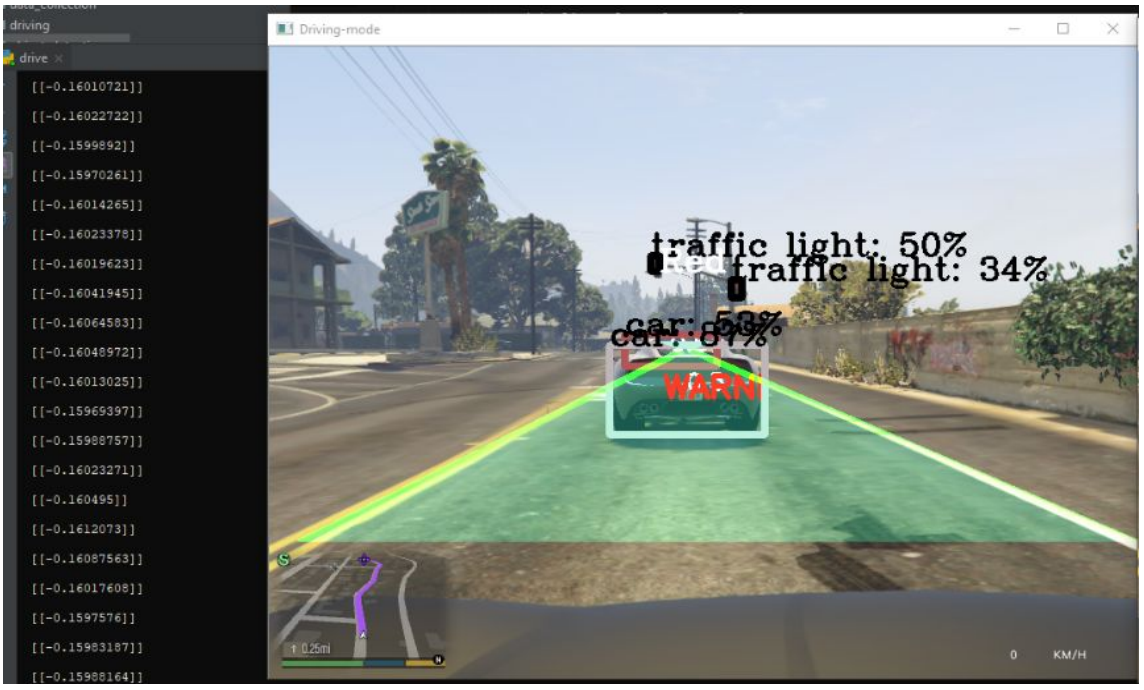
driving
drive
[[-0.16010721]]
[[-0.16022722]]
[[-0.1599892]]
[[-0.15970261]]
[[-0.16014265]]
[[-0.16023378]]
[[-0.16019623]]
[[-0.16041945]]
[[-0.16064583]]
[[-0.16048972]]
[[-0.16013025]]
[[-0.15969397]]
[[-0.15988757]]
[[-0.16023271]]
[[-0.160495]]
[[-0.1612073]]
[[-0.16087563]]
[[-0.16017608]]
[[-0.1597576]]
[[-0.15983187]]
[[-0.15988164]]
Driving-mode
traffic light: 50%
traffic light: 34%
car: 53%
car: 84%
WARN
0.25mi
0
KM/H

Figure 4.5

Chapter Five

5 Implementation and Testing

5.1 Test cases

5.1.1 Test case 1

Figure 5.1

5.1.2 Test case 2

Figure 5.2

5.1.3 Test case 3

Figure 5.3

5.1.4 Test case 4

Figure 5.4

5.1.5 Test case 5

Figure 5.5

5.1.6 Test case 6

Figure 5.6

5.1.7 Test case 7

Figure 5.7

5.1.8 test case 8

Figure 5.8

REFERENCES

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, April 25, 2016. URL: http://arxiv.org/abs/1604. 07316, arXiv:arXiv:1604.07316.

[2] Mariusz Bojarski, Philip Yeres , Bernhard Firner, Beat Anna Choromanaska ,, Urs Muller, Bernhard Firner, "Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car ", April 25, 2017 .

[3] Joseph Redmon, Ali Farhadi " YOLO9000: Better, Faster, Stronger" , April 25, 2016

[4] S Ontanon, Y. Lee, S. Snodgrass, D. Bonfiglio, F.Winston, C. McDonald and A. Gonzalez, "Case-BasedPrediction of Teen Driver Behavior and Skill" ICCBR 2014,LNCS 8765, pp. 375389, 2014.

[5] C. MacAdam, "Understanding and Modeling the HumanDriver" Vehicle System Dynamics 2003, Vol. 40, Nos. 13,pp. 101134

[6] G. Markkula, O Benderius, K. Wolff and M. Wahde, "A review of near-collision driver behavior models"

[7] C. Urmson, J. Anhalt, H. Bae, T. Brown, D. Demitrish, J.Struble, M. Taylor, M. Darms, D. Ferguson ,"AutonomousDriving in Urban Environments: Boss and the UrbanChallenge", June 2008.