Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

Autonomous Driving System in Virtual Environment

Ameya Morajkar
Department of Computer Engineering
Vidyalankar Institute of Technology
Mumbai,India

Dhananjay Nikalje
Department of Computer Engineering
Vidyalankar Institute of Technology
Mumbai, India

Pankaj Vanwari
Department of Computer Engineering
Vidyalankar Institute of Technology
Mumbai, India

Arsh Ahmed Jamadar
Department of Computer Engineering
Vidyalankar Institute of Technology
Mumbai, India

Abstract-Development of Autonomous driving system is now possible due to End-to-end deep learning, which is a modeling strategy that is a response to the success of deep neural networks. Unlike traditional methods, this strategy is not built on feature engineering. Instead, it leverages the power of deep neural networks, along with recent hardware advances (like GPUs) to harness the incredible potential of large amounts of data. It is closer to a human-like learning approach than traditional ML as it lets a neural network map raw input to direct output.

Keywords— Autonomous Driving Systems, End-to-End Deep Learning, Neural Networks, Unity Game Engine, Virtual-Simulators, SDC.

I. INTRODUCTION

Thanks to the availability of various sensors including radars, lidars, camera systems and also wireless communications, driver assistance systems autonomous vehicles have made significant advances in recent years. The main requirements that are imposed to autonomous vehicles are ability to cover long distances in a safer way, while decreasing the rate of accidents and traffic jams, and obeying the traffic rules, all without human interaction. There are 10 trillion automobile miles driven each year worldwide, with complex and novel conditions generating millions of situations in which autonomous vehicles face failure.

Highly intelligent autonomous systems in vehicle are required to take into account a broader range of information about the current road situation and the car itself on the same way as a human driver would process information. For enabling autonomous vehicles to handle adverse driving conditions, such as rain and wet roads, the control algorithm must be able to recognize roads within a tolerable margin of error, using measuring instruments, such as cameras and laser sensors. Autonomous vehicles must quickly make decisions based on incomplete information in situations that programmers often will not have considered, using ethics that must be encoded all too literally in software .

This paper attempts to provide a Autonomous driving system for an virtual environment. We are using two virtual environment for training and developing the autonomous system. The first one is an photo-realistic virtual simulator provided by Microsoft named 'Airsim' , as for now while while developing this project Airsim doesn't have signal system implemented yet, so instead of airsim simulator we are also developing our own simple simulator with signal system using Unity Game Engine.

Using such simulators it is now possible to collect a large amount of data to train autonomous driving models without having to use an actual car. These models can then be fine-tuned using a comparably lesser amount of real-world data and used on actual cars. This technique is called Behavioral Cloning. In this tutorial, we are training a model to learn how to steer a car through a portion of the Landscape map in these simulators using only one of the front facing webcams on the car as visual input and four distance measuring sensor. Our strategy, is to train an end-to-end deep learning model to predict the correct driving control signal (in this case the steering angle) given a frame from the webcam, and the car's current state parameters (speed, steering angle, throttle, distance senor readings etc.).

In the initial part of the paper, we will know about other relevant projects and then understand the process of developing/using virtual environment for training and testing self-driving car model. In the later part of the paper, we would discuss more about our how our model is actually being trained and evaluated. We would present the steps that would be used by our system. And the details of concepts which would be used.

II. BACKGROUND & RELATED WORK

After studying the various available autonomous driving systems we aim to execute our proposed system. In order to do that we studied various papers based on the same to understand different concepts adopted in each one of them.

Nvidia Developers [1] 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

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

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).

Zheng Wang [2] Modified a RC car to handle three tasks: self-driving on the track, stop sign and traffic light detection, and front collision avoidance. His system consisted of three subsystems: input unit (camera, ultrasonic sensor), processing unit (computer) and RC car control unit. The camera was connected to Raspberry Pi which sent data back Processing Unit via TCP server, the on-processing side Neural networks are used to predict next move of the car.

There are 38,400 (320×120) nodes in the input layer and 32 nodes in the hidden layer and 4 nodes in output layer. The number of nodes in the hidden layer is chosen fairly arbitrary. There are four nodes in the output layer where each node corresponds to the steering control instructions: left, right, forward and reverse respectively i.e. the possible moves for car.

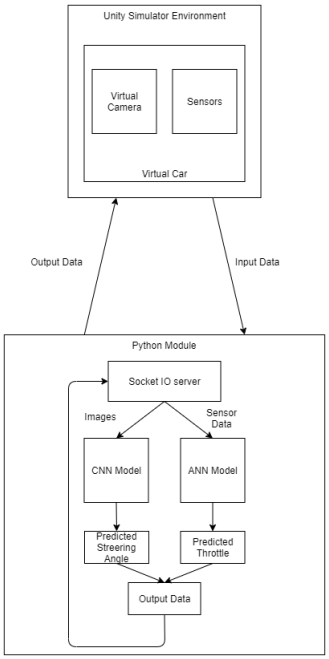
The system also had object detection feature which detected stop sign and red signal. This project adapted the shape-based approach and used Haar feature-based cascade classifiers for object detection. Since each object requires its own classifier and follows the same process in training and detection, this project only focused on stop sign and traffic light detection.

"How to Train Your Self Driving Car Using Deep Learning" by
Towardsdatascience.com [5] demonstrates method on how to Train an end-to-end deep learning model that would let a car drive by itself around the track in a driving simulator. It is a supervised regression problem between the car steering angles and the road images in real-time from the cameras of a car. In this project, Udacity driving simulator has been used which has two different tracks. One of them was used for collecting training data, and the other one — never seen by the model — as a substitute for the test set. This project was also inspired from [1] Nvidia's End-to-End learning Deep learning for self-driving cars.

"3D Modelling and Visualization based on the Unity game engine – Advantages and Challenges". [7] The paper has been organized with a brief introduction on 3D GIS modelling standards, description of project area and Unity3D game engine as project implementation platform.

III. PROPOSED SYSTEM (IMPLEMENTATION)

The diagram below shows the overall flow of how the system is going to work.



The system mainly consists of two parts:
1. Unity Environment
2. Python Module (Server)

We can manually drive the car in the simulated environment to collect the required data that is stored in the local folder. On training the model on collected data it can be tested using the python module. The system is based on Client-Server architecture where the python module act as server and the virtual car act as a client.

In autonomous driving mode, virtual camera images along with sensor data are transferred to the server. Using the received data, the model predicts the corresponding steering angle and acceleration to be applied which is transferred to the virtual car controller.

A. Simulator / Virtual Environment

For generating data we first need to setup a virtual environment, we are making use of 2 virtual simulator

1) AirSim Simulator

Fig.1.AirSim Simulator

AirSim (Aerial Informatics and Robotics Simulation) is an open-source, cross platform simulator for drones, ground

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

vehicles such as cars and various other objects, built on Epic Games' Unreal Engine 4 as a platform for AI research.

There are two ways you can generate training data from AirSim for deep learning. The easiest way is to simply press the record button provided in the lower right corner. This will start writing pose and images for each frame. The data logging code was modified to get data as per our training model requirement.

2) Developing simulator in Unity 3D

We used the unity game engine to develop a simple 3d simulator environment, which consists of roads and signal systems. The environment is developed using Road Architect. Road architect is a professional quality road system creator featuring dynamic intersections, bridges and many other road objects. The various road objects help to train the neural model with different environmental conditions. This includes roads, bridges, stop-signals, speed-limit signals, etc.

The simulated autonomous car is attached with a virtual camera that helps to collect images required for training the CNN model. As images do not provide enough information about the exact distances of the objects it is necessary to have sensors that directly provides such distances. Such sensors are implemented using Unity's 'Raycast' property. It continuously sends rays from a given point in specific direction. The length of such rays is used and stored as the distance of the objects from the car.

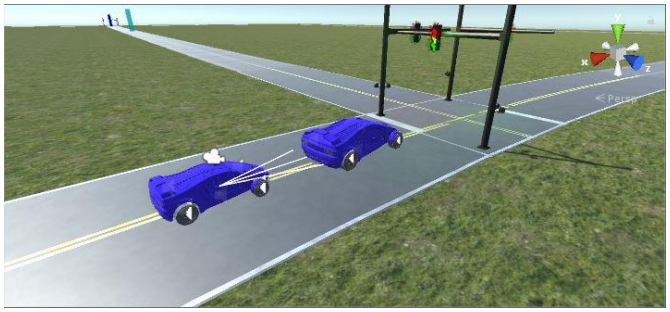


Fig.2. Unity Game Engine

The environment also includes various other cars running at different speeds.

IV. DATASET COLLECTION

The main part of creating a more precise Neural Network Model begins with the collecting of relevant data. Our Untity Simulator has more precisely two modes, one is the Training Mode and the other is the Evaluation Mode.

The dataset collection takes place in the Training Mode. There are three cameras mounted in front of our car in Unity. The location of these cameras are front left, front center and front right to more precisely capture the surrounding environment. In order to collect the dataset, we manually drive the car in the Training Mode. In Training Mode, the simulator captures the images from the three front cameras and stores it in a database. The corresponding human action pertaining to every frame is also captured by the simulator which is stored in an Excel file.

As a result the dataset required to train a Neural Network Model is collected, wherein the input to the model will be the input frame image and the output will be the steering angle. The images captured by the three front cameras are as follows:-

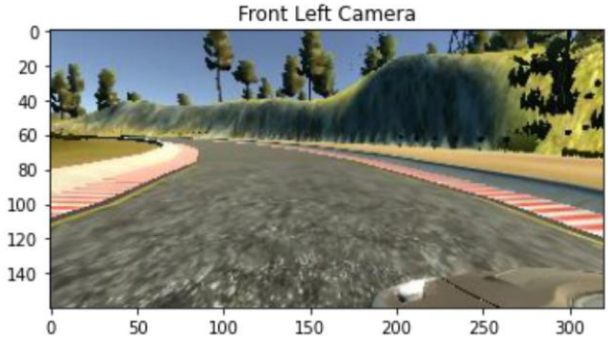


Fig.3. Front Left Camera View

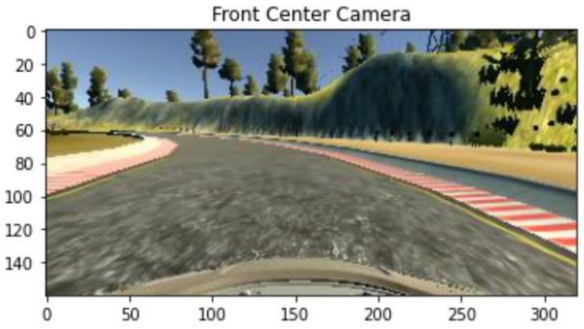


Fig.4. Front Center Camera View

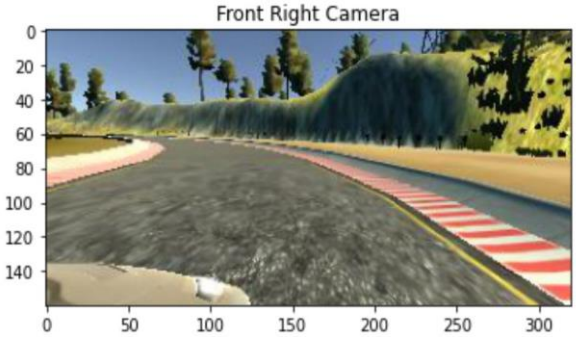


Fig.5. Front Right Camera View

The corresponding steering angle, throttle, deacceleration (reverse) and the current speed recorded pertaining to the input images which is stored in the Excel file is as follows:-

	center	left	right	steering	throttle	reverse	speed
0	center_2019_05_13_18_25_47_377.jpg	left_2019_05_13_18_25_47_377.jpg	right_2019_05_13_18_25_47_377.jpg	0.0	0.0	0	0.000078
1	center_2019_05_13_18_25_47_479.jpg	left_2019_05_13_18_25_47_479.jpg	right_2019_05_13_18_25_47_479.jpg	0.0	0.0	0	0.000079
2	center_2019_05_13_18_25_47_581.jpg	left_2019_05_13_18_25_47_581.jpg	right_2019_05_13_18_25_47_581.jpg	0.0	0.0	0	0.000079
3	center_2019_05_13_18_25_47_684.jpg	left_2019_05_13_18_25_47_684.jpg	right_2019_05_13_18_25_47_684.jpg	0.0	0.0	0	0.000079
4	center_2019_05_13_18_25_47_783.jpg	left_2019_05_13_18_25_47_783.jpg	right_2019_05_13_18_25_47_783.jpg	0.0	0.0	0	0.000078

Fig.6. Recorded Data Table

We collected around 20,000 images by driving the car for around 20 minutes. We tried to capture various different scenarios by driving the car in different ways so as to have a robust dataset.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

The sensor data helps to determine the acceleration to be applied. Currently the car consists of front sensors (Unity Raycast) attached to it. The length of the sensor is stored in CSV file along with the corresponding acceleration of the car at a given point in time. We can attach multiple sensors to car at various positions. The collected data is used to train the deep neural network.

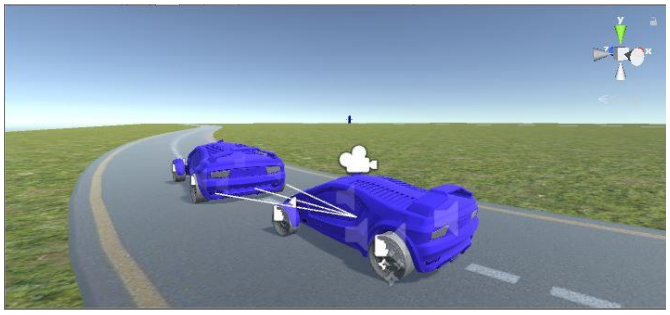


Fig.7. Sensors in Unity Game Engine

	A	B	C	D	E	F	G
1	Steer	Accel	Center	Left	Right	LeftMid	RightMid
2	0	0	1	1	1	1	1
3	0	0	1	1	1	1	1
4	0	0	1	1	1	1	1
5	0	0	1	1	1	1	1
6	0	0	1	1	1	1	1
7	0	0	1	1	1	1	1

Fig.8. Recorded data from Unity Game Environment

V. PREPROCESSING

We applied various preprocessing techniques on the dataset collected to make the model training task easier. Our main task was to predict the steering angle, so it was important for us to have an equal balance of left steering angle as well as right steering angle, so as to avoid making the model biased in favoring the outputs in a particular direction. As a result we wrote a python script, which sets a threshold and selects left steering and right steering angle images in equal proportion. The histogram depicting the steering angle and frequency of images with respect to that steering angle in our training dataset is as follows:-

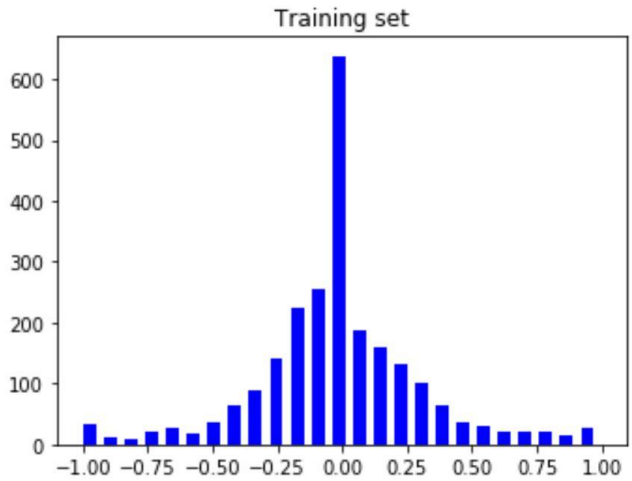


Fig.9 Steering Angle Distribution

In the above histogram, x-axis represents the steering angle wherein negative values denote left steering angle and positive values denote right steering angle. The y-axis represents the frequency of images with respect to a particular steering angle. We can see the proportion of steering angle in both the direction is almost the same. Moreover, we decided to keep steering angle of majority of the images in our training dataset close to zero. The main reason to do so was to make the model learn to drive straight, as in an ideal drive, majority of the driving takes place on a straight road as compared to the number of turns the car takes.

We also used various image augmentation techniques like image flipping, random zooming, altering the brightness intensities, on the dataset collected. The main reason for doing this was to make a versatile model which would be more robust to any changes in the environment. Few of the data augmented images used in the training is as follows:-

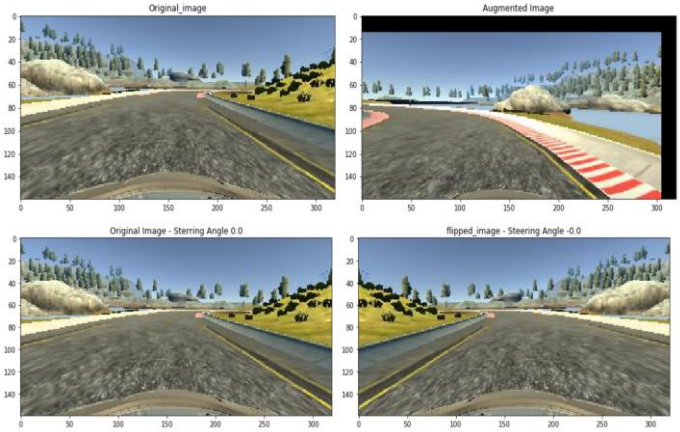


Fig.11.Augemented Images from dataset

The second image shows flipped representation of the original first image. When the image is flipped, the corresponding steering angle is reversed. The fourth image is the combination of image flipping, image panning and brightness alteration applied on the third image. These data augmenting techniques helps the model more robust in predicting the output even after slight changes in the input images. As a result, the model more precisely learns the mapping of input images to its corresponding steering angle.

VI. MODEL TRAINING

The model architecture we chose is nearly identical to the NVIDIA's network for End-to-End Learning for Self Driving Cars. Before feeding the input images to the model, we applied some more preprocessing techniques to make the input images appropriate as per the model requirements.

We first removed the top part and the bottom part (hood of the car) of the image to get our region of interest. We then converted the image from RGB to YUV format as the model tends to learn faster in this format. We applied Gaussian Blur with filter size of 3x3 and then resized the image to have shape (66, 200, 3) which is the input requirement for the model. We finally then normalized the input image pixels. One of the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

images after applying the preprocessing steps before feeding to the model is as shown below:

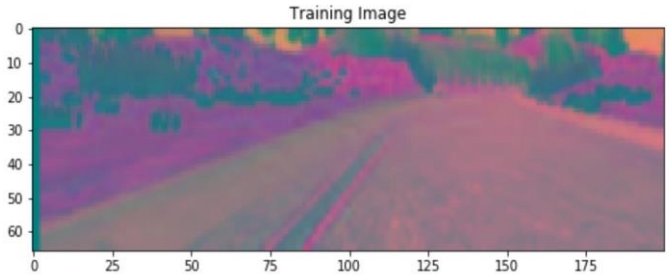


Fig.12.YUV format Training image.

Our model features 5 convolutional layers and 4 fully connected layers which looks similar to the architecture as shown below:-

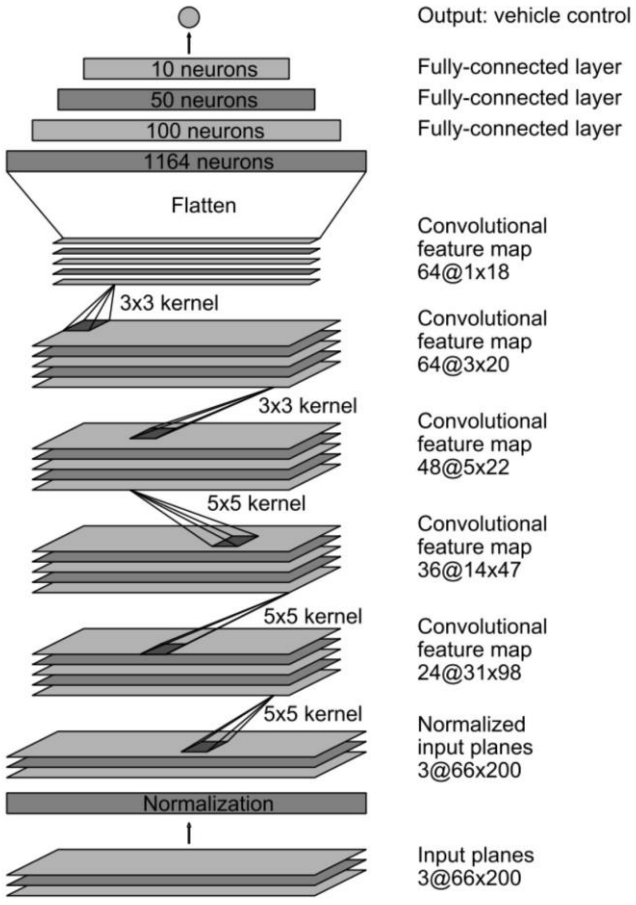


Fig.13 CNN Model

Source: [1]

Abstract features from the images are captured by the convolutional layers. These abstracted features are fed into the fully-connected layers which serve as a steering controller.

We used ELU activation in both the convolutional layer and fully-connected layer. We chose ELU over ReLU as our output lies between -1 to 1 and ReLU gives 0 as output for negative values which causes hinderance in models performance.

We used keras to build and train the model, which provided quick and simple implementation. Our keras model summary is as follows:-

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_7 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_8 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_9 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_10 (Conv2D)	(None, 1, 18, 64)	36928
flatten_2 (Flatten)	(None, 1152)	0
dense_5 (Dense)	(None, 100)	115300
dense_6 (Dense)	(None, 50)	5050
dense_7 (Dense)	(None, 10)	510
dense_8 (Dense)	(None, 1)	11

Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0

Fig.14.Model Summary

The model was trained using Adam Optimizer with a learning rate of 0.001. As our output was steering angle, we used mean squared-error as the loss function. We used batch generator to generate minibatches on the fly with a batch size of l00 and taking 300 steps for epoch in training. For validation, we used batch generator with a batch size of 100 taking 200 validation steps.

We trained the model for 30 epochs. In total, the model was trained on around 30,000 images per epoch. The following graph shows the variation in the loss with respect to the epochs in training as well as validation

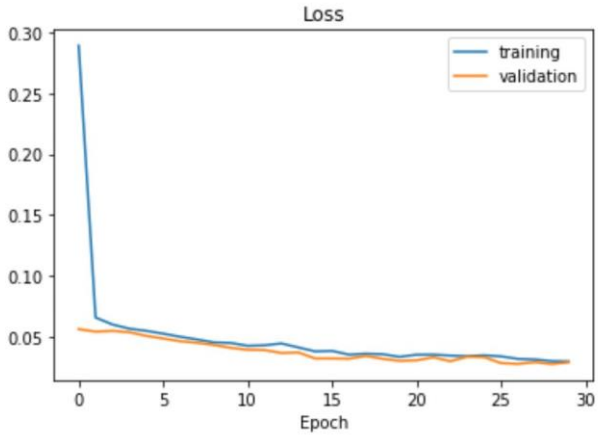


Fig.15.Loss vs Epoch

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

VII. OBJECT DETECTION

We used YOLOv3 (You Only Look Once) [8] for object Detection. YOLOv3 is an object detection algorithm which is based on neural nets which can be used to detect objects in live videos or static images, it is one of the fastest and accurate object detection method to date. YOLOv3 can identify 80 different classes which also include person, bicycle, traffic light, car, motorbike, stop sign, etc which are very useful in self driving task. As a result, we used YOLOv3 in our simulation, to mainly detect traffic lights, vehicle and pedestrians and to take appropriate decisions based on the detection while navigating the car.

At test time, the simulation sends the current captured frame to the server. The Neural Network Model is run on this input image to get the corresponding steering angle. Apart from that, YOLOv3 is also run on the same image to detect objects in the given image. Based on the output of detection, the corresponding acceleration is determined.

Fig.16.Signal Detection using YOLO

In the above image, the yolo model detects the traffic light with 73% probability. Using image processing, the color of the traffic light is deciphered. If the color is red, the car is deaccelerated so as to stop else if the color is green, the car is accelerated. YOLOv3 can detect multiple objects in the image as well

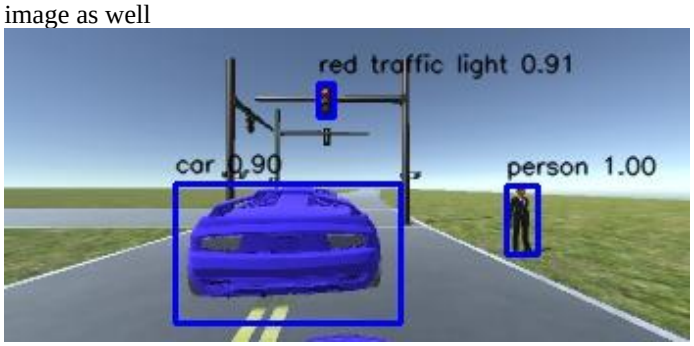


Fig.17.Object Detection

As in the above image, the YOLO model detects traffic light as well as car and person and thus appropriate decision is taken.

VIII. CONCLUSION

Simulator is the safest and cheapest way to test the autonomous car prototype before deploying it in real life environment. Our project provides such a flexible simulated environment. With the help of CNN model, the car is able to drive on various road conditions. It also successfully detected and classified various road objects.

IX. FUTURE SCOPE

Simulators can further be used to create a much more realistic environment, it can be used to simulate real life traffic situations, complex road architectures, human driver errors etc. This Simulations then can be used to train Autonomous driving systems. Such models then can tested and then be incorporated in real world cars.

X. REFERENCES

[1] End-to-End Deep Learning for Self-Driving Cars
By Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jackel, Urs Muller, Karol Zieba and Davide Del Testa – Nvidia Developers
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

[2] Self-Driving RC Car – Zheng Wang
https://zhengludwig.wordpress.com/projects/self-driving-rc-car/

[3] Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research https://microsoft.github.io/AirSim/.

[4] Introduction to Udacity Self-Driving Car Simulator
https://medium.com/activating-robotic-minds/introduction-to-udacity-self-driving-car-simulator-4d78198d301d -Naoki Shibuya

[5] How to Train Your Self Driving Car Using Deep Learning
https://towardsdatascience.com/how-to-train-your-self-driving-car-using-deep-learning-ce8ff76119cb

[6] http://neuralnetworksanddeeplearning.com/ By Michael Nielsen / Dec 2019 [E-book].

[7] Ismail Buyuksaliha, Serdar Bayburta, Gurcan Buyuksaliha, A.P. Baskaracaa, Hairi Karimb and Alias Abdul Rahmanb – " 3D Modelling and Visualization based on the Unity game engine – Advantages and Challenges", ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume IV-4/W4, 2017 4th International GeoAdvances Workshop, 14–15 October 2017, Safranbolu, Karabuk, Turkey.

[8] YOLOv3: An Incremental Improvemnet
By Joseph Redmond, Ali Farhadi – University of Washington
https://pjreddie.com/media/files/papers/YOLOv3.pdf