सी डैक | प्रगत संगणन विकास केंद्र, नोएडा

CDAC | Centre for Development of Advanced Computing, Noida

A PROJECT REPORT

ON

SELF DRIVING CAR SIMULATOR USING DEEP LEARNING

Submitted In Partial Fulfillment of the Requirement for The Award of

Post Graduate Diploma in Artificial Intelligence (PG-DAI)

Under the Guidance of

MR. RAVI PAYAL	**MRS. SARUTI GUPTA**
(Joint Director)	**(Project Engineer)**
(Program Coordinator)	**(Project Guide)**

Submitted By

SNEHAL THORAVE	**(PG-DAI)**	**PRN: 210920528040**
VYANKATESH PANDIT	**(PG-DAI)**	**PRN: 210920528040**
ANIKET KAKADE	**(PG-DAI)**	**PRN: 210920528006**

CONTENTS

INDEX	TITLE	PAGE NUMBER
I	Certificate	3
II	Acknowledgement	4
III	Abstract	5
IV	Introduction to the problem statement and the possible solution	6
V	Technology Used	7
VI	Data Collection	8
VII	Data Preprocessing	9
VIII	Network Architecture	11
IX	Coding	13
X	Results	20
XI	Conclusion & Future Scope	21-22
XII	Reference	23

CDAC, B-30, Institutional Area, Sector-62

Noida (Uttar Pradesh)-201307

CERTIFICATE

CDAC, NOIDA

This is to certify that Report entitled **"SELF DRIVING CAR SIMULATOR USING DEEP LEARNING"** which is submitted by Vyankatesh pandit in partial fulfillment of the requirement for the award of **Post Graduate Diploma in Artificial Intelligence** (PG-DAI) to **CDAC, Noida** is a record of the candidates own work carried out by them under my supervision.

The documentation embodies results of original work, and studies are carried out by the student themselves and the contents of the report do not from the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

MRS. SARUTI GUPTA.

(Project Engineer)

(Project Guide)

ACKNOWLEDGEMENT

I would like to express my best sense of gratitude & endeavor with respect to **Mrs. Saruti Gupta (Project Engineer)** CDAC, Noida for suggesting the problems scholarly guidance and expert supervision during the course of this project. Special thanks to **Mr. Ravi Payal (Program Coordinator).**

I am very thankful to **Mrs. Saruti Gupta (Project Engineer)** the project guide for constant simulated discussion, encouraging new ideas about this project.

Snehal Thorave	**210920528040**	[illegible]
Vyankatesh Pandit	**210920528042**	[illegible]
Aniket Kakade	**210920528006**	[illegible]

ABSTRACT

Self-driving cars have become a trending topic with great development technology over the past decade. The aim of the project is to train the neural network to drive an independent car agency on the natural trails of Udacity's Car Simulator. Misconduct has released the template as an open source software and a competition (challenge) to teach a car to drive using only camera images and depth reading.

Driving an independent vehicle requires learning to control the steering angle, throttle and brakes. Behavioral cloning technique is used to mimic human driving behavior in training mode on the track. That means the database is created in the template by the user a car driven in training mode, and a model of deep neural network and drive the car in the middle auto mode. Three building blocks are compared in terms of their functionality.

INTRODUCTION TO THE PROBLEM STATEMENT AND THE POSSIBLE SOLUTION

The concept of self-driving car has been debated since the 1920s, but the first semi auto-mated car was developed. Japan in 1977. A historic self-driving car was developed in the 1980s with an average speed of up to 31 kmph. And after some development in 2013 Tesla started their private car project. In 2014 they released the Tesla S with semi-autopilot mode. And some autopilot software developments are being released in the following similar models Tesla X etc. Researchers and analysts they have already begun to consider the consequences of independence carbon emissions vehicles, number of vehicles per person, etc. and provided their views on automotive automation. Self-driving cars will need more than human driving large consumer market protection capabilities. But surely, will have a significant impact on the timeline of transportation and a milestone in human history.

Technology Used

Technologies that are used in the implementation of this project and the motivation behind using these are described in this section.

TensorFlow: This an open-source library for dataflow programming. It is widely used for machine learning applications. It is also used as both a math library and for large computation. For this project Keras, a high-level API that uses TensorFlow as the backend is used. Keras facilitate in building the models easily as it more user friendly.

Different libraries are available in Python that helps in machine learning projects. Several of those libraries have improved the performance of this project. Few of them are mentioned in this section. First, “Numpy” that provides with high-level math function collection to support multi-dimensional metrices and arrays. This is used for faster computations over the weights (gradients) in neural networks. Second, “scikit-learn” is a machine learning library for Python which features different algorithms and Machine Learning function packages. Another one is OpenCV (Open Source Computer Vision Library) which is designed for computational efficiency with focus on real-time 4 applications. In this project, OpenCV is used for image preprocessing and augmentation techniques.

The project makes use of MiniConda Environment which is an open source distribution for Python which simplifies package management and deployment. It is best for large scale data processing.

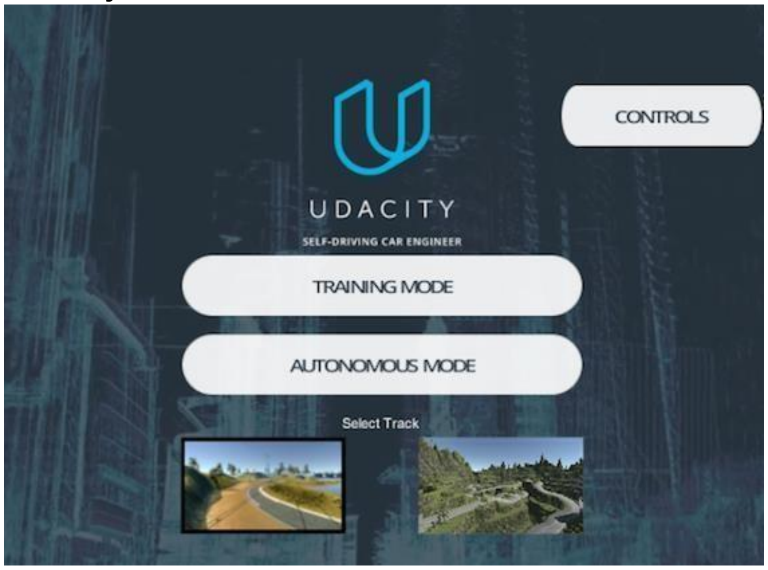
The machine on which this project was built, is a personal computer with following configuration:

- Processor: Intel(R) Core i5-7200U @ 2.7GHz
- RAM: 8GB
- System: 64bit OS, x64 processor

DATA COLLECTION

First for this project we need data. So we collect the data from the car simulator software. Its provided by Udacity. There is two mode training and autonomous. From training mode we collect the data in CSV format.

In traing mode we drive a car for 10 to 15 min for collecting the data. We take left turn , right turn and drive straithly. This all data stored in CSV file and we collect the image of every screen.



Data Selection

The first step in training the neural network is to choose the frames you will use. Our collected data is based on the type of road, weather, and driver activity (route, routing, turning, turn, etc.). To train CNN to create the next route, simply select the data where the driver lives on the route, and discard the rest. Removing bias from driving straight training data includes the upper part of the frames that represent the curves of the road.

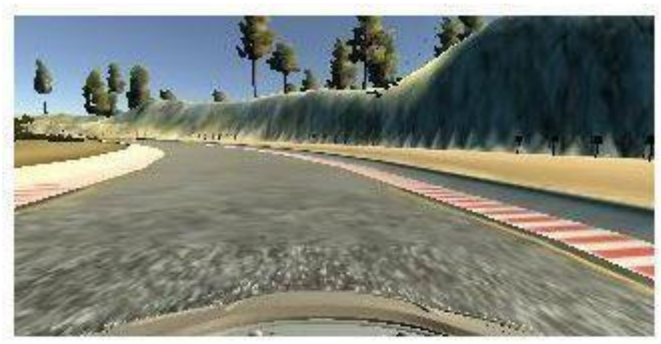
Data Preprocessing

1. Data Visualization and Balancing
   We drive a car manually so we have to balance left turn same as right turn so we visualize the data using matplotlib. And we balance the data.

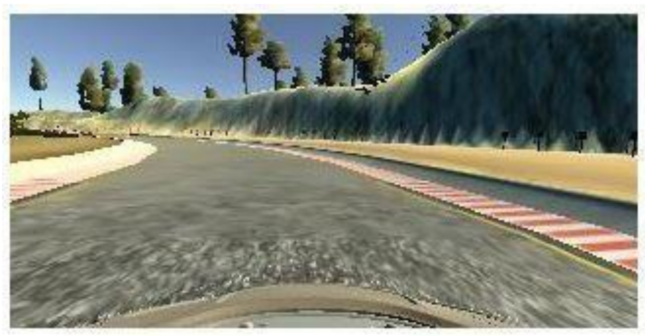
2. Augmentation

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 artifacts as the magnitude increases (as mentioned previously).

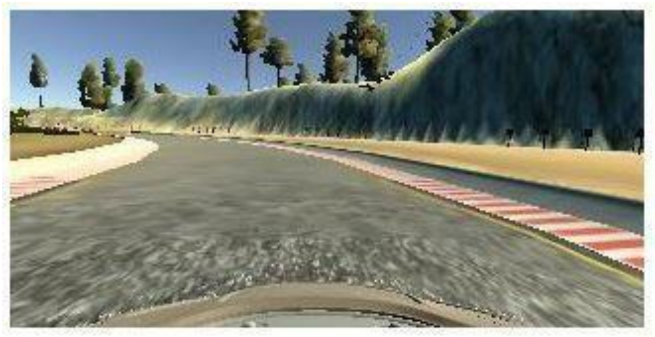
Zoom



Flip

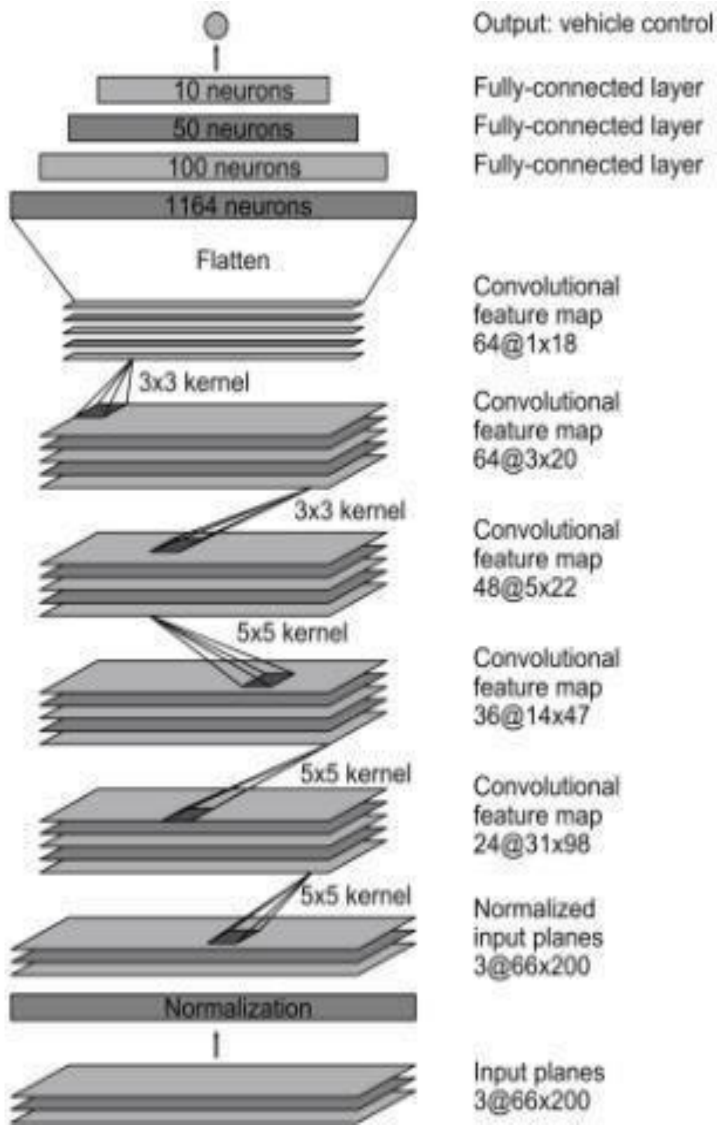


Brightness



Network Architecture

We train the weights of our network to minimize the mean-squared error between the steering command output by the network, and either the command of the human driver or the adjusted steering command for off-center and rotated images .Figure shows the network architecture, which 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 convolutional layers are designed to perform feature extraction, and are chosen empirically through a series of experiments that vary layer configurations. We then use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel, and a non-strided convolution with a 3×3 kernel size in the final two convolutional layers.

Coding:

1. Main.py

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import matplotlib.image as mpimg
from imgaug import augmenters as iaa
import cv2
import random
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D,Flatten,Dense
from tensorflow.keras.optimizers import Adam


def  getName(filePath):
    return filePath.split('\\')[-1]


def importDataInfo(path):
    columns = ['Center', 'Left', 'Right', 'Steering', 'Throttle', 'Brake', 'Speed']
    data = pd.read_csv(os.path.join(path, 'driving_log.csv'), names=columns)
    #### REMOVE FILE PATH AND GET ONLY FILE NAME
    print(data['Center'][0])
    print(getName(data['Center'][0]))
    data['Center'] = data['Center'].apply(getName)
    print(data.head())
    print('Total Images Imported', data.shape[0])
    return data


def balanceData(data,display=True):
    nBin = 31
    samplesPerBin = 500
    hist, bins = np.histogram(data['Steering'], nBin)
    #print(bins)
    if display:
        center = (bins[:-1] + bins[1:]) * 0.5
        #print(center)
        plt.bar(center, hist, width=0.06)
        plt.plot((-1,1),(samplesPerBin,samplesPerBin))
        plt.show()

    removeindexList = []
     for j in range(nBin):
        binDataList = []
        for i in range(len(data['Steering'])):
            if data['Steering'][i] >= bins[j] and data['Steering'][i] <= bins[j + 1]:
                binDataList.append(i)
        binDataList = shuffle(binDataList)
        binDataList = binDataList[samplesPerBin:]
        removeindexList.extend(binDataList)
    print('Removed Images:', len(removeindexList))
    data.drop(data.index[removeindexList], inplace=True)
    print('Remaining Images:', len(data))

    if display:
        hist, _ = np.histogram(data['Steering'], nBin)
        plt.bar(center, hist, width=0.06)
        plt.plot((-1,1),(samplesPerBin,samplesPerBin))
        plt.show()



def loadData(path, data):
    imagesPath = []
    steering = []
    for i in range(len(data)):
        indexed_data = data.iloc[i]
        #print(indexed_data)
        imagesPath.append(os.path.join(path,'IMG',indexed_data[0]))
        #print(os.path.join(path,'IMG',indexed_data[0]))
        steering.append(float(indexed_data[3]))
    imagesPath = np.asarray(imagesPath)
    steering = np.asarray(steering)
    return imagesPath,steering
```

```
def augmentImage(imgPath,steering):
    img =  mpimg.imread(imgPath)
    #print(np.random.rand(),np.random.rand(),np.random.rand(),np.random.rand())
    #PAN
    if np.random.rand() < 0.5:
        pan = iaa.Affine(translate_percent={"x": (-0.1, 0.1), "y": (-0.1, 0.1)})
        img = pan.augment_image(img)
    #ZOOM
    if np.random.rand() < 0.5:
        zoom = iaa.Affine(scale=(1, 1.2))
        img = zoom.augment_image(img)
    #brightness
    if np.random.rand() < 0.5:
        brightness = iaa.Multiply((0.4, 1.2))
        img = brightness.augment_image(img)
    ## Flip
    if np.random.rand() < 0.5:
        img = cv2.flip(img, 1)
        steering = -steering
    return img, steering

#imgRe, st =augmentImage('test.jpg',0)
#plt.imshow(imgRe)
#plt.show()



def preProcess(img):
    img = img[60:135,:,:] #cropping
    img=cv2.cvtColor(img,cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img / 255
    return img
imgRe=preProcess(mpimg.imread('test.jpg'))
plt.imshow(imgRe)
plt.show()
```

```
def batchGen(imagesPath, steeringList, batchSize, trainFlag):
    while True:
        imgBatch=[]
        steeringBatch=[]

        for i in range(batchSize):
            index = random.randint(0, len(imagesPath) - 1)
            if trainFlag:
                img, steering = augmentImage(imagesPath[index], steeringList[index])
            else:
                img = mpimg.imread(imagesPath[index])
                steering = steeringList[index]
            img = preProcess(img)
            imgBatch.append(img)
            steeringBatch.append(steering)
        yield (np.asarray(imgBatch),np.asarray(steeringBatch))



def createModel():
    model = Sequential()

    model.add(Convolution2D(24, (5, 5), (2, 2), input_shape=(66, 200, 3),
activation='elu'))
    model.add(Convolution2D(36, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(48, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(64, (3, 3), activation='elu'))
    model.add(Convolution2D(64, (3, 3), activation='elu'))

    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))

    model.compile(Adam(lr=0.0001), loss='mse')
    return model
```

2. Training Simulation:

```
import matplotlib.pyplot as plt

print('Setting UP')
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
from utlis import *
from sklearn.model_selection import train_test_split
#step 1
path = 'myData'
data = importDataInfo(path)

#step 2
balanceData(data,display=False)

#step 3
imagesPath,steering =loadData(path,data)
#print(imagesPath[0],steering[0])

### Step 4
xTrain, xVal, yTrain, yVal = train_test_split(imagesPath, steering,
test_size=0.2,random_state=10)
print('Total Training Images: ',len(xTrain))
print('Total Validation Images: ',len(xVal))

#step 8
model=createModel()
model.summary()

#STEP 9
history=model.fit(batchGen(xTrain,yTrain,100,1),steps_per_epoch=300,epochs=1
0,
        validation_data=batchGen(xVal,yVal,100,0),validation_steps=200)

#Step 10
model.save('model.h5')
print('Model saved')

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training','Validation'])
plt.ylim([0,1])
plt.title('Loss')
plt.xlabel('Epoch')
plt.show()
```

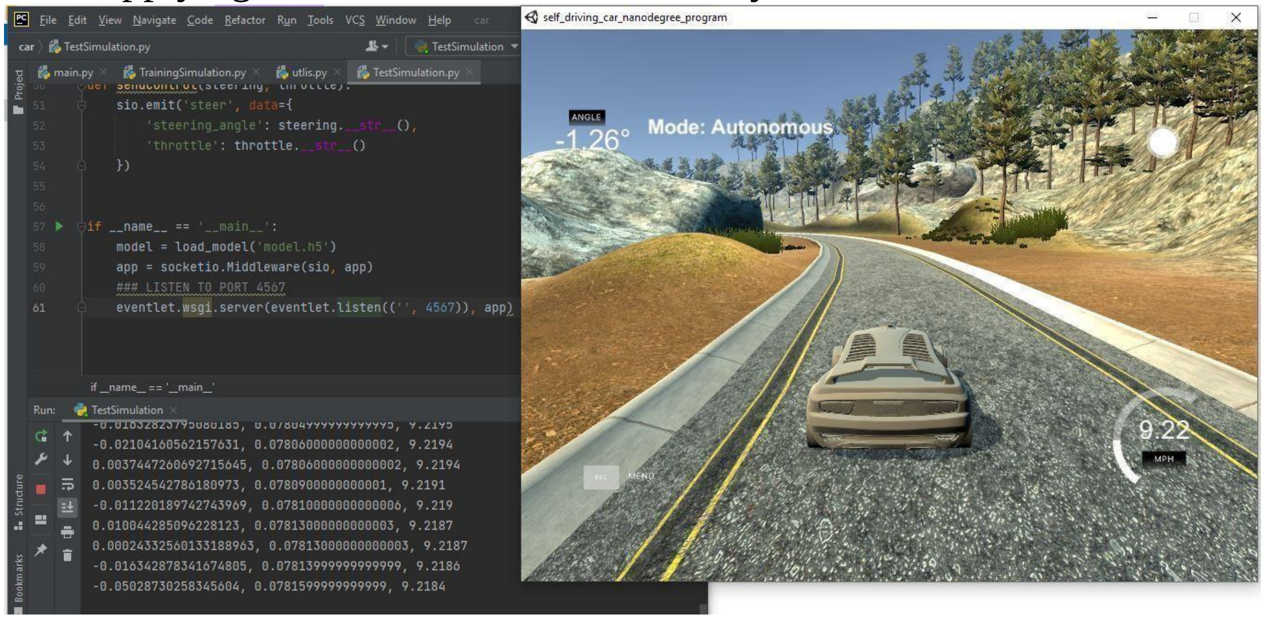
3. Testing Simulation

```
print('Setting UP')
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import socketio
import eventlet
import numpy as np
from flask import Flask
from tensorflow.keras.models import load_model
import base64
from io import BytesIO
from PIL import Image
import cv2

#### FOR REAL TIME COMMUNICATION BETWEEN CLIENT AND
SERVER
sio = socketio.Server()
#### FLASK IS A MICRO WEB FRAMEWORK WRITTEN IN PYTHON
app = Flask(__name__)  # '__main__'

maxSpeed = 10



def preProcess(img):
    img = img[60:135, :, :]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img / 255
    return img


@sio.on('telemetry')
```

```
def telemetry(sid, data):
    speed = float(data['speed'])
    image = Image.open(BytesIO(base64.b64decode(data['image'])))
    image = np.asarray(image)
    image = preProcess(image)
    image = np.array([image])
    steering = float(model.predict(image))
    throttle = 1.0 - speed / maxSpeed
    print(f'{steering}, {throttle}, {speed}')
    sendControl(steering, throttle)


@sio.on('connect')
def connect(sid, environ):
    print('Connected')
    sendControl(0, 0)


def sendControl(steering, throttle):
    sio.emit('steer', data={
        'steering_angle': steering.__str__(),
        'throttle': throttle.__str__()
    })


if__name__== '__main__':
    model  = load_model('model.h5')
    app = socketio.Middleware(sio, app)
    ### LISTEN TO PORT 4567
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

Result:

After creating Model we apply that model in our car in car simulator software. This time we use autonomous mode.
After applying model car drive automatically.

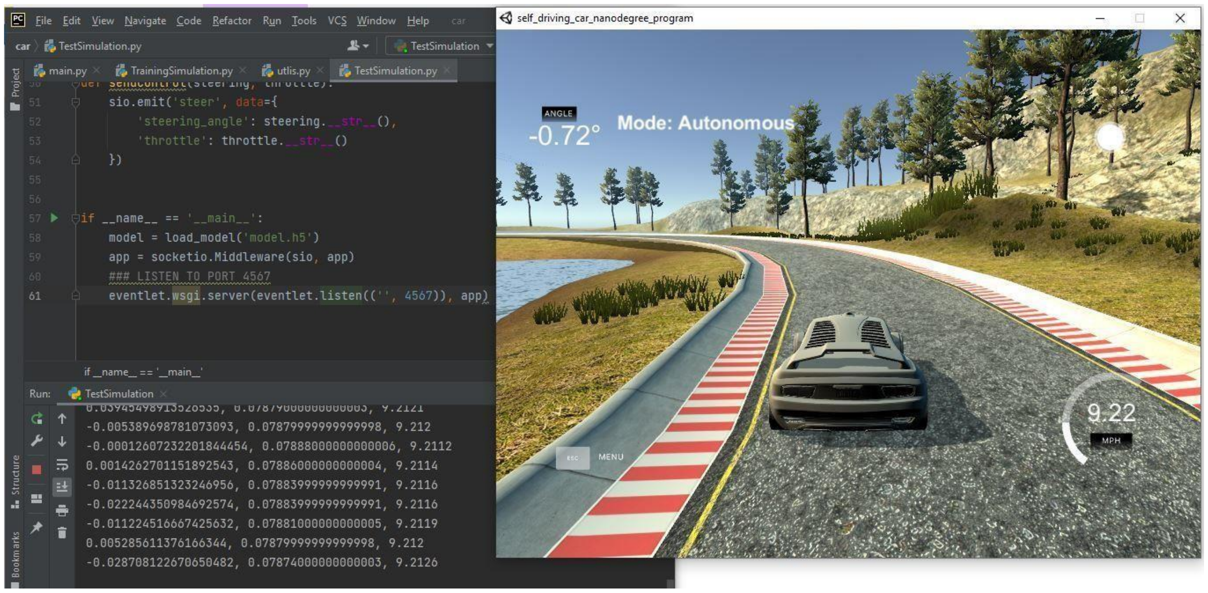


File Edit View Navigate Code Refactor Run Tools VCS Window Help car
car TestSimulation.py
TestSimulation
main.py TrainingSimulation.py utlis.py TestSimulation.py
self_driving_car_nanodegree_program
ANGLE
-1.26°
Mode: Autonomous
9.22
MPH
MENU

```python
def sendControl(steering, throttle):
    sio.emit('steer', data={
        'steering_angle': steering.__str__(),
        'throttle': throttle.__str__()
    })


if __name__ == '__main__':
    model = load_model('model.h5')
    app = socketio.Middleware(sio, app)
    ### LISTEN TO PORT 4567
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

Run: TestSimulation

```
-0.01632823795080185, 0.07804999999999995, 9.2195
-0.02104160562157631, 0.07806000000000002, 9.2194
0.0037447260692715645, 0.07806000000000002, 9.2194
0.003524542786180973, 0.0780900000000001, 9.2191
-0.011220189742743969, 0.07810000000000006, 9.219
0.010044285096228123, 0.07813000000000003, 9.2187
0.00024332560133188963, 0.07813000000000003, 9.2187
-0.016342878341674805, 0.07813999999999999, 9.2186
-0.05028730258345604, 0.0781599999999999, 9.2184
```



File Edit View Navigate Code Refactor Run Tools VCS Window Help car
car TestSimulation.py
TestSimulation
main.py TrainingSimulation.py utlis.py TestSimulation.py
self_driving_car_nanodegree_program
ANGLE
-0.72°
Mode: Autonomous
9.22
MPH
MENU

```python
def sendControl(steering, throttle):
    sio.emit('steer', data={
        'steering_angle': steering.__str__(),
        'throttle': throttle.__str__()
    })


if __name__ == '__main__':
    model = load_model('model.h5')
    app = socketio.Middleware(sio, app)
    ### LISTEN TO PORT 4567
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

Run: TestSimulation

```
0.03945498913528535, 0.07879000000000003, 9.2121
-0.005389698781073093, 0.07879999999999998, 9.212
-0.00012607232201844454, 0.07888000000000006, 9.2112
0.0014262701151892543, 0.07886000000000004, 9.2114
-0.011326851323246956, 0.07883999999999991, 9.2116
-0.022244350984692574, 0.07883999999999991, 9.2116
-0.011224516667425632, 0.07881000000000005, 9.2119
0.005285611376166344, 0.07879999999999998, 9.212
-0.028708122670650482, 0.07874000000000003, 9.2126
```

Conclusions

It is interesting to find the use of combinations of real world dataset and simulator data to train these models. Then I can get the true nature of how a model can be trained in the simulator and generalized to the real world or vice versa. There are many experimental implementations carried out in the field of self-driving cars and this project contributes towards a significant part of it.

This project started with training the models and tweaking parameters to get the best performance on the tracks and then trying to generalize the same performance on different tracks. The models that performed best on 1 track did poorly on Track_2, hence there was a need to use image augmentation and processing to achieve real time generalization.

The program learns for example to find a roadmap without the need for clear labels during training.

Future Work

In future work to extend CNN current to regional-based CNNs to monitor passenger safety as welldone. More work is needed to develop resilience network, and a better understanding of how CNN works inside. In addition, including other cars dynamics Database parameters such as speed and velocity will do provide for real-life navigation based on the car.

Placing counterfeit vehicles and obstacles along the way, can increase the level of challenges facing solving this problem, however, will bring it much closer to real-time the environment self-driving cars would face in the real world. How well modeling performance in real-world data can be a good challenge. The model tried with a real world database, but there was no way to test it in a similar location.

The big players in the automotive industry should have tried this private cars. This would be a good test to see how this model really works in real time

REFERENCES

- Autonomous Vehicles: Levels, Technologies, Impacts and Concerns - International Journal of Applied Engineering Research ISSN Paper by Mohsin Raza Student, Department of Mechanical Engineering, JSS Academy of Technical Education, Noida, India.

- Mariusz Bojarski, “End-to-End Deep Learning for Self-Driving Cars”, Published on 17 Aug, 2016, https://devblogs.nvidia.com/deep-learning-self-driving-cars/, accessed Nov 2017

- Brustein, J., 2014, “Self-Driving Cars Will Mean More Traffic”, Bloomberg.

- Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backprop- agation applied to handwritten zip code recognition. Neural Computation, 1(4):541–551, Winter 1989.

- 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

Sites:
https://developer.nvidia.com/blog/deep-learning-self-driving-cars/