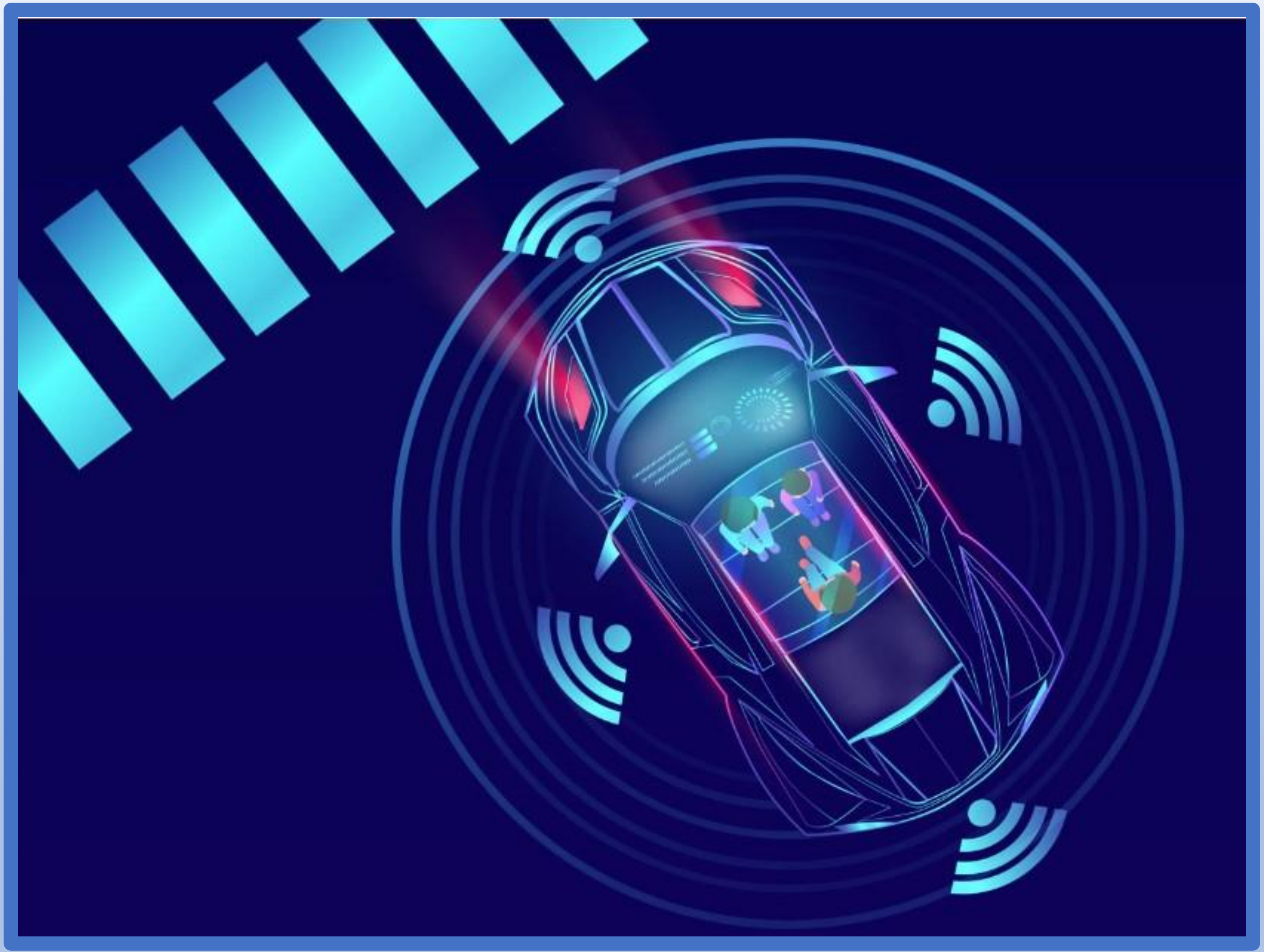

SELF DRIVING CAR
- Futuristic self
- Komal Reddy Koukuntla

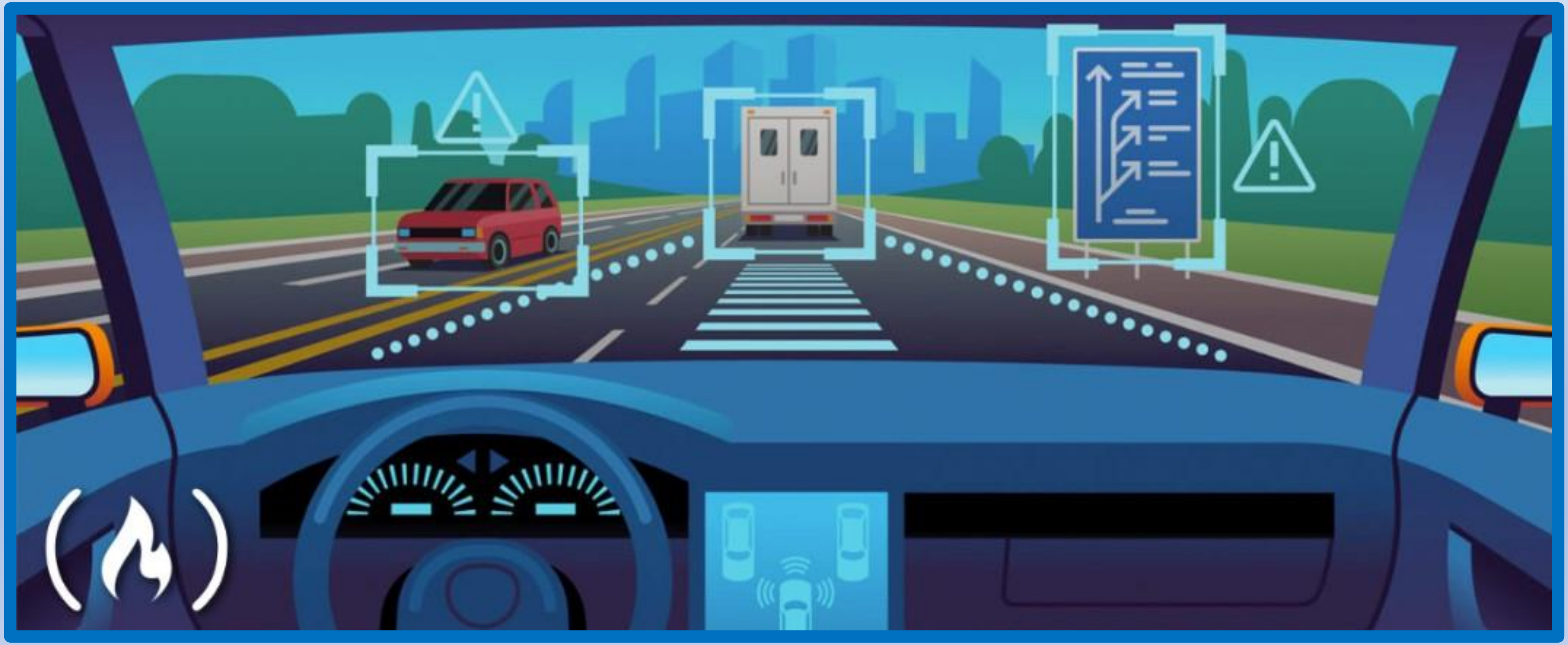


ABSTRACT

Self-Driving car, a car capable of sensing its surrounding and moving on its own through traffic and other obstacles with minimum or no human input. This is the current upcoming technology in the automobile industry and even though it has been discussed and worked on for a long time, it was successfully manufactured by TESLA. In recent years, these cars began to roll out in foreign markets as private and public vehicles(taxis etc.). Many companies like Waymo, UBER, Nissan, Nvidia are involved in this product development. With this type of car, the whole automotive transportation's safety, security, efficiency is increased and the human errors can be eradicated whilst the drive is made to its best. This project has infused the idea of traffic signal resposding which is absent in the current models and the above mentioned advantages can be achieved with much more ease and at a low cost. This type of system can bring a revolution in transporting for differently abled people and also help blind people travel independently.

OBJECTIVE

- Demonstrate automated driving in complex traffic environments. Test integrated applications in all possible scenarios taking into account the full range of automation levels.
- Enhance the perception performance in complex scenarios by using advanced sensors supported by cooperative and communication technologies.



INTRODUCTION

- When we manually drive a car we see front at first then at back(rear) if required, and also we look at the side view mirrors, these images act as inputs

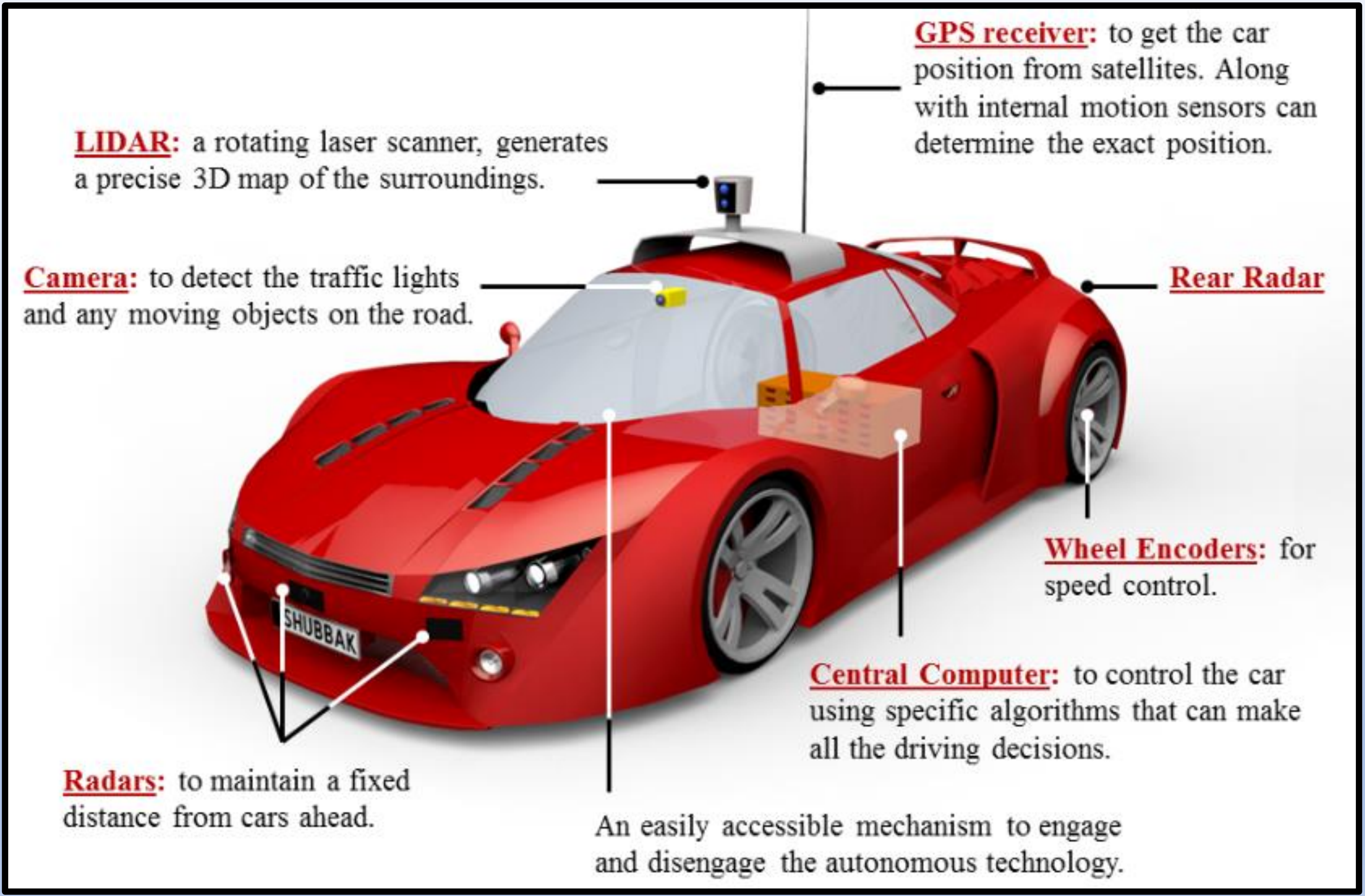
Front view

Rear view

Side view

1. Input will be given by cameras(Front view, Rear view, Side view)
2. LIDAR(3D Geometry, little expensive)
3. RADAR(how far the object is)
4. Ultrasonic sensors
5. GPS

Depending on company to company we can have as many inputs as possible



How these inputs exactly work?

- The inputs defined before are sent to computing box
- This computing box gives us the outputs

OUTPUTS FOR DRIVING CAR

1. Steering wheel angle
2. Acceleration we need to apply
3. Brakes if required
4. Indicators or signals which are needed to be handled properly

Data Generation

- Records images from center, left, and right cameras w/ associated steering angle, speed, throttle and brake.
- Saves to CSV
- Ideally you have a joystick, but keyboard works too

Udacity recently open sourced their self driving car simulator originally built for SDND students

1. Built in Unity
2. Added a bunch of scripts like gravity, momentum and acceleration

Training Mode- Behavioural cloning

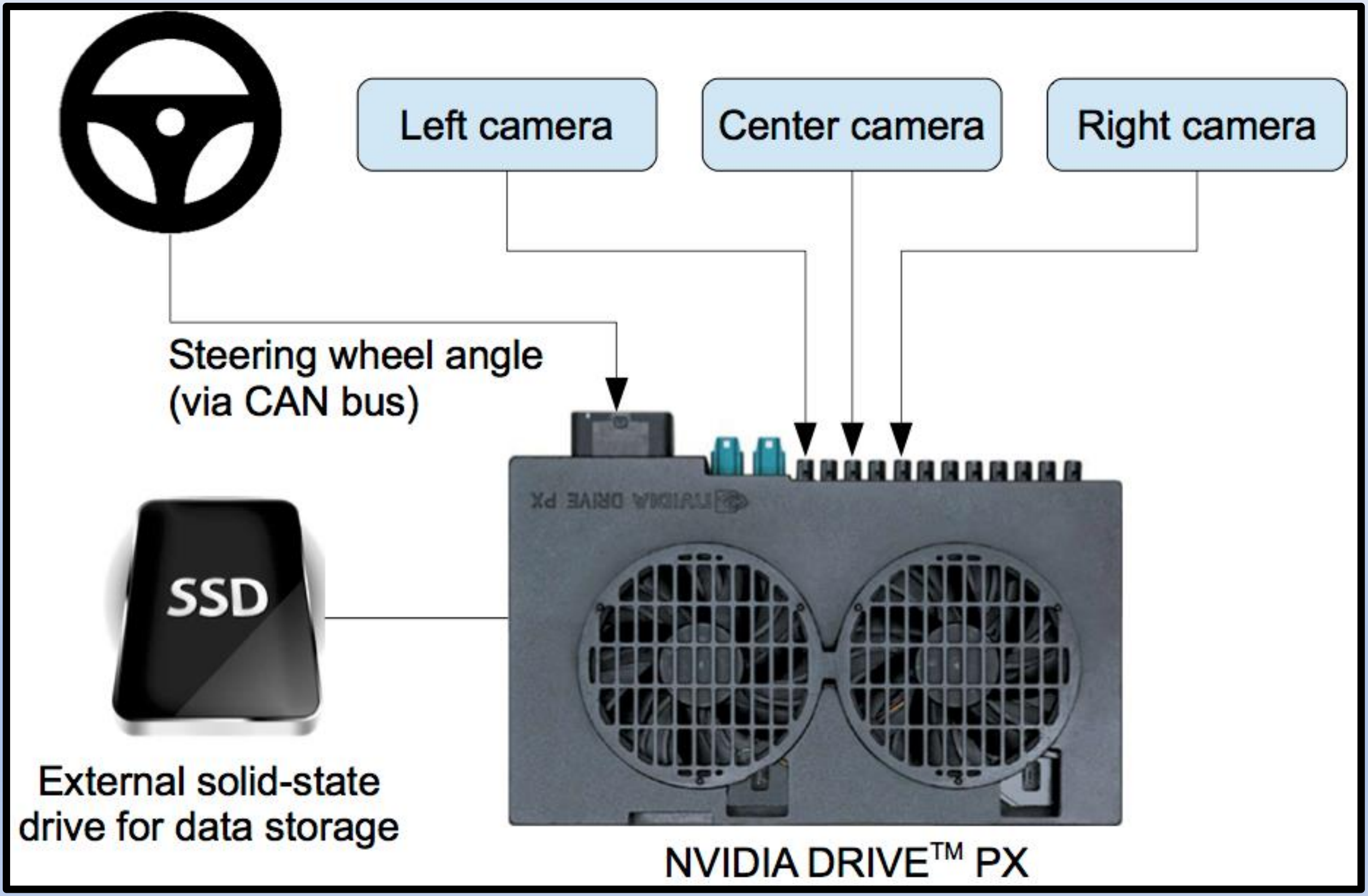
We use a 9 layer convolutional network(in keras its literally 9 lines of code), based off of Nvidia's end-to-end learning for self driving car paper. 72 hours of driving data was collected in all sorts of conditions from human drivers

How is data organized in CSV File?

1. First three columns are images captured by center , left , right cameras respectively
2. Next four columns are steering angle, throttle,reverse,speed respectively
3. We split the data into training part of 80% and testing part of 20%

METHODOLOGY

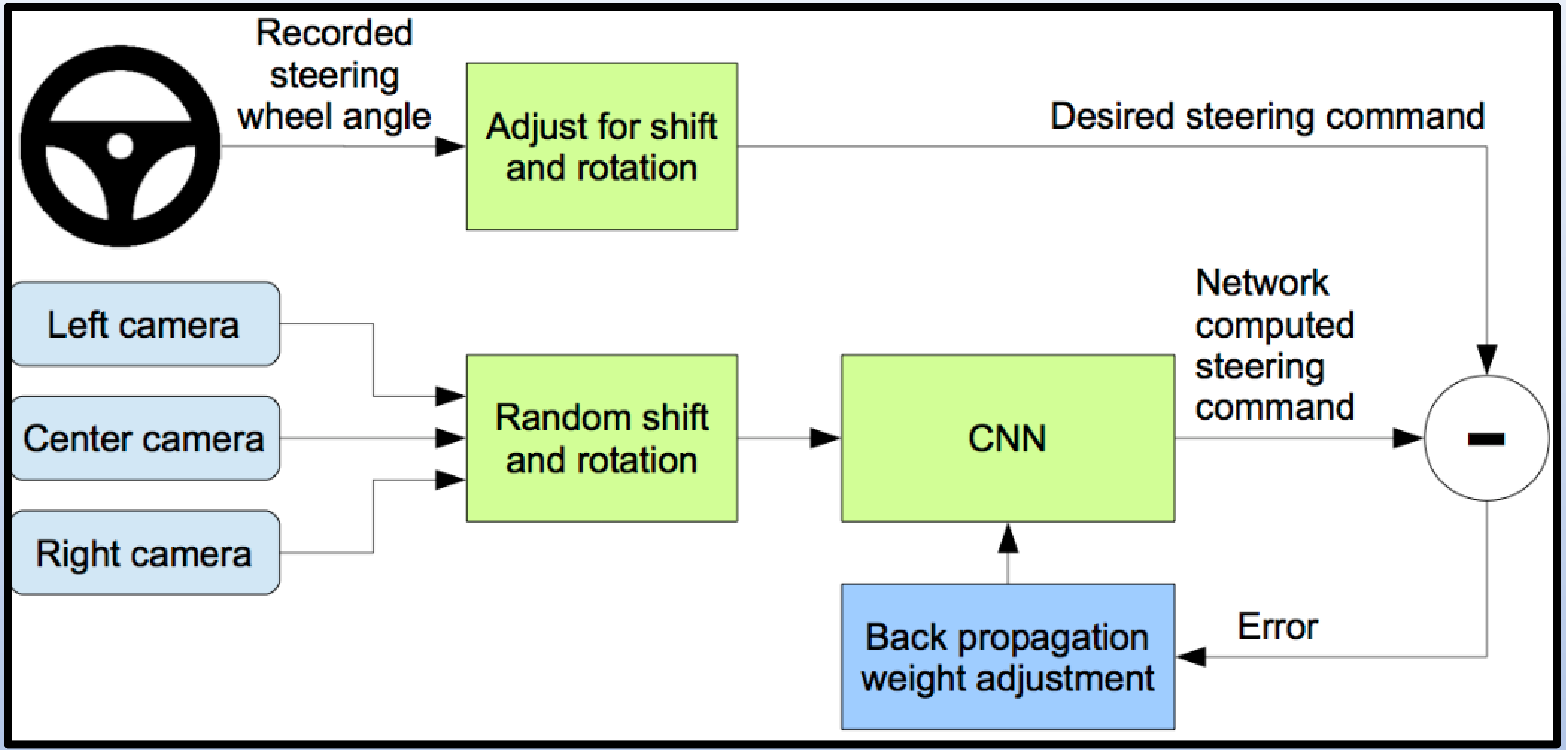
Hardware design

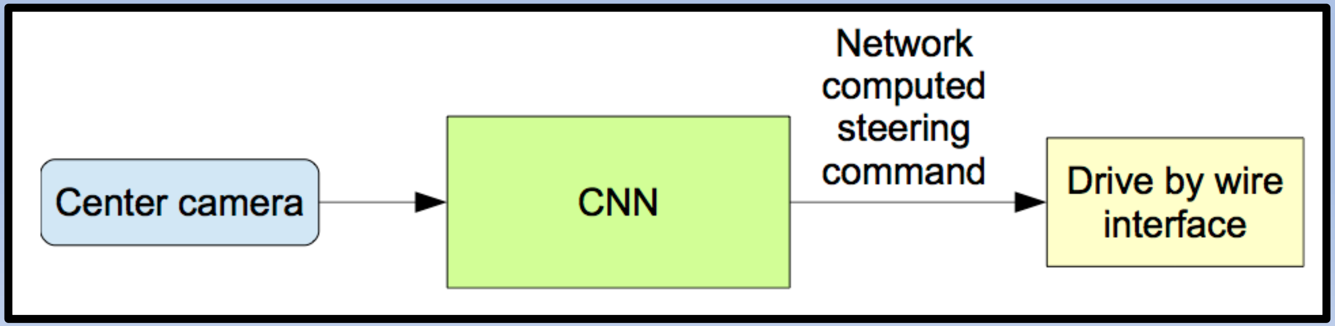


- 3 cameras
- The steering command is obtained by tapping into the vehicle’s Controller Area Network (CAN) bus.
- Nvidia's Drive PX onboard computer with GPUs

In order to make the system independent of the car geometry, the steering command is 1/r, where r is the turning radius in meters. 1/r was used 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).

Software design





Images are fed into a CNN that 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. The weight adjustment is accomplished using back propagation

Eventually, it generated steering commands using just a single camera

Testing

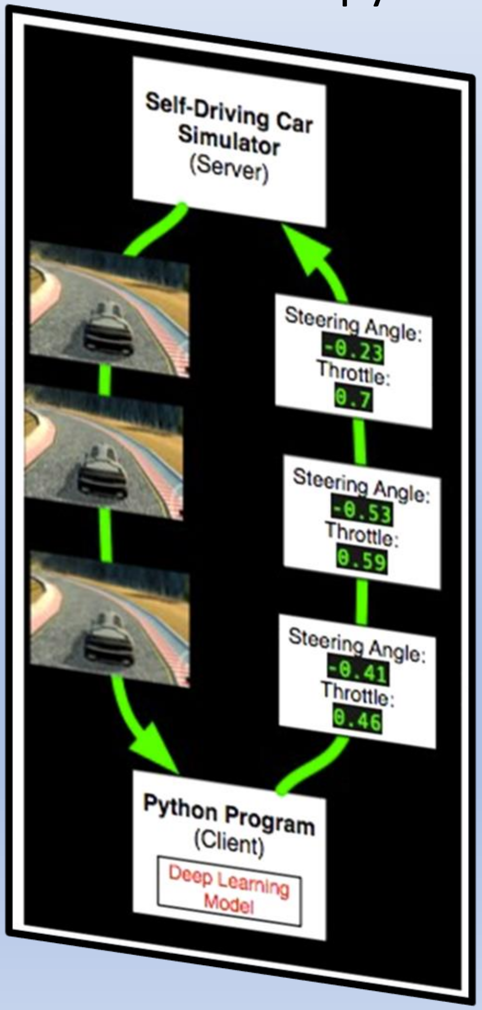
1. For testing we could sort of think as a server client architecture
2. The server is going to be the simulator itself that is our Udacity app and the client is the python program

We breakdown dataset into two parts one is for training and the other for testing

80% of the dataset is used for training

20% is used for testing

Training takes much time compared to testing



We will just run autonomous mode, then run our model and the car will start driving

CODE

Lets dive into training the model
The code displayed here is written in the file name model(model.py)

```
## IMPORTING LIBRARIES ##
##########################
import os
import csv
import cv2
import numpy as np

from PIL import Image
from keras.models import Sequential
from keras.layers import Flatten, Dense, Lambda
from keras.layers.core import Dense, Activation,
Flatten, Dropout

from keras.layers import Cropping2D
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras import backend as K
```

We import necessary libraries which we will be using later

Versions used:

H5py==2.7.1
Keras==2.1.6
Numpy==1.13.3
Opencv-python==3.3.0.10
Pandas==0.21.0
Tensorflow-gpu==1.10.1

```
## READING IMAGE DATA FROM THE LOG ##
#####################################
lines = []
header = True
camera_images = []
steering_angles = []

with open('/content/driving_log.csv', 'r') as
f:
    reader = csv.reader(f, delimiter=';')
    for row in reader:
        if header:
            header = False
            continue
        steering_center = float(row[3])
```

Now we will be reading the data column wise

As the data file is of format CSV we will open it and access via columns one by one

Which we can see in the following picture

1. First three columns are images captured by center , left , right cameras respectively
2. Next four columns are steering angle, throttle,reverse,speed respectively

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	/IMG/center_2020_07_14_14_18_36_008.jpg;/IMG/left_2020_07_14_14_18_36_008.jpg;/IMG/right_2020_07_14_14_18_36_008.jpg;-0.1588235;0;0;8.075665															
2	/IMG/center_2020_07_14_14_18_36_090.jpg;/IMG/left_2020_07_14_14_18_36_090.jpg;/IMG/right_2020_07_14_14_18_36_090.jpg;-0.1705882;0;0;7.992766															
3	/IMG/center_2020_07_14_14_18_36_168.jpg;/IMG/left_2020_07_14_14_18_36_168.jpg;/IMG/right_2020_07_14_14_18_36_168.jpg;-0.1705882;0;0;7.944249															
4	/IMG/center_2020_07_14_14_18_36_247.jpg;/IMG/left_2020_07_14_14_18_36_247.jpg;/IMG/right_2020_07_14_14_18_36_247.jpg;-0.1705882;0.2630496;0;8.021999															
5	/IMG/center_2020_07_14_14_18_36_343.jpg;/IMG/left_2020_07_14_14_18_36_343.jpg;/IMG/right_2020_07_14_14_18_36_343.jpg;-0.1705882;0.2413888;0;8.324794															
6	/IMG/center_2020_07_14_14_18_36_419.jpg;/IMG/left_2020_07_14_14_18_36_419.jpg;/IMG/right_2020_07_14_14_18_36_419.jpg;-0.1529412;0.1462076;0;8.380316															
7	/IMG/center_2020_07_14_14_18_36_492.jpg;/IMG/left_2020_07_14_14_18_36_492.jpg;/IMG/right_2020_07_14_14_18_36_492.jpg;-0.1235294;0.3666444;0;8.599499															

```
#Steering angle (sa) correction factor for stereo cameras
        sa_cor = 0.2
        steering_left = steering_center + sa_cor
        steering_right = steering_center - sa_cor

        #Reading camera images from their paths
        path_src1 = row[0]
        path_src2 = row[1]
        path_src3 = row[2]
        img_name1 = path_src1.split('/')[-1]
        img_name2 = path_src2.split('/')[-1]
        img_name3 = path_src3.split('/')[-1]
        path1 = '/content/data/IMG/' + img_name1
        path2 = '/content/data/IMG/' + img_name2
        path3 = '/content/data/IMG/' + img_name3

        #Image and Steering Dataset
        img_center = np.asarray(Image.open(path1))
        img_left = np.asarray(Image.open(path2))
        img_right = np.asarray(Image.open(path3))
        camera_images.extend([img_center, img_left, img_right])
        steering_angles.extend([steering_center, steering_left,
steering_right])
```

We are accessing each and every image via Path which is stored in a zip file called data in which we have images captured(IMG) and CSV file(driving_log.csv)

This is how Images had been stored while capturing in Data zip file

data	15-12-2022 21:05	Compressed (zipped)...	1,15,159 KB

data	File folder

IMG	File folder					15-12-2022 20:26
driving_log	Microsoft Excel Comma Separ...	42 KB	No	350 KB	89%	15-12-2022 20:18

center_2020_07_14_14_18_36_008	JPG File	15 KB	No
center_2020_07_14_14_18_36_090	JPG File	15 KB	No
center_2020_07_14_14_18_36_168	JPG File	15 KB	No
center_2020_07_14_14_18_36_247	JPG File	15 KB	No
center_2020_07_14_14_18_36_343	JPG File	15 KB	No
center_2020_07_14_14_18_36_419	JPG File	15 KB	No
center_2020_07_14_14_18_36_492	JPG File	15 KB	No
center_2020_07_14_14_18_36_585	JPG File	15 KB	No
center_2020_07_14_14_18_36_679	JPG File	15 KB	No
center_2020_07_14_14_18_36_751	JPG File	15 KB	No
center_2020_07_14_14_18_36_830	JPG File	15 KB	No
center_2020_07_14_14_18_36_903	JPG File	15 KB	No
center_2020_07_14_14_18_36_972	JPG File	15 KB	No
center_2020_07_14_14_18_37_044	JPG File	15 KB	No
center_2020_07_14_14_18_37_116	JPG File	15 KB	No
center_2020_07_14_14_18_37_184	JPG File	15 KB	No
center_2020_07_14_14_18_37_276	JPG File	15 KB	No
center_2020_07_14_14_18_37_371	JPG File	15 KB	No
center_2020_07_14_14_18_37_453	JPG File	15 KB	No
center_2020_07_14_14_18_37_557	JPG File	15 KB	No

left_2020_07_14_14_18_38_529	JPG File	16 KB
left_2020_07_14_14_18_38_613	JPG File	16 KB
left_2020_07_14_14_18_38_704	JPG File	16 KB
left_2020_07_14_14_18_38_810	JPG File	16 KB
left_2020_07_14_14_18_38_889	JPG File	16 KB
left_2020_07_14_14_18_38_975	JPG File	16 KB
left_2020_07_14_14_18_39_074	JPG File	16 KB
left_2020_07_14_14_18_39_164	JPG File	16 KB
left_2020_07_14_14_18_39_254	JPG File	16 KB
left_2020_07_14_14_18_39_333	JPG File	16 KB
left_2020_07_14_14_18_39_421	JPG File	16 KB
left_2020_07_14_14_18_39_489	JPG File	15 KB
left_2020_07_14_14_18_39_563	JPG File	16 KB
left_2020_07_14_14_18_39_649	JPG File	15 KB
left_2020_07_14_14_18_39_732	JPG File	16 KB
left_2020_07_14_14_18_39_815	JPG File	16 KB
left_2020_07_14_14_18_39_887	JPG File	16 KB

Data is augmented via this code

```
## DATA AUGMENTATION ##
#######################
augmented_imgs, augmented_sas= [],[]

for aug_img,aug_sa in
zip(camera_images,steering_angles):
    augmented_imgs.append(aug_img)
    augmented_sas.append(aug_sa)

    #Flipping the image
    augmented_imgs.append(cv2.flip(aug_img
,1))

    #Reversing the steering angle
    augmented_sas.append(aug_sa*-1.0)
```

```
## INDEPENDENT VARIABLES and LABELS ##
######################################
X_train, y_train = np.array(augmented_imgs),
np.array(augmented_sas)
X_train, y_train = np.array(camera_images),
np.array(steering_angles)
```

```
## IMAGE PRE-PROCESSING ##
##########################
def preprocess(image):
    import tensorflow as tf
    #Resizing the image
    return tf.image.resize(image, (200, 66))
```

The images which are stored in zip file named Data are being processed using tensorflow GPU

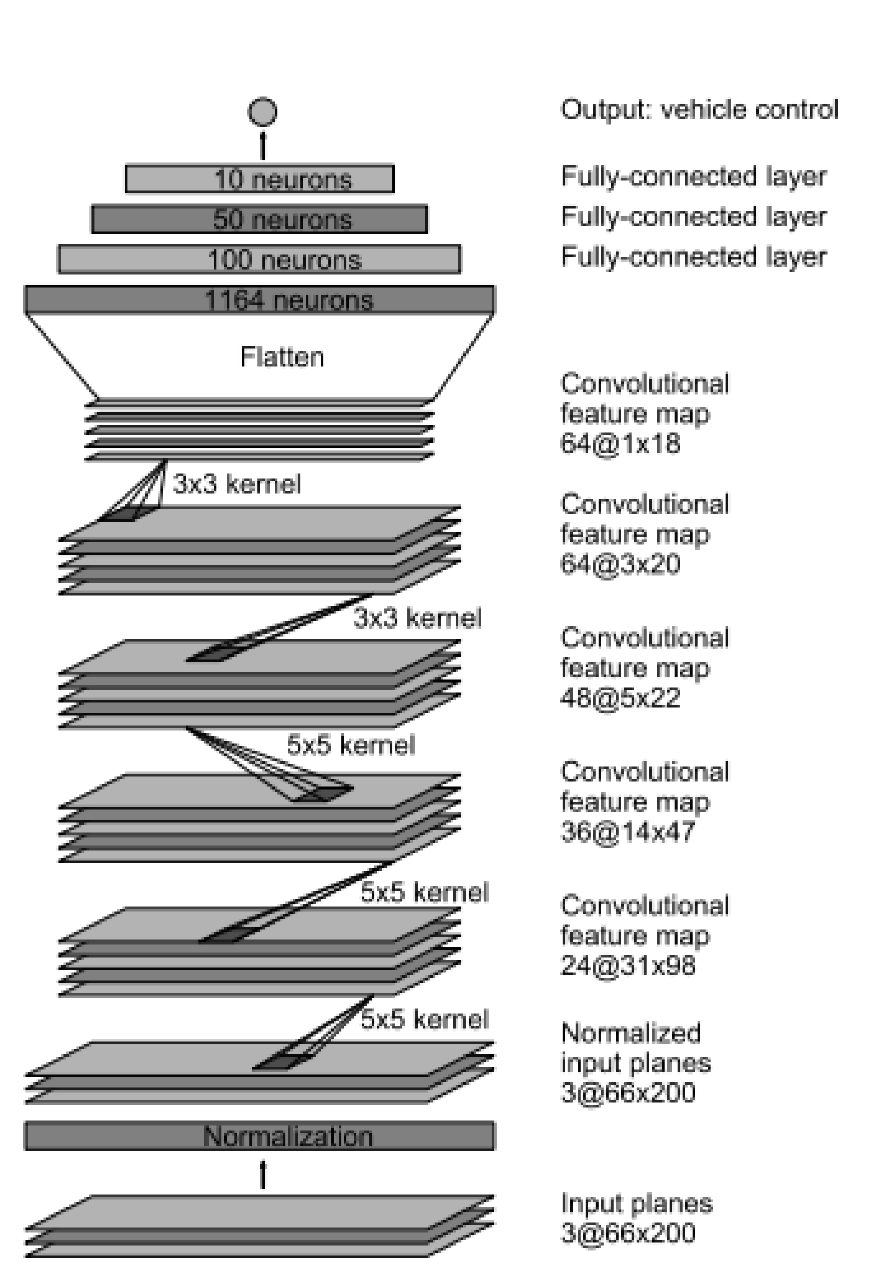
```python
## THE CNN ARCHITECTURE ##
##########################
'''
The CNN architecture is used from NVIDIA's End
to End Learning for Self-Driving Cars paper.
Reference:
https://arxiv.org/pdf/1604.07316v1.pdf
'''
#Keras Sequential Model
model = Sequential()

#Image cropping to get rid of the irrelevant
parts of the image (the hood and the sky)
model.add(Cropping2D(cropping=((50,20), (0,0)),
input_shape=(160,320,3)))

#Pre-Processing the image
model.add(Lambda(preprocess))
model.add(Lambda(lambda x: (x/ 127.0 - 1.0)))
```

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-center and rotated images

```
#The layers
model.add(Conv2D(filters=24, kernel_size=(5,
5), strides=(2, 2),activation='relu'))
model.add(Conv2D(filters=36, kernel_size=(5,
5),strides=(2, 2), activation='relu'))
model.add(Conv2D(filters=48, kernel_size=(5,
5), strides=(2, 2),activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3)
,activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3,
3),activation='relu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(units=100, activation='relu'))
model.add(Dense(units=50, activation='relu'))
model.add(Dense(units=10, activation='relu'))
model.add(Dense(units=1))
print(model.summary())
```

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 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 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.

```
#Compiling and Saving the Model
model.compile(loss='mse',optimizer='adam')
#adaptive moment estimation.
model.fit(X_train,y_train,validation_split=0.2,
shuffle=True,epochs=10)
model.save('model.h5')

print('The model.h5 file has been created!')
## END OF THE CODE ##
```

The training code which is model.py is executed in google colab as !python model.py

Now the model has been saved into model.h5 file which we will be using for testing

In next slide we have the outputs of training code once executed

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 cropping2d (Cropping2D)     (None, 90, 320, 3)        0

 lambda (Lambda)             (None, 200, 66, 3)        0

 lambda_1 (Lambda)           (None, 200, 66, 3)        0

 conv2d (Conv2D)             (None, 98, 31, 24)        1824

 conv2d_1 (Conv2D)           (None, 47, 14, 36)        21636

 conv2d_2 (Conv2D)           (None, 22, 5, 48)         43248

 conv2d_3 (Conv2D)           (None, 20, 3, 64)         27712

 conv2d_4 (Conv2D)           (None, 18, 1, 64)         36928

 dropout (Dropout)           (None, 18, 1, 64)         0

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 100)               115300

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 10)                510

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
```

```
_________________________________________________________________
None
Epoch 1/10
187/187 [==============================] - 55s 284ms/step - loss: 0.0163 - val_loss: 0.0540
Epoch 2/10
187/187 [==============================] - 52s 277ms/step - loss: 0.0083 - val_loss: 0.0529
Epoch 3/10
187/187 [==============================] - 53s 284ms/step - loss: 0.0055 - val_loss: 0.0600
Epoch 4/10
187/187 [==============================] - 51s 274ms/step - loss: 0.0038 - val_loss: 0.0526
Epoch 5/10
187/187 [==============================] - 54s 289ms/step - loss: 0.0029 - val_loss: 0.0552
Epoch 6/10
187/187 [==============================] - 51s 272ms/step - loss: 0.0023 - val_loss: 0.0537
Epoch 7/10
187/187 [==============================] - 50s 270ms/step - loss: 0.0020 - val_loss: 0.0557
Epoch 8/10
187/187 [==============================] - 51s 272ms/step - loss: 0.0016 - val_loss: 0.0537
Epoch 9/10
187/187 [==============================] - 51s 273ms/step - loss: 0.0015 - val_loss: 0.0532
Epoch 10/10
187/187 [==============================] - 51s 273ms/step - loss: 0.0013 - val_loss: 0.0607
The model.h5 file has been created!
```

Lets dive into testing the model
The code displayed here is written in the file name drive(drive.py)

```
import argparse
import base64
from datetime import datetime
import os
import shutil

import numpy as np
import socketio
import eventlet
import eventlet.wsgi
from PIL import Image
from flask import Flask
from io import BytesIO

from keras.models import load_model
import h5py
from keras import __version__ as keras_version
```

We import necessary libraries which we will be using later

Versions used:

H5py==2.7.1
Keras==2.1.6
Numpy==1.13.3
Opencv-python==3.3.0.10
Pandas==0.21.0
Tensorflow-gpu==1.10.1
Matplotlib==2.1.0
Flask-SocketIO==2.9.2
Eventlet==0.21.0

```python
sio = socketio.Server()
app = Flask(__name__)
model = None
prev_image_array = None
```

```python
class SimplePIController:
    def __init__(self, Kp, Ki):
        self.Kp = Kp
        self.Ki = Ki
        self.set_point = 0.
        self.error = 0.
        self.integral = 0.

    def set_desired(self, desired):
        self.set_point = desired

    def update(self, measurement):
        # proportional error
        self.error = self.set_point - measurement

        # integral error
        self.integral += self.error

        return self.Kp * self.error + self.Ki *
self.integral
```

```python
controller = SimplePIController(0.1, 0.002)
set_speed = 9
controller.set_desired(set_speed)
```

```python
@sio.on('telemetry')
def telemetry(sid, data):
    if data:
        # The current steering angle of the car
        steering_angle = data["steering_angle"]
        # The current throttle of the car
        throttle = data["throttle"]
        # The current speed of the car
        speed = data["speed"]
        # The current image from the center camera of the car
        imgString = data["image"]
        image = Image.open(BytesIO(base64.b64decode(imgString)))
        image_array = np.asarray(image)
        steering_angle = float(model.predict(image_array[None, :, :, :], batch_size=1))

        throttle = controller.update(float(speed))

        print(steering_angle, throttle)
        send_control(steering_angle, throttle)

        # save frame
        if args.image_folder != '':
            timestamp = datetime.utcnow().strftime('%Y_%m_%d_%H_%M_%S_%f')[:-3]
            image_filename = os.path.join(args.image_folder, timestamp)
            image.save('{}.jpg'.format(image_filename))
    else:
        # NOTE: DON'T EDIT THIS.
        sio.emit('manual', data={}, skip_sid=True)
```

```python
@sio.on('connect')
def connect(sid, environ):
    print("connect ", sid)
    send_control(0, 0)


def send_control(steering_angle, throttle):
    sio.emit(
        "steer",
        data={
            'steering_angle': steering_angle.__str__(),
            'throttle': throttle.__str__()
        },
        skip_sid=True)
```

```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote Driving')
    parser.add_argument(
        'model',
        type=str,
        help='/content/model.h5'
    )
    parser.add_argument(
        'image_folder',
        type=str,
        nargs='?',
        default='',
        help='/content/img.zip'
    )
    args = parser.parse_args()

    # check that model Keras version is same as local Keras version
    f = h5py.File(args.model, mode='r')
    model_version = f.attrs.get('keras_version')
    keras_version = str(keras_version).encode('utf8')

    if model_version != keras_version:
        print('You are using Keras version ', keras_version,
              ', but the model was built using ', model_version)

    model = load_model(args.model)

    if args.image_folder != '':
        print("Creating image folder at {}".format(args.image_folder))
        if not os.path.exists(args.image_folder):
            os.makedirs(args.image_folder)
        else:
            shutil.rmtree(args.image_folder)
            os.makedirs(args.image_folder)
        print("RECORDING THIS RUN ...")
    else:
        print("NOT RECORDING THIS RUN ...")

    # wrap Flask application with engineio's middleware
    app = socketio.Middleware(sio, app)

    # deploy as an eventlet WSGI server
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

The testing code which is drive.py is executed in google colab along with model.h5 which is our trained model as !python drive.py model.h5

Before executing the code we have to open the beta simulator and then select the mode as autonomous , then run the above command, we can see that the car is moving accordingly

```
!python drive.py model.h5
```

```
2022-12-15 16:07:29.659137: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dle
2022-12-15 16:07:29.659290: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.
2022-12-15 16:07:29.659310: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you would like t
You are using Keras version  b'2.11.0' , but the model was built using  2.11.0
```

Car in autonomous mode driving one lap

CONCLUSION

The testing has been done successfully using 7467 (2488 images per camera) images recorded

Car drives almost perfect (as we have trained it using the data it may not always predict accurately)

In this project, only the steering angle is added as a dependent variable (label) for the corresponding camera images. However, it can be understood that this alone is not sufficient, because the autonomous mode driving speed is low, and that the vehicle wobbles around the center line during turns. At this point, speed and braking data can also be added to the model to train a driving behavior that can make smoother turns. And we can also train the model to follow certain traffic rules too


THANK YOU