Final Report on

Self-Driving Car Model

Based of Deep-Learning and IoT

(For Minor-II)

Session of 2018 January-May

Submitted By –

Apoorv Goel 15102237

Kunal Das 15103262

III Year

VI Semester

Submitted To - Manish Kumar Thakur

1. **Introduction –**

Udacity open-sourced their Self-Driving Car Simulator originally built for their Self-Driving Car Nanodegree Students, to teach them how to train cars, how to navigate road courses using deep learning.

The simulator is built in Unity and provides a single car with a couple of tracks to begin with. Since the project has been open-sourced, more tracks and cars can be programmed into the simulator. Any kinds of changes can be made into the simulator. The simulator also comes in executable form, ready to test and run any deep-learning, neural-net script on it for Car-Behavioural-Cloning.

The simulator includes two modes –

a. **Training** (Manual control given to user for generating and recording of data)
b. **Autonomous** (Provides a server-client API to feed data to the car)

The data generated by the simulator includes –

a. **Images** from **centre**, **left** and **right**: Supposedly from camera mounted on top of the hood of the car

b. **Steering angle**: Current steering angle of the car (Floating value ranging between -1 and 1)
c. **Throttle**: Acceleration provided by user at the time of training or fed through the API
d. **Speed**: Speed of the car regardless of throttle
e. **Brake**: Reduces speed

All the training data is stored in a csv file for simple import and usage while training and testing convolutional neural network.

2. **Abstract from Nvidia's Paper** (see references) –
"CNNs have revolutionized the computational pattern recognition process. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The important breakthrough of CNNs is that features are now learned automatically from training examples. The CNN approach is especially powerful when applied to image recognition tasks because the convolution operation captures the 2D nature of images. By using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

While CNNs with learned features have been used commercially for over twenty years, their adoption has exploded in recent years because of two important developments. First, large, labeled data sets such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) are now widely available for training and validation. Second, CNN learning algorithms are now implemented on massively parallel graphics processing units (GPUs), tremendously accelerating learning and inference ability.

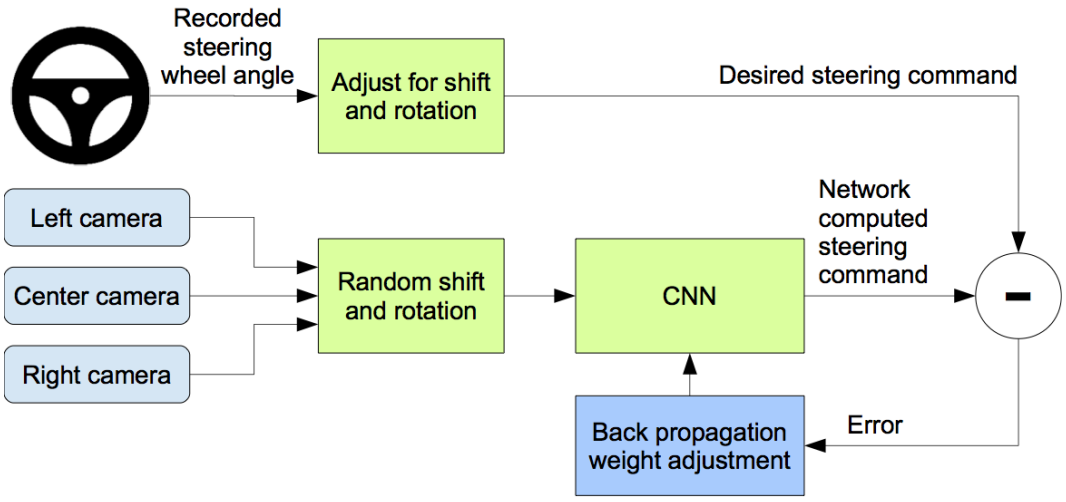
The CNNs that we describe here go beyond basic pattern recognition. We developed a system that learns the entire processing pipeline needed to steer an automobile. The groundwork for this project was actually done over 10 years ago in a Defense Advanced Research Projects Agency (DARPA) seedling project known as DARPA Autonomous Vehicle (DAVE), in which a sub-scale

radio control (RC) car drove through a junk-filled alley way. DAVE was trained on hours of human driving in similar, but not identical, environments. The training data included video from two cameras and the steering commands sent by a human operator.

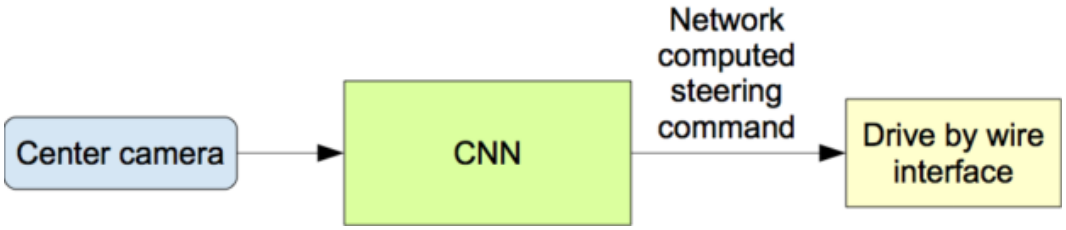
In many ways, DAVE was inspired by the pioneering work of Pomerleau, who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. ALVINN is a precursor to DAVE, and it provided the initial proof of concept that an end-to-end trained neural network might one day be capable of steering a car on public roads. DAVE demonstrated the potential of end-to-end learning, and indeed was used to justify starting the DARPA Learning Applied to Ground Robots (LAGR) program, but DAVE's performance was not sufficiently reliable to provide a full alternative to the more modular approaches to off-road driving. (DAVE's mean distance between crashes was about 20 meters in complex environments.)

About a year ago we started a new effort to improve on the original DAVE, and create a robust system for driving on public roads. The primary motivation for this work is to avoid the need to recognize specific human-designated features, such as lane markings, guard rails, or other cars, and to avoid having to create a collection of "if, then, else" rules, based on observation of these features. We are excited to share the preliminary results of this new effort, which is aptly named: DAVE–2."

3. Convolutional Neural Network –



Images are fed into the convolutional neural network that then computes a proposed steering command. The proposed command is compared to the desired command for the image, and the weights of the convolutional neural network are adjusted to bring the convolutional neural network's output closer to the desired output. The weight adjustment is accomplished using back propagation. Once trained, the network is able to generate commands from the images of a single centre camera.



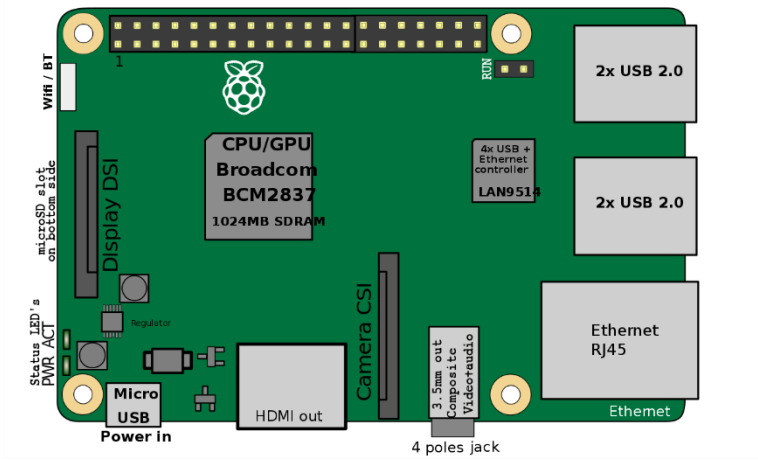
4. Aim –

Based on the simulator and the scripts we wrote to train and test the car, we set out to build a small toy car and implement the same CNN model (without training a new one) on the toy car. The variables used while testing on the simulator are too many to mechanically build and handle on the car. Steering angle is the most important one, which we will cover first. Speed, throttle and brakes require more subtle motor controls which we will try in the end. We suspect, even after placing all things where they need to go, there'd still be bugs and problems that would be tough to overcome.

5. Apparatus Used in Building Car –

a. Raspberry Pi 3

On-board machine responsible for sending images and receiving control inputs based on those images



b. Raspberry Pi Camera Module

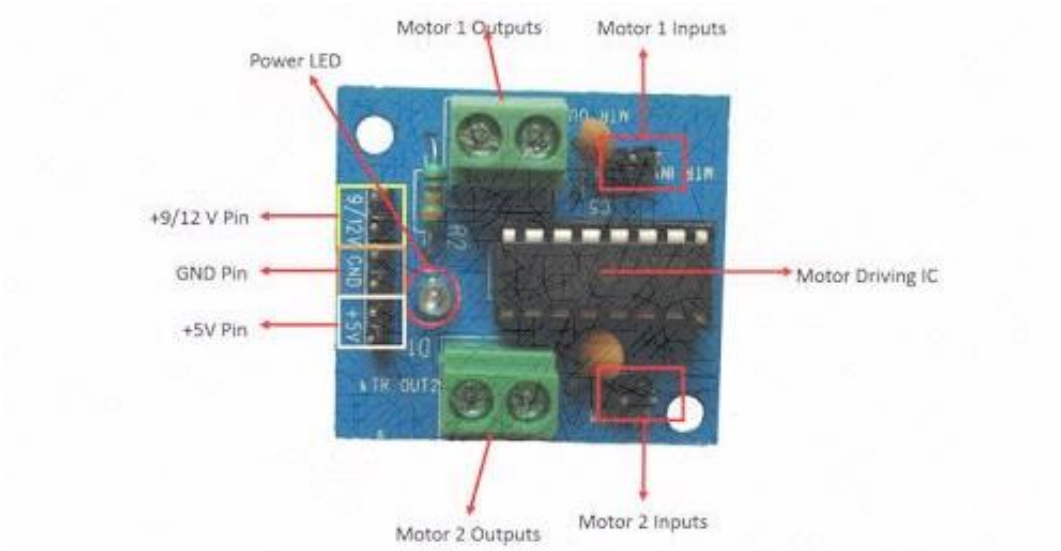
Used for taking pictures on Raspberry Pi

c. DC Motors (x2)

Steering front wheels of the car and Driving the Rear wheels of the car

d. L293D DC Motor Controller Board

Used to easily control the DC motors for bot clockwise and anti-clockwise direction.



e. Power Source (x2)

A higher voltage power source for powering the entire circuit; another power bank for regulated supply for powering the Raspberry Pi 3

f. Connecting Wires

Sending out control feeds from Raspberry Pi 3 to the DC Motor Controller

g. Toy Car

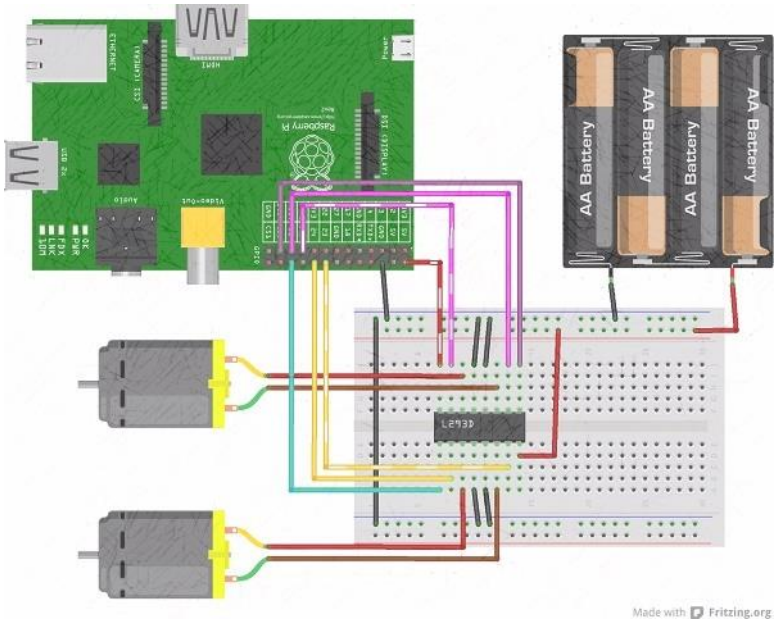
For mounting all the above mentioned apparatus for a wireless experience of Self-Driving Car

6. Hardware Issues Faces –

i. Placing heavy power bank reduced the overall speed and performance of the car.

ii. Replacing a good power bank with any feasible alternative power source led to current leaks in the circuit and lower voltage which again forced the car not to work to its best performance.

iii. Controlling the motor required making a circuit from scratch. Thankfully it was saved by replacing the idea by a simple IC (L293D).



iv. Bread-board was let go because it consumed too much space.

v. IC was soldered with connections to make the setup more light-weight. Connection stopped working after this. This idea was dropped then and there.

vi. Motor Controller board was brought in place to fix all issues.

vii. Smaller power bank was placed to power the entire setup.

viii. Still less power was drawn by the circuit when powered by the new smaller power bank. Much less even when both Raspberry Pi and the circuit was powered at once. Replaced one power bank with two.

7. On-board Script for Receiving Manual Controls (Server) –

```
import base64
import datetime
import picamera
import RPi.GPIO as GPIO
from io import BytesIO
from flask import Flask, request
from flask_socketio import SocketIO, emit

#Rear Wheels for Driving
Motor1A = 23
Motor1B = 24

#Front Wheels for Steer
Motor2A = 26
Motor2B = 19
```

```
def accelerate():
    GPIO.output(Motor1A, GPIO.HIGH)
    GPIO.output(Motor1B, GPIO.LOW)

def reverse():
    GPIO.output(Motor1A, GPIO.LOW)
    GPIO.output(Motor1B, GPIO.HIGH)

def left():
    GPIO.output(Motor2A, GPIO.LOW)
    GPIO.output(Motor2B, GPIO.HIGH)

def right():
    GPIO.output(Motor2A, GPIO.HIGH)
    GPIO.output(Motor2B, GPIO.LOW)
def leftTurn():
    GPIO.output(Motor1A, GPIO.HIGH)
    GPIO.output(Motor1B, GPIO.LOW)
    GPIO.output(Motor2A, GPIO.LOW)
    GPIO.output(Motor2B, GPIO.HIGH)

def rightTurn():
    GPIO.output(Motor1A, GPIO.HIGH)
    GPIO.output(Motor1B, GPIO.LOW)
    GPIO.output(Motor2A, GPIO.HIGH)
    GPIO.output(Motor2B, GPIO.LOW)

def leftRev():
    GPIO.output(Motor1A, GPIO.LOW)
    GPIO.output(Motor1B, GPIO.HIGH)
    GPIO.output(Motor2A, GPIO.LOW)
    GPIO.output(Motor2B, GPIO.HIGH)

def rightRev():
    GPIO.output(Motor1A, GPIO.LOW)
    GPIO.output(Motor1B, GPIO.HIGH)
    GPIO.output(Motor2A, GPIO.HIGH)
    GPIO.output(Motor2B, GPIO.LOW)

def brakes():
    GPIO.output(Motor1A, GPIO.LOW)
    GPIO.output(Motor1B, GPIO.LOW)

def release():
    GPIO.output(Motor2A, GPIO.LOW)
    GPIO.output(Motor2B, GPIO.LOW)
```

```
app = Flask(__name__)
socketio = SocketIO(app)
socketio_connections = set()

@socketio.on('throttle', namespace='/sdc')
def on_throttle(msg):
    if msg['key']=='38':
        accelerate()
        print("@"+str(datetime.datetime.now().time())+" : Accelerate")
    elif msg['key']=='40':
        reverse()
        print("@"+str(datetime.datetime.now().time())+" : Reverse")

@socketio.on('steer', namespace='/sdc')
def on_steer(msg):
    if msg['key']=='37':
        left()
        print("@"+str(datetime.datetime.now().time())+" : Steering Left")
    elif msg['key']=='39':
        right()
        print("@"+str(datetime.datetime.now().time())+" : Steering Right")

@socketio.on('turn', namespace='/sdc')
def on_turn(msg):
    if msg['key']=='75':
        leftTurn()
        print("@"+str(datetime.datetime.now().time())+" : Turning Left")
    elif msg['key']=='77':
        rightTurn()
        print("@"+str(datetime.datetime.now().time())+" : Turning Right")

@socketio.on('reverse', namespace='sdc')
def on_reverse(msg):
    if msg['key']=='77':
        leftRev()
        print("@"+str(datetime.datetime.now().time())+" : Reversing Left")
    elif msg['key']=='79':
        rightRev()
        print("@"+str(datetime.datetime.now().time())+" : Reversing Right")

@socketio.on('brakes', namespace='/sdc')
def on_brakes(msg):
    brakes()
    print("@"+str(datetime.datetime.now().time())+" : Brakes")

@socketio.on('release', namespace='/sdc')
def on_release(msg):
    release()
```

```
        print("@"+str(datetime.datetime.now().time())+" : Released Steer")

    @socketio.on('connect', namespace='/sdc')
    def on_connect():
        socketio_connections.add(request.sid)
        print("@"+str(datetime.datetime.now().time())+" : Remote Connected")

    @socketio.on('disconnect', namespace='/sdc')
    def on_disconnect():
        socketio_connections.remove(request.sid)
        print("@"+str(datetime.datetime.now().time())+" : Remote Disconnected")

    if __name__ == '__main__':
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(Motor1A, GPIO.OUT)
        GPIO.setup(Motor1B, GPIO.OUT)
        GPIO.setup(Motor2A, GPIO.OUT)
        GPIO.setup(Motor2B, GPIO.OUT)
        socketio.run(app, host='192.168.43.55', port=9999)
        GPIO.cleanup()
```

8. **Client-Side Script Acting as Remote for Manual Control –**

```
    import datetime
    from tkinter import *
    from socketIO_client import SocketIO, BaseNamespace

    class Namespace(BaseNamespace):

        def on_connect(self, *args):
            print("@"+str(datetime.datetime.now().time())+" : Car Connected")

        def on_disconnect(self, *args):
            print("@"+str(datetime.datetime.now().time())+" : Car Disconnected")

    socketIO = SocketIO('192.168.43.55', 9999, Namespace)
    myNamespace = socketIO.define(Namespace, '/sdc')

    flagUp = False
    flagDown = False
    flagLeft = False
    flagRight = False

    def keyPressUp(e):
        global flagUp
        global myNamespace
        if flagUp==False:
            myNamespace.emit('throttle', {'key': str(e.keycode)})
            flagUp=True
```

```
def keyReleaseUp(e):
    global flagUp
    global myNamespace
    if flagUp==True:
        myNamespace.emit('brakes', {'key': str(e.keycode)})
        flagUp=False
def keyPressDown(e):
    global flagDown
    global myNamespace
    if flagDown==False:
        myNamespace.emit('throttle', {'key': str(e.keycode)})
        flagDown=True

def keyReleaseDown(e):
    global flagDown
    global myNamespace
    if flagDown==True:
        myNamespace.emit('brakes', {'key': str(e.keycode)})
        flagDown=False

def keyPressLeft(e):
    global flagLeft
    global myNamespace
    if flagLeft==False:
        myNamespace.emit('steer', {'key': str(e.keycode)})
        flagLeft=True

def keyReleaseLeft(e):
    global flagLeft
    global myNamespace
    if flagLeft==True:
        myNamespace.emit('release', {'key': str(e.keycode)})
        flagLeft=False

def keyPressRight(e):
    global flagRight
    global myNamespace
    if flagRight==False:
        myNamespace.emit('steer', {'key': str(e.keycode)})
        flagRight=True

def keyReleaseRight(e):
    global flagRight
    global myNamespace
    if flagRight==True:
        myNamespace.emit('release', {'key': str(e.keycode)})
        flagRight=False
```

```
if __name__ == '__main__':
    print("@"+str(datetime.datetime.now().time())+" : Remote client started")
    window = Tk()
    window.title("Self Driving Car")
    window.bind("<KeyPress-Up>", keyPressUp)
    window.bind("<KeyRelease-Up>", keyReleaseUp)
    window.bind("<KeyPress-Down>", keyPressDown)
    window.bind("<KeyRelease-Down>", keyReleaseDown)
    window.bind("<KeyPress-Left>", keyPressLeft)
    window.bind("<KeyRelease-Left>", keyReleaseLeft)
    window.bind("<KeyPress-Right>", keyPressRight)
    window.bind("<KeyRelease-Right>", keyReleaseRight)
    window.mainloop()
```

9. Software Side Issues –

i. Simultaneously controlling both front and back wheel: Although unnecessary on the autonomous part, but still to simplify each function call for simultaneous throttle and steer, key inputs were mentioned uniquely to fix the problem.

ii. Server-Client communication: Up until this point we had prepared a remote for testing the speed and steering of the car with various setup. This required creating a server (Raspberry Pi/Car) to handle input fed by the client (Laptop/Remote). Reverse communication, i.e. messages sent by the server and received by the client were difficult to handle through the libraries used: Flask and SocketIO. This problem was overcome by the following setup of 2-way server-client system.

10. Script for Training Model based on Data Collected –

```
import os
import argparse
import numpy as np
import pandas as pd
from keras.optimizers import Adam
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
from utils import INPUT_SHAPE, batch_generator
from sklearn.model_selection import train_test_split
from keras.layers import Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten

np.random.seed(0)
```

```
def load_data(args):
    data_df = pd.read_csv(os.path.join(os.getcwd(), args.data_dir,
'driving_log.csv'), names=['center', 'left', 'right', 'steering', 'throttle',
'reverse', 'speed'])
    X = data_df[['center', 'left', 'right']].values
    y = data_df['steering'].values
    X_train, X_valid, y_train, y_valid = train_test_split(X, y,
test_size=args.test_size, random_state=0)
    return X_train, X_valid, y_train, y_valid


def build_model(args):
    model = Sequential()
    model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
    model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    model.add(Dropout(args.keep_prob))
    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.summary()
    return model


def train_model(model, args, X_train, X_valid, y_train, y_valid):
    checkpoint = ModelCheckpoint('model-{epoch:03d}.h5', monitor='val_loss',
verbose=0, save_best_only=args.save_best_only, mode='auto')
    model.compile(loss='mean_squared_error',
optimizer=Adam(lr=args.learning_rate))
    model.fit_generator(batch_generator(args.data_dir, X_train, y_train,
args.batch_size, True), args.samples_per_epoch, args.nb_epoch, max_q_size=1,
validation_data=batch_generator(args.data_dir, X_valid, y_valid,
args.batch_size, False), nb_val_samples=len(X_valid), callbacks=[checkpoint],
verbose=1)

def s2b(s):
    s = s.lower()
    return s == 'true' or s == 'yes' or s == 'y' or s == '1'

def main():
    parser = argparse.ArgumentParser(description='Behavioral Cloning Training
Program')
    parser.add_argument('-d', help='data directory', dest='data_dir',
type=str, default='data')
```

```
    parser.add_argument('-t', help='test size fraction', dest='test_size',
type=float, default=0.2)
    parser.add_argument('-k', help='drop out probability', dest='keep_prob',
type=float, default=0.5)
    parser.add_argument('-n', help='number of epochs', dest='nb_epoch',
type=int, default=10)
    parser.add_argument('-s', help='samples per epoch',
dest='samples_per_epoch', type=int, default=20000)
    parser.add_argument('-b', help='batch size', dest='batch_size', type=int,
default=40)
    parser.add_argument('-o', help='save best models only',
dest='save_best_only', type=s2b, default='true')
    parser.add_argument('-l', help='learning rate', dest='learning_rate',
type=float, default=1.0e-4)
    args = parser.parse_args()

    print('-' * 30)
    print('Parameters')
    print('-' * 30)
    for key, value in vars(args).items():
        print('{:<20} := {}'.format(key, value))
    print('-' * 30)

    data = load_data(args)
    model = build_model(args)
    train_model(model, args, *data)

if __name__ == '__main__':
    main()
```

11.Script for Testing Model on the Simulator –

```
import os
import utils
import base64
import argparse
import eventlet
import socketio
import numpy as np
from PIL import Image
from io import BytesIO
from flask import Flask
from datetime import datetime
from keras.models import load_model

sio = socketio.Server()
app = Flask(__name__)

model = None
```

```
prev_image_array = None

MAX_SPEED = 25
MIN_SPEED = 10
speed_limit = MAX_SPEED

@sio.on('telemetry')
def telemetry(sid, data):
    if data:
      speed = float(data["speed"])
      throttle = float(data["throttle"])
      steering_angle = float(data["steering_angle"])
      image = Image.open(BytesIO(base64.b64decode(data["image"])))
      try:
            image = np.asarray(image)
            image = utils.preprocess(image)
            image = np.array([image])
            steering_angle = float(model.predict(image, batch_size=1))
            global speed_limit
            if speed > speed_limit:
                speed_limit = MIN_SPEED
            else:
                speed_limit = MAX_SPEED
            throttle = 1.0 - steering_angle**2 - (speed/speed_limit)**2
            #print('{} {} {}'.format(steering_angle, throttle, speed))
            send_control(steering_angle, throttle)
        except Exception as e:
            print(e)
    else:
        sio.emit('manual', data={}, skip_sid=True)

@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)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote Driving')
```

```
    parser.add_argument(
        'model',
        type=str,
        help='Path to model h5 file. Model should be on the same path.'
    )
    args = parser.parse_args()
    model = load_model(args.model)
    app = socketio.Middleware(sio, app)
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

12.On-board Script for Receiving Controls (Server) -

```
import base64
import datetime
import picamera
import RPi.GPIO as GPIO
from io import BytesIO
from flask import Flask, request
from flask_socketio import SocketIO, emit

#Rear Wheels for Driving
Motor1A = 23
Motor1B = 24

#Front Wheels for Steer
Motor2A = 26
Motor2B = 19

def accelerate():
    GPIO.output(Motor1A, GPIO.HIGH)
    GPIO.output(Motor1B, GPIO.LOW)

def reverse():
    GPIO.output(Motor1A, GPIO.LOW)
    GPIO.output(Motor1B, GPIO.HIGH)

def left():
    GPIO.output(Motor2A, GPIO.LOW)
    GPIO.output(Motor2B, GPIO.HIGH)

def right():
    GPIO.output(Motor2A, GPIO.HIGH)
    GPIO.output(Motor2B, GPIO.LOW)

def brakes():
    GPIO.output(Motor1A, GPIO.LOW)
    GPIO.output(Motor1B, GPIO.LOW)

def release():
```

```
    GPIO.output(Motor2A, GPIO.LOW)
    GPIO.output(Motor2B, GPIO.LOW)

app = Flask(__name__)
socketio = SocketIO(app)
socketio_connections = set()

def display(msg):
    print("@"+str(datetime.datetime.now().time())+" : "+str(msg))

@socketio.on('accelerate', namespace='/sdc')
def on_accelerate(msg):
    accelerate()
    display("Accelerating")

@socketio.on('left', namespace='/sdc')
def on_left(msg):
    left()
    display("Turning Left")

@socketio.on('right', namespace='/sdc')
def on_right(msg):
    right()
    display("Turning Right")

@socketio.on('brakes', namespace='/sdc')
def on_brakes(msg):
    brakes()
    display("Applied Brakes")

@socketio.on('release', namespace='/sdc')
def on_release(msg):
    release()
    display("Released Steer")

@socketio.on('connect', namespace='/sdc')
def on_connect():
    socketio_connections.add(request.sid)
    display("Remote Connected")

@socketio.on('disconnect', namespace='/sdc')
def on_disconnect():
    socketio_connections.remove(request.sid)
    display("Remote Disconnected")

if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(Motor1A, GPIO.OUT)
```

```
        GPIO.setup(Motor1B, GPIO.OUT)
        GPIO.setup(Motor2A, GPIO.OUT)
        GPIO.setup(Motor2B, GPIO.OUT)
        #GPIO.setwarnings(False)
        socketio.run(app, host='192.168.43.55', port=6666)
        GPIO.cleanup()
```

13.On-board Script for Sending Images (Client) -

```
import base64
import datetime
import picamera
import RPi.GPIO as GPIO
from time import sleep
from io import BytesIO
from socketIO_client import SocketIO, BaseNamespace

def display(msg):
    print("@"+str(datetime.datetime.now().time())+" : "+str(msg))

class Namespace(BaseNamespace):

    def on_connect(self, *args):
        display("Connected with Server")

    def on_disconnect(self, *args):
        display("Disconnected from Server\n")

display("Starting Connection with Server")
socketIO = SocketIO('192.168.43.227', 6668, Namespace)
myNamespace = socketIO.define(Namespace, '/sdc')
display("Connection Established with Server")

camera = picamera.PiCamera()
camera.resolution = (640, 320)
camera.rotation = 180
camera.start_preview()

def capture():
    myStream = BytesIO()
    camera.capture(myStream, 'jpeg')
    myString = base64.b64encode(myStream.getvalue())
    myNamespace.emit('image', {'value': str(myString)})

if __name__ == '__main__':
    display("Client Started")
    for i in range(0,250):
        display("Taking Picture "+str(i))
        capture()
```

```
        camera.stop_preview()
```

14.Script for a Separate Machine for Processing –

```
import utils
import base64
import datetime
import numpy as np
from PIL import Image
from io import BytesIO
from flask_socketio import emit
from flask import Flask, request
from keras.models import load_model
from socketIO_client import BaseNamespace
from flask_socketio import SocketIO as fsio
from socketIO_client import SocketIO as sioc

app = Flask(__name__)
serverSocket = fsio(app)
socketio_connections = set()

def display(msg):
    print("@"+str(datetime.datetime.now().time())+" : "+str(msg))


class Namespace(BaseNamespace):

    def on_connect(self, *args):
        display("Connected with Remote Server")

    def on_disconnect(self, *args):
        display("Disconnected from Remote Server\n")

model = load_model("model.h5")
clientSocket = sioc('192.168.43.55', 6666, Namespace)
myNamespace = clientSocket.define(Namespace, '/sdc')

def send_control(steering_angle):
    steering_angle = 100*steering_angle
    #steering_angle = 100
    if steering_angle > 0:
        #steering_angle = int(round(steering_angle))
        myNamespace.emit('right', {})
        #right
    elif steering_angle < 0:
        steering_angle = -1*steering_angle
        #steering_angle = int(round(steering_angle))
        myNamespace.emit('left', {})
        #left
```

```
        else:
            myNamespace.emit('release', {})

    @serverSocket.on('connect', namespace='/sdc')
    def on_connect():
        socketio_connections.add(request.sid)
        display("Connected with Image Client")
        display("Starting Car")
        myNamespace.emit('accelerate', {})

    @serverSocket.on('disconnect', namespace='/sdc')
    def on_disconnect():
        socketio_connections.remove(request.sid)
        display("Disconnected from Image Client")
        display("Stopping Car")
        myNamespace.emit('brakes', {})

    @serverSocket.on('image', namespace='/sdc')
    def on_image(msg):
        display("Received Image from Client")
        myString = base64.b64decode(msg['value'])
        myStream = BytesIO(myString)
        image = Image.open(myStream)
        image = np.asarray(image)
        image = utils.preprocess(image)
        image = np.array([image])
        steering_angle = float(model.predict(image, batch_size=1))
        display(steering_angle)
        send_control(steering_angle)

    if __name__ == '__main__':
        display("Server Started")
        serverSocket.run(app, host='192.168.43.227', port=6668)
```

15.Code Novelty –

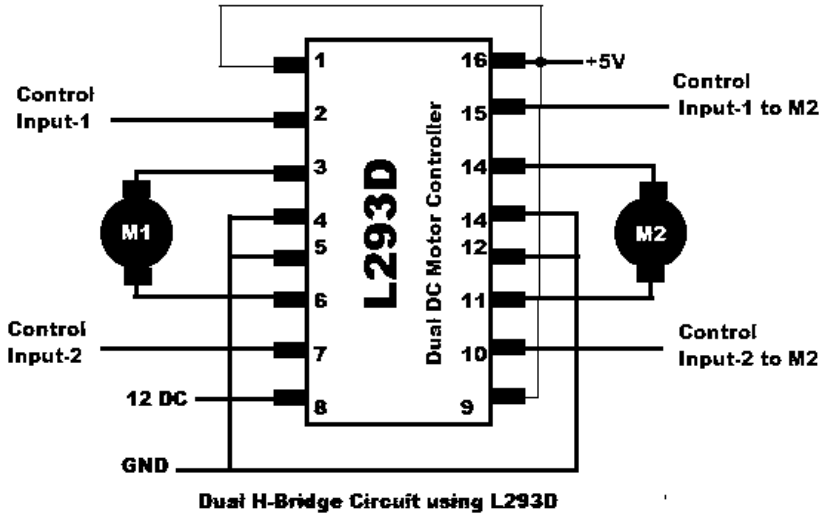
- Raspberry Pi was responsible for all the connections and inputs fed to the Motors to control the car. We ran two separate scripts with two separate ports numbers on the same IP address, but it didn't mean there was only one script in control at a time. One script ran to capture images and send at its own pace, while the other script ran to listen for control received from the processing done on the laptop.
- On the laptop the single script ran, both as a Client and Server at the same time; reason being we needed to allocate the same IP and port to this process responsible for calculating the steering angle. The server is always stuck in loop to listen for messages (i.e. the images), while the client part of the same script ready to send message (steering angle) only when the image has been processed.

16.Limitations –

i. 30 pictures per second processing speed is required for a seamless drive and turning experience. We could only achieve 1 picture per second. Biggest factor of this was we couldn't run the scripts on Raspberry Pi because of processing limitations.

ii. Network delay: Server-Client communication took delay than usual. This added to the processing delay made the whole thing more slow.

iii. No speed control for motor: The L293D motor control board has shorted enable with power input. This made us unable to control the speed of the motor which in turn couldn't regulate speed and turning angle.



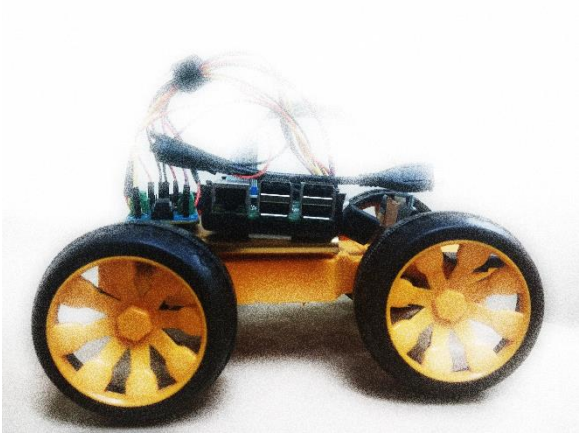
Dual H-Bridge Circuit using L293D

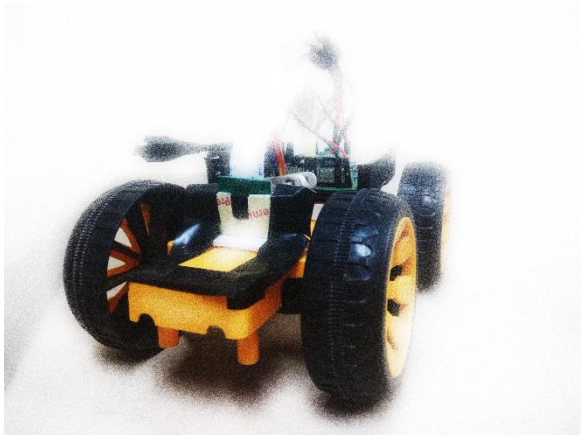
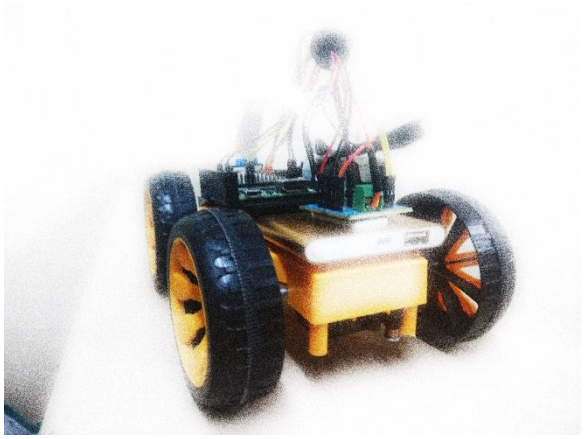
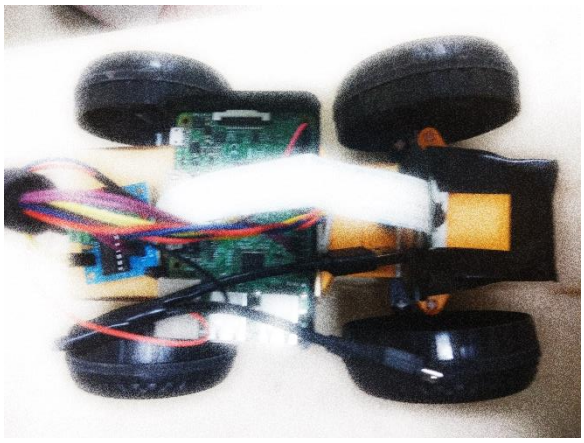
iv. Camera adjustment took time. Judging the simulation pictures and mounting the camera accordingly was a tricky job.

17.Future Scope –

- There is still room for improvement in the project. For starters all the limitations must be overcome.
- The next thing that needs to be done is the complete conversion of the project from simulation to real model. Only steering command was successfully integrated into the car. Speed and Throttle were neglected while demonstrating in final evaluation.
- Pre-trained model was used to test the car, training the car on its own would be much better for its own testing.
- Training while testing or at the least fixing its mistakes while testing and collecting the data and re training in background would a huge step for this project.
- More sensors can always be added for course correction and object collision detection.
- Making the car completely wireless by providing it with a power supply that can hold the entire setup was our biggest obstacle we faced. If this would have been eliminated, then many other problems would have went away with it.

18.Gallery –




19.References –

i. https://devblogs.nvidia.com/deep-learning-self-driving-cars/
Nvidia's paper titled End-to-End Deep Learning for Self-Driving Cars

ii. https://github.com/udacity/self-driving-car-sim
Udacity open-sourced Self-Driving Car Simulator

iii. https://github.com/udacity/CarND-Behavioral-Cloning-P3
Udacity provided a skeleton code for Car-Behavioural-Cloning

iv. https://github.com/kunal26das/Projects/tree/master/Minor%202
Github link for the project source code