**SELF DRIVING CAR**

**CREATE A SELF DRIVING CAR USING YOUR OWN DATA SET**

**On**

**APPLICATION USING PYTHON**

**SUBMITTED BY:**

**GANAGALA CHALAPATHI**

**UNDER**

[](https://coincent.ai/)**Coincent in association with Microsoft brings to you Certification, Industrial Training, Projects & Internship from our Partnered Companies.**

**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 responsing 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.

Keywords--- Self Driving Car, Human Input, Human Error, Traffic Signal Responsing.

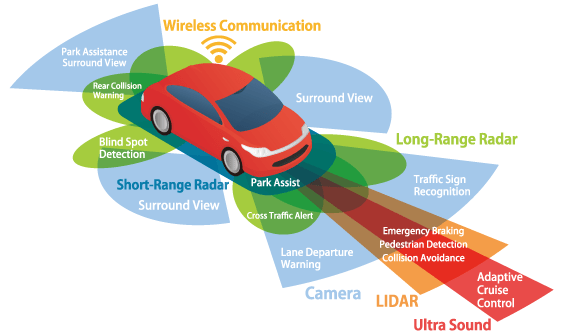
**OBJECTIVE :**

We develops various automated driving functions for daily traffic by dynamically adapting the level of automation to situation and driver status. Further, the project addresses legal issues that might impact successful market introduction.

**TARGETS FOR RESEARCH AND DEVELOPMENT**

* 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.
* Provide guidelines for the implementation of cooperative controls involving both drivers and automation.
* Define and validate specific evaluation methodologies.
* Assess the impact of automated driving on European road transport.
* Evaluate the legal framework with regards to existing implementation barriers.

INTRODUCTION TO SELF DRIVING CAR:



The concept of self driving car is discussed since 1920s, however first semi auto-mated car was developed by Japan in 1977.A major landmark self driving car was developed in 1980s with speed limited upto 31 kmph. And after some developments in 2013 Tesla started their autonomous car project. In 2014 they released Tesla S with semi-autopilot mode. And further improvements in the autopilot software are released in following models like Tesla X etc.A survey of opinion from public about self driving cars was conducted by Brandon Schoettle and Michael Sivak in top three major English speaking countries and intial response from the public is positive about upcoming technology in automobile industry. [1] But in our country there was no such development, this project analysed and took this as base. So with in this background we decided to develop a autonomous car at a low cost that could afforded by many citizens. And with the existing system we included Traffic signal response which is not present in Tesla and many other companies. With this the transportation quality and safety is improved as human errors made will be significantly reduced and road safety will be increased. And it also brings revolution in helping differently abled and blind people to travel independently.

**METHODOLOGY:**

After conducting experiments since 1920s, the first truly autonomous cars were developed in 1980s with Carnegie Mellon University’s Navlab and ALV projects in 1984 and Mercedes-Benz and Bundeswehr university Munich’s Eureka Prometheus Project in 1987. Following these, many companies like general motors, Nissan, Toyota, Audi, Volvo, Google, Tesla etc have started working on self driving vehicles project. After many years, in 1991, the United States Congress passed a bill ISTEA Transportation Authorization which stated that USDOT to demonstrate an automated vehicle and highway system by 1997.The Federal Highway Administration with some companies close headway platooning intended to operate in segregated traffic and free agent vehicle to operate in mixed traffic. Following that in 1995 Navlab project completed 3100 miles where 98% was automated. In 2005 world’s first driverless car, Parkshuttle came out. It uses artificial reference points in road to verify its position. Three Military projects Demo I, Demo II, Demo III funded by US Government in 2000s is a unmanned vehicle to navigate long difficult off road terrain. After some years the idea of autonomous car was brought into Grand Challenges but idea couldn’t be implemented in first attempt but in following events they achieved it.In US partial automation system(Level2 Systems) is made available by several automobile companies but conditional automation with some human input was under development(Level3 Systems). [2] The first complete test urban test for the self-driving car was made in 2007. [3] Although many prototypes where introduced by many companies even Google and Nissan , the revolution was made by TESLA in 2014. They launched first version of Model S equipped with system capable of lane detection with autonomous steering, braking and speed limit adjustment based on signals image recognition. But it is made suitable only for limited highway access nit for urban driving.They accumulatedover 1.2billion miles in Autopilot mode since 2015 As of now in 2017 and 2018 Volvo and Audi launched their model of cars wih autopilot. 2017 Waymo One a self-driving car was launched which was owned by Google once. It was launched as a limited taxi service in Arizona. Most of the models mentioned above are not fully autonomous since fully autonomous car are not made legal in US. So as of now, Tesla is yet to release their fully autonomous car which is expected in 2019 or in following years. But in India there is no development of autonomous car models. Still they have partial features of self driving like starting from cruise control, self-parking, automatic emergency brakes, lane change assist.

**CODE:**

Build My CNN (Convolutional Neural Network) Model based on the NVIDIA model

Designed and Run in Google Colaboratory.

Original file is located at

!ls myNewTrack

!pip3 install imgaug

import os

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from sklearn.utils import shuffle

from sklearn.model\_selection import train\_test\_split

import tensorflow.keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense

from imgaug import augmenters as iaa

import cv2

import pandas as pd

import ntpath

import random

# Load CSV file into Data Object

datadir = 'myNewTrack'

columns = ['center','steering','throttle']

data = pd.read\_csv(os.path.join(datadir, 'data\_img.csv'), names = columns)

pd.set\_option('display.max\_colwidth',None)

data.head()

# Remove image paths

def path\_leaf(path):

head,tail = ntpath.split(path)

return tail

data['center'] = data['center'].apply(path\_leaf)

data.head()

# Make sure steering data is unbiased

num\_bins = 25

samples\_per\_bin = 650

hist,bins = np.histogram(data['steering'], num\_bins)

center = (bins[:-1]+bins[1:]) \* 0.5 #center bins

plt.bar(center, hist, width=0.25)

plt.plot((np.min(data['steering']),np.max(data['steering'])), (samples\_per\_bin, samples\_per\_bin))

plt.show()

# Load image paths and steering values of servo motor

def load\_img\_steering(datadir1,datadir2,data):

img\_path = []

steering = []

for i in range(len(data)): #Each row of data object

indexed\_data = data.iloc[i]

center = indexed\_data[0]

if i < 1098:

img\_path.append(os.path.join(datadir1,center.strip()))

steering.append(float(indexed\_data[1]))

elif (i < 1182 or i > 1204): #else:

img\_path.append(os.path.join(datadir2,center.strip()))

steering.append(float(indexed\_data[1]))

img\_paths = np.asarray(img\_path)

steerings = np.asarray(steering)

return img\_paths,steerings

image\_paths, steerings = load\_img\_steering(datadir + '/IMG',datadir + '/IMG2',data)

print(image\_paths)

print(steerings)

# Split data randomly with 80% to be used as training data and 20% as validation data

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(image\_paths,steerings,test\_size=0.2,random\_state=6)

#print(X\_train)

#print(y\_train)

print('Training Sample: {}\nValid Samples: {}'.format(len(X\_train), len(X\_valid)))

# Make sure data split kept steering unbiased

fig,axs = plt.subplots(1,2,figsize=(12,4))

axs[0].hist(y\_train,bins=num\_bins,width=0.25,color='blue')

axs[0].set\_title('Training Set')

axs[1].hist(y\_valid,bins=num\_bins,width=0.25,color='red')

axs[1].set\_title('Validation Set')

# Data augmentation techniques

# Zoom image randomly by 30%

def zoom(img):

zoom = iaa.Affine(scale=(1,1.3))

img = zoom.augment\_image(img)

return img

# Visualize zoom function

image = image\_paths[random.randint(0,1500)]

original\_image = mpimg.imread(image)

zoomed\_image = zoom(original\_image)

fig,axs = plt.subplots(1,2,figsize=(15,10))

fig.tight\_layout()

axs[0].imshow(original\_image)

axs[0].set\_title('Original Image')

axs[1].imshow(zoomed\_image)

axs[1].set\_title('Zoomed Image')

# Pan image randomly +-10%

def pan(img):

pan = iaa.Affine(translate\_percent={"x": (-0.1,0.1), "y": (-0.1, 0.1)})

img = pan.augment\_image(img)

return img

# Visualize pan function

image = image\_paths[random.randint(0,1500)]

original\_image = mpimg.imread(image)

panned\_image = pan(original\_image)

fig,axs = plt.subplots(1,2,figsize=(15,10))

fig.tight\_layout()

axs[0].imshow(original\_image)

axs[0].set\_title('Original Image')

axs[1].imshow(panned\_image)

axs[1].set\_title('Panned Image')

# Randomly alter image brightness

def img\_random\_brightness(img):

brightness = iaa.Multiply((0.2, 1.2))

img = brightness.augment\_image(img)

return img

# Visualize brightness alteration

image = image\_paths[random.randint(0,1500)]

original\_image = mpimg.imread(image)

rand\_image\_brightness = img\_random\_brightness(original\_image)

fig,axs = plt.subplots(1,2,figsize=(15,10))

fig.tight\_layout()

axs[0].imshow(original\_image)

axs[0].set\_title('Original Image')

axs[1].imshow(rand\_image\_brightness)

axs[1].set\_title('Random Image Brightness')

# Randomly flip image horizontally and flip steering angle as well to match flipped image

def img\_random\_flip(img,steering\_angle):

img=cv2.flip(img,1) #1 = horizontal flip

steering\_angle = 12.75\*2-steering\_angle #13 means straight forward

return img, steering\_angle

# Visualize image & steering flip

random\_idx = random.randint(0,1500)

image = image\_paths[random\_idx]

steering\_angle = steerings[random\_idx]

original\_image = mpimg.imread(image)

rand\_image\_flip, flipped\_steering\_angle = img\_random\_flip(original\_image, steering\_angle)

fig,axs = plt.subplots(1,2,figsize=(15,10))

fig.tight\_layout()

axs[0].imshow(original\_image)

axs[0].set\_title('Original Image - ' + 'Steering Angle ' + str(steering\_angle))

axs[1].imshow(rand\_image\_flip)

axs[1].set\_title('Random Image Flip - ' + 'Steering Angle ' + str(flipped\_steering\_angle))

# Randomly combine the assortment of image augmentation techniques on any one image

def random\_augment(image,steering\_angle):

img = mpimg.imread(image)

if np.random.rand() < 0.5: #occurs about 50% of the time

img = pan(img)

if np.random.rand() < 0.5:

img = zoom(img)

if np.random.rand() < 0.5:

img = img\_random\_brightness(img)

if np.random.rand() < 0.5:

img,steering\_angle = img\_random\_flip(img,steering\_angle)

return img, steering\_angle

# Visualize random\_augment function

numcols = 2

numrows = 10

fig,axs = plt.subplots(numrows,numcols,figsize=(15,50))

fig.tight\_layout

for ii in range(numrows):

randnum = random.randint(0,len(image\_paths)-1)

random\_image = image\_paths[randnum]

random\_steering = steerings[randnum]

original\_image = mpimg.imread(random\_image)

augmented\_image, aug\_steering = random\_augment(random\_image, random\_steering)

axs[ii][0].imshow(original\_image)

axs[ii][0].set\_title("Original Image")

axs[ii][1].imshow(augmented\_image)

axs[ii][1].set\_title("Augmented Image")

# Preprocess image to match dimensions and color scheme used in the NVIDIA model

def img\_preprocess(img):

#img = mpimg.imread(img\_path)

img = img[20:240, :, :]

#YUV is important when using NVIDIA Model

img = cv2.cvtColor(img, cv2.COLOR\_RGB2YUV) #Y=luminescence; U,V=chrominance

img = cv2.GaussianBlur(img, (3,3), 0) #helps get rid of noise

img = cv2.resize(img, (200, 66)) #matches size used in NVIDIA model architecture

img = img/255 #normalize image

return img

# Visualize preprocessed image

random\_idx = random.randint(0,1500)

image = image\_paths[random\_idx]

orig\_image = mpimg.imread(image)

preprocessed\_image = img\_preprocess(orig\_image)

fig,axs = plt.subplots(1,2,figsize=(15,10))

fig.tight\_layout()

axs[0].imshow(orig\_image)

axs[0].set\_title('Original Image')

axs[1].imshow(preprocessed\_image)

axs[1].set\_title('Preprocessed Image')

print(orig\_image.shape)

# Randomly choose batch\_size number of images from image\_paths

def batch\_generator(image\_paths, steering\_ang, batch\_size, istraining):

while(True):

batch\_img = []

batch\_steering = []

for ii in range(batch\_size):

random\_index = random.randint(0, len(image\_paths)-1)

if istraining:

img, steering = random\_augment(image\_paths[random\_index], steering\_ang[random\_index])

else:

img = mpimg.imread(image\_paths[random\_index])

steering = steering\_ang[random\_index]

img = img\_preprocess(img)

batch\_img.append(img)

batch\_steering.append(steering)

yield (np.asarray(batch\_img), np.asarray(batch\_steering))

# Test batch\_generator function

#x\_train\_gen, y\_train\_gen = next(batch\_generator(X\_train, y\_train, 1, 1))

#x\_valid\_gen, y\_valid\_gen = next(batch\_generator(X\_valid, y\_valid, 1, 0))

#fig,axs = plt.subplots(1,2,figsize=(15,10))

#fig.tight\_layout()

#axs[0].imshow(x\_train\_gen[0])

#axs[0].set\_title('Training Image')

#axs[1].imshow(x\_valid\_gen[0])

#axs[1].set\_title('Validation Image')

# The buidling and layers inside my NVIDIA model for machine learning

def nvidia\_model():

model = Sequential()

model.add(Conv2D(24, (5, 5), strides=(2,2), input\_shape=(66,200,3), activation='elu'))

model.add(Conv2D(36, (5,5), strides=(2,2), activation='elu'))

model.add(Conv2D(48, (5,5), strides=(2,2), activation='elu'))

model.add(Conv2D(64, (3,3), activation='elu'))

model.add(Conv2D(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(Dropout(0.5))

model.add(Dense(1))

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

return model

model = nvidia\_model()

print(model.summary())

# Train & validate trained machine learning model on data

h = model.fit(batch\_generator(X\_train, y\_train, 300, 1),

steps\_per\_epoch=300,

epochs=15,

validation\_data = batch\_generator(X\_valid, y\_valid, 150, 0),

validation\_steps=200,

verbose=1,

shuffle=1)

# Plot resultant loss of model

plt.plot(h.history['loss'])

plt.plot(h.history['val\_loss'])

plt.legend(['Training Data', 'Validation Data'])

plt.title('Loss')

plt.xlabel('Epoch')

model.save('model.h5')

# Only used in Google Colab

#from google.colab import files

#files.download('model.h5')

# Convert model to a TensorFlow-Lite model to use in Raspberry Pi

from tensorflow import lite

converter = lite.TFLiteConverter.from\_keras\_model(model)

tflite\_model = converter.convert()

open("model.tflite","wb").write(tflite\_model)

#files.download('model.tflite')

###############################

#### Stop Car at Stop Sign ####

#### --------------------- ####

#### Objective: ####

#### > Drive car forward ####

#### until a stopsign is ####

#### spotted on the cam ####

###############################

from picamera.array import PiRGBArray

from picamera import PiCamera

import RPi.GPIO as GPIO

from time import sleep

import numpy as np

import cv2

# Inializations: #

#Camera fps/size

camera=PiCamera()

camera.resolution=(640,480)

camera.framerate=20

rawCapture=PiRGBArray(camera,size=(640,480))

sleep(0.1)

#Car PIN setups

enA=14

in1=15

in2=18

temp1=1

servo=22

angle=45

GPIO.setmode(GPIO.BCM)

GPIO.setup(enA,GPIO.OUT)

GPIO.setup(in1,GPIO.OUT)

GPIO.setup(in2,GPIO.OUT)

GPIO.setup(servo,GPIO.OUT)

GPIO.output(in1,GPIO.LOW)

GPIO.output(in2,GPIO.LOW)

pwm\_servo=GPIO.PWM(servo,100)

pwm\_motor=GPIO.PWM(enA,1000)

pwm\_servo.start(12.5)

pwm\_motor.start(25)

def make\_coordinate(img,line\_parameters):

slope,intercept=line\_parameters

y1=img.shape[0]

y2=int(y1\*(3/5))

x1=int((y1-intercept)/slope)

x2=int((y2-intercept)/slope)

return np.array([x1,y1,x2,y2])

def average\_slope\_intercept(img,lines):

left\_fit=[]

right\_fit=[]

for line in lines:

x1,y1,x2,y2 = line.reshape(4)

# returns slope 1st and y-intercept 2nd

parameters = np.polyfit((x1,x2),(y1,y2),1)#degree 1

slope=parameters[0]

intercept=parameters[1]

if slope < 0:

left\_fit.append((slope,intercept))

else:

right\_fit.append((slope,intercept))

left\_fit\_average=np.average(left\_fit,axis=0)

right\_fit\_average=np.average(right\_fit,axis=0)

left\_line=make\_coordinate(img, left\_fit\_average)

right\_line=make\_coordinate(img, right\_fit\_average)

return np.array([left\_line, right\_line])

def region\_of\_interest(img, vertices):

mask = np.zeros\_like(img)

match\_mask\_color = 255 #only one color because it is a gray image

cv2.fillPoly(mask, vertices, match\_mask\_color)

masked\_image = cv2.bitwise\_and(img, mask)

return masked\_image

def draw\_the\_lines(img,lines):

img=np.copy(img)

blank\_img = np.zeros((img.shape[0],img.shape[1],3), dtype=np.uint8)

for line in lines:

for x1,y1,x2,y2 in line:

#line(image,pt1,pt2,color,thickness=None,lineType=None,shift=None)

cv2.line(blank\_img,(x1,y1),(x2,y2),(0,255,0),thickness=4)

img=cv2.addWeighted(img,0.8,blank\_img,1,0.0)

return img

def process(img):

#image dimensions

height = img.shape[0]

width = img.shape[1]

#Bottom triangle region

#region\_of\_interest\_vertices = [

# (0, height),

# (width/2, height/2),

# (width, height)

#]

#Bottom half of image

region\_of\_interest\_vertices = [

(0, height),

(0, height/2),

(width, height/2),

(width, height)

]

gray\_img = cv2.cvtColor(img,cv2.COLOR\_RGB2GRAY)

#Gaussian Blur will remove noise in image

blur = cv2.GaussianBlur(gray\_img, (5,5),0)

#Canny(img,minThreshold,maxThreshold) 1:2 or 1:3

canny\_image = cv2.Canny(gray\_img,50,150)

cropped\_image = region\_of\_interest(canny\_image,

np.array([region\_of\_interest\_vertices], np.int32),)

lines = cv2.HoughLinesP(cropped\_image,

#smaller rho/theta=more accurate longer processing time

rho=6, #number of pixels

theta=np.pi/60,

threshold=160,

lines=np.array([]),

minLineLength=40,

maxLineGap=25)

#image\_w\_lines=draw\_the\_lines(img,lines)

averaged\_lines=average\_slope\_intercept(img,lines)

image\_w\_lines=draw\_the\_lines(img,averaged\_lines)

return image\_w\_lines

# Begin Camera video and driving forward #

camera.start\_preview()

for frame in camera.capture\_continuous(rawCapture,format="rgb",use\_video\_port=True):

# Begin driving at medium speed

#pwm\_motor.ChangeDutyCycle(50) #medium speed

#pwm\_motor.ChangeDutyCycle(80) #high speed

pwm\_motor.ChangeDutyCycle(70) #med-hi speed

GPIO.output(in1,GPIO.HIGH)

GPIO.output(in2,GPIO.LOW)

pwm\_servo.ChangeDutyCycle(12.5)

# grab raw NumPy array representing image - 3D array

image=frame.array

#Find lanes in image region of interest and draw lines on them

image = process(image)

# Wait 1 ms and read key input

if cv2.waitKey(1) & 0xFF == ord('q'):

break

#clear the stream in preparation for the next frame

rawCapture.truncate(0)

# End camera functions

camera.stop\_preview()

cv2.destroyAllWindows()

camera.close()

# End car functions

pwm\_servo.stop()

pwm\_motor.stop()

GPIO.cleanup()

###############################

#### Stop Car at Stop Sign ####

#### --------------------- ####

#### Objective: ####

#### > Drive car forward ####

#### until a sotpsign is ####

#### spotted on the cam ####

###############################

from picamera.array import PiRGBArray

from picamera import PiCamera

import RPi.GPIO as GPIO

from time import sleep

import cv2

# Inializations: #

#Camera fps/size

camera=PiCamera()

camera.resolution=(640,480)

camera.framerate=20

rawCapture=PiRGBArray(camera,size=(640,480))

sleep(0.1)

stopsign\_cascade=cv2.CascadeClassifier('/home/pi/Desktop/oldRPi/RPi/stopsign\_good.xml')

#Car PIN setups

enA=13

in1=27

in2=17

temp1=1

servo=12

angle=45

GPIO.setmode(GPIO.BCM)

GPIO.setup(enA,GPIO.OUT)

GPIO.setup(in1,GPIO.OUT)

GPIO.setup(in2,GPIO.OUT)

GPIO.setup(servo,GPIO.OUT)

GPIO.output(in1,GPIO.LOW)

GPIO.output(in2,GPIO.LOW)

pwm\_servo=GPIO.PWM(servo,100)

pwm\_motor=GPIO.PWM(enA,1000)

pwm\_servo.start(13)

pwm\_motor.start(25)

# Begin Camera video and driving forward #

camera.start\_preview()

camera.start\_recording('stopsign\_video.h264')

for frame in camera.capture\_continuous(rawCapture,format="bgr",use\_video\_port=True):

# Begin driving at medium speed

pwm\_motor.ChangeDutyCycle(35) #medium speed

#pwm\_motor.ChangeDutyCycle(80) #high speed

#pwm\_motor.ChangeDutyCycle(70) #med-hi speed

GPIO.output(in1,GPIO.HIGH)

GPIO.output(in2,GPIO.LOW)

pwm\_servo.ChangeDutyCycle(13)

# grab raw NumPy array representing image - 3D array

image=frame.array

#convert image to grayscale

gray\_img=cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

# Wait and read key input

key=cv2.waitKey(1) & 0xFF

# Find stopsign in image

print("Before stop sign finding")

found\_stopsigns=stopsign\_cascade.detectMultiScale(gray\_img,1.1,5)

print("Found "+str(len(found\_stopsigns))+" stop sign(s)")

if len(found\_stopsigns)>0:

for (x,y,w,h) in found\_stopsigns:

cv2.rectangle(image,(x,y),(x+w,y+h),(255,255,0),2)

camera.add\_overlay(image)

cv2.imwrite("found\_stopsign\_Jun\_25\_20.jpg",image)

sign\_width=w

sign\_height=h

print("width of stop sign:",w,"and height:",h)

if(sign\_width>65 or sign\_height>65):

print("Turn on brake lights")

print("Decrease motor speed")

print("stop car")

GPIO.output(in1,GPIO.LOW)

GPIO.output(in2,GPIO.LOW)

sleep(1)

break

#clear the stream in preparation for the next frame

rawCapture.truncate(0)

# if the 'q' key was pressed or a stop sign was found

# break from the loop

if key == ord("q"): #or len(found\_stopsigns)>0:

break

# End camera functions

camera.stop\_recording()

camera.stop\_preview()

cv2.destroyAllWindows()

camera.close()

# End car functions

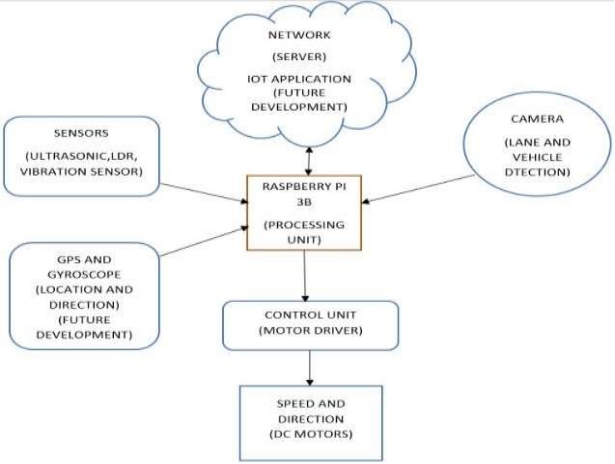
pwm\_servo.stop()

pwm\_motor.stop()

GPIO.cleanup()

***PROPOSED SYSTEM:***

This project plans to planning to provide a self driving car with a system that can navigate between two places on the map, detect any obstacles, lane detection [5], accident avoidance and emergency services. And our project’s uniqueness is we are implementing traffic signal responsing which is not present in Tesla and other companies car. Above figure is the basic block diagram of this project.



**Components used in this project :**

Raspberry Pi The below shown image is Raspberry Pi .It enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.

Specifications

· Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz

· 1GB, 2GB or 4GB LPDDR4-3200 SDRAM (depending on model)

· 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE

· Gigabit Ethernet · 2 USB 3.0 ports; 2 USB 2.0 ports

· 2 × micro-HDMI ports (up to 4kp60 supported) · OpenGL ES 3.0 graphics.



***Ultrasonic Sensor***

The figure given below is Ultrasonic sensors. It measure distance by using ultrasonic waves. The sensor heademitsan ultrasonic wave and receives the wave reflected back from thetarget.



***Ultrasonic Sensor***

### ***LDR Sensor***

The figure given below is LDR Sensor. It is basically a photocell that works on the principle of photoconductivity.



Fig f: LDR Sensor

### ***Vibration Sensor***

The Fig(g) is Vibration Sensor. It is a piezoelectric accelerometer that sense vibration. They are used for measuring fluctuating accelerations or speeds or for normal vibration measurement.

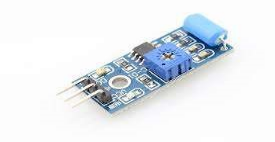


Fig g: Vibration Sensor

### ***Camera***

* Logitech c270
* For lane and vehicle detection

### ***Technical Support***

* Max Resolution: 720p/30fps
* Focus type: fixed focus
* Lens technology: standard
* Built-in mic: mono
* FoV: 60°
* Universal clip fits laptops, LCD or monitors



* Fig h: Logitech c270

### ***GPS***

A GPS navigation system is a GPS receiver designed for a specific purpose such as a car-based or hand-held device or a smartphone app. The global positioning system (GPS) is a 24- vigation system that uses multiple satellite signals to find a receiver’s position on earth.

### ***Gyroscope***

A device used for measuring or maintaining orientation and angular velocity.



Fig i: Gyroscope Sensor

### ***Motor Drivers***

The figure given below is Motor drives.It is used for motor interfacing. These drive circuits can be easily interfaced with the motor and their selection depends upon the type of motor being used and their ratings(Current, Voltage).

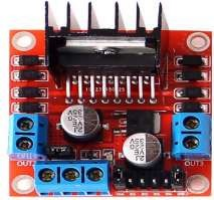


Fig j: Motor Driver Circuit

### ***DC Motor***

The figure given below is DC Motor.It is used for speed and direction.



Fig k: DC Motor

### ***Network(IOT Application*)**

Internet of Things(IoT) describes the large and growing set of digital devices – now numbering in the billions – that operate across networks of potentially global scale. As opposed to the regular Internet (of people), the IoT is comprised only of smart sensors another devices. Its gathers operational data from remote sensors for collecting and controlling.

**RESULT**

The below figure is the output for Lane Detection from running the python code for our model in Compiler. The YELLOW LINES indicate the detection of lanes.



Fig l: Lane Detection Output

The below figure in the picture captured via camera integrated in the model where the blue strip indicates lane and green line indicates the lane detection.



Fig m: Output from Camera Mounted in Model

The below figure is the output for Traffic signal responding from running the python code for our model in Compiler. With Hough gradient, the colour of the light from signal is indicated whether it is red or yellow or green.



Fig n: Output fot Traffic Signal Responding

The below figure is the complete model of self-driving car.With all the above mentioned components integrated and connected with chassis.



Fig o: Self Driving Car Model

## 

## CONCLUSION:

Since Self Driving Car is the major upgradation in automatable industry in future, this project focuses on bring changes in road safety and commuting and significantly reduce accidents and human errors through continuous learning by the system. This project will be a revolution in transporting differently abled people and blind people can drive independently. With our product as base mobile applications can be developed where owner summon the vehicle via the app and produce a fully autonomous car on passing the law (Fully autonomous cars are still illegal, but will be the future mode of transport).

**GitHub URL:**

[**https://github.com/ganagalachalapathi75/self-driving-car.git**](https://github.com/ganagalachalapathi75/self-driving-car.git)

**COINCENT Registered ID:**

[**Ganagalachalapathi829@gmail.com**](mailto:Ganagalachalapathi829@gmail.com)

**PHONE NO:**

**9110395281**