Department of Electronics & Communication Engineering,

MANIT Bhopal

Minor Project Report (EC-326)

Group No. : 9

Project Title: ROBOT NAVIGATION USING IMAGE PROCESSING

S.No.	Scholar No.	Name	Signature
1	171114031	Rahul Kumar	Rahul Kumar
2	171114105	Neeteesh Kumar	Neeteesh kumar
3	171114119	Sachin Chauhan	Sachin chauhan
4	171114094	Nilabh Ranjan Singh	Nilabh Ranjan Singh
5	171114123	Amit Kumar	Amit kumar

Name & Signature of faculty mentor : DR. Dheeraj Agrawal

Introduction:

This project explains about an automated guided vehicle (robot vehicle) which will be developed for operating on the roads with real life like environment. The robot will be designed as an outdoor robot which can detect and follow lanes, traffic light and stop at an end point.The Robot will also detect obstacle coming in path and will make decision according to distance from obstacle.The robot will be controlled using image processing .Traffic signs carry a lot of information about the traffic andenvironmentetc. which provides great assistance in driving. Hence, traffic sign identification is also very important for intelligent transport system or unmanned ground vehicle. The aim of this paper is to present a method for visual servo control using onlyvisual images from a webcam. Visual servo is the use of image data in closed loop control of a robot. Without doubt, today, the use of vision in robotic applications is rapidly increasing. This is due to the fact that vision based sensors such as webcams are falling in price more rapidly than any other sensor. It is also a richer sensor than traditional ranging devices,particularly since a camera captures much more data simultaneously.

The image is recorded by a webcam which is installed above the robot. The image is then sent via a USB cable to a PC, to be processed by OpenCV.

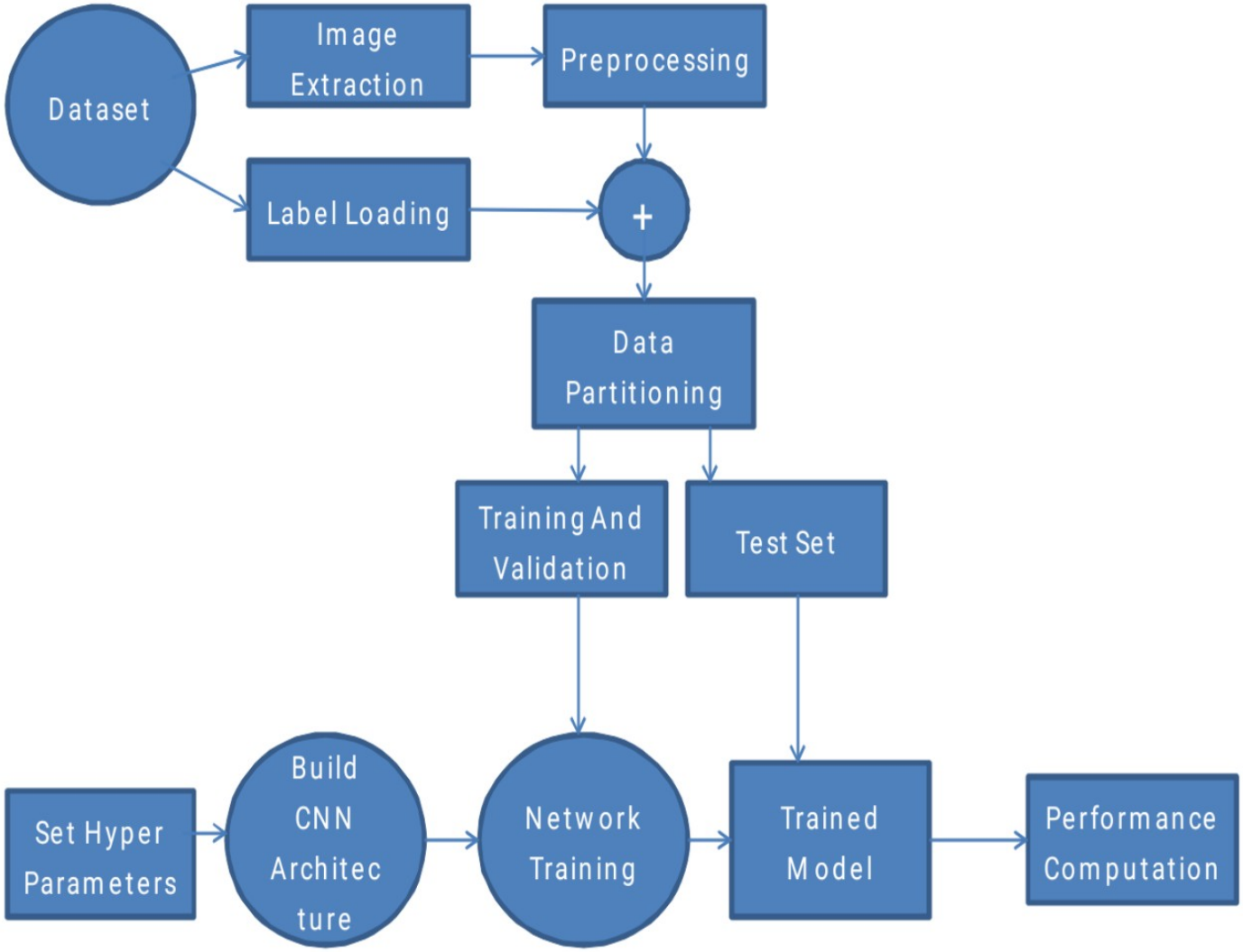
Objective:

To develop a robot which will navigate using image processing and will do the following tasks:

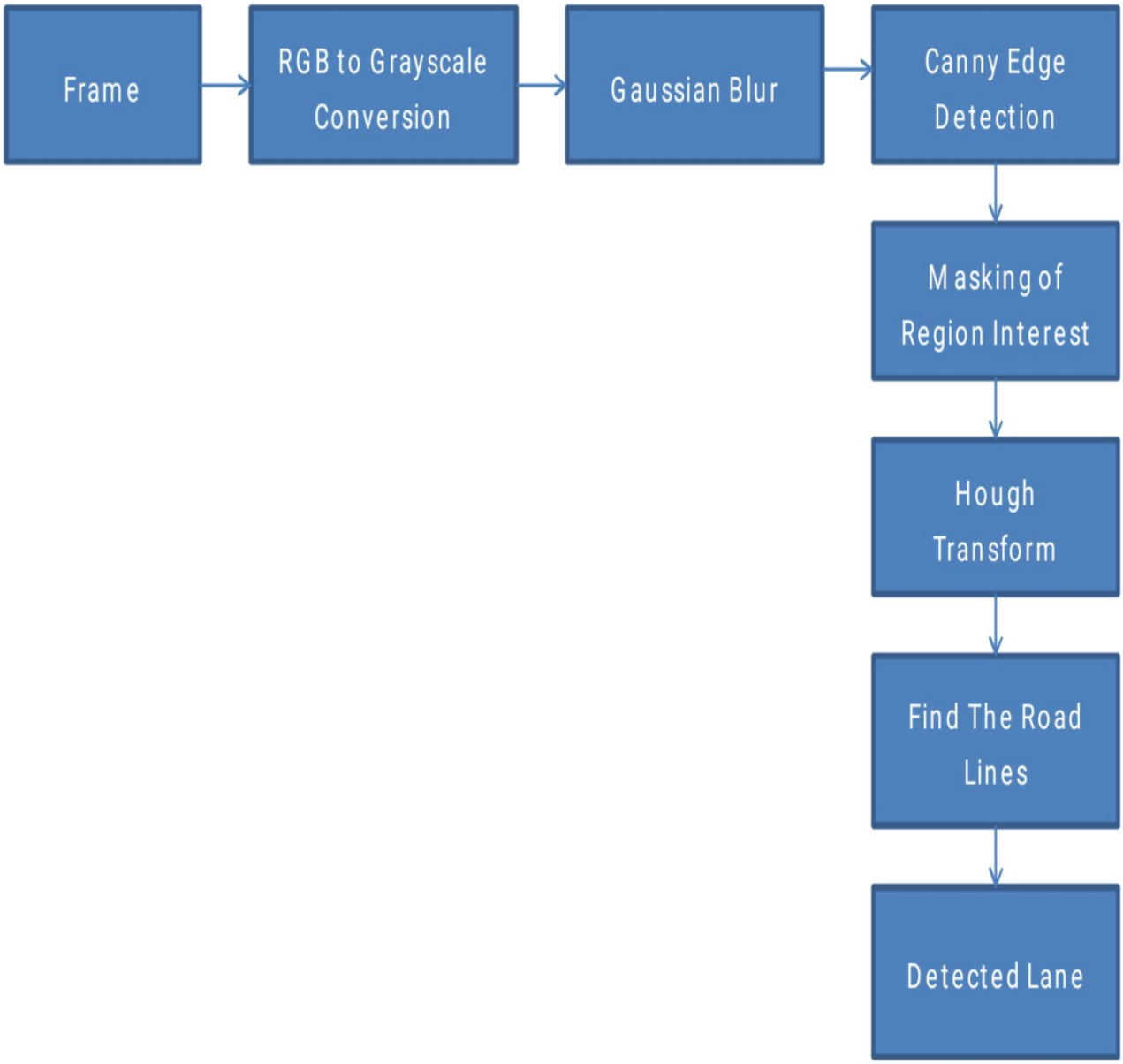
- *Traffic light ,lane and stop sign detection*
- *Obstacles Avoidance*

Block Diagram:

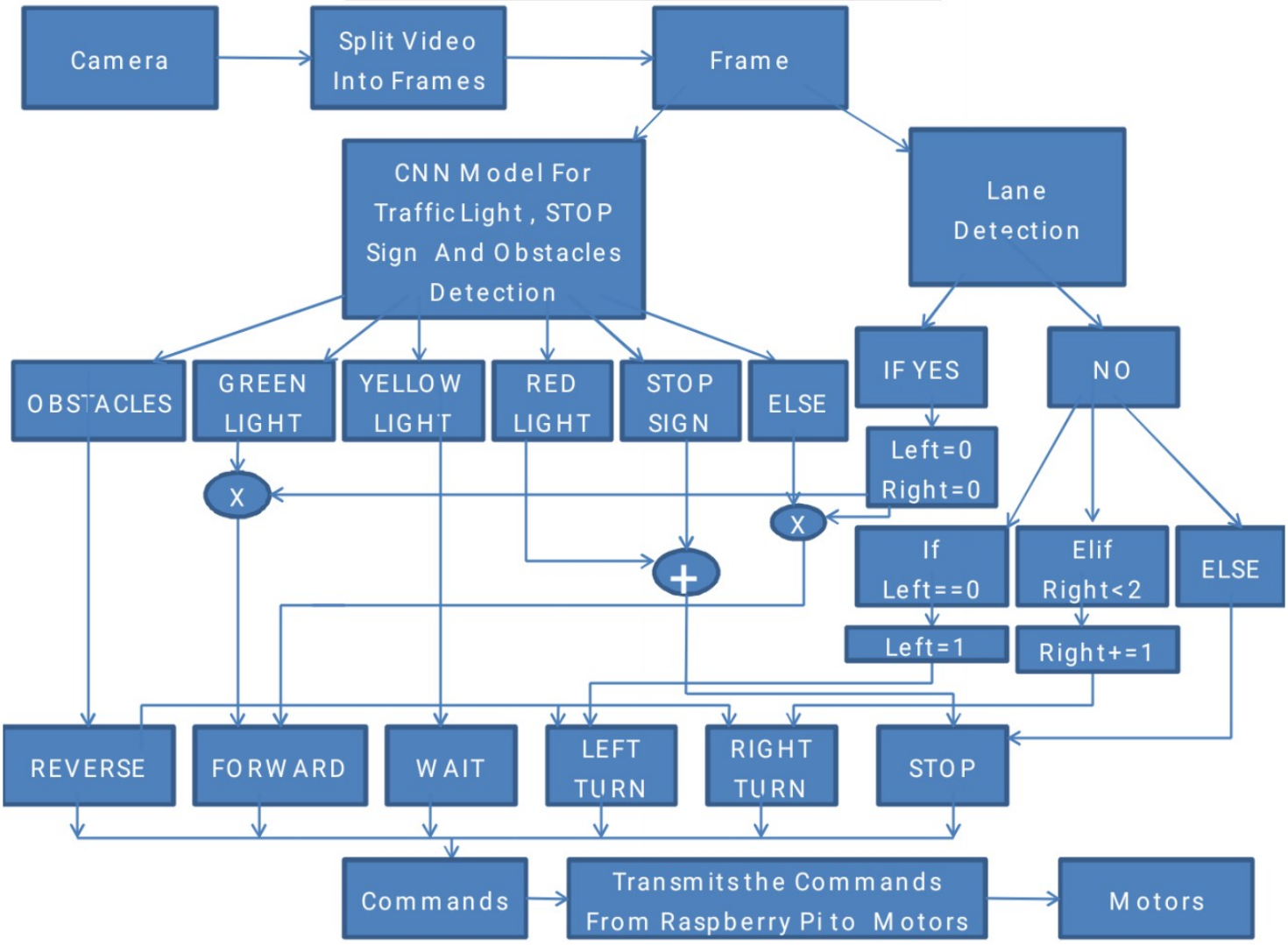
1.Block Diagram For CNN Model To Classify Traffic Lights, STOP Sign And Obstacles



2. Block Diagram For Lane Detection



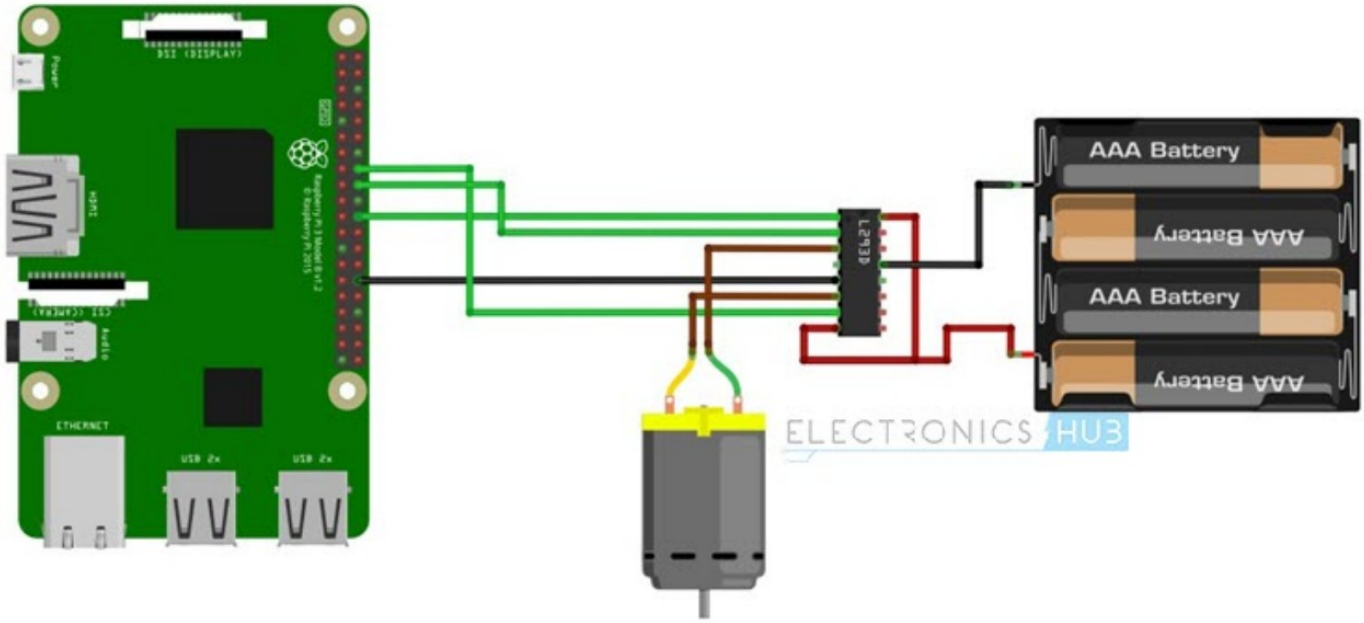
3. BLOCK DIAGRAM TO CONTROL ROBOT



Circuit Diagram

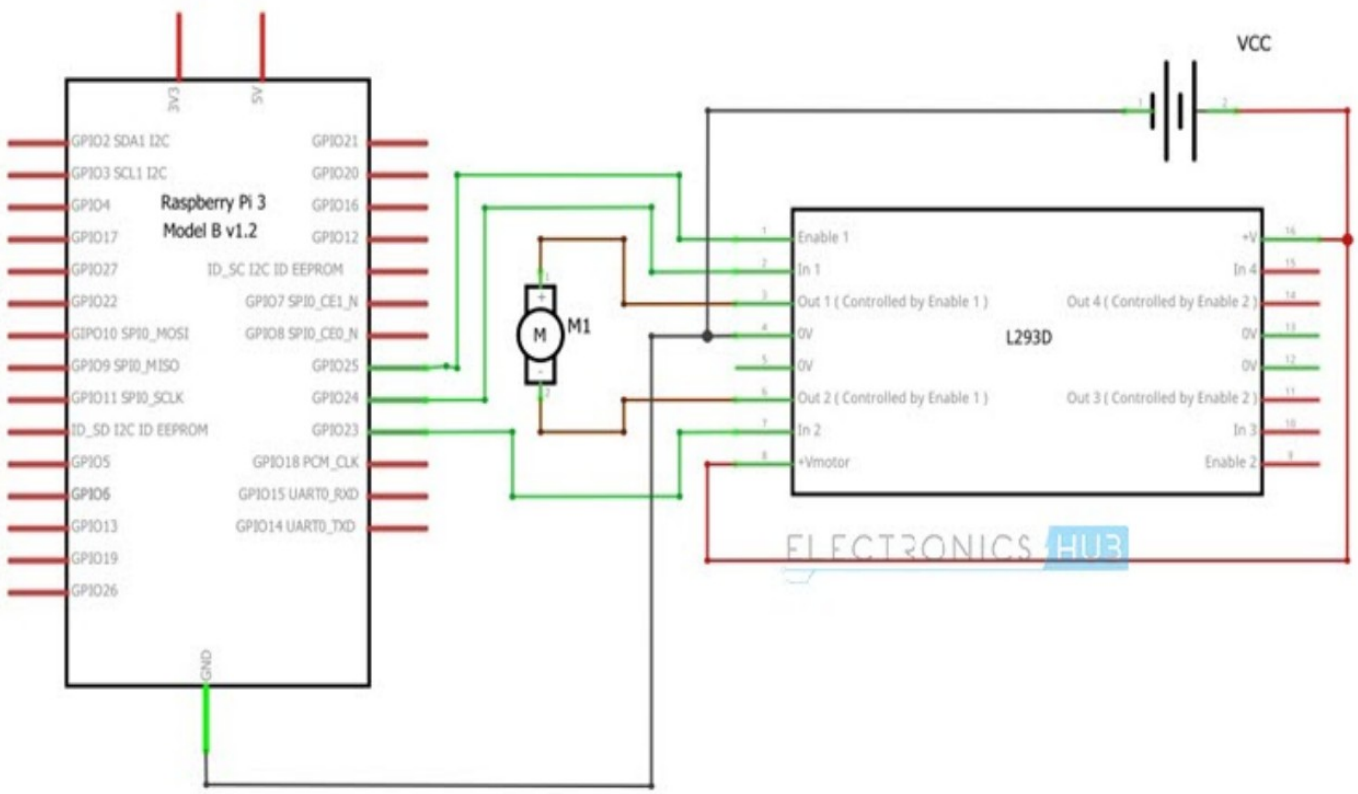
Fritzing Image

With the L293D Motor Driver IC, we can actually control two motors. For simplicity reasons, We will demonstrate the circuit, working and program for controlling a single DC Motor with Raspberry Pi. The following image is the Fritzing diagram of the project.



Circuit Diagram

The circuit wiring diagram of the project is shown below. We can easily configure this circuit and the program for controlling two DC Motors with Raspberry Pi and L293D Motor Driver IC.



The Building Blocks :

Hardware

- Chassis from an old Nikko RC Car
- Raspberry Pi 3 + Camera Module
- Arduino Uno + Custom Shield
- DC Motor Controller
- 3D printed parts
- Servomotor
- *Emergency Stop Button* (Essential !)
- Batteries: original Nikko battery for the motor + External battery for
- the Raspberry Pi + Arduino

1. HARDWARE DESIGN

1.1 List of Hardware

A pre-built four wheel drive (4WD) chassis is used as a base on which following hardware components are fit :

- Raspberry Pi (rev B) for GPU and CPU computations
- Wi-Fi 802.11n dongle to connect to Pi remotely
- Motor driver IC L293D which can control two motors

- 8 AAA batteries to provide power
- Jumper wires to connect individual components
- L shaped aluminium strip to support camera
- Pi camera
- Ultrasonic sensor to detect obstacles
- Servo motor to make the head (camera) flexible to rotation

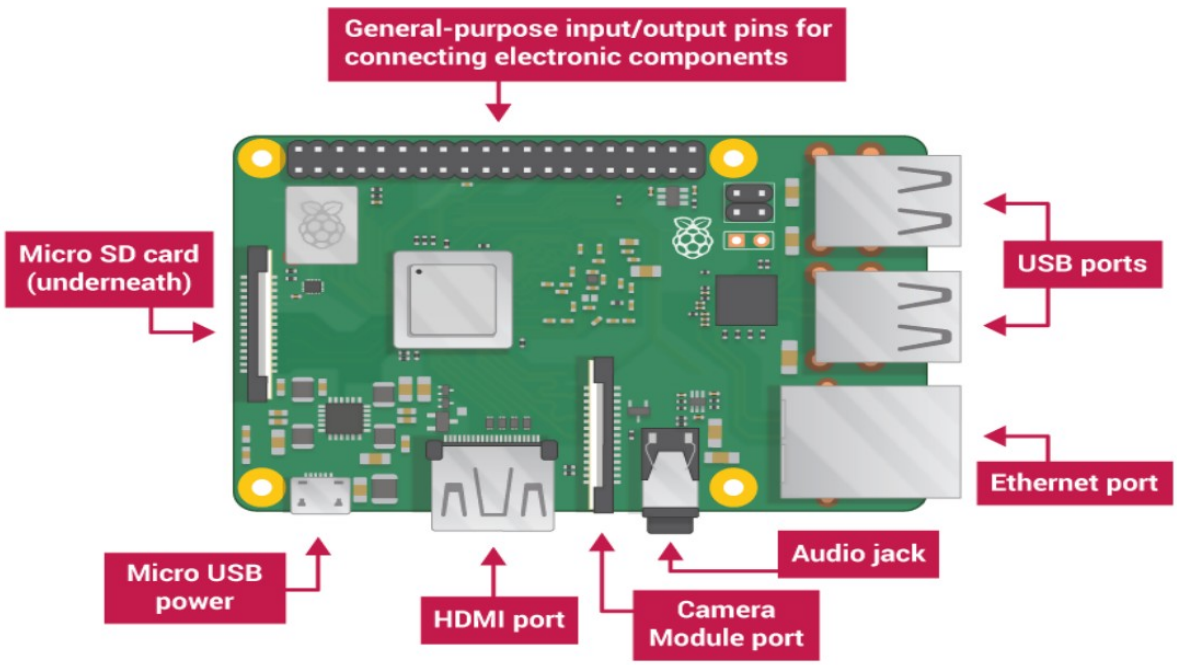
1.2 Hardware Components Description

1.2.1 Raspberry Pi

The Raspberry Pi is a credit card-sized single-board computer. There are currently five Raspberry Pi models in market i.e. the Model B+, the Model A+, the Model B, the Model A, and the Compute Module (currently only available as part of the Compute Module development kit). All models use the same SoC (System on Chip - combined CPU & GPU), the BCM2835, but other hardware features differ.

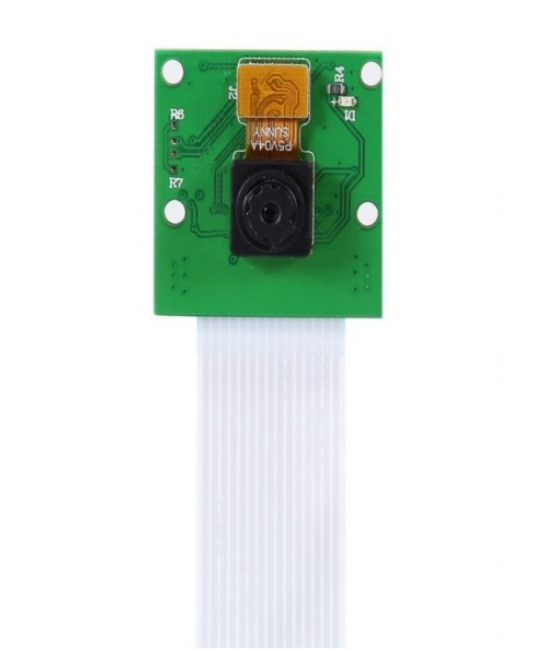
The A and B use the same PCB, whilst the B+ and A+ are a new design but of very similar form factor .The Compute Module is an entirely different form factor and cannot be used standalone.

In this project, we have used the model B Rev 2. It comprises of a 512 MB RAM model with two USB ports and a 10/100 Ethernet controller .



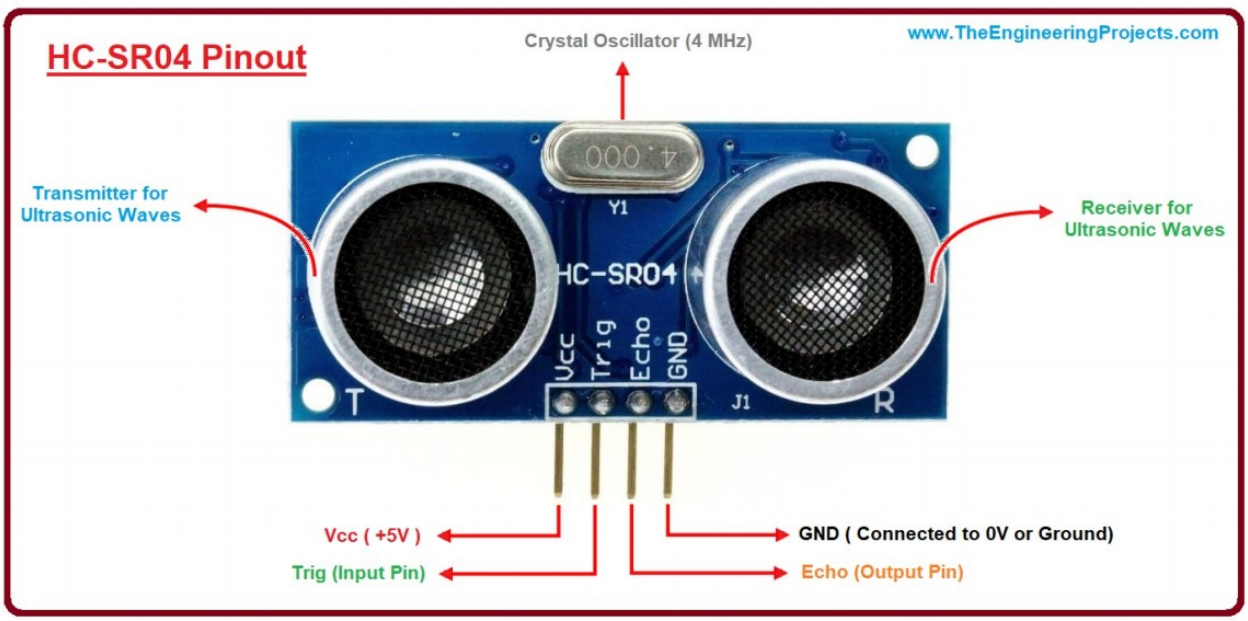
1.2.2 Pi Camera

It is the camera shipped along with Raspberry Pi . Pi camera module is also available to which can be used to take high-definition videos as well as still photographs .



1.2.3 Ultrasonic Sensors

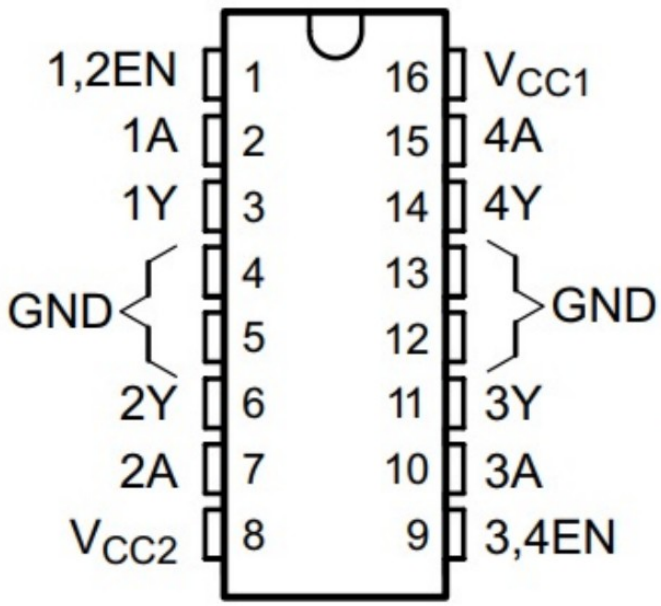
Ultrasonic sensors (also known as transceivers when they both send and receive, but more generally called transducers) evaluate attributes of a target by interpreting the echoes from radio or sound waves respectively . In this project, they are used to detect the distance of obstacles from the car .



1.2.4 L293D Motor Driver IC

We will used L293D Motor Driver IC for controlling a DC Motor with Raspberry Pi. It is a very common motor driver IC which is capable of driving two motors with individual currents up to 600mA.

The Pin diagram of the L293D Motor Driver IC, along with the pin description is shown in the following image.



PIN		DESCRIPTION
NAME	NO.	
1,2EN	1	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	Driver outputs
3,4EN	9	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	Device ground
V_{CC1}	16	5-V supply for internal logic translation
V_{CC2}	8	Power VCC for drivers 4.5 V to 36 V

1.3 Hardware Components Connection

The 4 wheels of the chassis are connected to 4 separate motors. The motor driver IC L293D is capable of driving 2 motors simultaneously . The rotation of the wheels is synchronized on the basis of the sides i.e. the left front and left back wheels rotate in sync and right front and right back wheel rotate in sync. Thus the pair of motors on each side is given the same digital input from L293D at any moment. This helps the car in forward, backward movements when both side wheels rotate in same direction with same speed. The car turns when the left side wheels rotate in opposite direction to those in right . The chassis has two shelves over the wheels separated by 2 inch approx. The IC is fixed on the lower shelf with the help of two 0.5 inch screws. It is permanently connected to the motor wires and necessary jumper wires are drawn from L293D to connect to Raspberry Pi . The rest of the space on the lower shelf is taken by 8 AA batteries which provide the power to run the motors. To control the motor connected to pin 3 (O1), pin 6 (O2), the pins used

are pin 1, pin 2 and pin 7 which are connected to the GPIOs of Raspberry pi via jumper wires.

Table I Truth Table to Control the Left Motor

Pin1	Pin2	Pin7	Function
High	High	Low	Anti-Clockwise
High	Low	High	Clockwise
High	High	High	Stop
High	Low	Low	Stop
Low	X	X	Stop

High +5V, Low 0V, X=either high or low (don't care)

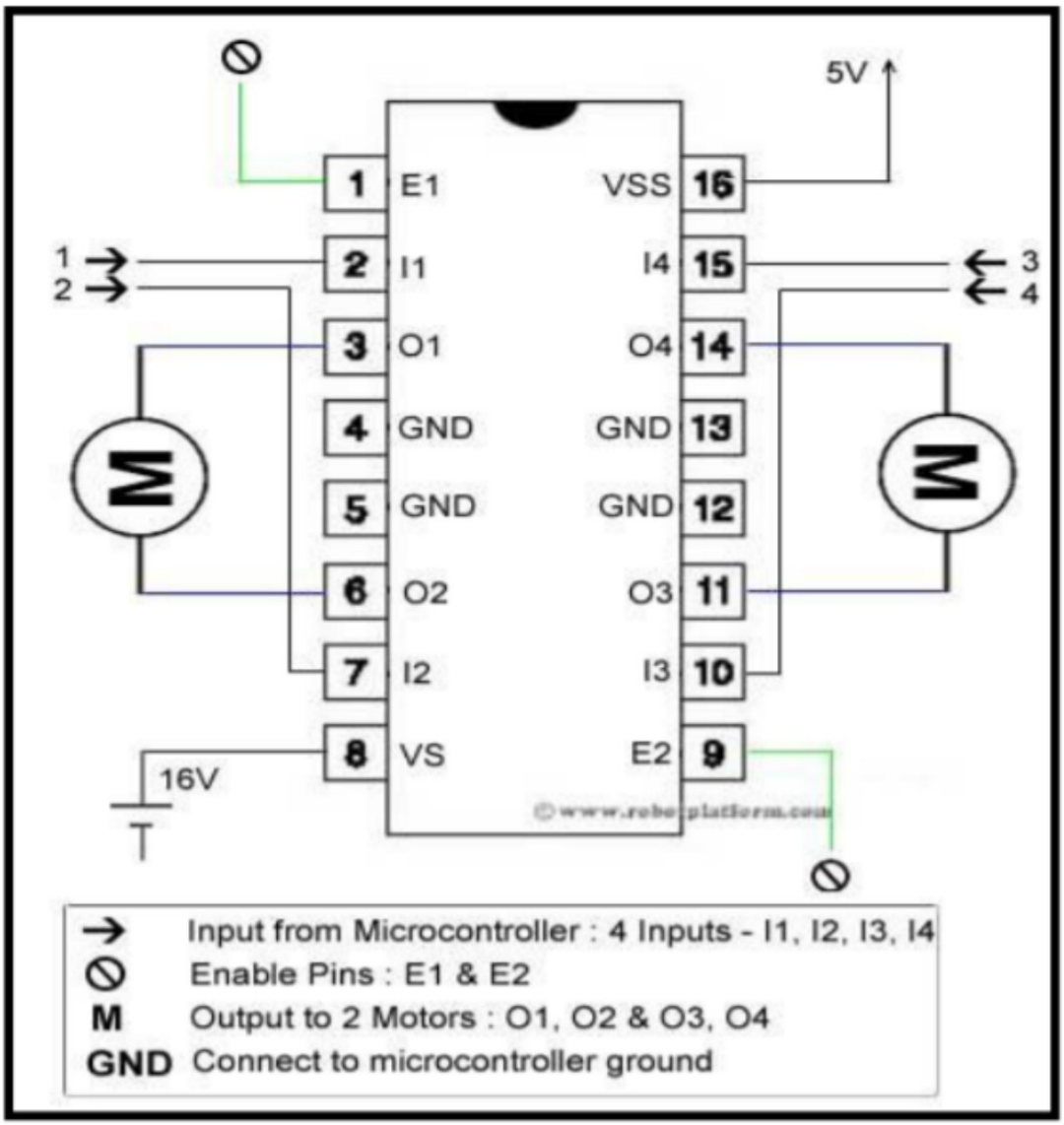


Fig 2: Hardware Connections

The raspberry pi case is glued on the top shelf along with the L shaped aluminum strip. The pi is fit in the case and the aluminum strip gives the support to the camera fit on servo motor and the ultrasonic sensor. The Wi-Fi dongle is attached to the USB port in Raspberry Pi in order to connect to it wirelessly. The complete connection of the raspberry pi with motor controller L293D can be found in fig 1. Since raspberry pi needed its own IP, it needs to be connected to a Wi-Fi router or Hotspot. For the same we need to make some changes in the field specified so as to make raspberry pi recognize the

router every time it boots up. Navigate to the file “/etc/network/interfaces” and add following lines to make the PI connect with your router after reboot.

iface wlan0 inetdhcp

wpa-ssid "Your Network SSID" wpa-psk "Your Password"

Software Tools And Libraries :

- C++ on the Arduino
- Python + Numpy + OpenCV on the Raspberry Pi
- Keras on the Raspberry Pi
- Homemade serial protocol for Arduino <-> Raspberry Pi
- communication
- Convolutional Neural Network
- Hough Transform
- VLC

Description Of Tools And Libraries:

Raspbian OS

- Of all the operating systems Arch, Risc OS, Plan 9 or Raspbian available for Raspberry Pi, Raspbian comes out on top as being the most user-friendly, best-looking, has the best range of default softwares and optimized for the Raspberry Pi hardware . Raspbian is a free operating system based on
- Debian (LINUX), which is available for free from the Raspberry Pi website.

Python

- Python is a widely used general-purpose, high-level programming language. Its syntax allows the programmers to express concepts in fewer lines of code when compared with other languages like C, C++or java .

RPi.GPIO Python Library

- The RPi.GPIO Python library allows you to easily configure and read-write the input/output pins on the Pi's GPIO header within a Python script . This package is not shipped along with Raspbian.

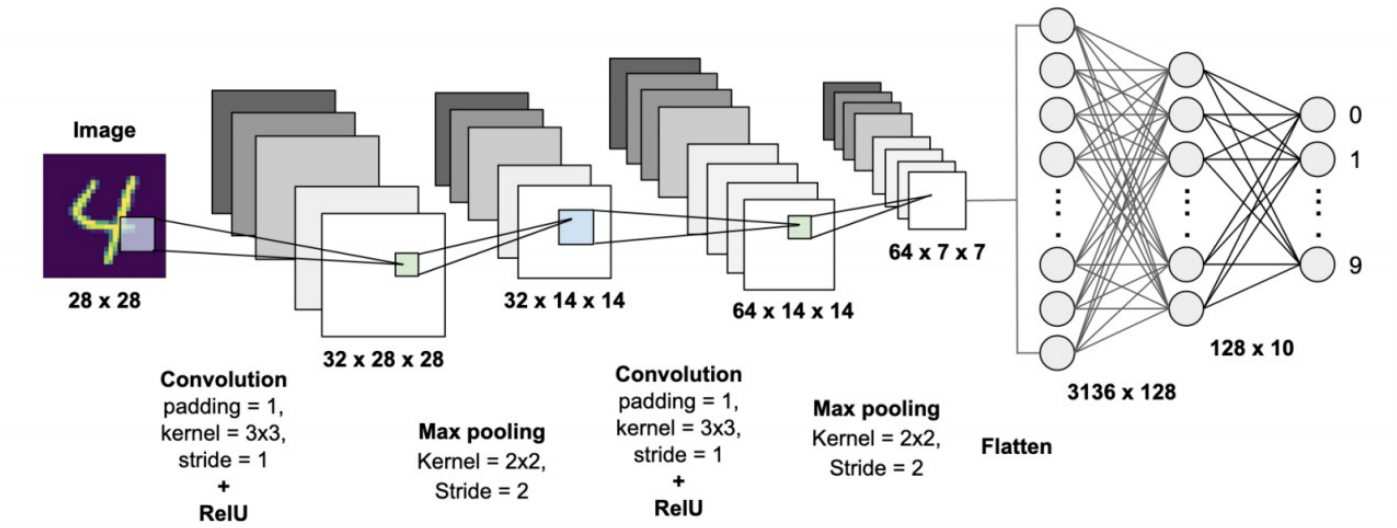
OpenCV

- It (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. It has over 2500 optimized algorithms, including both a set of classical algorithms and the state of the art algorithms in Computer Vision, which can be used for image processing, detection and face recognition, object identification, classification actions, traces, and other functions .
- This library allows these features be implemented on computers with relative ease, provide a simple computer vision infrastructure to prototype quickly sophisticated applications .
- The library is used extensively by companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, and startups area as Applied Minds, Video Surf and Zeitera. It is also used by many research groups and government . It is based on C++ but wrappers are available in python

as well. In our project is used to detect the roads and guide the car on unknown roads .

Convolutional Neural Network

A Convolutional Neural Network is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



Hough Transform

The Hough transform is a popular feature extraction technique that converts an image from Cartesian to polar Cordinates. Any point within the image space is represented by a sinusoidal curve in the Hough space. In addition, two points in a line segment generate two curves, which are overlaid at a location that corresponds with a line through the image space. Even though this model form is very easy, it is deeply complicated for the case of complex shapes due to noise and shape imperfection, as well as the problem of finding slopes of vertical lines. The CHT solved this problem by putting a transformation of the centroid of the shape in the x-y plane to the parameter space.

Used Algorithms:

.CNN Model For Image Calssification:

.Lane Detection Algorithm:

CNN Model For Image Classification

We applied CNN on Datasets which having the five classes like Green, Red,Yellow For traffic Lights , STOP sign and Obstacle, to classify them and we got the accuracy of the classifier is approximately 98%.

```
classifier.fit_generator(training_set,steps_per_epoch=260,epochs=5,validation_data=test_set,validation_steps=260)
```

```
C:\Users\pc\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype fro
m `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
WARNING:tensorflow:From C:\Users\pc\Anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_wit
h (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

```
C:\Users\pc\Anaconda3\lib\site-packages\ipykernel_launcher.py:18: UserWarning: Update your `Dense` call to the Keras 2 API: `De
nse(activation="relu", units=128)`
C:\Users\pc\Anaconda3\lib\site-packages\ipykernel_launcher.py:19: UserWarning: Update your `Dense` call to the Keras 2 API: `De
nse(activation="sigmoid", units=5)`
```

```
Found 266 images belonging to 5 classes.
Found 78 images belonging to 5 classes.
WARNING:tensorflow:From C:\Users\pc\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorfl
ow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
260/260 [==============================] - 580s 2s/step - loss: 0.1721 - acc: 0.9264 - val_loss: 0.1039 - val_acc: 0.9410
Epoch 2/5
260/260 [==============================] - 498s 2s/step - loss: 0.0710 - acc: 0.9696 - val_loss: 0.0715 - val_acc: 0.9642
Epoch 3/5
260/260 [==============================] - 517s 2s/step - loss: 0.0483 - acc: 0.9802 - val_loss: 0.0541 - val_acc: 0.9769
Epoch 4/5
260/260 [==============================] - 536s 2s/step - loss: 0.0241 - acc: 0.9923 - val_loss: 0.0406 - val_acc: 0.9796
Epoch 5/5
260/260 [==============================] - 522s 2s/step - loss: 0.0167 - acc: 0.9949 - val_loss: 0.0460 - val_acc: 0.9794
```



```
1]: <keras.callbacks.History at 0x1d14043ab38>
```

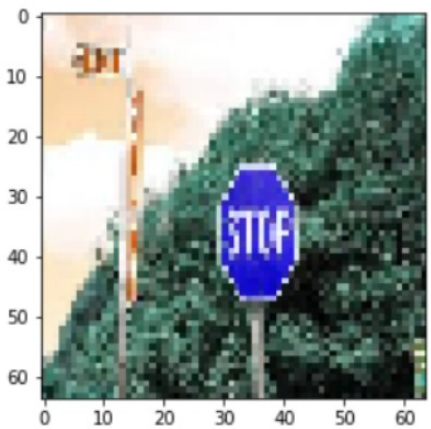
In [3]:
```
classifier.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30752)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               3936384
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
=================================================================
Total params: 3,937,925
Trainable params: 3,937,925
Non-trainable params: 0
```

Now for Traffic lights,Stop sign and Obstacles Prediction

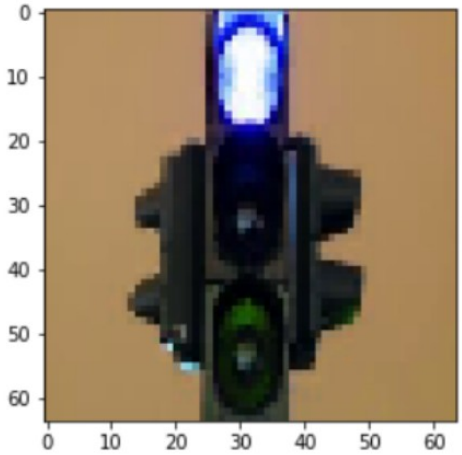
```
img=cv2.imread('C:/Users/pc/Downloads/datasets/test/Stop/s4.jpg')
img = cv2.resize(img,(64,64))

predict(img)
```

```
STOP SIGN
```



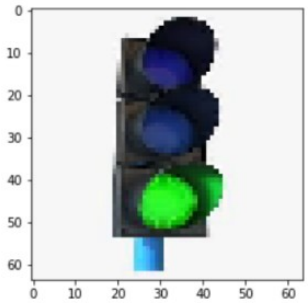
```
img=cv2.imread('C:/Users/pc/Downloads/datasets/test/Red/r1.jpg')
img = cv2.resize(img,(64,64))

predict(img)
```

```
Red Light
```



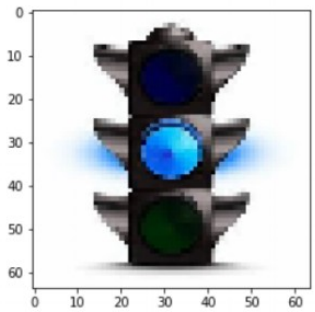
```
img=cv2.imread('C:/Users/pc/Downloads/datasets/test/Green/g1.jpg')
img = cv2.resize(img,(64,64))

predict(img)
```

```
GREEN LIGHT
```



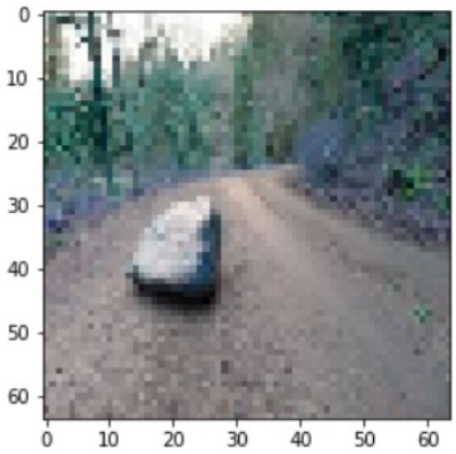
```
img=cv2.imread('C:/Users/pc/Downloads/datasets/test/Yellow/y1.jpg')
img = cv2.resize(img,(64,64))

predict(img)
```

```
YELLOW LIGHT
```



```
img=cv2.imread('C:/Users/pc/Downloads/datasets/test/Obstacle/b6.jpg')
img = cv2.resize(img,(64,64))

predict(img)
```

```
Obstacle Found
```



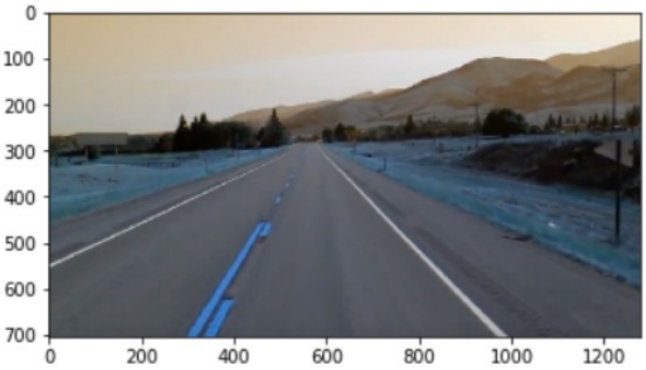
Lane Detection Algorithm:

Step 1: Reading an Image

Input Image

```
In [3]: import math
        import cv2
        import numpy as np
        import matplotlib.pyplot as plt
        img=cv2.imread('C:/Users/pc/Downloads/datasets/test_image2.jpg')
        print(" INPUT IMAGE")
        plt.imshow(img)
```

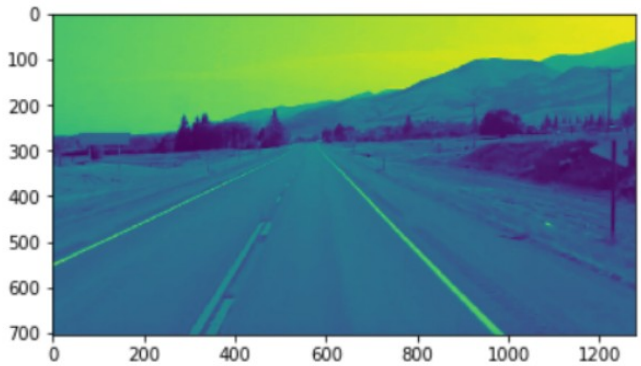
```
 INPUT IMAGE
Out[3]: <matplotlib.image.AxesImage at 0x18be571f518>
```



Step 2: Color image to Grayscale Conversion: This will help us with the identification of edges an

```
In [5]: def lane_detection(img):
            ## Color image to GrayScale image conversion
            gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
            plt.imshow(gray_img)

        print(" Grayscale Image")
        lane_detection(img)
```

```
 Grayscale Image
```



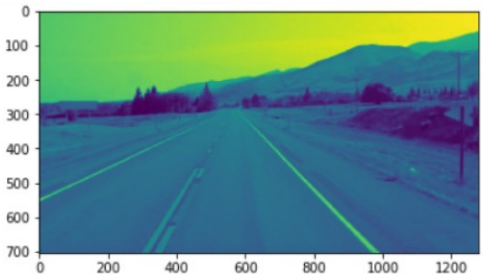
d corners.

Step 3: Gaussian Blur

Adding Gaussian noise to an image, it very useful as it smooths the interpolation between the pixels and is a way to super-pass noise and spurious gradients.

In [7]:
```
def lane_detection(img):
    ## Color image to GrayScale image conversion

    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)


    ## Gaussian Blur
    # Higher the kernel, the more blur the outcome image will be.
    kernel_size = 5
    gauss_img = cv2.GaussianBlur(gray_img,(kernel_size, kernel_size), 0)
    plt.imshow(gauss_img)

print(" After Gaussina Blur ")
lane_detection(img)
```

```
After Gaussina Blur
```



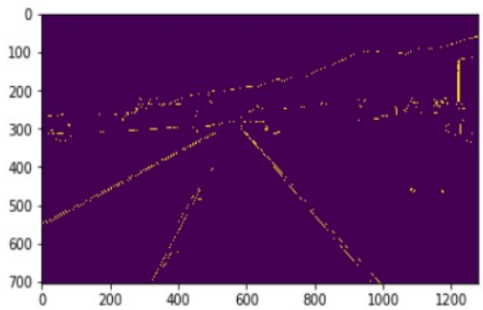
In [ ]:

Step 4: Canny Edge Detection

Canny Edge Detection offers a way to detect the boundaries of an image. This is done through the gradients of the image.

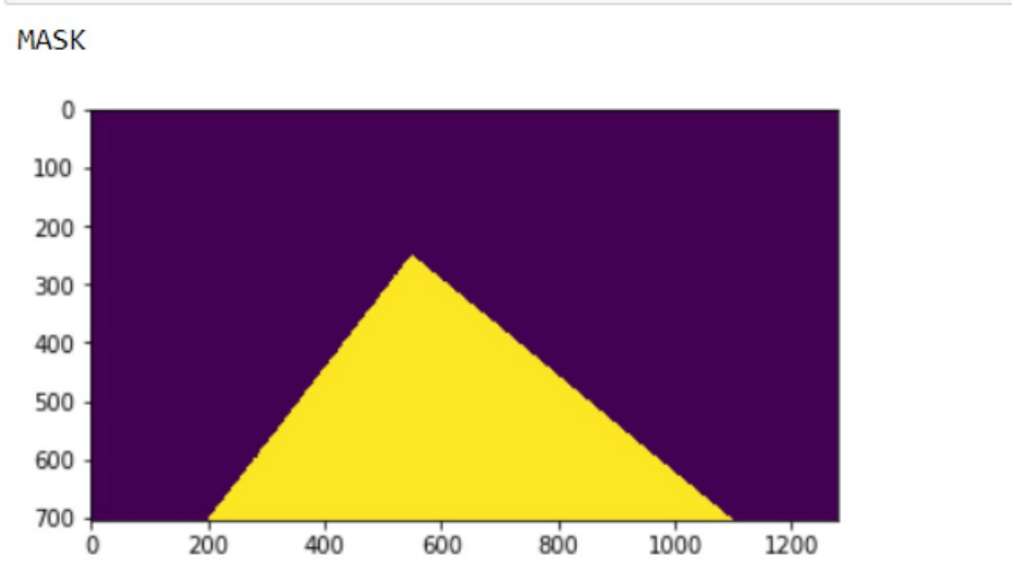
```
    ## Canny Edge Detection : It is used to detect boundaries of an image, through the gredients of the image
    low_threshold, high_threshold = [50, 150]
    canny_img = cv2.Canny(gauss_img, low_threshold, high_threshold)
    plt.imshow(canny_img)



    #plt.imshow(cropped_img)
print("Image After Canny Edge Detection")
lane_detection(img)
```

```
Image After Canny Edge Detection
```

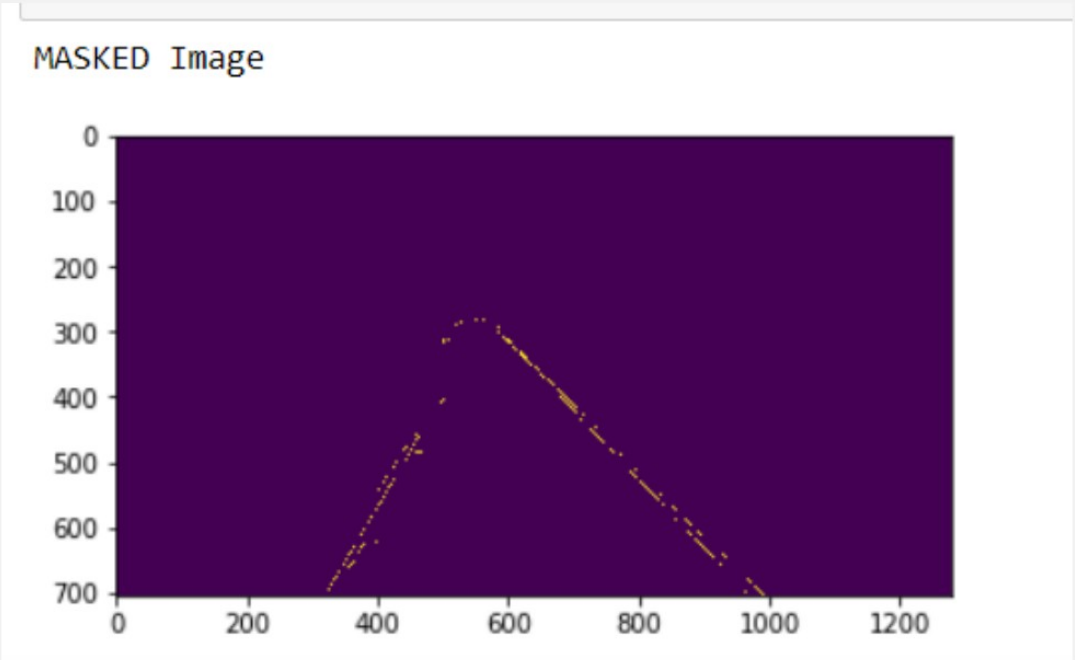


Step 5: Mask a region of interest

In the above Image, there are some outliers; some edges from the other part of the road, As our camera will be fixed, we can put a mask upon the image and keep only these lines that are interesting for our task. Thus, it will be very natural to draw a *trapezium* in order to keep onlyon where we should expect the road lines to be.



Created Mask



After Taking Bitwise AND with input image And Mask

Step 6: Hough lines Detector

The above image represents only the dots of the edges. What is left is to connect the edges. In this case, we are looking for lines and we are going to do this by transporting the images to the parameter space, called Hough Space. We will now deal with polar coordinates (rho and theta), in which we will search for intersecting

lines.

Step 7: Find the Road Lines

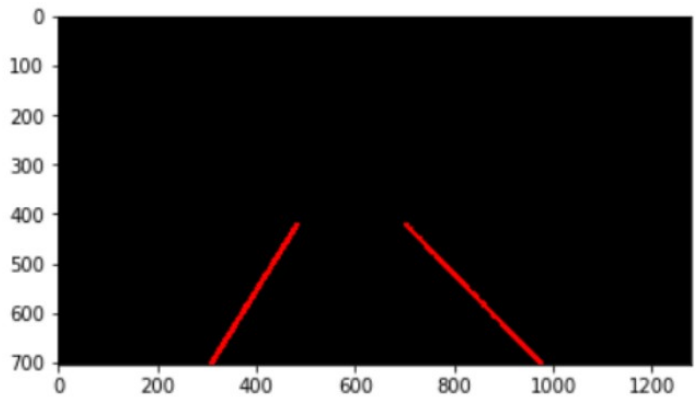
Our strategy will be the following:

• Break the image in half with reference to the x-axis

• Fit a linear regression model to the points, in order to find a smooth line.

```
    line_image=display_lines(img,averaged_lines)
    plt.imshow(line_image)

    combo_image=cv2.addWeighted(img,0.8,line_image,1,1)


print("Line Image")
lane_detection(img)
```

```
Line Image
```

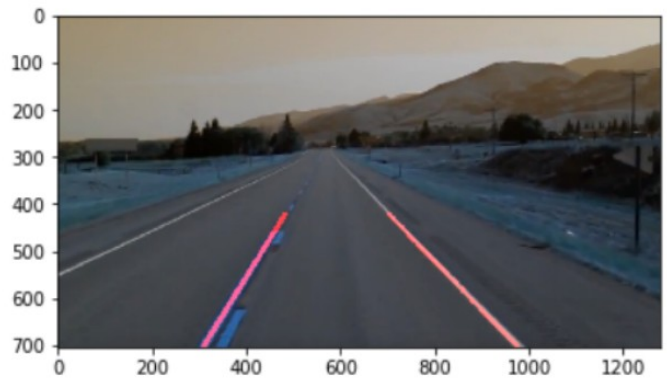


Step 8: Concatenate the lines with the original image

By giving weight to the two images, we can just add them. The black areas of the img_line have value 0 , thus that addition will not alter the output one.

```
    combo_image=cv2.addWeighted(img,0.8,line_image,1,1)
    plt.imshow(combo_image)

    #plt.imshow(cropped_img)
print("Concatenated Image")
lane_detection(img)
```

```
Concatenated Image
```



Result And Conclusion:

We have theoretically built a Robot Navigation System. We have theoretically implemented the CNN image classification algorithm for traffic lights ,stop sign and Obstacles detection and Lane detection algorithm on the assembled different hardware components.The Robot is able to take the various decisions like move forward, turn left,turn right,stop and reverse back, according to different situations like 1.when green light detected then move forward, 2. Red Light detected then STOP 3.When Any obstacles detected the reverse back and the left turn and then right turn or reverse back and then right turn and then left turn. 4.If Lane is not detected then left turn or right turn.

Future scope & advancements:

The work could be enhanced by improving the algorithm by adding machine learning to it. The present algorithm performs the operations on all the frames. It is accurate but its efficiency could be further enhanced if it starts learning by itself and avoid unnecessary calculations of the regions which are already known or familiar.

We can work on the path planning of Robot by which Robot follow the shortest distance path from source to destination. For this we will use the Shortest Path Dijkstras Algorithm.At Every Junction Robot will able to decide which path is the shortest for reaching the destination.

We can also work on the Distance measurement.When Robot will move from one place to Another Place then it will able to measure the distance travelled by the Robot.

As we Know that the importance of the Artificial Intelligence in the future will become more and more .Due to this , many systems will work autonomously .Robot Navigation system is similar to Self Driving Car .

References:

[1]. Richard Szeliski, Computer Vision: Algorithms and Applications, Springer, 09 2010.

[2]. Shangbing Gao and Yan Zhang, "The automatic detection and recognition of the Traffic Sign" , *International Conference on Virtual Reality and Visualization*, 09 2016.

[3]. P. Shopa, N. Sumitha and P.S.K Patra, "Traffic Sign Detection and Recognition Using OpenCV", *International Conference on Information Communication and Embedded*, 02 2014.

[4]. [online] Available: http://www.slideshare.net/hanneshapke/introduction-to-convolutional-neural-networks.

[5]. S. Liu et al., Creating Autonomous Vehicle Systems, Morgan Claypool Publishers, 2017.

Code Implementation

#1.This is the CNN Model For Classification Of Traffic Light Signals ,STOP Sign And Obstacles.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
classifier=Sequential()
classifier.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))
classifier.add(Flatten())
classifier.add(Dense(output_dim=128,activation='relu'))
classifier.add(Dense(output_dim=5,activation='sigmoid'))
classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
training_set=train_datagen.flow_from_directory('C:/Users/pc/Downloads/datasets/training',target_size=(64,64),batch_size=32,class_mode='categorical')
test_set=test_datagen.flow_from_directory('C:/Users/pc/Downloads/datasets/test',target_size=(64,64),batch_size=32,class_mode='categorical')
classifier.fit_generator(training_set,steps_per_epoch=260,epochs=10,validation_data=test_set,validation_steps=260)
```

#2. Green, Yellow, Red, STOP signs and Obstacles prediction function from Above Built CNN model.

```
def predict(test_image):
test_image=image.img_to_array(test_image)
test_image=np.expand_dims(test_image,axis=0)
   result=classifier.predict(test_image)
training_set.class_indices
   if result[0][0]==1: # command for Green light
      command="GO"
```

```
elif result[0][1]==1: # command for Yellow light
        command="WAIT"
elif result[0][2]==1: # command for Red Light
        command="STOP"
elif result[0][3]==1:                # command for STOP sign
        command="STOP"
else:
     command="Obstacle"
return command
```

#3. Code Implementation for Lane detection through Hough Transform

```
import math
import cv2
import numpy as np
import matplotlib.pyplot as plt

def display_lines(image,lines):
line_image=np.zeros_like(image)
    if lines is not None:
        for line in lines:
            x1,y1,x2,y2=line.reshape(4)
            cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)
    return line_image

def make_coordinates(image,line_parameters):
slope,intercept=line_parameteres
    y1=image.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(image,lines):
left_fit=[]
right_fit=[]
    for line in lines:
        x1,y1,x2,y2=line.reshape(4)
        parameters=np.polyfit((x1,x2),(y1,y2),1)
        slope=parameters[0]
        intercept=parameters[1]
        if slope<0:
left_fit.append((slope,intercept))
        else:
right_fit.append((slope,intercept))
left_fit_avg=np.average(left_fit,axis=0)
right_fit_avg=np.average(right_fit,axis=0)
left_line=make_coordinates(image,left_fit_avg)
right_line=make_coordinates(image,right_fit_avg)
    return np.array([left_line,right_line])

def region_of_interest(canny_img):
    height=canny_img.shape[0]
    polygons=np.array([
        [(0,height),(canny_img.shape[1],height),(canny_img.shape[1]//2,250)]
    ])
    mask=np.zeroes_like(canny_img)
    cv2.fillPoly(mask,polygons,255)

    ## Bitwise & b/w mask and canny image,to show only region of interest traced
     # by the polygon contour of the mask
masked_img=cv2.bitwise_and(canny_img,mask)
    return masked_img
```

```
def lane_detection(img):
    # Color image to GrayScale image conversion
gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)


    ## Gaussian Blur
    # Higher the kernel, the more blur the outcome image will be.
kernel_size = 5
gauss_img = cv2.GaussianBlur(gray_img,(kernel_size, kernel_size), 0)


    ## Canny Edge Detection : It is used to detect boundaries of an image, through the gredientsof the image
low_threshold, high_threshold = [200, 300]
canny_img = cv2.Canny(gauss_img, low_threshold, high_threshold)
cropped_img=region_of_interest(canny_img)
    ## Hough Transform: to connect the dots of images by transporting the images to the Parameter Space
     # We have taken polar coordinates(rho,theta),in which we searched for intersecting lines.
    lines = cv2.HoughLinesP(cropped_img, rho=1, theta=math.pi/180,
                    threshold=15, np.array([]),
minLineLength=30,
maxLineGap=40)


averaged_lines=average_slope_intercept(img,lines)


line_image=display_lines(img,averaged_lines)
combo_image=cv2.addWeighted(img,0.8,line_image,1,1)
    return combo_image
```

#4. Code To Control Motors : Robot follows the command Forward , Left_Turn , Right_Turn and STOP.

```
import RPi.GPIO as gpio
import time
```

```
def init():
    gpio.setmode(gpio.BOARD)
    gpio,setup(7,gpio.OUT)
    gpio,setup(11,gpio.OUT)
    gpio,setup(13,gpio.OUT)
    gpio,setup(15,gpio.OUT)
def stop(tf):
    init()
    gpio.output(7,False)
    gpio.output(11,False)
    gpio.output(13,False)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()

def  forward(tf):   # Move Forward
    init()
    gpio.output(7,False)
    gpio.output(11,True)
    gpio.output(13,True)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()
def  reverse(tf):   # Move Reverse
  def  reverse(tf):   # Reverse back
    init()
    gpio.output(7,True)
    gpio.output(11,False)
    gpio.output(13,False)
    gpio.output(15,True)
    time.sleep(tf)
    gpio.cleanup()
```

```
def  turn_left(tf):   # Turn Left
    init()
    gpio.output(7,True)
    gpio.output(11,True)
    gpio.output(13,True)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()
def turn_right(tf):  # Turn Right
    init()
    gpio.output(7,False)
    gpio.output(11,False)
    gpio.output(13,False)
    gpio.output(15,True)
    time.sleep(tf)
    gpio.cleanup()
def  pivot_left(tf):   # Pivot  Left
    init()
    gpio.output(7,True)
    gpio.output(11,False)
    gpio.output(13,True)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()
def  Pivot_right(tf):  #  Pivot Right
    init()
    gpio.output(7,False)
    gpio.output(11,True)
    gpio.output(13,False)
    gpio.output(15,True)
    time.sleep(tf)
    gpio.cleanup()
```

#5. Driver Function Of Lane detection ,Traffic Signals Detection And Give the Command to System

```
## Driver Function


if __name__=="__main__":
    command="RUN"
    cap=cv2.VideoCapture(0)
    left=0
    right=0
    while True:
ret,frame=cap.read()
lane_detected_img=lane_detection(frame)
        if lane_detected_img is not None:
           left=0
           right=0
           if predict(frame)=="GO": ## Green sign detected
              forward(100)  # move forward Function Call
           elif predict(frame)=="WAIT":
              command="WAIT"
           elif  predict(frame)== "STOP"          # RED Sign or STOP sign detected
              stop(1)
          else:       # if any Obstacles found
                reverse(1)
                pivot _left(1)
                forward(1)
               pivot_right(1)
```

```
        else:
            # Now turn left and check whether lane is present or not
            if left==0:
                turn_left(1)    # 1 is the time frame  for left turn  function call
                left+=1
                continue
            else:
                if right==0:
                    turn_right(1)  # Right Turn
                    right+=1
                else:
                    stop(1)       #stop function Call
        if cv2.waitKey(1)==13:
            break

    #relaese camera and close windows
cap.release()
    cv2.destroyAllWindows()
```