Final Project Report
Capstone Design (SOC4150-002)

IUT Capstone Design Comprehensive Assignment

Submitted by:
Team: tripLM 16-3
U1610146 Mirzashomol Karshiev
U1610137 Mardon Zarifjonov
U1610125 Lazizbek Qahhorov
U1610143 Mirpulatjon Shukurov

Inha University in Tashkent
2020

Abstract:

Nowadays, the term self-driving car is getting popular, proof that all big companies are trying to create their car that will be totally on electric energy. For that reason, most science-related universities have a diploma work to build a self-driving car prototype. The purpose of this report is to understand how we can detect the lines and predict the turns based on detected lines.

For detecting lines on the road from the video that is captured from the camera, the python programming language was chosen for its simplicity and easiness of syntax. OpenCV and Tensorflow libraries were used for image processing and training the lines.

Results showed that using these 2 technologies mentioned above you can achieve accurate calculations. With the trained linear regression model we obtained correct lines that overlap with real lines in the video. It predicted the turn as well with the help of lines but it is not fully reliable.

To make the model perfectly correct deep learning must be used.

Acknowledgements:

The goal of our project was to detect the lines on the road from the video and predict turn based on these lines. These tasks were completed due to the help given by the professor and teacher assistant. We would like to thank them for their lectures that played a key role in finishing the project successfully. Thank you for our team members for their dedication and hard work.

Table of contents:

1. Introduction:

The purpose of this report is to show the easiness of using TensorFlow and OpenCV libraries for detecting lines or other shapes. We have been given the task to detect the lines on the road from the video. For that reason, we chose those libraries because they contain a wide spectrum of algorithms to deal with images and training models. We had to perform several operations on the image for detecting lines such as converting it into grayscale, denoising, then converting to binary format after which using filtering to detect edges of the region you are interested in. And applying the HoughLinesP algorithm to find lines from edges and giving these lines to the machine learning model to correct the lines.

2. Algorithm:

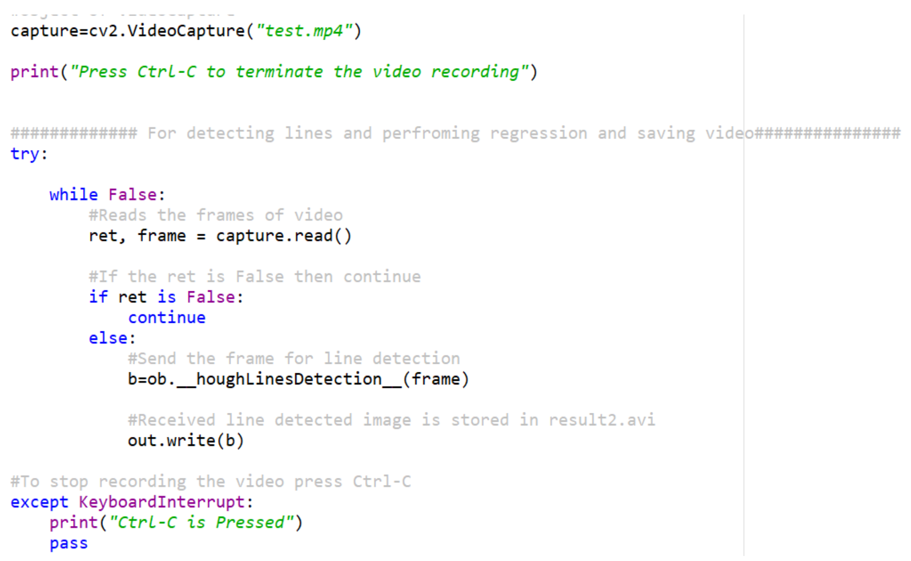
The algorithm consist of 2 steps:

- 2.1 Detecting lines, performing regression of lines, and saving video of it.
- 2.2 Detecting lines and predicting turn from recorded video.

2.1 Detecting lines and performing regression of lines:

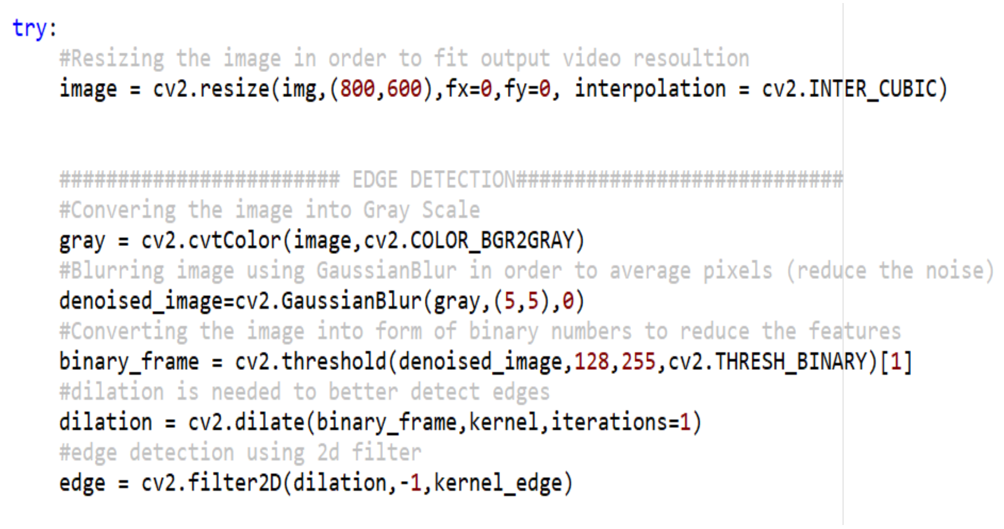
First of all we have to mention we are working on video for line detection. As it is known the video consist of frames, frames are simple images. For detecting the lines we performed operations on each image.

1. Read the video from folder by cv2.VideoCapture(). Then with help of capture.read() it will read each frame of the video. If there occurred some error during the reading of frames it will skip. Received frames are sent as argument to method for line detection.



```
capture=cv2.VideoCapture("test.mp4")

print("Press Ctrl-C to terminate the video recording")


############# For detecting lines and perfroming regression and saving video##############
try:

    while False:
        #Reads the frames of video
        ret, frame = capture.read()

        #If the ret is False then continue
        if ret is False:
            continue
        else:
            #Send the frame for line detection
            b=ob.__houghLinesDetection__(frame)

            #Received line detected image is stored in result2.avi
            out.write(b)

#To stop recording the video press Ctrl-C
except KeyboardInterrupt:
    print("Ctrl-C is Pressed")
    pass
```

2. Detection of edges in the image. The image is resized into 800x600 resolution due to requirement and it converted into grayscale the denoised using GaussianBlur algorithm which averages the outlier pixels. After which it converted into binary format then dilated for better detecting edges. Finally, using filter2D we obtain edges.



```
try:
    #Resizing the image in order to fit output video resoultion
    image = cv2.resize(img,(800,600),fx=0,fy=0, interpolation = cv2.INTER_CUBIC)


    ######################## EDGE DETECTION###########################
    #Convering the image into Gray Scale
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    #Blurring image using GaussianBlur in order to average pixels (reduce the noise)
    denoised_image=cv2.GaussianBlur(gray,(5,5),0)
    #Converting the image into form of binary numbers to reduce the features
    binary_frame = cv2.threshold(denoised_image,128,255,cv2.THRESH_BINARY)[1]
    #dilation is needed to better detect edges
    dilation = cv2.dilate(binary_frame,kernel,iterations=1)
    #edge detection using 2d filter
    edge = cv2.filter2D(dilation,-1,kernel_edge)
```

3. Creating our region of interest for detecting the lines inside this ROI. First we create dark image of size as our image 800x600. Then we create polygon using 3 points and combine this polygon with dark

image. After which we mask the dark image with real image and we get our ROI.

```
###################### REGION OF INTEREST #########################
#takes the height of numpy array image
height=edge.shape[0]
#takes the width of numpy array image
width=edge.shape[1]

#Create the dark image of size as original image
dark_image1=np.zeros([height, width, 3], dtype=np.uint8)
#Creating polyong using 3 points for region of interest
polygon1=np.array([[(140,600),(750,600),(420,400)]])
#Masking the dark image wiht polygon
cv2.fillPoly(dark_image1,polygon1,(255,255,255))
#Bitwise the image and mask to get region of interest
masked_image1=cv2.bitwise_and(image,dark_image1)
```

4. Detecting the edge of ROI. We convert roi into grayscale then use denoising method after which convert into binary format. After all use filter2D for edge detection.

```
############# EDGE DETECTION OF REGION OF INTEREST ################
#Converting ROI (region of interest) into gray scale
roi_gray=cv2.cvtColor(masked_image1,cv2.COLOR_BGR2GRAY)
#Denoising the ROI
roi_blur=cv2.GaussianBlur(roi_gray,(5,5),0)
#Converting denoised ROI into binary format
roi_binary= cv2.threshold(roi_blur,128,255,cv2.THRESH_BINARY)[1]
#Detectin the edges from binary ROI
roi_edge=cv2.filter2D(roi_binary,-1,kernel_edge)
```

5. Detecting the lines from ROI.

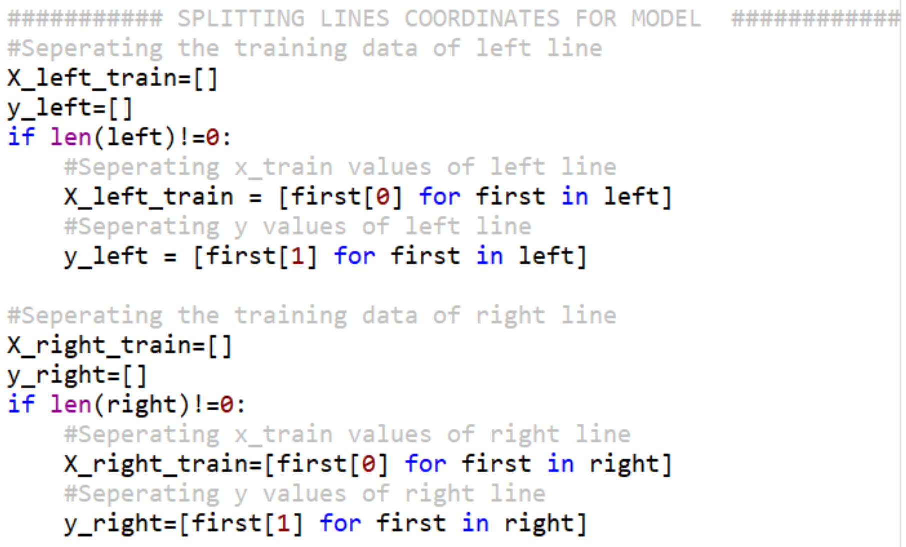
```
############# LINE DETECTION OF REGION OF INTEREST ################
#Detectin the lines from edges that are obtained from filter2D
lines=cv2.HoughLinesP(roi_edge,1,np.pi/180,45,np.array([]),minLineLength=2,maxLineGap=150)
```

6. Separating the lines into left and right lines.

```
############# SEPERATING LINES INTO LEFT AND RIGHT ################
#Checks whether frame has lines
if lines is not None:
    #Left line coordinates list
    left=[]
    #Right line coordintes list
    right=[]
    #Gettin the the center of image or half of width
    img_center = roi_edge.shape[1]//2
    #If lines is not empty
    if len(lines)!=0:
        #Seperating Lines into left and right
        for line in lines:
            #Coordinates of line
            for x2, y2, x1, y1 in line:
                #If coordinates are righter from center of image
                if x1>img_center and x2>img_center:
                    #Store the coordinates into right line
                    right.append(np.float32([x1,y1]))
                    right.append(np.float32([x2,y2]))

                #If coordinates are lefter from center of image
                elif x1<img_center and x2<img_center:
                    #Store the coordinates into left line
                    left.append(np.float32([x1,y1]))
                    left.append(np.float32([x2,y2]))
                else:
                    continue
```

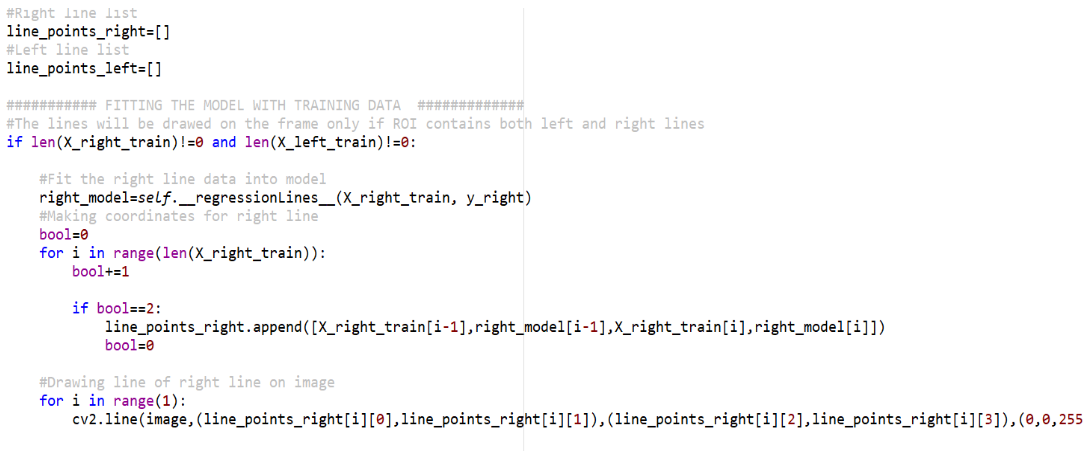
The lines are separated with help of center of image if the coordinates are righter that means right line if less than center of image it belongs to left line.

7. Preparing training data from line coordinates. For training the regression lines we have to split the line into x and y values.



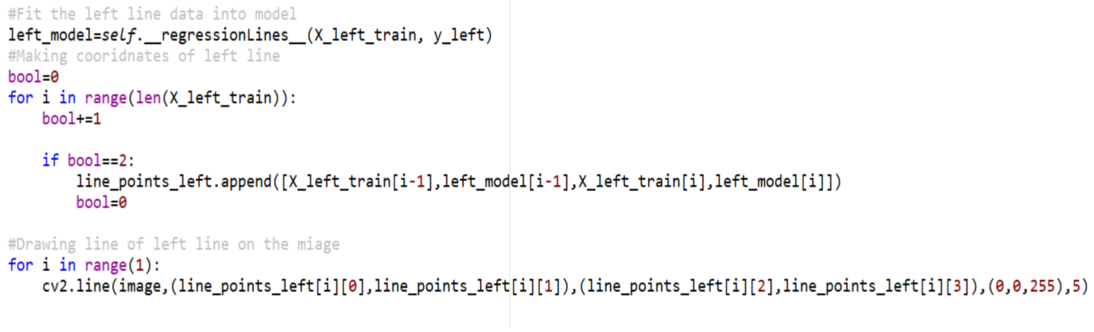
```
########### SPLITTING LINES COORDINATES FOR MODEL   #############
#Seperating the training data of left line
X_left_train=[]
y_left=[]
if len(left)!=0:
    #Seperating x_train values of left line
    X_left_train = [first[0] for first in left]
    #Seperating y values of left line
    y_left = [first[1] for first in left]

#Seperating the training data of right line
X_right_train=[]
y_right=[]
if len(right)!=0:
    #Seperating x_train values of right line
    X_right_train=[first[0] for first in right]
    #Seperating y values of right line
    y_right=[first[1] for first in right]
```

8. Training the right line based on regression. After returning predicted points of regression model we combine them back to draw the right line on the image.



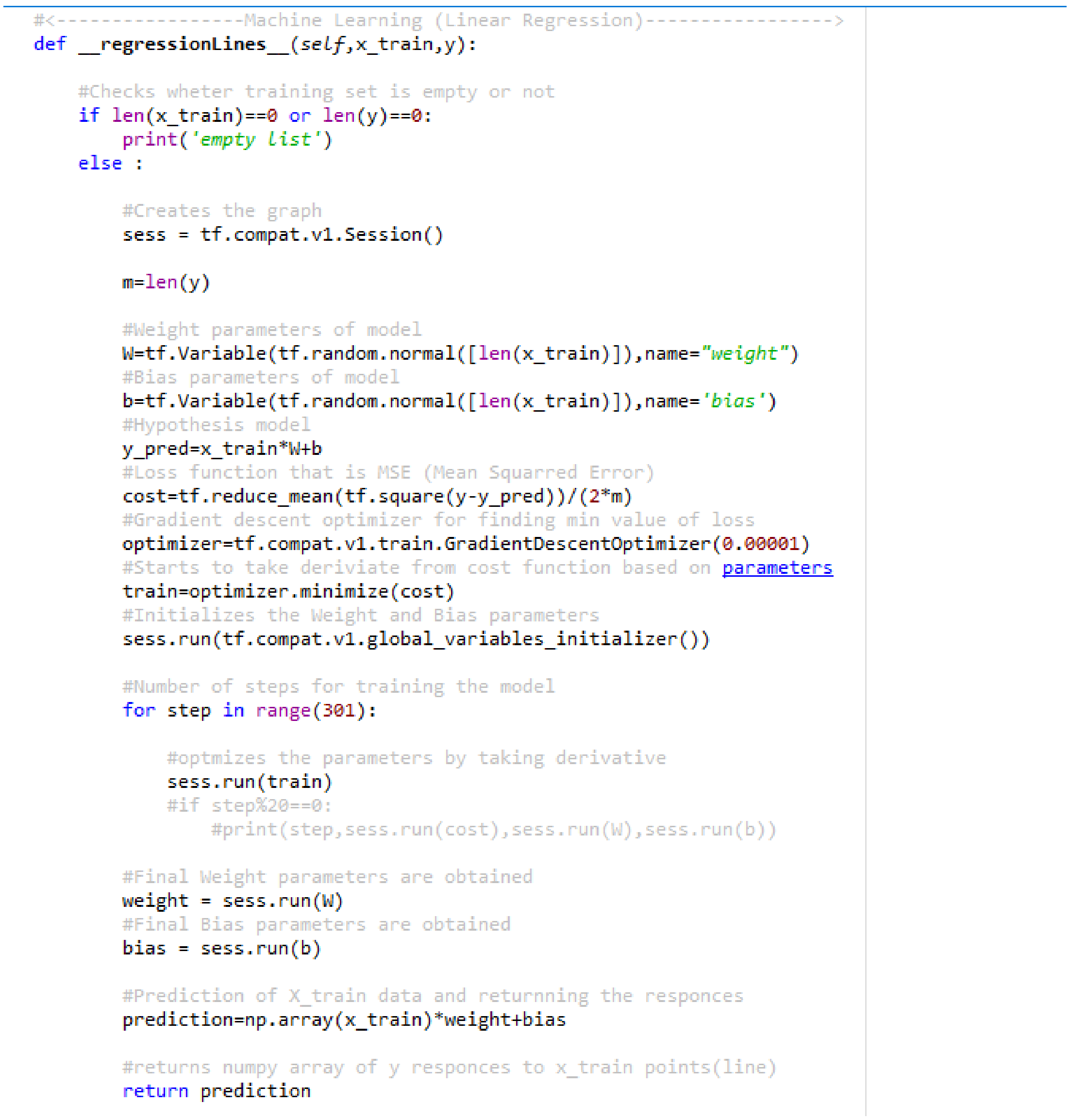
```
#Right line list
line_points_right=[]
#Left line list
line_points_left=[]

########### FITTING THE MODEL WITH TRAINING DATA  #############
#The lines will be drawed on the frame only if ROI contains both left and right lines
if len(X_right_train)!=0 and len(X_left_train)!=0:

    #Fit the right line data into model
    right_model=self.__regressionLines__(X_right_train, y_right)
    #Making coordinates for right line
    bool=0
    for i in range(len(X_right_train)):
        bool+=1

        if bool==2:
            line_points_right.append([X_right_train[i-1],right_model[i-1],X_right_train[i],right_model[i]])
            bool=0

    #Drawing line of right line on image
    for i in range(1):
        cv2.line(image,(line_points_right[i][0],line_points_right[i][1]),(line_points_right[i][2],line_points_right[i][3]),(0,0,255
```

9. Training the left line based on regression and drawing the predicted lines on image.



```
#Fit the left line data into model
left_model=self.__regressionLines__(X_left_train, y_left)
#Making cooridnates of left line
bool=0
for i in range(len(X_left_train)):
    bool+=1

    if bool==2:
        line_points_left.append([X_left_train[i-1],left_model[i-1],X_left_train[i],left_model[i]])
        bool=0

#Drawing line of left line on the miage
for i in range(1):
    cv2.line(image,(line_points_left[i][0],line_points_left[i][1]),(line_points_left[i][2],line_points_left[i][3]),(0,0,255),5)
```

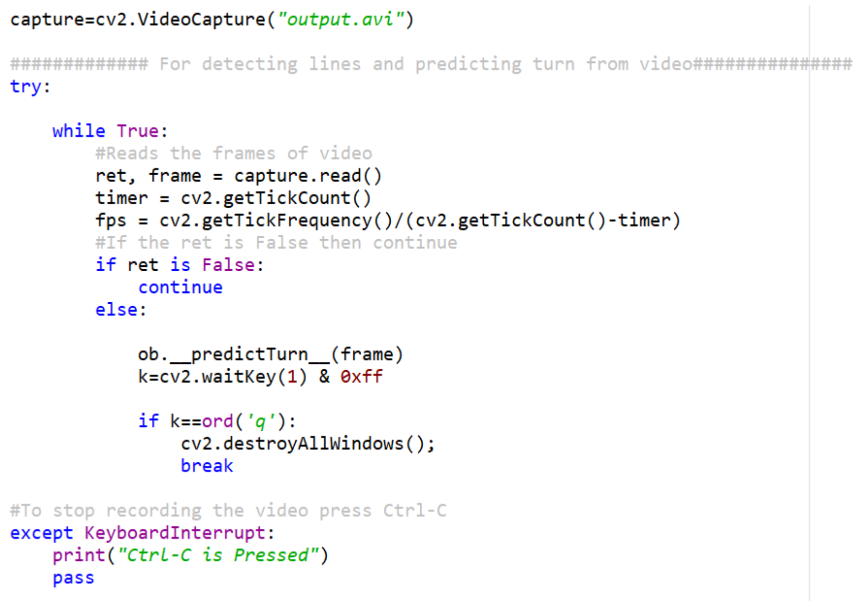
10. Machine learning model for training lines.



```python
#<-----------------Machine Learning (Linear Regression)----------------->
def __regressionLines__(self,x_train,y):

    #Checks wheter training set is empty or not
    if len(x_train)==0 or len(y)==0:
        print('empty list')
    else :

        #Creates the graph
        sess = tf.compat.v1.Session()

        m=len(y)

        #Weight parameters of model
        W=tf.Variable(tf.random.normal([len(x_train)]),name="weight")
        #Bias parameters of model
        b=tf.Variable(tf.random.normal([len(x_train)]),name='bias')
        #Hypothesis model
        y_pred=x_train*W+b
        #Loss function that is MSE (Mean Squarred Error)
        cost=tf.reduce_mean(tf.square(y-y_pred))/(2*m)
        #Gradient descent optimizer for finding min value of loss
        optimizer=tf.compat.v1.train.GradientDescentOptimizer(0.00001)
        #Starts to take deriviate from cost function based on parameters
        train=optimizer.minimize(cost)
        #Initializes the Weight and Bias parameters
        sess.run(tf.compat.v1.global_variables_initializer())

        #Number of steps for training the model
        for step in range(301):

            #optmizes the parameters by taking derivative
            sess.run(train)
            #if step%20==0:
                #print(step,sess.run(cost),sess.run(W),sess.run(b))

        #Final Weight parameters are obtained
        weight = sess.run(W)
        #Final Bias parameters are obtained
        bias = sess.run(b)

        #Prediction of X_train data and returnning the responces
        prediction=np.array(x_train)*weight+bias

        #returns numpy array of y responces to x_train points(line)
        return prediction
```

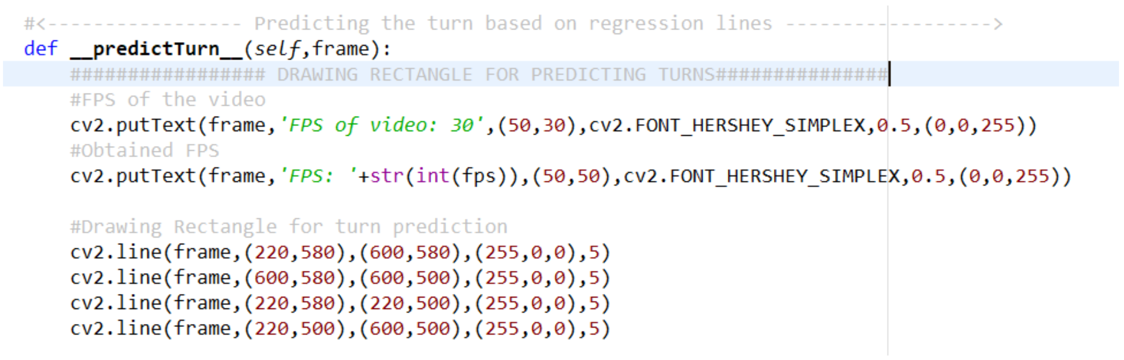
2.2 Detecting lines and predicting turn from recorded video.

1. We read the video with detected lines. Received frames are passed to function for predicting the turn.



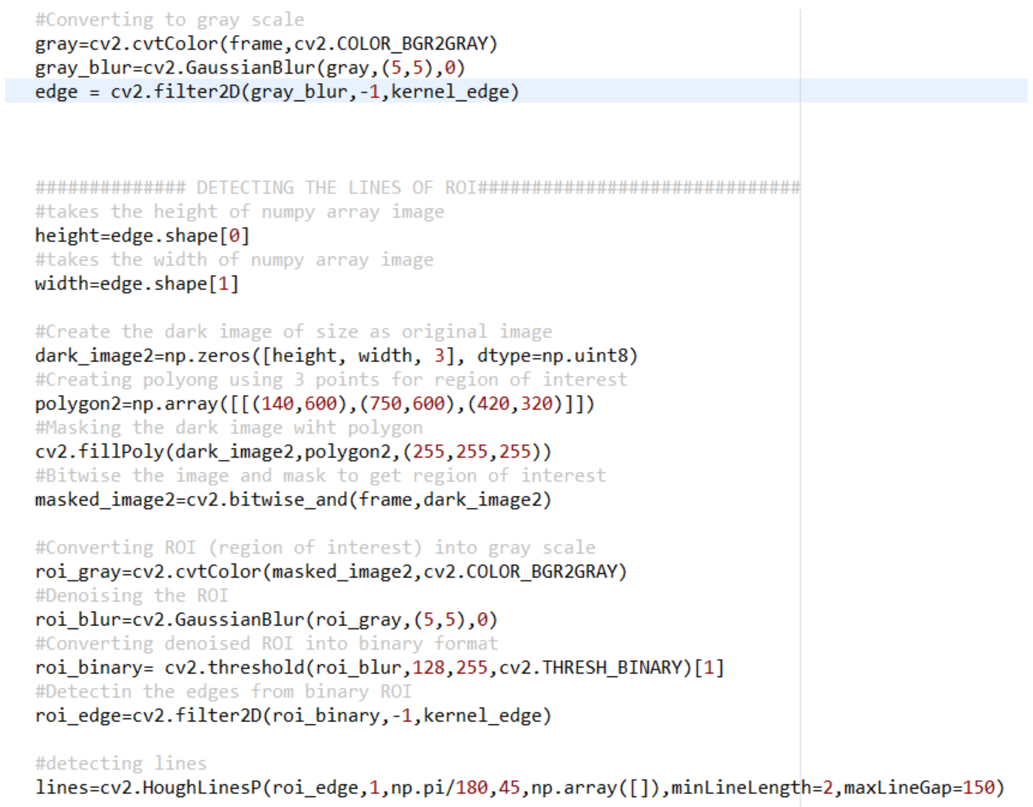
```
capture=cv2.VideoCapture("output.avi")

############# For detecting lines and predicting turn from video##############
try:

    while True:
        #Reads the frames of video
        ret, frame = capture.read()
        timer = cv2.getTickCount()
        fps = cv2.getTickFrequency()/(cv2.getTickCount()-timer)
        #If the ret is False then continue
        if ret is False:
            continue
        else:

            ob.__predictTurn__(frame)
            k=cv2.waitKey(1) & 0xff

            if k==ord('q'):
                cv2.destroyAllWindows();
                break

#To stop recording the video press Ctrl-C
except KeyboardInterrupt:
    print("Ctrl-C is Pressed")
    pass
```

2. Drawing the rectangle for distance calculation between center of frame and center of lane.



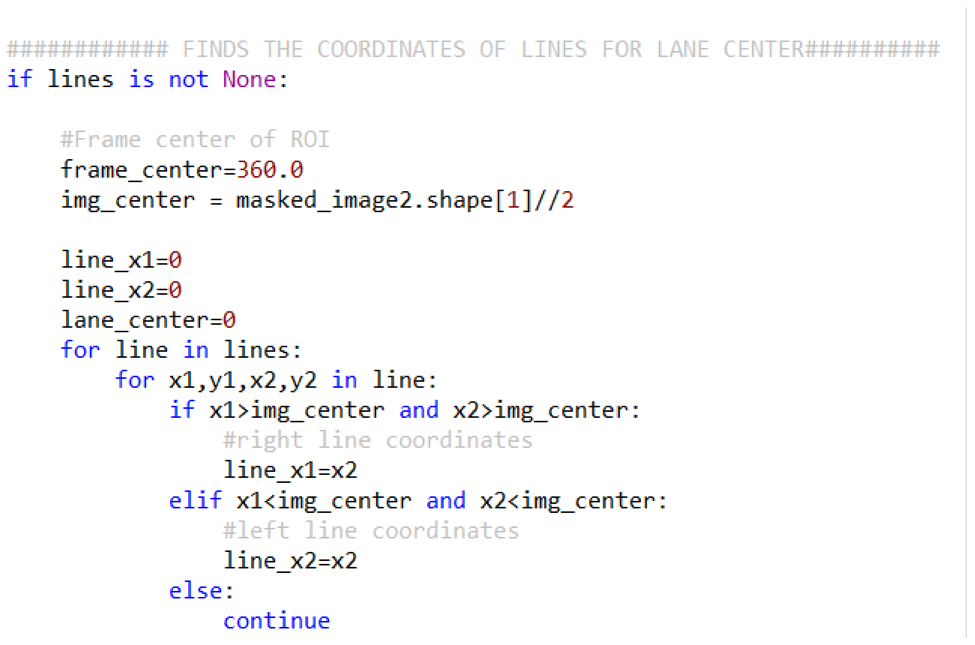
```
#<---------------- Predicting the turn based on regression lines ----------------->
def __predictTurn__(self,frame):
    ################ DRAWING RECTANGLE FOR PREDICTING TURNS###############
    #FPS of the video
    cv2.putText(frame,'FPS of video: 30',(50,30),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255))
    #Obtained FPS
    cv2.putText(frame,'FPS: '+str(int(fps)),(50,50),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255))

    #Drawing Rectangle for turn prediction
    cv2.line(frame,(220,580),(600,580),(255,0,0),5)
    cv2.line(frame,(600,580),(600,500),(255,0,0),5)
    cv2.line(frame,(220,580),(220,500),(255,0,0),5)
    cv2.line(frame,(220,500),(600,500),(255,0,0),5)
```

3. Detecting the lines of ROI. The frame is converted to grayscale then it is blurred. After which it is filtered for edge detection. The same procedure is applied fro ROI for detecting the lines using HoughLinesP algorithm.



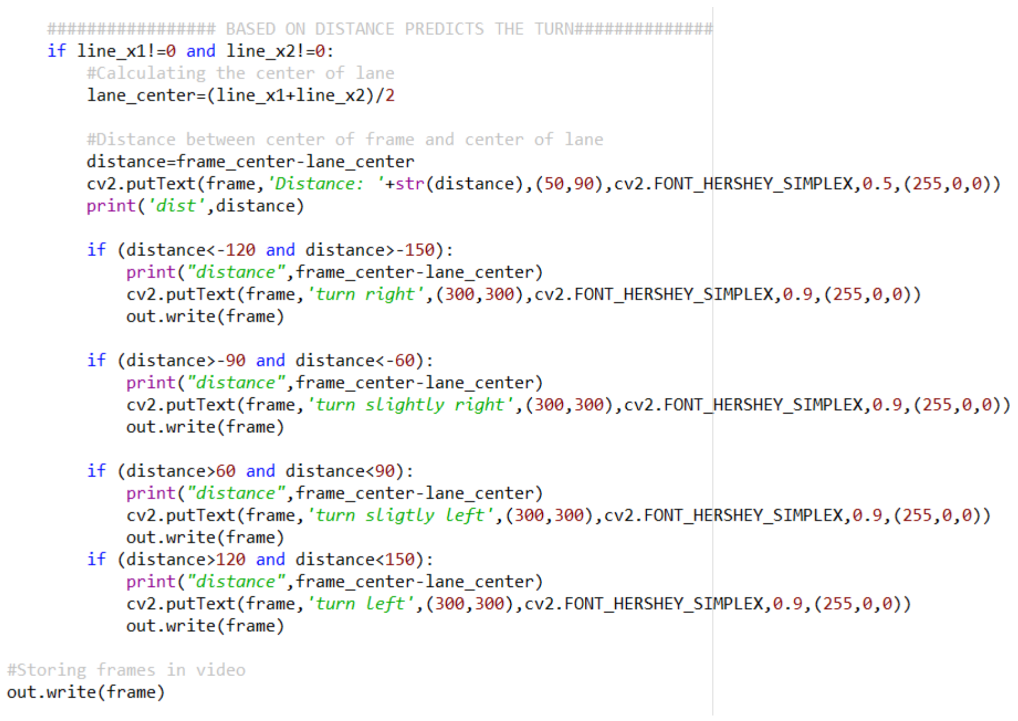
```
#Converting to gray scale
gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
gray_blur=cv2.GaussianBlur(gray,(5,5),0)
edge = cv2.filter2D(gray_blur,-1,kernel_edge)



############## DETECTING THE LINES OF ROI##############################
#takes the height of numpy array image
height=edge.shape[0]
#takes the width of numpy array image
width=edge.shape[1]

#Create the dark image of size as original image
dark_image2=np.zeros([height, width, 3], dtype=np.uint8)
#Creating polyong using 3 points for region of interest
polygon2=np.array([[(140,600),(750,600),(420,320)]])
#Masking the dark image wiht polygon
cv2.fillPoly(dark_image2,polygon2,(255,255,255))
#Bitwise the image and mask to get region of interest
masked_image2=cv2.bitwise_and(frame,dark_image2)

#Converting ROI (region of interest) into gray scale
roi_gray=cv2.cvtColor(masked_image2,cv2.COLOR_BGR2GRAY)
#Denoising the ROI
roi_blur=cv2.GaussianBlur(roi_gray,(5,5),0)
#Converting denoised ROI into binary format
roi_binary= cv2.threshold(roi_blur,128,255,cv2.THRESH_BINARY)[1]
#Detectin the edges from binary ROI
roi_edge=cv2.filter2D(roi_binary,-1,kernel_edge)

#detecting lines
lines=cv2.HoughLinesP(roi_edge,1,np.pi/180,45,np.array([]),minLineLength=2,maxLineGap=150)
```

4. Calculating the center of lane for predicting the turn



```
############ FINDS THE COORDINATES OF LINES FOR LANE CENTER##########
if lines is not None:

    #Frame center of ROI
    frame_center=360.0
    img_center = masked_image2.shape[1]//2

    line_x1=0
    line_x2=0
    lane_center=0
    for line in lines:
        for x1,y1,x2,y2 in line:
            if x1>img_center and x2>img_center:
                #right line coordinates
                line_x1=x2
            elif x1<img_center and x2<img_center:
                #left line coordinates
                line_x2=x2
            else:
                continue
```

5. Calculating the difference between the center of frame and center of lane based on this predicting the turn of car. Lane center is found by line coordinate and finding center of them.
   Distance is equal frame_center – lane_center which gives number by that number we decide to turn.



```
    ################# BASED ON DISTANCE PREDICTS THE TURN##############
    if line_x1!=0 and line_x2!=0:
        #Calculating the center of lane
        lane_center=(line_x1+line_x2)/2

        #Distance between center of frame and center of lane
        distance=frame_center-lane_center
        cv2.putText(frame,'Distance: '+str(distance),(50,90),cv2.FONT_HERSHEY_SIMPLEX,0.5,(255,0,0))
        print('dist',distance)

        if (distance<-120 and distance>-150):
            print("distance",frame_center-lane_center)
            cv2.putText(frame,'turn right',(300,300),cv2.FONT_HERSHEY_SIMPLEX,0.9,(255,0,0))
            out.write(frame)

        if (distance>-90 and distance<-60):
            print("distance",frame_center-lane_center)
            cv2.putText(frame,'turn slightly right',(300,300),cv2.FONT_HERSHEY_SIMPLEX,0.9,(255,0,0))
            out.write(frame)

        if (distance>60 and distance<90):
            print("distance",frame_center-lane_center)
            cv2.putText(frame,'turn sligtly left',(300,300),cv2.FONT_HERSHEY_SIMPLEX,0.9,(255,0,0))
            out.write(frame)
        if (distance>120 and distance<150):
            print("distance",frame_center-lane_center)
            cv2.putText(frame,'turn left',(300,300),cv2.FONT_HERSHEY_SIMPLEX,0.9,(255,0,0))
            out.write(frame)

#Storing frames in video
out.write(frame)
```

3. Division of tasks:

We had to divide our tasks into 3 parts so that our job would be efficient.

Task Division		
Team Name	tripLM 16-3	**Date:** May 6, 2020
Role	**Name**	**Main Responsibility**
Main Coder	**Mirzashomol Karshiev**	Implemented code on Spider using TensorFlow and OpenCV libraries.
Facilitator	**Mardon Zarifjonov**	Helped with writing report, requirement of analysis.
Project Analyst	**Mirpulatjon Shukurov**	Helped with finding useful information and writing SW design specification.
Coder	**Lazizbek Qahorov**	Helped with optimizing the code and fixing bugs.