***Main Algorithms***

detection\_graph = tf.Graph()  
with detection\_graph.as\_default():  
 od\_graph\_def = tf.GraphDef()  
 with tf.gfile.GFile(PATH\_TO\_CKPT, 'rb') as fid:  
 serialized\_graph = fid.read()  
 od\_graph\_def.ParseFromString(serialized\_graph)  
 tf.import\_graph\_def(od\_graph\_def, name='')sess = tf.Session(graph=detection\_graph)# Define input and output tensors (i.e. data) for the object detection classifier# Input tensor is the image  
image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0')# Output tensors are the detection boxes, scores, and classes  
# Each box represents a part of the image where a particular object was detected  
detection\_boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0')# Each score represents level of confidence for each of the objects.  
# The score is shown on the result image, together with the class label.  
detection\_scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0')  
detection\_classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0')# Number of objects detected  
num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0')

Main Logic:

def region\_of\_interest(img, vertices):  
mask = np.zeros\_like(img)   
if len(img.shape) > 2:  
channel\_count = img.shape[2]  
ignore\_mask\_color = (255,) \* channel\_count  
else:  
ignore\_mask\_color = 255   
cv2.fillPoly(mask, vertices, ignore\_mask\_color)  
masked\_image = cv2.bitwise\_and(img, mask)  
return masked\_image

video = cv2.VideoCapture("test\_video.mp4")fourcc = cv2.VideoWriter\_fourcc(\*'XVID')  
frame\_width = int(video.get(3))  
frame\_height = int(video.get(4))  
# out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))  
# out = cv2.VideoWriter('outpy.mp4',cv2.VideoWriter\_fourcc('M','J','P','G'), 20, (frame\_width,frame\_height))  
out = cv2.VideoWriter('output\_video.avi',cv2.VideoWriter\_fourcc('M','J','P','G'), 20 , (frame\_width,frame\_height))try:  
   
 while(video.isOpened()):  
 ret, frame = video.read()  
 stime = time.time()  
 objects = []  
 class\_str = ""  
 frame\_width = frame.shape[0]  
 frame\_height = frame.shape[1] rows, cols = frame.shape[:2] left\_boundary = [int(cols\*0.40), int(rows\*0.95)]  
 left\_boundary\_top = [int(cols\*0.40), int(rows\*0.20)]  
 right\_boundary = [int(cols\*0.60), int(rows\*0.95)]  
 right\_boundary\_top = [int(cols\*0.60), int(rows\*0.20)] bottom\_left = [int(cols\*0.20), int(rows\*0.95)]  
 top\_left = [int(cols\*0.20), int(rows\*0.20)]  
 bottom\_right = [int(cols\*0.80), int(rows\*0.95)]  
 top\_right = [int(cols\*0.80), int(rows\*0.20)]  
 vertices = np.array([[bottom\_left, top\_left, top\_right, bottom\_right]], dtype=np.int32) cv2.line(frame,tuple(bottom\_left),tuple(bottom\_right), (255, 0, 0), 5)  
 cv2.line(frame,tuple(bottom\_right),tuple(top\_right), (255, 0, 0), 5)  
 cv2.line(frame,tuple(top\_left),tuple(bottom\_left), (255, 0, 0), 5)  
 cv2.line(frame,tuple(top\_left),tuple(top\_right), (255, 0, 0), 5) copied = np.copy(frame)  
 interested=region\_of\_interest(copied,vertices) frame\_expanded = np.expand\_dims(interested, axis=0)  
  
 (boxes, scores, classes, num) = sess.run(  
 [detection\_boxes, detection\_scores, detection\_classes, num\_detections],  
 feed\_dict={image\_tensor: frame\_expanded}) vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(  
 frame,  
 np.squeeze(boxes),  
 np.squeeze(classes).astype(np.int32),  
 np.squeeze(scores),  
 category\_index,  
 use\_normalized\_coordinates=True,  
 line\_thickness=8,  
 min\_score\_thresh=0.78) print(frame\_width,frame\_height) ymin = int((boxes[0][0][0]\*frame\_width))  
 xmin = int((boxes[0][0][1]\*frame\_height))  
 ymax = int((boxes[0][0][2]\*frame\_width))  
 xmax = int((boxes[0][0][3]\*frame\_height)) Result = np.array(frame[ymin:ymax,xmin:xmax])  
  
 ymin\_str='y min = %.2f '%(ymin)  
 ymax\_str='y max = %.2f '%(ymax)  
 xmin\_str='x min = %.2f '%(xmin)  
 xmax\_str='x max = %.2f '%(xmax)  
 cv2.putText(frame,ymin\_str, (50, 50),cv2.FONT\_HERSHEY\_SIMPLEX,0.6,(255,0,0),2)  
 cv2.putText(frame,ymax\_str, (50, 70),cv2.FONT\_HERSHEY\_SIMPLEX,0.6,(255,0,0),2)  
 cv2.putText(frame,xmin\_str, (50, 90),cv2.FONT\_HERSHEY\_SIMPLEX,0.6,(255,0,0),2)  
 cv2.putText(frame,xmax\_str, (50, 110),cv2.FONT\_HERSHEY\_SIMPLEX,0.6,(255,0,0),2)  
 print(scores.max())  
 print("left\_boundary[0],right\_boundary[0] :", left\_boundary[0], right\_boundary[0]) print("left\_boundary[1],right\_boundary[1] :", left\_boundary[1], right\_boundary[1]) print("xmin, xmax :", xmin, xmax)  
 print("ymin, ymax :", ymin, ymax) if scores.max() > 0.78:  
 print("inif") if(xmin >= left\_boundary[0]):  
 print("move LEFT - 1st !!!")  
 cv2.putText(frame,'Move LEFT!', (300, 100),cv2.FONT\_HERSHEY\_SIMPLEX,1.5,(0,255,0),2) elif(xmax <= right\_boundary[0]):  
 print("move Right - 2nd !!!")  
 cv2.putText(frame,'Move RIGHT!', (300, 100),cv2.FONT\_HERSHEY\_SIMPLEX,1.5,(0,255,0),2) elif(xmin <= left\_boundary[0] and xmax >= right\_boundary[0]):  
 print("STOPPPPPP !!!! - 3nd !!!")  
 cv2.putText(frame,' STOPPPPPP!!!', (300, 100),cv2.FONT\_HERSHEY\_SIMPLEX,1.5,(0,255,0),2) cv2.line(frame,tuple(left\_boundary),tuple(left\_boundary\_top), (255, 0, 0), 5)  
 cv2.line(frame,tuple(right\_boundary),tuple(right\_boundary\_top), (255, 0, 0), 5)out.write(frame)  
except:  
 pass