Traffic Count

The traffic counts provide a baseline of traffic volume that is used to perform analysis on a roadway or intersection. Traffic counts are used to determine the number of lanes needed on a roadway. At an intersection, traffic counts are used to determine the number and length of turn lanes needed, and the appropriate traffic control measure (stop sign, traffic signal, roundabout, etc.). If there is a traffic signal involved, traffic counts are critical to developing appropriate signal timing. As discussed above counting of traffic is import for the controlled and smooth flow of vehicles on the road. The traffic is counted either automatically or manually. There are many different devices and services to count the vehicles automatically. One of the popular ways to count traffic is through laying Pneumatic tubes across the road for more detailed explanation visit https://www.youtube.com/watch?time_continue=42&v=Y4RUnJ0EiFk . There are even services dedicated to do the traffic counts one of the popular one is "Data From Sky" http://datafromsky.com/ .

No special explanation is required for the manual counts, it is literally sitting beside a road counting the vehicles passing by.

The method employed in this project is similar to what "Data From Sky" delivers on traffic counts. A traffic video is analyzed using opencv2 and python3 . First the video is converted into the data, python can work with and this conversion is done by using opencv. As per my experience from this project, considering the factors like angle of the video, sunlight , disturbances , shaking of the camera etc will all impact the counting. Every video or image will require customized conversions using opencv and also different python algorithms to count the vehicles. Of course, if machine learning is used to train the models to count the vehicles in any given condition then the process is easy. However, using machine learning is out of this project's scope as I am in initial stages of learning opencv and new to this traffic counting.

My initial experiments using opencv was done in jupyter notebook, since it is easy to view the results from the code executed. The first problem I encountered using opencv is that it does not work well with jupyter notebook. The code opens a window to show the changes done to the video/image. After the execution is completed the window should close by itself because I added cv2.destroyAllWindows() or cv2.destroyWindow("window name") in the code. This works well through shell but gets stuck when using notebook. I referred http://jorgemoreno.xyz/pycvtraffic.php for the code. The code from that website works well with the video posted in same blog which is "traffic3.mp4" , but modification were required to count the vehicles for other video named "traffic4.mp4". The initial experiments to understand the way opencv works was done in this notebook https://github.com/bhavyaramgiri/Traffic-count/blob/master/Opencv_trials.ipynb . Detailed explanation of all the opencv functions are explained in the notebook with relevant web-links to learn more about the topics.

Next, I have tried to count vehicles in an intersection of the road. This video file is names as "traffic4.mp4" and the code for the same is executed through the script. For counting the vehicles 3 libraries are imported Numpy, Pandas and Opencv2. The entire code is divided into 6 parts. 1st part is where variables are initialized, 2nd part is to identify the centroids of the vehicles in the frame under the blue line, 3rd part is to record the vehicles identified into a dataframe, 4th part to record the the number of vehicles in current frame only, 5th part is to identify the vehicles which crossed green line which is how the count is performed and the 6th part is where the computed values from all the parts above are displayed on the frame.

1st part:

In this part it is mainly variable initialization. The video variable is one them, which needs to be configured before the frames starts executing through the while loop. The syntax to create the video objects is (“intended name of file”, format of saving, fps, (height, width)) . Frames per second, height and width of the frame of the video has to be found before executing the cv2.VideoWriter() function. There are 1586 number of frames for the entire video, 29 frames are displayed per second, the actual width of the frame is 1920 and height of the video is 1080. Inside the while loop the frames are reduced 50% of the size. This loop will execute the codes for one frame at a time as an image. So initially the first image will be converted to gray, back ground is subtracted, closing,opening and dilation operations are carried over foreground objects, then the image is converted into binary to remove any shadows. The contours of the foreground objects is identified, using hull; contours are drawn around the objects. The blue and green line set are drawn for straight road and other set at the turn of the road.

```
import numpy as np
import cv2
import pandas as pd

fgbg = cv2.createBackgroundSubtractorMOG2()# create background subtractor
cap = cv2.VideoCapture('traffic4.mp4') # captures the video
fps = cap.get(cv2.CAP_PROP_FPS) # frames per second
fps = int(fps)
frames_count = cap.get(cv2.CAP_PROP_FRAME_COUNT) # total frame count
frames_count = int(frames_count)
width = cap.get(cv2.CAP_PROP_FRAME_WIDTH) # Width of the frame
height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT) # height of the frame

print("Number of frames in entire video :{}".format(frames_count))
print("Number of frames per second:{}".format(fps))
print("Width of the video:{}".format(int(width)))
print("Height of the video:{}".format(int(height)))

# we will require dataframe once we are inside the while loop to store the centroids of the vehicles
# one data frame for the cars continuing going straight and other dataframe for the cars joing the
# straight road from the turn.
df = pd.DataFrame(index=range(int(frames_count)))
df.index.name = "Frames"
df_turn = pd.DataFrame(index=range(int(frames_count)))
df_turn.index.name = "Frames"

# all the variables below will be updated frame by frame once inside the while loop.
framenumber = 0  # frame number will be updated for each loop
carscrossed_straight = 0
carscrossed_turn = 0
carids = [] # vehicle unique id are appended in the empty list which enters the frame
carids_turn = []
caridscrossed = [] # vehicles which cross the green line are noted down
caridscrossed_turn = []
totalcars = 0 # Total number of the cars as per the ids detected will be updated inside the loop.
totalcars_turn = 0

# To capture the changes made on the video, creating "video" object
ret, frame = cap.read()  # import image
h = int(frame.shape[0])
w = int(frame.shape[1])
height = int(h/2)
```

```
width = int(w/2)
dsize = (width, height)
image = cv2.resize(frame,dsize)  # resize image
width2, height2, channels = image.shape
# Above codes are for finding fps, height and width of the frame which are the inputs to  cv2.VideoWriter() function.
video = cv2.VideoWriter('traffic_count.avi', cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), fps, (height2, width2))


while True: # while loop will execute the code below as long as check returns True.
    ret, frame = cap.read() # check/ret have same function of storing boolean values
    if ret: # if ret is True the following code will execute or else it will break

        #reducing every frame by 50%
        scale_percent = 50
        width = int(frame.shape[1] * scale_percent / 100)
        height = int(frame.shape[0] * scale_percent / 100)
        dsize = (width, height)
        image = cv2.resize(frame,dsize)  # resize image

        # processing every frame to draw contours
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # converts image to gray
        fgmask = fgbg.apply(gray) # applying background subtractor
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15, 15)) # initializing ellipse kernel
        closing = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
        opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
        dilation = cv2.dilate(opening, kernel)
        ret, bins = cv2.threshold(dilation, 220, 255, cv2.THRESH_BINARY) # converson to binary
        contours, hierarchy = cv2.findContours(bins, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # fining contours
        hull = [cv2.convexHull(c) for c in contours] # creating hull around contour points
        cv2.drawContours(image, hull, -1, (255,255,255), 2) # drawing contour

        # drawing the lines for carscrossed_straight
        # all the ids are considered for the vehicles below blue line
        # all the vehicles which cross green line will be counted
        blue_x_line1a = 290
        blue_y_line1a = 190
        blue_x_line1b = 540
        blue_y_line1b = 260
        cv2.line(image, (blue_x_line1a, blue_y_line1a), (blue_x_line1b, blue_y_line1b), (255,0,0), 2)
        green_x_line1a = 270
        green_y_line1a = 220
        green_x_line1b = 510
        green_y_line1b = 290
        cv2.line(image, (green_x_line1a, green_y_line1a), (green_x_line1b, green_y_line1b), (0, 255, 0), 2)
        # drawing the line for carcrossed_turn
        blue_x_line2a = 750
        blue_y_line2a = 480
        blue_x_line2b = 750
        blue_y_line2b = 180
        cv2.line(image, (blue_x_line2a,blue_y_line2a), (blue_x_line2b, blue_y_line2b), (255, 0, 0), 2)
        green_x_line2a = 795
        green_y_line2a = 490
        green_x_line2b = 795
        green_y_line2b = 200
        cv2.line(image, (green_x_line2a,green_y_line2a), (green_x_line2b,green_y_line2b), (0,255,0), 2)
```

2nd part:

If the image has contours, parent contour is identified, area of the parent contour is acquired and compared to be between 7000 and 60000. 60000 is for the big vehicles like trucks and 7000 for vehicles small like cars. Once the contours satisfying the above condition is filtered in , the centroids of these contours are identified. The contours once drawn on the vehicles will be presents till the vehicles leaves the frame. When the vehicles moves forward their size decreases and so does their contours and in distance their contours merge together, in order to avoid that and make process simple vehicles only below the blue line are considered which is near to the camera angle thus individual cars can be identified easily. To achieve that each x and y coordinate of the centroids of all the parent contours below the green line are selected. Any cars above green line will not be considered for counting. These filtered centroids are then stored in cxx and cyy list. This whole process is repeated again for vehicles which join the straight road through a turn.

```
minarea = 7000 # minimum area of the contours allowed
maxarea = 60000 # maximum area of the contour allowed
cxx = [] # The x co-ordinate of centroid will be updated
cyy = [] # the y co-ordinate of the centroid will be updated

for i in range(len(contours)): # of all the contours in the frame
    if hierarchy[0, i, 3] == -1: # choosing only the parent ones
        area = cv2.contourArea(contours[i]) # area of the parent contour is identified
        if minarea < area < maxarea: # the area is checked if is in between the contraints
            cnt = contours[i]
            M = cv2.moments(cnt)
            cx = int(M['m10'] / M['m00']) # x co-ordinate of centroid
            cy = int(M['m01'] / M['m00']) # y co-ordinate of centroid
            # by adding markers, it can be verified that the contours are drawn for one vehicles only.
            if cy > blue_y_line1a and cx < blue_x_line1b:
                cv2.drawMarker(image, (cx, cy), (0, 0, 255), cv2.MARKER_STAR, markerSize=5,
thickness=10,line_type=cv2.LINE_AA)
                cxx.append(cx)
                cyy.append(cy)


# same process is applied for vehicle joing from the turn to the straight road
minarea_turn = 1000
maxarea_turn = 40000
cxx_turn = []
cyy_turn = []

for i in range(len(contours)):
    if hierarchy[0, i, 3] == -1:
        area = cv2.contourArea(contours[i])
        if minarea_turn < area < maxarea_turn:
            cnt = contours[i]
            M = cv2.moments(cnt)
            cx = int(M['m10'] / M['m00'])
            cy = int(M['m01'] / M['m00'])
            # by adding markers, it can be verified that the contours are drawn for one vehicles only.
            if cx >= blue_x_line2a:
                cv2.drawMarker(image, (cx, cy), (0, 0, 255), cv2.MARKER_STAR, markerSize=5,
thickness=10,line_type=cv2.LINE_AA)
```

```
                cxx_turn.append(cx)
                cyy_turn.append(cy)
```

3rd part:

This part will execute only for frames with vehicles in them, i.e if centroids are captured from 2nd part of the code otherwise the 4th part will execute skipping the 3rd part. Once the centroids are captured from the second frame, carids list is checked to be empty, if it is then none of the frame before the current frame could be having vehicles in them. So any vehicles in current frame are the new one. An id is assigned to these vehicles, a column numbered the same id will be created in dataframe and the centroids of the vehicles will be added below its id column and in the row of its current frame. Totalcars count is increased accordingly. For examples if first car appears at 16th frame of the video, then its centroid will be stored in dataframe at 0th column and 16th row. Next if carid has ids stored in it from any of the previous frames then the else statement is executed. The purpose for this “else” part of the code is to track the vehicles and not double count them. An empty array of zeros is created which will be of dimension of number of rows equal to number of vehicles below blue line in current frame and number of columns equal to number of all vehicles identified from first to previous frame which are stored in carid. So, if 3 vehicles are in carid lis and 4 in current frame, we have to make sure if any of the vehicles from the carid list are still in current frame. If any of the centroids of the vehicles from previous frame is same to one of the centroid in current frame than its clear that the vehicles did not move in between frames and its the same vehicle. Then the centroid is stored in df under the same id and in current frame.
There may be a case where previous frame may be empty because there were no vehicles in it, in such case oldcxcy will be an empty. So if there is no ids from the previous frame then loop will execute for next centroid. After we get the centroids of previous and current cars their differences is stored in dx and dy array. The difference of x and y coordinates are added column wise, and the minimum values is selected using np.argmin() function. The argmin() function will give the index number of the min value within the array. Once we have the index of minimum value, that difference is extracted using index and stored in mindx and mindy variable. mindx and mindy can be zero if the centroids are same or the values were never changed from initial zeros in dx and dy array.

df	0th carid	1st carid	2nd carid
0th frame	[x1,y1]	[x2,y2]	
1st frame			[x3,y3]

To explain that lets consider that the table above is the df , we had 2 vehicles in first frame and one new vehicle in second frame and for third frame we have one vehicle of centroid [x4,y4]. Now the length of the carids is 3. So the dx and dy array will be of shape 1x3 all filled with zeros. If [x4,y4] values is same as [x3,y3] then those centroids will be saved in the row below [x3,y3]. If it is not same , difference between [x4,y4] and [x3,y3] is calculated and stored in dx and dy. Because [x1,y1] and [x2,y2] are from frame 0 , their centroids are never pulled in “oldcxcy” variable to subtract with [x4,y4] , thus the zeros assigned for them in dx and dy are never replaced by the difference and are left to zero. We don't want to consider these zeros, because in array if we have multiple zeros the argmin() function will pull out the index of first zero.
Now if we actually found some difference between olcxcy and curcxcy then those differences have to be within a min radius range of 70 to be considered as the same car, that is the vehicles moved from one frame to another at a distance of less than 70. If this condition is met, those centroids will be saved in df in the same carid column. Such vehicles whose centroids are saved in df so far, their indexes are

stored in “min_index” list. Next the vehicles whose centroid difference is more than 70, such cars are new and are added to a new column in df. The carids and totalcars is updated accordingly.
Same process is conducted for the vehicles joining the straight road from the turn.

```
min_index = []
maxrad = 70
# this will only execute for the frames with contours and centroids
if len(cxx):  # if cxx is filled and not empty
    if not carids: # if there are no vehicles "ids" recorded yet
        for i in range(len(cxx)):
            carids.append(i)
            df[str(carids[i])] = "" # creating an empty column of current carid
            df.at[int(framenumber), str(carids[i])] = [cxx[i], cyy[i]] # adding the centroids inside the df
            totalcars = carids[i] + 1  # updating the total cars recognized.

    else:  # if there are already car ids in the list
        dx = np.zeros((len(cxx), len(carids)))  # new arrays to calculate the difference between oldcent and current
        dy = np.zeros((len(cyy), len(carids)))

        for i in range(len(cxx)):
            for j in range(len(carids)):  # loops through all recorded car ids
               # acquires centroid from previous frame for specific carid
                oldcxcy = df.iloc[int(framenumber – 1)][str(carids[j])]
                # acquires current frame centroid that doesn't necessarily line up with previous frame centroid
                curcxcy = [cxx[i], cyy[i]]
                if oldcxcy == curcxcy: # if older and current centroid is same, then they are the same vehicle.
                    df.at[int(framenumber), str(carids[j])] = curcxcy
                    min_index.append(i)

                if not oldcxcy:  # checks if old centroid is empty
                    continue  # continue to next carid
                else:  # difference between all current centroids of this frame and old centroids of previous are stored.
                    dx[i, j] = np.abs(oldcxcy[0] - curcxcy[0])
                    dy[i, j] = np.abs(oldcxcy[1] - curcxcy[1])

        for j in range(len(carids)):
            sumsum = dx[:, j] + dy[:, j] # array of difference of a particular column
            # choosing the minimum value from array above and storing index number
            correctindextrue = np.argmin(sumsum)


            # The index is used to extract the minimum difference in order to check if it is within radius later on
            mindx = dx[correctindextrue, j]
            mindy = dy[correctindextrue, j]

            if mindx == 0 and mindy == 0: # this is for not considering the unfilled dx and dy elements
                continue
            else:
                # if the difference is less than the maximum radius than its the same car
                if mindx < maxrad and mindy < maxrad:
                   #adds centroid to corresponding previously existing carid
                    df.at[int(framenumber), str(carids[j])] = [cxx[correctindextrue], cyy[correctindextrue]]
                    min_index.append(correctindextrue)
        for i in range(len(cxx)):
            if i not in min_index:
                df[str(totalcars)] = ""  # create another column with total cars
                carids.append(totalcars)  # append to list of car ids
                df.at[int(framenumber), str(totalcars)] = [cxx[i], cyy[i]]  # add centroid to the new car id
                totalcars = totalcars + 1  # adds another total car the count
```

code below for vehicles coming in from the turn

```
        min_index_turn = []
        maxrad_turn = 70

        if len(cxx_turn):
            if not carids_turn:
                for i in range(len(cxx_turn)):
                    carids_turn.append(i)
                    df_turn[str(carids_turn[i])] = ""
                    df_turn.at[int(framenumber), str(carids_turn[i])] = [cxx_turn[i], cyy_turn[i]]
                    totalcars_turn = carids_turn[i] + 1

            else:
                dx_turn = np.zeros((len(cxx_turn), len(carids_turn)))
                dy_turn = np.zeros((len(cyy_turn), len(carids_turn)))

                for i in range(len(cxx_turn)):
                    for j in range(len(carids_turn)):
                        oldcxcy_turn = df_turn.iloc[int(framenumber - 1)][str(carids_turn[j])]
                        curcxcy_turn = [cxx_turn[i], cyy_turn[i]]
                        if oldcxcy_turn == curcxcy_turn:
                            df_turn.at[int(framenumber), str(carids_turn[j])] = curcxcy_turn
                            min_index_turn.append(i)

                        if not oldcxcy_turn:
                            continue
                        else:
                            dx_turn[i, j] = np.abs(oldcxcy_turn[0] - curcxcy_turn[0])
                            dy_turn[i, j] = np.abs(oldcxcy_turn[1] - curcxcy_turn[1])

                for j in range(len(carids_turn)):
                    sumsum_turn = dx_turn[:, j] + dy_turn[:, j]
                    correctindextrue_turn = np.argmin(sumsum_turn)

                    mindx_turn = dx_turn[correctindextrue_turn, j]
                    mindy_turn = dy_turn[correctindextrue_turn, j]


                    if mindx_turn == 0 and mindy_turn == 0:
                        continue
                    else:
                        if mindx_turn < maxrad_turn and mindy_turn < maxrad_turn:
                          #adds centroid to corresponding previously existing carid
                            df_turn.at[int(framenumber), str(carids_turn[j])] = [cxx_turn[correctindextrue_turn],
cyy_turn[correctindextrue_turn]]

                            min_index_turn.append(correctindextrue_turn)


                for i in range(len(cxx_turn)):
```

```
    if i not in min_index_turn:
        df_turn[str(totalcars_turn)] = ""
        carids_turn.append(totalcars_turn)
        df_turn.at[int(framenumber), str(totalcars_turn)] = [cxx_turn[i], cyy_turn[i]]
        totalcars_turn = totalcars_turn + 1
```

4th part:

In this part the total number of vehicles in the current frame is added to the " currentcars" variable and index of such vehicles is appended into the "currentcarsindex" variable. Same process is conducted for the vehicles joining the straight road from the turn.

```
currentcars = 0  # current cars on screen
currentcarsindex = []  # current cars on screen carid index


for i in range(len(carids)):  # loops through all carids

    if df.at[int(framenumber), str(carids[i])] != '':

        currentcars = currentcars + 1  # adds another to current cars on screen
        currentcarsindex.append(i)  # adds car ids to current cars on screen

currentcars_turn = 0  # current cars on screen
currentcarsindex_turn = []  # current cars on screen carid index


for i in range(len(carids_turn)):  # loops through all carids

    if df_turn.at[int(framenumber), str(carids_turn[i])] != '':

        currentcars_turn = currentcars_turn + 1  # adds another to current cars on screen
        currentcarsindex_turn.append(i)  # adds car ids to current cars on screen
```

5th part:

This is the part where we add "ID" text in the video and also count the number of vehicles crossing the green line. The current vehicle's centroids and the centroid of the vehicle of the previous frames are stored in variables "curcent" and "oldcent". The if statement in line 292 is very important, in case the previous frame is 0th , the frame number will (0-1) I.e "-1" , in python that will be the last row. So to avoid that we add a if statement specifically for 0th frame. A circle is drawn using "oldcent" points to visually if oldcent and curcent are within the circle, it also means that oldcent is not empty . Next is the condition which counts the vehicles, if any "curcent" is below the green line and "oldcent" is above the green line that means the vehicles traveled forward and such vehicles count is stored in the variables. Same process is conducted for the vehicles joining the straight road from the turn.

```
        for i in range(currentcars):
            curcent = df.iloc[int(framenumber)][str(carids[currentcarsindex[i]])]

            if framenumber != 0: # This is to avoid for choosing the last frame from the df if the frame is 0
                oldcent = df.iloc[int(framenumber - 1)][str(carids[currentcarsindex[i]])]
            else:
                oldcent = []

            if curcent:
                cv2.putText(image, "ID:" + str(carids[currentcarsindex[i]]), (int(curcent[0]), int(curcent[1] -
10)),cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 255, 255), 2)


                if oldcent:  # checks if old centroid exists
                # the circle below is visualization for area within which curcent and oldcent are of the same vehicle.
                    cv2.circle(image,(tuple(oldcent)), maxrad, (0,0,255), 1)


                # checks if old centroid is on or below line and curcent is on or above line
                    if oldcent[0] <= green_x_line1b and oldcent[1] >= green_y_line1a and curcent[0] <= green_x_line1b and
curcent[1] < green_y_line1a and carids[currentcarsindex[i]] not in caridscrossed:
                        carscrossed_straight = carscrossed_straight + 1
                        cv2.line(image,(green_x_line1a, green_y_line1a), (green_x_line1b, green_y_line1b), (0, 0, 255), 5)
                        caridscrossed.append(currentcarsindex[i])  # adds car id to list of count cars to prevent double counting


# 5th part for vehicles joining the straight road from the turn.
        for i in range(currentcars_turn):  # loops through all current car ids on current frame
            curcent_turn = df_turn.iloc[int(framenumber)][str(carids_turn[currentcarsindex_turn[i]])]

            if framenumber != 0:
                oldcent_turn = df_turn.iloc[int(framenumber - 1)][str(carids_turn[currentcarsindex_turn[i]])]
            else:
                oldcent_turn = []

            if curcent_turn:  # if there is a current centroid
                cv2.putText(image, "ID:" + str(carids_turn[currentcarsindex_turn[i]]), (int(curcent_turn[0]), int(curcent_turn[1] -
10)),cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 255, 255), 2)

                if oldcent_turn:  # checks if old centroid exists
                # the circle below is visualization for area within which curcent and oldcent are of the same vehicle.
                    cv2.circle(image,(tuple(oldcent_turn)), maxrad_turn, (0,0,255), 1)

                # checks if old centroid is on or below line and curcent is on or above line
                    if oldcent_turn[0] >= green_x_line2a and curcent_turn[0] <= green_x_line2a and
carids_turn[currentcarsindex_turn[i]] not in caridscrossed_turn:
                        carscrossed_turn = carscrossed_turn + 1
                        cv2.line(image, (green_x_line2a,green_y_line2a), (green_x_line2b,green_y_line2b), (0, 0, 255), 5)
                        caridscrossed_turn.append(currentcarsindex_turn[i])
```

6th part:

Using opencv now the findings are displayed on the frame, number of vehicles going straight, number of vehicles joining the straight road from the turn , total number of cars in the video, count of frames and display of the time of video. All the changes done from the beginning of the code will only be displayed on the video when cv2.imshow() command is used. Next frame number is increased by one for the next loop. “key” will keep on displaying the video until it ends, however the video can be terminated by pressing “q” key. The else in line 359 execute if the “ret” in line 58 is false. After all the all the frames are completed executing, the df with all the centroids is saved in traffic.csv file. The capture object created in line 11 is closed and all the windows displaying video is closed too.

```
        cv2.rectangle(image, (0, 0), (250, 100), (169,169,169), -1)  # background rectangle for on-screen text
        cv2.putText(image, "Cars Crossed straight: " + str(carscrossed_straight), (0, 30),
cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 0, 0),2)
        cv2.putText(image, "Cars Crossed turn: " + str(carscrossed_turn), (0, 45), cv2.FONT_HERSHEY_SIMPLEX, .5,(0, 0,
0), 2)

        cv2.putText(image, "Total Cars Detected: " + str(len(carids) + len(carids_turn)), (0, 60),
cv2.FONT_HERSHEY_SIMPLEX, .5,(0, 0, 0), 2)

        cv2.putText(image, "Frame: " + str(framenumber) + ' of ' + str(frames_count), (0, 75),
cv2.FONT_HERSHEY_SIMPLEX,.5, (0, 0, 0), 2)

        cv2.putText(image, 'Time: ' + str(round(framenumber / fps, 2)) + ' sec of ' + str(round(frames_count / fps, 2))+ ' sec', (0,
90), cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 0, 0), 2)

        cv2.imshow("contour_image", image)

        framenumber = framenumber + 1

        # All the changes made to the video will be written and stored to the video itself
        video.write(image)

        key = cv2.waitKey(int(1000 / fps)) & 0xFF
        if key == ord('q'):
            break
    # if ret inside the while loop of 1st part returned False then the code would break here.
    else:

        break

# saves dataframe to csv file for later analysis
df.to_csv('traffic.csv', sep=',')
df_turn.to_csv('traffic_turn.csv', sep=",")
cap.release()
cv2.destroyAllWindows()
```

<u>Issues and challeges:</u>

The most time consuming but important part of this code was to draw the blue and green line appropriately on the video frame such as it covers all the vehicles. For instance for the straight road I had to draw the blue and green line at an angle not straight, this is because I don't want to add the vehicle’s centroids in the carid list which are going backwards at the other side of the road and also to avoid the centroids of the vehicles joining the straight road from the turn. Further, if we observe in the video the first car did not show the centroid marker up to few frames, I had to try several coordinates to

draw the lines after the frame where the first car id appears. Similar problem was faced during drawing the lines for vehicles coming from the turn, due to morphological transformation not being processed properly, the ids were created for the same car multiple times in multiple frame. This error can be viewed in the video " turn_issues.avi" file. So I had to figure out by trial and error the appropriate coordinates which will capture all the carids created before it changes in upcoming frames.
Other issue is the 1st car abruptly changes from carid 0 to carid 1, this could have been fixed by increasing the maximum radius from 70 to above, but when tried to do so, many other cars which were together in closer proximity in other frames combined to create a single carid.
In addition to above, the total car detected in the video which is displayed on the top left corner shows the total count to be 29, which is correct because we have 25 vehicles going straight and 4 joining from the right but the way it is achieved is wrong. The first car has 2 carids assigned and couple frames later there are two car driving in the same speed beside each other, which opencv considered as one car and gave them together single carid. That is how total count 29 is achieved.

Conclusion:

In total for proper traffic count, the contours area should cover all small to big vehicles. The coordinates should be used accurately for counting the vehicles. To assign the same id to same car in multiple frames the maximum radius should be used appropriately and the lines though not necessary to draw have to be choose in a way that is capturing all carids before they disappear and reappear again with new id because of morphological transformation issues.

The entire code to count vehicles from "traffic4.mp4": https://github.com/bhavyaramgiri/Traffic-count/blob/master/Vehicle_count.py
The link to view the video used in opencv trials https://www.youtube.com/watch?v=dTdsjKRyMuU
The link to see the actual video of this code "traffic4.mp4" is here https://youtu.be/vgvQIanWWB4
For "turn_issues.avi" https://youtu.be/H6dyVp9hBlE
The output video of this exercise is in https://youtu.be/-FK4wy6Tp10