Traffic Count

Summary:

In cities and towns the traffic is counted to study traffic volume on a given road and to provide better traffic solutions for smoother traffic flow. Traffic count data helps the city and towns to make traffic management decisions, such as when to implement traffic-calming measures, walking and cycling improvements, traffic and parking regulations, bus stop locations etc . But how do we count vehicles on the road? There are two ways to do that. One is manually and second automatically. Manual counting is a tedious process where one has to physically be at the location beside the road counting vehicles one by one. Second is counting automatically which can be done using devices such as traffic counters and counting through video.
The counting of vehicles using counters is the contact based system, whereas using video/image processing for counting is the contact less system. Contact based systems involve placing a tube or loop across a road and when a vehicle passes over the tube the pulse of air pressure closes an air switch, producing an electrical signal that is transmitted to a counter where the count data is stored and transferred. Whereas, the contact less system can extract the count information from the video or image using relevant softwares.
Traffic counters are expensive, as we need to install one at every location of the road wherever counting is required. Additionally, counters are effective only when vehicles follow strict lane discipline and they are unable to consistently detect non motorized vehicles. Counters also fail in providing directional count, that is the data cannot be classified into vehicles going straight, left, right or backwards. Two or more vehicles going abreast will be recorded as single unit.
For counting through videos, we need cameras to be installed on the traffic poles which are relatively inexpensive and we can use freely available software to count the vehicles. For this project I have used free python library and free image processing library called opencv for counting the vehicles in the video. Counting using video and free software is more effective and less expensive than the counters and indeed less tedious than manually counting it for hours.

Introduction:

The purpose of this project is to count the vehicles from a video captured by the cameras mounted on the poles beside the road. This approach of counting vehicles is not new, there are services such as “Data From Sky”[2] who provide solutions for vehicle counting. But I have made an attempt to do it myself using python programming and opencv image processing techniques. To explain my approach to count vehicles in simple words, I have transformed the video using opencv into the binary data which is then used to extract the counts using python codes. The code for this project is limited to the video “traffic4.mp4” alone. For any other video or image customized morphological conversions has to be carried using opencv and also different python codes may be required 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.

Methodology:

libraries required for this project are numpy, cv2, pandas and matplotlib.

For understanding the operation and functioning of opencv following codes/components are essential :

Note: The codes below are executed on video “traffic3.mp4”

1. To display a video using opencv

```
import numpy as np
import cv2
cap = cv2.VideoCapture('traffic3.mp4')
while True:
    check, frame = cap.read()
    cv2.imshow("trafficvideo", frame)
    key = cv2.waitKey(30)
    if key == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

cv2.VideoCapture() is the function which reads in the video in mp4 or avi or in any other format. Object "cap" is created to store video reading. The while loop will execute for every frame in the video. For this video we have 1500 frames. "check" is boolean data type, returns True if python is able to read the VideoCapture object and if not False is returned and the execution stops. "frame" is the NumPy array, it represents the first image of the video captured. 1500 such frames/images will be captured one by one. once the "check" returns true, the frame will be displayed by cv2.imshow() function. "key" is to generate a new frame after every 30 milliseconds. Once "q" is entered the windows will be destroyed. cap.release() will stop action of reading the video and cv2.destroyAllWindows() will close the window showing the video.

2. Displaying nth frame in the video. Say for 515th page,

o/p:

```
cap = cv2.VideoCapture('traffic3.mp4')
cap.set(1, 515)
ret, frame = cap.read()
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)
plt.axis('off')
plt.imshow(image)
```

Every frame in the video will be resized by 50% by using cv2.resize() function in order to reduce lag.

The image component above has three items in it. image.shape[0] is the height , image.shape[1] is the width and image.shape[2] gives the channels.

```
i/p: image.shape
o/p: (360, 640, 3)
```

3. Basic information on the video

OpenCv reads the video in the form of frames, for instance "traffic3.mp4" has 1500 frames. Each frame will be read in the form of numpy array. For gray image/frame the numpy matrix will be of 2 dimensions i.e rows x columns (default here it is a single channel). For colour image/frame the numpy matrix will be of 3 dimensions i.e rows x columns x channels.

To get the information on the video such has total number of frames, frames appearing in every second, width and height of a frame are achieved using following opencv commands.

Total frames:

```
i/p:  cap.get(cv2.CAP_PROP_FRAME_COUNT)
o/p: 1500
```

Frames per second:

```
i/p: cap.get(cv2.CAP_PROP_FPS)
o/p: 25
```

Width of the frame:

```
i/p: cap.get(cv2.CAP_PROP_FRAME_WIDTH)
o/p: 1280
```

Height of the frame:

```
i/p: cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
o/p: 720
```

4. Video writer:

We can draw on the video using opencv functions, drawing like adding a marker on vehicles, drawing a line or drawing a contour around the vehicles all such changes are written/drawn above the video frame using cv2.VideoWriter() function which will record all the drawing over the video and will save the edited video.
syntax for VideoWriter: cv2.VideoWriter(“desired filename”, fourcc, fps, (height,width))[4]
FourCC[5] is a 4-byte code used to specify the video codec. MJPG[6] is a safe choice for saving video in ".avi" format. For saving video in mp4 format use "*'mp4v'" as argument for fourcc[7] , however I found mp4 does not save properly.

```
video = cv2.VideoWriter('traffic_counter.avi', cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), fps, (height, width))
```

5. Background subtractor:

There are three background subtractor to choose from which are Background Subtractors MOG, MOG2 and GMG. MOG and GMG seems to be older version and are not functioning properly, however MOG2 works fine. For this project we are MOG2 algorithm which stands for Gaussian Mixture-based Background/Foreground Segmentation Algorithm[8]. Following is the code to subtract the background.

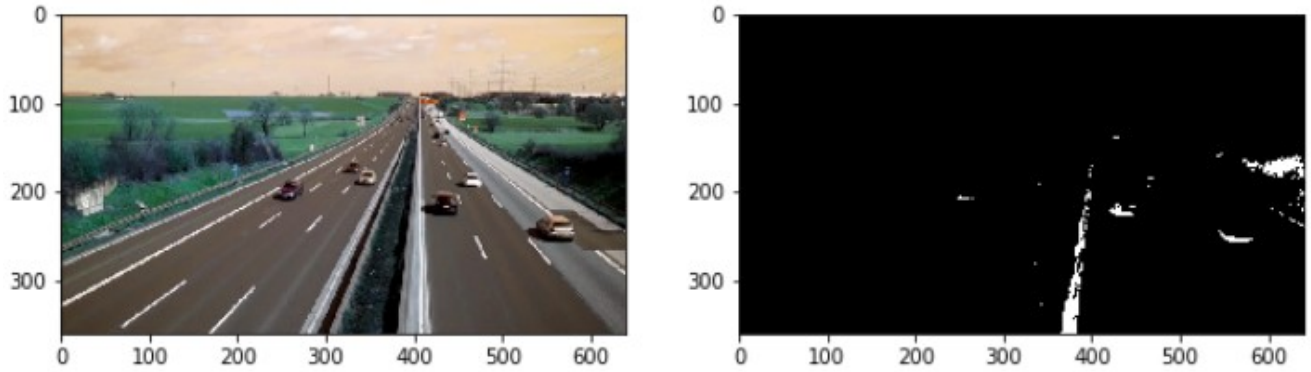
fgbg = cv2.createBackgroundSubtractorMOG2()

The background subtractor will subtract the static background and focus on moving foreground. It is the better approach to first convert the image/video to gray and then use background subtractor. The visual comparison between color frame with background subtracted (fgmask) of the same frame is shown below using matplotlib.

```
cap = cv2.VideoCapture('traffic3.mp4')
ret, frame = cap.read()
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)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # converts image to gray
fgmask = fgbg.apply(gray)

rcParams['figure.figsize'] = 11 ,8
fig, ax = plt.subplots(1,2)
ax[0].imshow(image);
# cmap - This is the color mapping. If this is not used in this case matplotlib will try to plot the gray images as a RGB image
ax[1].imshow(fgmask, cmap='gray');
```

o/p:



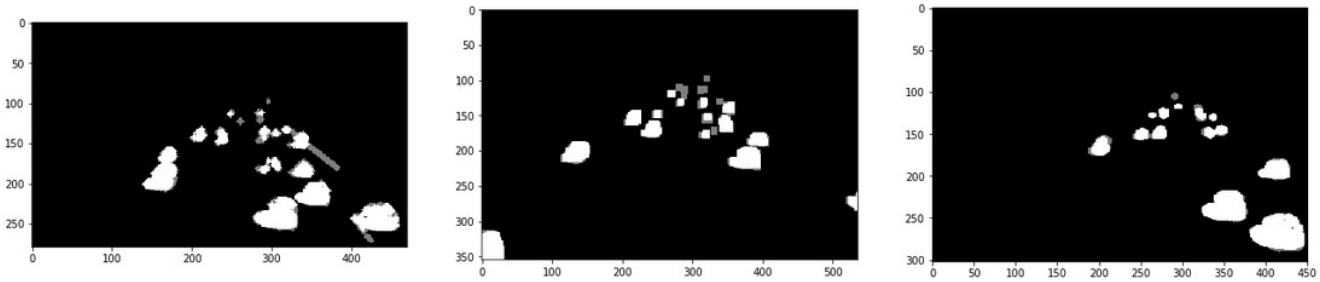
6. Processing the video for clarity

The following transformation is carried on the video before we use python to start counting the vehicles. The main point of applying the following mentioned transformations is to remove noise, isolate the cars, and make them into solid shapes which can be more easily tracked. All the below steps are the initial preparation of video before actual counting codes are written.

1. Converting each frame/video to gray scale using "cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)"

2. Applying background subtractor algorithm to the video which will take out the static background and show only the moving foreground, this is done by using "cv2.createBackgroundSubtractorMOG2()"[9]

3. Initiating the kernel [10], which will be used for morphological transformation of each frame in the video. The structural element [11] used in the kernel can be choosed from ellipse [12] , square or cross shaped. (5,5) is the shape of the kernel, in simple words matrix of 5 rows and 5 columns.
   Below are the example of cross, square and ellipse kernel initialization



4. After the kernel, morphological transformations like closing, opening and dilation is performed on the foreground video. [13]

5. In addition to the above transformations lastly binary threshold transformation [14,15,16] is applied on the top of all transformations. Threshold syntax is cv2.threshold(source image, threshold value, max value, threshold type) . Here threshold value is the pixel value and max value is assigned to pixel. Every value below 220(threshold value) will be set to zero and above 220 to value 255, 0 is for black, 255 is white and 220 is threshold closer to 255 .

The code for all the transformation mentioned above is as follows:

```
import numpy as np
import cv2
fgbg = cv2.createBackgroundSubtractorMOG2()# create background subtractor
cap = cv2.VideoCapture('traffic3.mp4')
fps = cap.get(cv2.CAP_PROP_FPS)
while True:
    check, frame = cap.read()
    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)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    fgmask = fgbg.apply(gray)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    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)
    cv2.imshow("fgmask+closing+opening+dilation+bins", bins)
    key = cv2.waitKey(int(1000 / fps)) & 0xFF
    if key == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

7. Contours and Hierarchy

We have so far made morphological conversions on the video, by doing that we have converted a color video into two dimensional binary image so that python can be now used to get information from the video or to make any changes over them. The next step if to identify the vehicles in the video. This is done by getting the contours, creating a hull on the objects. We will be drawing the contours for our visual understanding.

To locate the contours of the binary objects in image/video following syntax is used.

contours, hierarchy = cv2.findContours(bins, cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

Contours :

Contours[17] can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. All the contours saved is in a list. Each individual contour is a Numpy array of (x, y) coordinates of boundary points of the object. cv2.findContours() works best with binary images/video. We require contours here to mark the vehicles with convex hull in the video.

Hierarchy:

Sometimes due to the change in the sunlight/street light on the vehicles opencv may read multiple contours within a contour, that's when we need hierarchy[18]. Hierarchy gives ranking to the contours based on parent and child positions. All the hierarchy is stored in numpy.ndarray format. OpenCV represents hierarchy as an array of four values : [Next, Previous, First_Child, Parent] . Parent is the main contour which encompasses a vehicles all the contours inside this parent will be the child

RETR_TREE:
retrieves all of the contours and reconstructs a full hierarchy of nested contours [19].
CHAIN_APPROX_SIMPLE: removes all redundant points and compresses the contour, thereby saving memory.
Hull :
Using the contour points a hull [20] is to be created around objects in the image/video, syntax for the hull is hull = [cv2.convexHull(c) for c in contours] . cv2.convexHull() will use the contour points and create an outline i.e. a hull joining the points around the objects.
Drawing the contours :
So we have an outline hull around the objects, using this outline, contours are drawn using cv2.drawContours(). The syntax for drawing contours is explained: cv2.drawContours(image, hull, -1, (0, 255, 0), 3) here -1 is the used to draw all contours, individual contours can also be drawn by giving specific contours number other than -1 as an argument. (0,255,0) is the decimal code of (Blue,Green,Red) which is for lime color contour. 3 is the thickness of the contour line. One has to observe that opencv actually takes in the color arguments in (BGR) format rather than usual (RGB) [21]

The code below is executed to check the data and its form, stored in contours and hierarchy components. In order to view the captured items inside the contour and hierarchy above code is executed without while loop and for the first frame/ image of the video.

```
import numpy as np
import cv2
cap = cv2.VideoCapture('traffic3.mp4')
fps = cap.get(cv2.CAP_PROP_FPS)
fgbg = cv2.createBackgroundSubtractorMOG2()
check, frame = cap.read()
scale_percent = 50
#calculate the 50 percent of original dimensions
width = int(frame.shape[1] * scale_percent / 100)
height = int(frame.shape[0] * scale_percent / 100)
dsize = (width, height)
image = cv2.resize(frame,dsize)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
fgmask = fgbg.apply(gray)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
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)
contours, hierarchy = cv2.findContours(bins, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

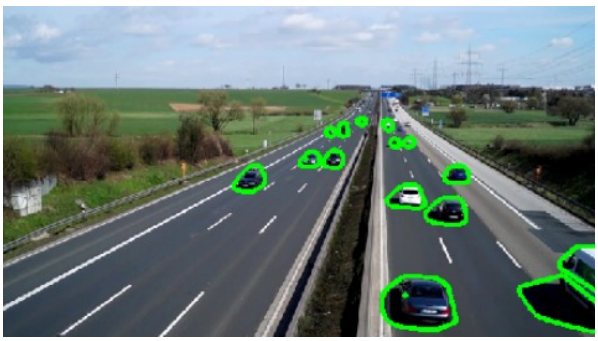
i/p: print(len(contours))
o/p: 9

so we have 9 contours in the first frame of the video. Its not necessary that all 9 have to be vehicles and we have an hierarchy of 9 items in it. Each item is ranked as parent or children, “-1” being children.

i/p: print("length of hierarchy:", len(hierarchy))
o/p: 1

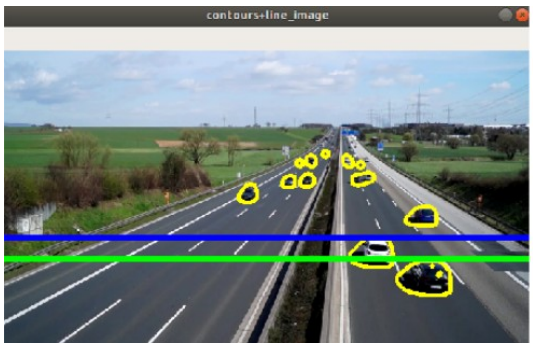
i/p: print(hierarchy)
o/p:
```
array([[[ 1, -1, -1, -1],
        [ 2,  0, -1, -1],
        [ 3,  1, -1, -1],
        [ 4,  2, -1, -1],
        [ 5,  3, -1, -1],
        [ 6,  4, -1, -1],
        [ 7,  5, -1, -1],
        [-1,  6,  8, -1],
        [-1, -1, -1,  7]]], dtype=int32)
```

When contours are drawn on the video, the syntax used is
cv2.drawContours(transformed image, hull, -1, (B,G,R), Thickness)



8. Drawing on the video frame.

For this project purpose we will be drawing lines, circles markers etc. on the video frame to help us visually comprehend the counting mechanism of our code. As you see from the image above the contours with vehicles are clearly visible at the bottom of the frame, the vehicles at the top part of the frame are minute and often merge together. A blue line will be drawn and the contours below it will only be considered for counting. The green line will be used to count the number of vehicles passing it upwards or downwards.
The syntax to draw line in the video frame is cv2.line(image, (x1, y1), (x2, y2), (B,G,R), line thickness).
Note that the line is from start of the frame and ends at the end of the frame. x1 will be 0 and x2 will be width of the frame. Y axis is inverted in opencv i.e 0 is at top and height is at bottom. I have also changed the color of contours to yellow to differentiate it from the green line.



9. Finding the centroids of the vehicles below blue line.

Parent is the main contour which is the outline of an foreground object (white object) and is assigned the value of “-1” by opencv in the hierarchy array . All the contours inside parent contours will be children contours. These children contours are assigned as next, previous or first child positions. Our focus is only to capture the parent contours of the objects which are below the blue line and then find their centroids. So first setting up a for loop for all contours one by one, then filtering in the contours which are parent only that is those with value “-1”. Then the area of that parent contour is calculated by using cv2.contourArea() function. Every foreground object in the frame will have contours which means we will also have contours for other objects in the frame along with the vehicles. To select contours of vehicles alone, the area of all the parent contours are compared between the limits of 300 to 50000 for this video, for any other video this range may change. All the area/contours coming under this constraints are selected. The centroids of such selected contours is identified using cv2.moments(contour) function [22]. The centroids consist of x and y coordinates. As per the code below cx and cy will be the centroid points of the contour being considered. Next, filtering in the contours whose centroid’s “y” coordinate is below the blue line cy > blueline. As mentioned earlier the y-axis for opencv frame starts from the height being at the bottom(origin) and 0 being at the top. So “cy” has to be greater than blueline which is “225” . After filtering the contours below blue line, a rectangle is created around such contours to differentiate them from the ones above the blue line. To create rectangle cv2.rectangle() [23] function is used. The arguments for this function includes the top left x and y coordinates and also the height and width of the rectangle which are all achieved by using cv2.boundingRect(contour) function. Once the rectangles are added to those contours, text giving the x

and y centroid coordinates and marker representing the centroid is included too. For cv2.putText() function font is FONT_HERSHEY_SIMPLEX [24] . Function cv2.drawMarker() is used to draw marker [25]. Once all the above operations are completed, the filtered contours centroids are saved in the the list of “cxx” and “cyy”.

All steps done so far will be combined together in the code below.

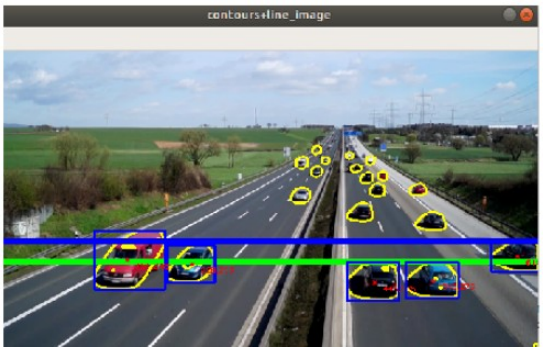
```
import numpy as np
import cv2
fgbg = cv2.createBackgroundSubtractorMOG2()
cap = cv2.VideoCapture('traffic3.mp4')
fps = cap.get(cv2.CAP_PROP_FPS)
while True:
    check, frame = cap.read()
    scale_percent = 50
    #calculate the 50 percent of original dimensions
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dsize = (width, height)
    image = cv2.resize(frame,dsize)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    fgmask = fgbg.apply(gray)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    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)
    contours, hierarchy = cv2.findContours(bins, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    hull = [cv2.convexHull(c) for c in contours]
    cv2.drawContours(image, hull, -1, (0,255,255), 2)
    blue_line = 225
    cv2.line(image, (0, blue_line), (width, blue_line), (255,0,0), 5)
    green_line = 250
    cv2.line(image, (0, green_line), (width, green_line), (0, 255, 0), 5)
    # min area for contours in case a bunch of small noise contours are created
    minarea = 300
    # max area for contours, can be quite large for buses
    maxarea = 50000
    # vectors for the x and y locations of contour centroids in current frame
    cxx = np.zeros(len(contours))
    cyy = np.zeros(len(contours))
    for i in range(len(contours)):
        if hierarchy[0, i, 3] == -1:
            area = cv2.contourArea(contours[i])
            if minarea < area < maxarea:
                # calculating centroids of contours
                cnt = contours[i]
                M = cv2.moments(cnt)
                cx = int(M['m10'] / M['m00'])
                cy = int(M['m01'] / M['m00'])
                if cy > blue_line:
                    x, y, w, h = cv2.boundingRect(cnt)
                    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
                    cv2.putText(image, str(cx) + "," + str(cy), (cx + 10, cy + 10), cv2.FONT_HERSHEY_SIMPLEX,.3, (0, 0, 255),
1)
                    cv2.drawMarker(image, (cx, cy), (0, 0, 255), cv2.MARKER_STAR, markerSize=5,
thickness=1,line_type=cv2.LINE_AA)
```

```
                cxx[i] = cx
                cyy[i] = cy

    cxx = cxx[cxx != 0]
    cyy = cyy[cyy != 0]

    cv2.imshow("contours+line_image", image)
    key = cv2.waitKey(int(1000 / fps)) & 0xFF
    if key == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

And the resulting video from the code above produces following result:



Once we get the centroids of the vehicles we can use it in any way to count the vehicles. I will be using the same steps above to transform “traffic4.mp4” video morphologically using opencv and then use the centroids to count the vehicles. In the other video I will be counting the vehicles which go straight and also other vehicles which joins the straight road from a turn.

The entire code written for counting the vehicles for “Traffic4.mp4” is divided into 6 parts and is in Appendix A of this report.
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 5 parts are displayed on the frame.

Note: The codes and explanation below are for the video “traffic4.mp4”

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.

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.

3 rd part:

This part will execute only for frames with vehicles in them, i.e if centroids are captured from 2 nd part of the code otherwise the 4 th part will execute skipping the 3 rd 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 16 th frame of the video, then its centroid will be stored in dataframe at 0 th column and 16 th 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.

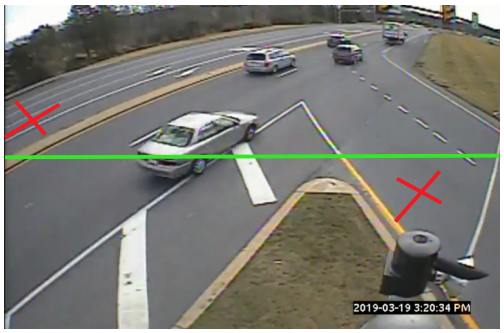
4th part:
In this part the total number of vehicles in the current frame is added to the “ currentcars” variable and index of such vehicles is appended into the “currentcarsindex” variable. Same process is conducted for the vehicles joining the straight road from the turn.

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 0 th , 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 0 th 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.
(To view the code of mentioned line number visit the script in the github link given in appendix B)

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 onefor 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.
(To view the code of mentioned line number visit the script in the github link given in appendix B)

Issues and challeges:

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. This issue is represented in the image below where the red cross marks show the possibility of unwanted vehicles getting counted.



Further, if we observe in the end result video "traffic_count.avi" 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 1 st 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.

Limitations:

The code for this project is limited to the video "traffic4.mp4" alone.
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 morphological transformation and in turn will affect 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 as I have added cv2.destroyAllWindows() or

cv2.destroyWindow(“window name”) command in the code. This works well through shell but the screen freezes when using notebook.[3]

Data provenance:

The data for this project is the video, which we received through Ms.Lindsey Klein. She is the president of Imperial Traffic and Data Collection. Her organization specializes in traffic data collection specifically vehicle counts. The location of the video is not provided.

Data Integrity:

The data for this project is not a csv file, so there is no issues of missing or incorrect data. Therefore, the integrity of the data is not an issue in this case.

Data Veracity:

Data veracity is not an issue here, since the data to be analyzed is in video format. The video is real and does not include any graphic elements affecting the count of the vehicles.

Conclusion:

The code as in Appendix A counts 24 vehicles out of 25 correctly those of which are traveling straight and are counting all 4 vehicles joining the straight road from the turn giving accuracy of 96.55% .
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.

Appendix A:

```
# the code is divided into 6 parts 3 common parts and 3 seperate parts for vehicles going straight
# and vehicles joining the straight road from a turn.
#-----------------------------------------------------------------------------------------------
# 1st part

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
       # criterias
        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 for vehicles continuing straight

        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
                        oldcxcy = df.iloc[int(framenumber - 1)][str(carids[j])]# acquires centroid from previous frame for specific
carid
                        curcxcy = [cxx[i], cyy[i]]# acquires current frame centroid that doesn't necessarily line up with previous
frame centroid
                        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
                    correctindextrue = np.argmin(sumsum) # choosing the minimum value from array above and storing index
number


                    # 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 mindx < maxrad and mindy < maxrad: # if the difference is less than the maximum radius than its the same
car
```

```
                    #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

#-------------------------------------------------------------------------------------
# 3rd part for vehicles joining the straight road from the turn.
# The code below is same as above with change in variables names
        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 for vehicles continuing straight

    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

#-------------------------------------------------------------------------------------
# 4th part for vehicles joining the straight road from the turn.

    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 for vehicles continuing straight
    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
   # Top left hand corner on-screen text
    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()
```

Appendix B:

1. The entire code to count vehicles from “traffic4.mp4” is in following link:
https://github.com/bhavyaramgiri/Traffic-count/blob/master/Vehicle_count.py

2. The link to view the jupyter notebook for opencv trials
https://github.com/bhavyaramgiri/Traffic-count/blob/master/Opencv_trials.ipynb

3. The link to view the video used in opencv trials
https://www.youtube.com/watch?v=dTdsjKRyMuU

4. The link to see the actual video of this code “traffic4.mp4”
https://youtu.be/vgvQIanWWB4

5. The link for “turn_issues.avi”
https://youtu.be/H6dyVp9hBlE

6. The output video of counting of vehicle of this project
https://youtu.be/-FK4wy6Tp10

References:

1. Code referred for this project : http://jorgemoreno.xyz/pycvtraffic.php
2. Data from sky website : http://datafromsky.com/
3. Jupyter notebook crashes with opencv : https://medium.com/@mrdatainsight/how-to-use-opencv-imshow-in-a-jupyter-notebook-quick-tip-ce83fa32b5ad
4. video writer : https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#videowriter-videowriter
5. fourcc: https://en.wikipedia.org/wiki/FourCC
6. MJPG : https://www.fourcc.org/mjpg/
7. mp4 : https://stackoverflow.com/questions/30103077/what-is-the-codec-for-mp4-videos-in-python-opencv
8. MOG2 : https://docs.opencv.org/3.3.0/db/d5c/tutorial_py_bg_subtraction.html
9. The reason for choosing MOG2 : https://docs.opencv.org/3.3.0/db/d5c/tutorial_py_bg_subtraction.html
10. kernel : https://en.wikipedia.org/wiki/Kernel_(image_processing)
11. structural element used in the kernel : https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#gac2db39b56866583a95a5680313c314ad
12. ellipse kernel : https://www.mathworks.com/help/images/ref/strel.html
13. morphological transformations : https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html
14. binary threshold: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#additional-resources
15. https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=threshold#threshold
16. https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html#what-is-thresholding
17. Contours: https://docs.opencv.org/3.3.1/d4/d73/tutorial_py_contours_begin.html
18. Hierarchy : https://docs.opencv.org/trunk/d9/d8b/tutorial_py_contours_hierarchy.html
19. RETR_TREE : https://docs.opencv.org/trunk/d3/dc0/group__imgproc__shape.html#gga819779b9857cc2f8601e6526a3a5bc71ab10df56aed56c89a026580adc9431f58
20. Hull : https://docs.opencv.org/trunk/dd/d49/tutorial_py_contour_features.html
21. (B,G,R) color in opencv : https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/
22. Centroids : https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html
23. Drawing rectangle : https://docs.opencv.org/3.1.0/dc/da5/tutorial_py_drawing_functions.html
24. Text font : https://docs.opencv.org/3.1.0/d0/de1/group__core.html#ga0f9314ea6e35f99bb23f29567fc16e11
25. Marker line type: https://docs.opencv.org/3.4/d0/de1/group__core.html#gaf076ef45de481ac96e0ab3dc2c29a777