# University of Sri Jayewardenepura

## FACULTY OF GRADUATE STUIES

## COMPUTER SCIENCE DEPARTMENT


**PRESENTED BY:**

**NAME: H. G NUWAN INDIKA**

**REGISTRATION NUMBER:**

6037PS2021014

**Index Number:**

GS/COMP/091


**CSC 616 2.0 Image Processing and Computer Vision:**

**Assignment 01**

# 1 Contents

# 2 Requirements

Count the number of vehicles that got away from the main road

## 2.1 Application

if you could integrate a vehicle detection system in a traffic light camera, you could easily track several useful things simultaneously:

- How many vehicles are present at the traffic junction during the day?
- What time does the traffic build up?
- What kind of vehicles are traversing the junction (heavy vehicles, cars, etc.)?
- Is there a way to optimize the traffic and distribute it through a different street?

And so on. The applications are endless!

# 3 constraints

- vehicles need to turn to sideways
- got away from the main road

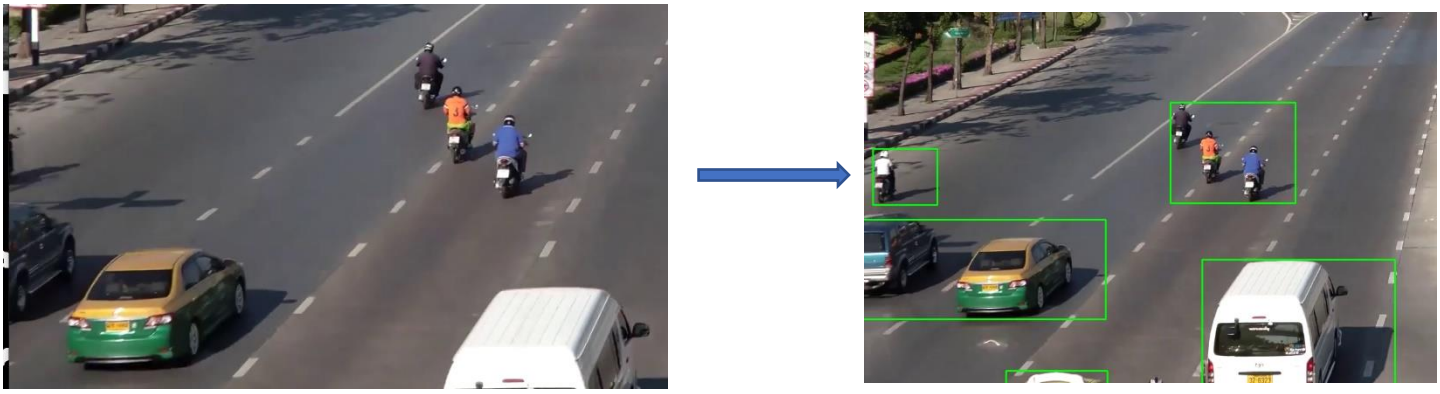
# 4 image processing pipeline

## 4.1 How it works

The vehicle counting system built is made up of three main components: a detector, tracker, and counter. The detector identifies vehicles in each frame of video and return an array of bounding boxes around the vehicles to the Trakker. The tracker uses the bounding boxes to track the vehicles in subsequent frames. The counter counts vehicles when they leave the frame or makes use of counting line drawn across a road.

## 4.2 Essential Concepts

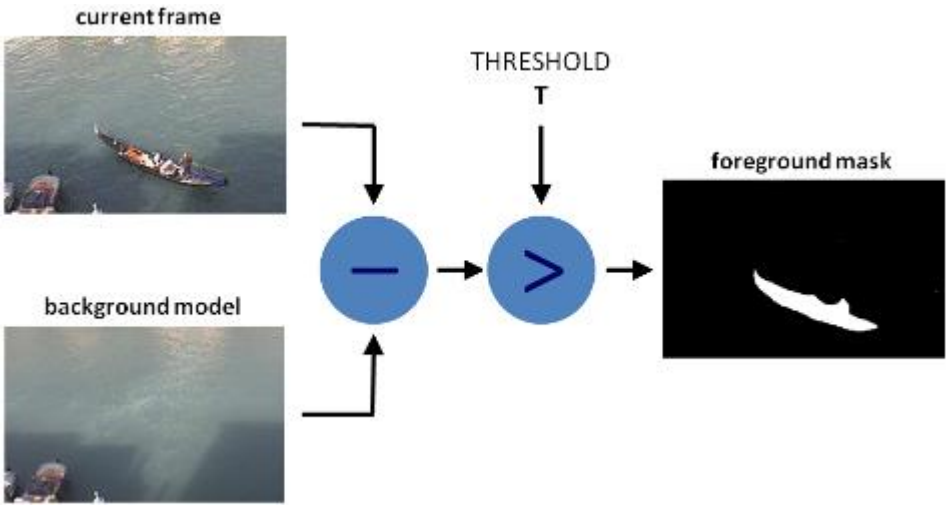
Our objective is to capture of the moving object and highlight that object in the video consider this frame from the video bellow



### 4.2.1 Background Subtraction Methods

Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.

As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model, containing the static part of the scene or, more in general, everything that can be considered as background given the characteristics of the observed scene.




#### *4.2.1.1 Our scenario*

Background modeling consists of two main steps:

1. Background Initialization.
2. Background Update.

In the first step, an initial model of the background is computed, while in the second step that model is updated to adapt to possible changes in the scene.

### 4.2.2 Image Dilation

Morphology is a set of image processing operations that process images based on predefined structuring elements known also as kernels. The value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. By choosing the size and shape of the kernel, you can construct a morphological operation that is sensitive to specific shapes regarding the input image.

Two of the most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of the object in an image, while erosion does exactly the opposite. The amount of pixels added or removed, respectively depends on the size and shape of the structuring element used to process the image. In general the rules followed from these two operations have as follows:

Erosion and Dilation are morphological image processing operations. OpenCV morphological image processing is a procedure for modifying the geometric structure in the image. In morphism, we find the shape and size or structure of an object. Both operations are defined for binary images, but we can also use them on a grayscale image. These are widely used in the following way:

- Removing Noise
- Identify intensity bumps or holes in the picture.
- Isolation of individual elements and joining disparate elements in image.

Dilation

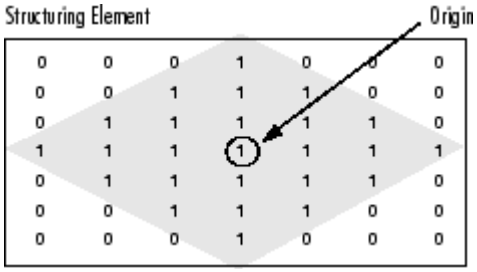
Dilation is a technique where we expand the image. It adds the number of pixels to the boundaries of objects in an image. The structuring element controls it. The structuring element is a matrix of 1's and 0's.

### 4.2.3 The concept of “structuring elements”

As it can be seen above and in general in any morphological operation the structuring element used to probe the input image, is the most important part.

A structuring element is a matrix consisting of only 0's and 1's that can have any arbitrary shape and size. Typically are much smaller than the image being processed, while the pixels with values of 1 define the neighborhood. The center pixel of the structuring element, called the origin, identifies the pixel of interest – the pixel being processed.

For example, the following illustrates a diamond-shaped structuring element of 7x7 size.



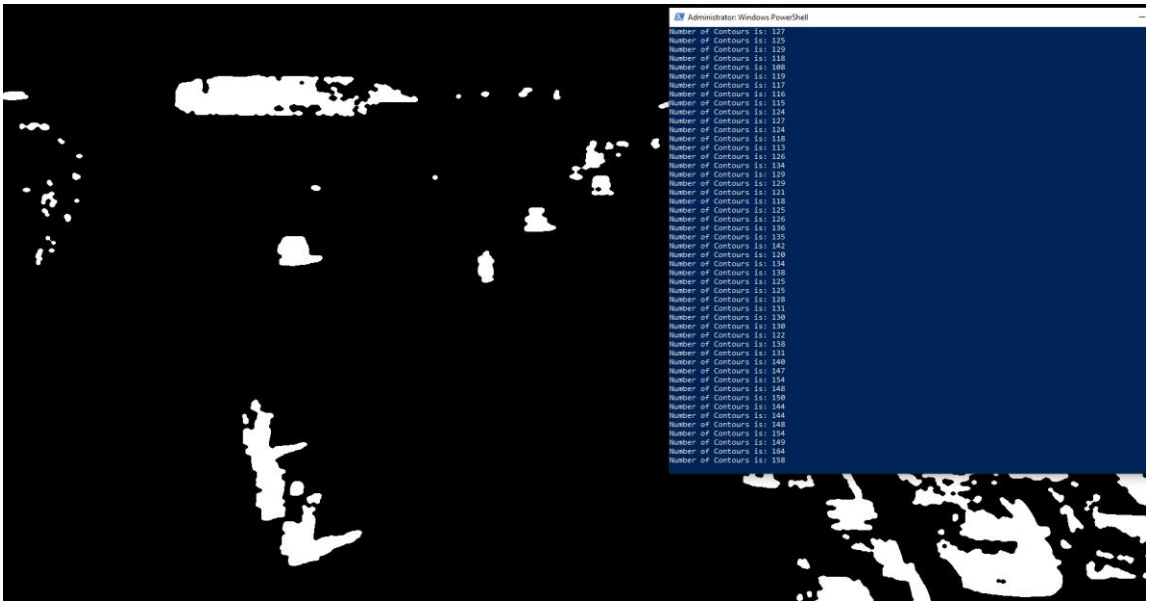

A Diamond-Shaped Structuring Element and its Origin

A structuring element can have many common shapes, such as lines, diamonds, disks, periodic lines, and circles and sizes. You typically choose a structuring element the same size and shape as the objects you want to process/extract in the input image.

### 4.2.4 Finding Contours

The contours are used to identify the shape of an area in the image having the same color or intensity. Contours are like boundaries around regions of interest. So, if we apply contours on the image after the thresholding step,

```
    dilat = cv2.dilate(img_sub,np.ones((5,5)))
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    dilate = cv2.morphologyEx (dilat, cv2. MORPH_OPEN , kernel)
    dilate = cv2.morphologyEx (dilate, cv2. MORPH_CLOSE , kernel)
    side,h=cv2.findContours(dilate,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    # Find Number of contours
    print("Number of Contours is: " + str(len(side)))
```




The white regions have been surrounded by grayish boundaries which are nothing but contours. We can easily get the coordinates of these contours. This means we can get the locations of the highlighted regions.

Note that there are multiple highlighted regions, and each region is encircled by a contour. In our case, the contour having the maximum area is the desired region. Hence, it is better to have as few contours as possible.

In the image above, there are still some unnecessary fragments of the white region. There is still scope of improvement. The idea is to merge the nearby white regions to have fewer contours and for that, we can use another technique known as image dilation.

# 5 Build a Vehicle Detection System using OpenCV and Python

Python version is 3.9.6 and used the computer vision library OpenCV version is 4.5.2

## 5.1 Import Libraries

```
import cv2
import numpy as np
from time import sleep
```

## 5.2 pre – defined parmters

```
width_min=50 #Minimum width of the rectangle
height_min=50 #Minimum height of the rectangle

offset=1 #Allowable error between pixels

post_line=620 #Count Line Position

delay=60 #Video FPS

detec = [] #array store the frame detected
vehicales= 0 #count the detected vehicles
```

## 5.3 catch the center of the rectangel

```
def catch_center(x, y, w, h):
    x1 = int(w / 2)
    y1 = int(h / 2)
    cx = x + x1
    cy = y + y1
    return cx,cy
```

## 5.4 Import the Video

```
cap = cv2.VideoCapture('videoplayback1080.mp4')
```

## 5.5 adding Background Subtraction Methods

```
subtract = cv2.bgsegm.createBackgroundSubtractorMOG()
```

up to now code

```
import cv2
import numpy as np
from time import sleep

width_min=50 #Minimum width of the rectangle
height_min=50 #Minimum height of the rectangle

offset=1 #Allowable error between pixel

post_line=620 #Count Line Position
```

```
delay=60 #Video FPS

detec = [] #array store the frame detected
vehicales= 0 #count the detected vehicles


def catch_center(x, y, w, h):
    x1 = int(w / 2)
    y1 = int(h / 2)
    cx = x + x1
    cy = y + y1
    return cx,cy

cap = cv2.VideoCapture('videoplayback1080.mp4')
subtract = cv2.bgsegm.createBackgroundSubtractorMOG()
```

## 5.6 extract the frame from the imported video

after we import and add the Background Subtraction Methods we need to each frame until the video ended  for the we need to extract the frame from the imported video

```
ret , frame1 = cap.read()
```

This code initiates an infinite loop (to be broken later by a break statement), where we have ret and frame being defined as the cap.read(). Basically, ret is a boolean regarding whether or not there was a return at all, at the frame is each frame that is returned. If there is no frame, you wont get an error, you will get None.

```
while True:
    ret , frame1 = cap.read()
    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
```

## 5.7 calculator the tempo

```
tempo = float(1/delay)
sleep(tempo)
```

## 5.8 OpenCV reads colors as RGB

```
grey = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
```

Here, we define a new variable, gray, as the frame, converted to gray. Notice this says BGR2GRAY. It is important to note that OpenCV reads colors as BGR (Blue Green Red), where most computer applications read as RGB (Red Green Blue). Remember this.

```
if cv2.waitKey(1) == ord('q'):
        break
```

This statement just runs once per frame. Basically, if we get a key, and that key is a q, we will exit the while loop with a break, which then runs:

## 5.9 OpenCV Python Gaussian Blur

OpenCV provides cv2.gaussianblur() function to apply Gaussian Smoothing on the input source image

```
blur = cv2.GaussianBlur(grey,(3,3),5)
```

## 5.10 adding Dilation

```
dilat = cv2.dilate(img_sub,np.ones((5,5)))
```

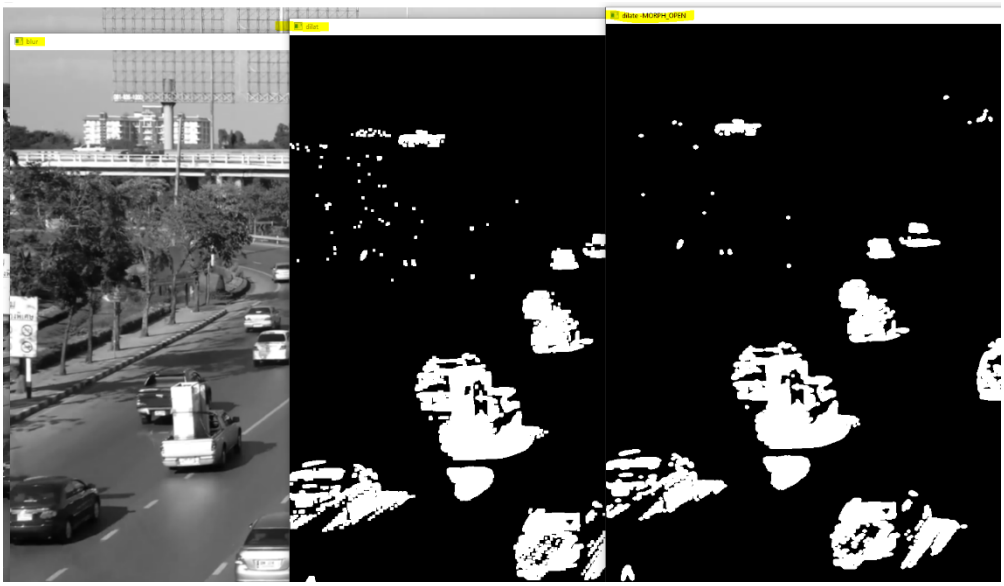
## 5.11 structuring element

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
```

## 5.12 Opening

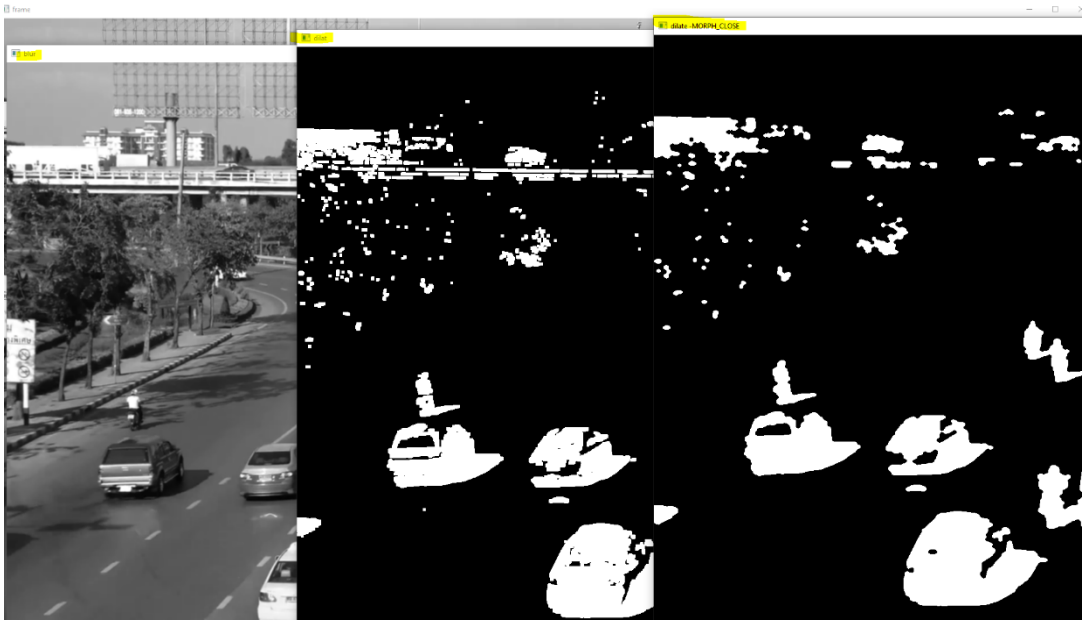
Opening is just another name of erosion followed by dilation. It is useful in removing noise, here we use the function cv2.morphologyEx

```
dilate = cv2.morphologyEx (dilat, cv2. MORPH_OPEN , kernel)
```



## 5.13 Closing

Closing is reverse of opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the



## 5.14 Find Contours

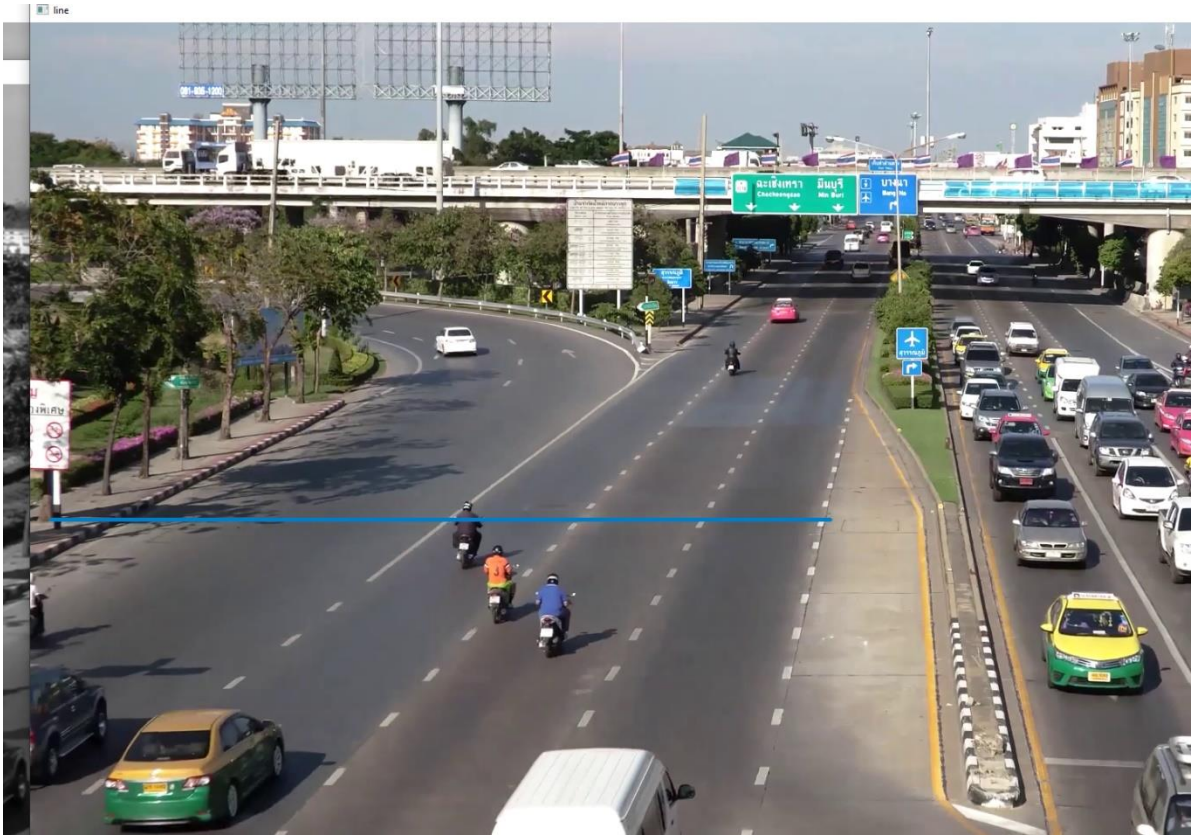
```
contours,h=cv2.findContours(dilate,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

there are three arguments in cv.findContours() function, first one is contour retrieval mode, second is contour approximation method. And it outputs a modified image, the `contours` and h(hierarchy). `contours` is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

## 5.15 Draw line

Draw line in '620' Line Position the current frame

```
cv2.line(frame1, (25, post_line), (1000, post_line), (200,127,0), 3)
```



## 5.16 contours of the image

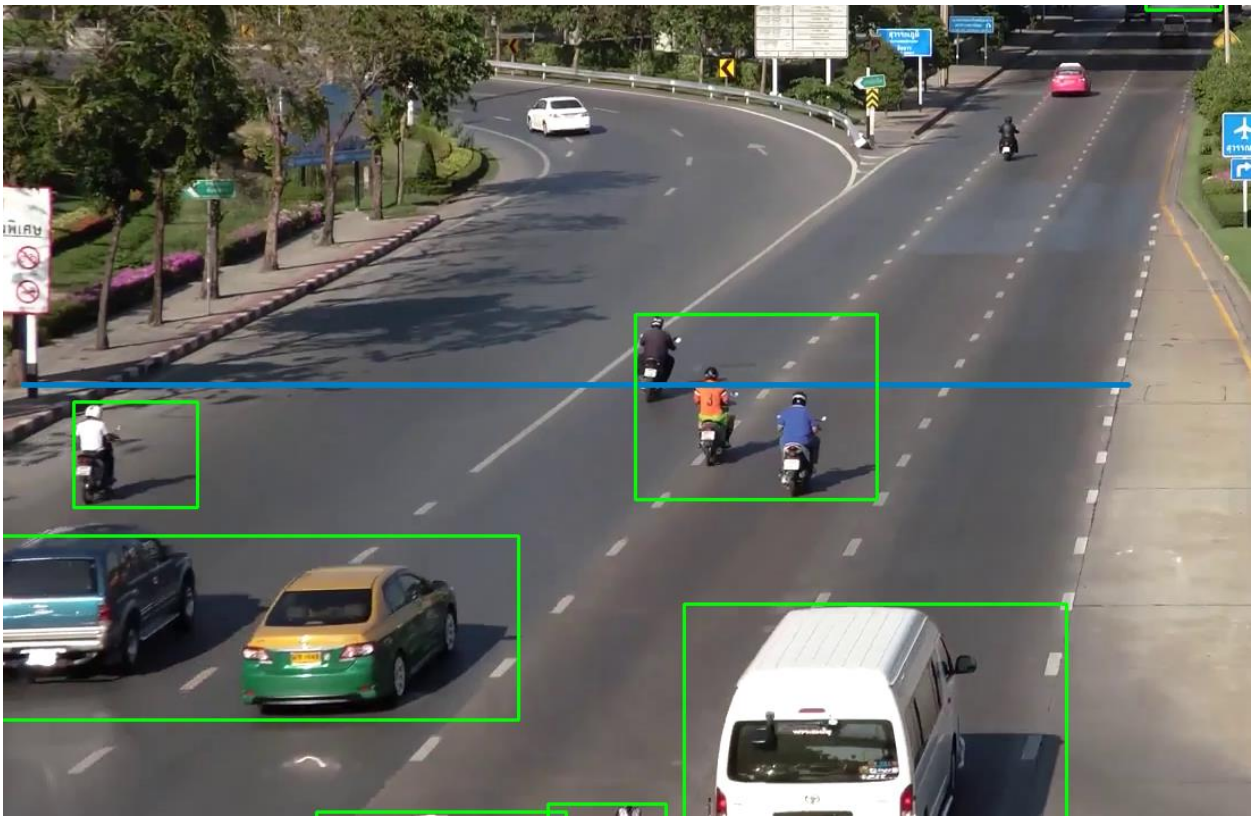
now we need go into each contours of the image for that we need us following for and validate the outline and also we need compute the bounding boxes of each contour, which is the starting (x,y,w,h) – coordinates of the bounding box followed by the width and height

```
for(i,c) in enumerate(contours):
        (x,y,w,h) = cv2.boundingRect(c)
        validate_outline = (w >= width_min) and (h >= height_min)
        if not validate_outline:
            continue
```

## 5.17 draw a rectangle with blue line borders of thickness of 2 px using `cv2.rectangle`

```
    for(i,c) in enumerate(contours):
        (x,y,w,h) = cv2.boundingRect(c)
        validate_outline = (w >= width_min) and (h >= height_min)
        if not validate_outline:
            continue

        cv2.rectangle(frame1,(x,y),(x+w,y+h),(0,255,0),2)
```



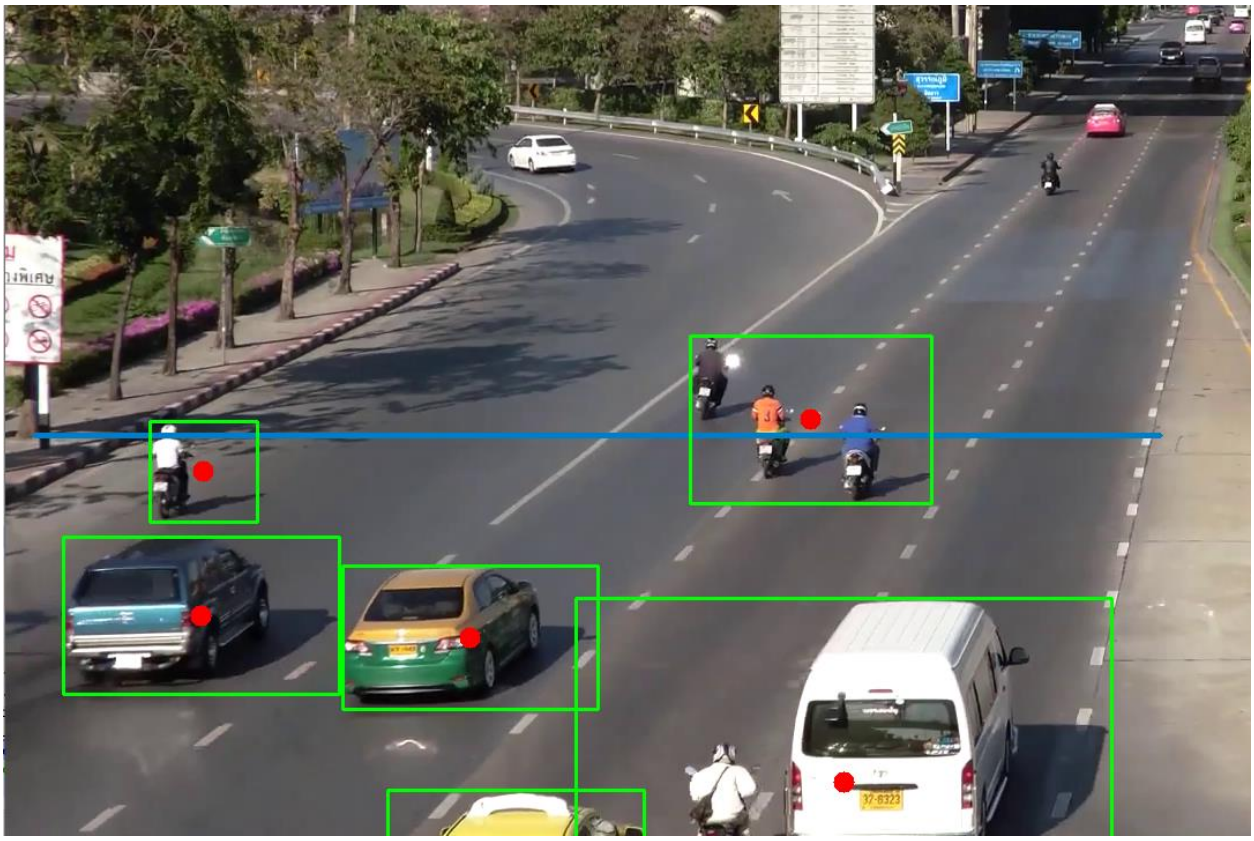
## 5.18 calculate the center of the current frame and draw a red 9px circle

### 5.18.1 calculate the center of the current frame

```
center = catch_center(x, y, w, h)
detec.append(center)
```

### 5.18.2 draw a red 9px circle

```
cv2.circle(frame1, center, 9, (0, 0,255), -1)
```



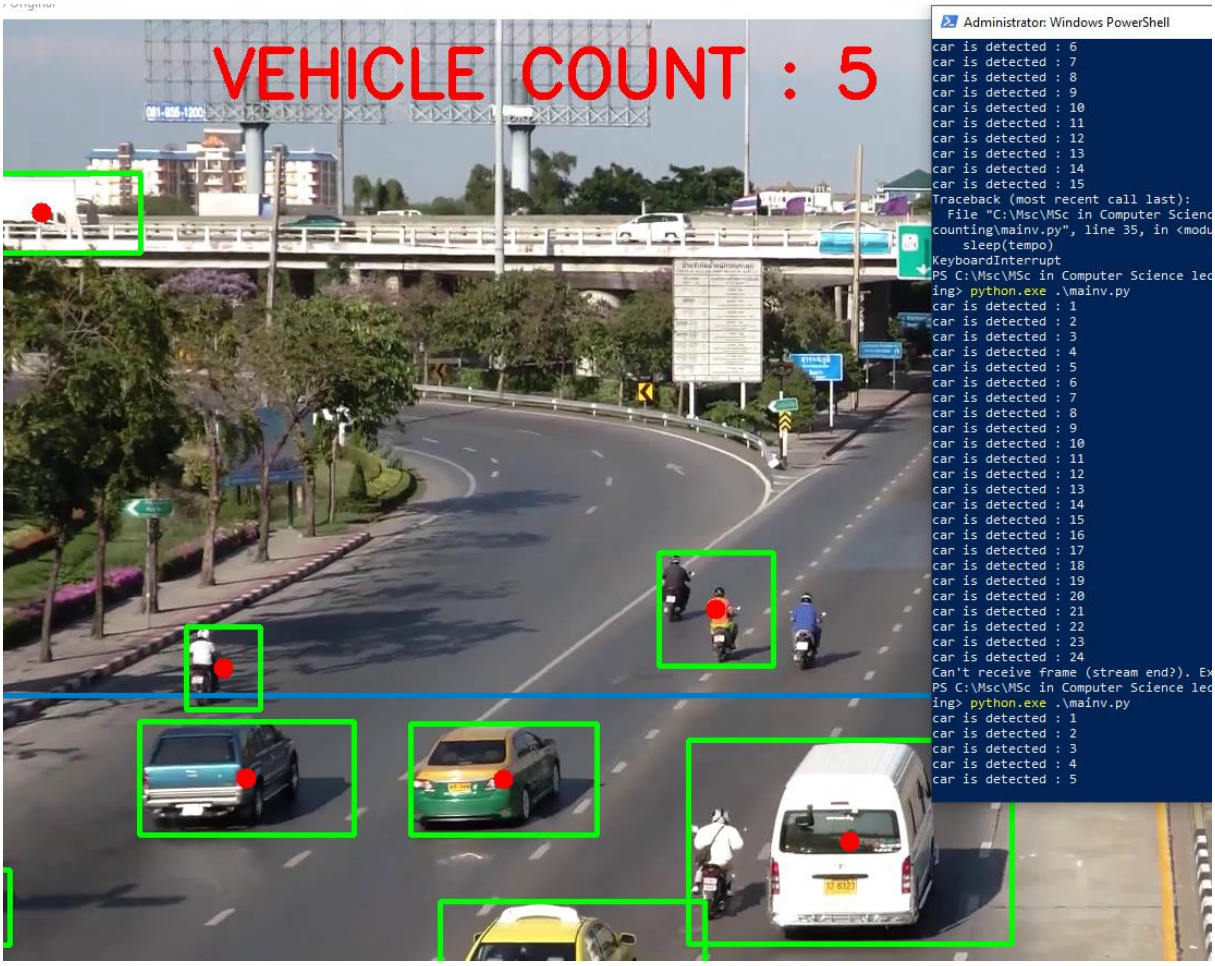
## 5.19 detect the moving object

now we detect the moving object of the current frame next we next to calculate the object that pass the blue line

```
        for (x,y) in detec:
            if y<(post_line+offset) and y>(post_line-offset):
                vehicales+=1
                cv2.line(frame1, (25, post_line), (1000, post_line), (0,127,255), 3)
                detec.remove((x,y))
                print("car is detected : "+str(vehicales))
```

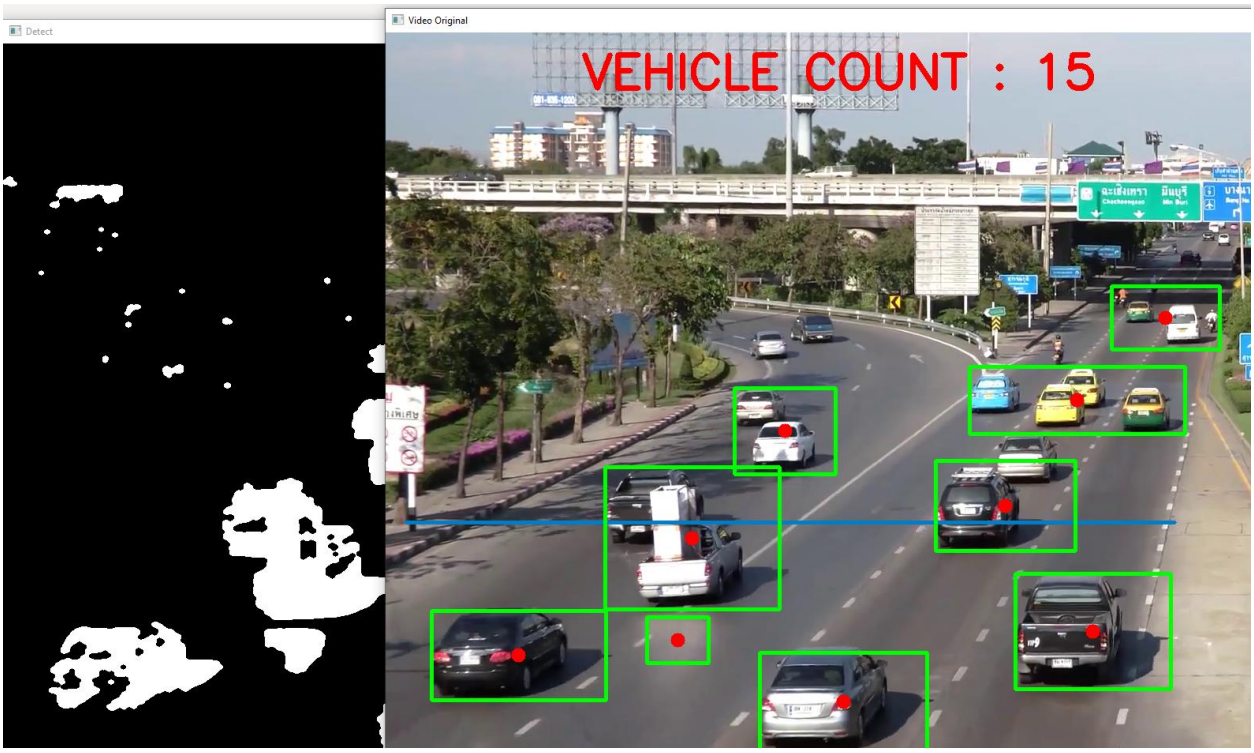
## 5.20 then we print it in the video

```
  cv2.putText(frame1, "VEHICLE COUNT : "+str(vehicales), (250, 70), cv2.FONT_HERSHEY_SIMPLEX, 2, (0,
0, 255),5)
  cv2.imshow("Video Original" , frame1)
```



## 5.21 print the Morphology is a set of image processing operations

```
cv2.imshow("Detect",dilate)
```




## 5.22 when the escape but press operation will stop

```
if cv2.waitKey(1) == 27:
        break
```

## 5.23 closes all of the imshow() windows then releases the video

```
cv2.destroyAllWindows()
cap.release()
```

# 6 final code

```
import cv2
import numpy as np
from time import sleep

width_min=50 #Minimum width of the rectangle
height_min=50 #Minimum height of the rectangle

offset=1 #Allowable error between pixel

post_line=620 #Count Line Position

delay=60 #Video FPS

detec = [] #array store the frame detected
vehicales= 0 #count the detected vehicles


def catch_center(x, y, w, h):
    x1 = int(w / 2)
    y1 = int(h / 2)
    cx = x + x1
    cy = y + y1
    return cx,cy

cap = cv2.VideoCapture('videoplayback1080.mp4')
subtract = cv2.bgsegm.createBackgroundSubtractorMOG()

while True:
    ret , frame1 = cap.read()
    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    tempo = float(1/delay)
    sleep(tempo)
    grey = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
    if cv2.waitKey(1) == ord('q'):
        break
    blur = cv2.GaussianBlur(grey,(3,3),5)
    img_sub = subtract.apply(blur)
    dilat = cv2.dilate(img_sub,np.ones((5,5)))
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    dilate = cv2.morphologyEx (dilat, cv2. MORPH_OPEN , kernel)
    dilate = cv2.morphologyEx (dilate, cv2. MORPH_CLOSE , kernel)
    contours,h=cv2.findContours(dilate,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

    cv2.line(frame1, (25, post_line), (1000, post_line), (200,127,0), 3)
    for(i,c) in enumerate(contours):
        (x,y,w,h) = cv2.boundingRect(c)
        validate_outline = (w >= width_min) and (h >= height_min)
        if not validate_outline:
            continue

        cv2.rectangle(frame1,(x,y),(x+w,y+h),(0,255,0),3)
        center = catch_center(x, y, w, h)
        detec.append(center)
        cv2.circle(frame1, center, 9, (0, 0,255), -1)

        for (x,y) in detec:
            if y<(post_line+offset) and y>(post_line-offset):
                vehicales+=1
                cv2.line(frame1, (25, post_line), (1000, post_line), (0,127,255), 3)
```

```
                detec.remove((x,y))
                print("car is detected : "+str(vehicales))

    cv2.putText(frame1, "VEHICLE COUNT : "+str(vehicales), (250, 70), cv2.FONT_HERSHEY_SIMPLEX, 2,
(0, 0, 255),5)
    cv2.imshow("Video Original" , frame1)
    cv2.imshow("Detect",dilate)

    if cv2.waitKey(1) == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

# 7 Evaluation

It is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It was introduced in the paper "An improved adaptive background mixture model for real-time tracking with shadow detection" by P. KadewTraKuPong and R. Bowden in 2001. It uses a method to model each background pixel by a mixture of K Gaussian distributions (K = 3 to 5). The weights of the mixture represent the time proportions that those colors stay in the scene. The probable background colors are the ones which stay longer and more static.

While coding, we need to create a background object using the function, cv.bgsegm.createBackgroundSubtractorMOG(). It has some optional parameters like length of history, number of gaussian mixtures, threshold etc. It is all set to some default values.

This method can use in limited computer and fewer storage resources. For future work and to overcome the drawbacks of this system. I'm hoped to improve the accuracy of the object detection and counting.

# 8 Reference


[1] Vehicle Detection and Counting Method Based on Digital Image Processing in Python Reha Justin1, Dr. Ravindra Kumar2
1Intern, 2Principal Scientist, CSIR-Central Road Research Institute, Transportation Planning Division Delhi, India
reha.justin96@gmail.com, 2ravindra261274@gmail.com

[2]I built a video-based vehicle counting system — here's how - https://alphacoder.xyz/vehicle-counting/

[3]Loading Video Source OpenCV Python Tutorial - https://pythonprogramming.net/loading-video-python-opencv-tutorial/

[4]Object detection and tracking in PyTorch - https://towardsdatascience.com/object-detection-and-tracking-in-pytorch-b3cf1a696a98

[5]Sheng-Yi Chiu 1,* ID , Chung-Cheng Chiu 1 and Sendren Sheng-Dong Xu 2 A Background Subtraction Algorithm in Complex Environments Based on Category Entropy Analysis

[6] opencv documentation python - https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

[7]Practical Image Process with OpenCV - https://towardsdatascience.com/practical-image-process-with-opencv-8405772c603e

[8]An Evaluation of Background Subtraction for
Object Detection Vis-a-Vis Mitigating
Challenging Scenarios
SUMAN KUMAR CHOUDHURY, PANKAJ KUMAR SA, (Member, IEEE),
SAMBIT BAKSHI, (Member, IEEE), AND BANSHIDHAR MAJHI, (Member, IEEE)

[9]Research on Vehicle Detection and Tracking Algorithm Based on the Methods of Frame Difference and Adaptive Background Subtraction Difference
Yiqin Cao*, Xiao Yun, Tao Zhong and Xiaosheng Huang
School of Software, East China Jiaotong University, Nanchang, 330013, China
*Corresponding author