Chap 1. INTRODUCTION

In the era of technology, the number of vehicles on the road is increasing. Besides that, with the advance technology, the vehicle manufacturers produce the vehicle with the best quality. At the same time also produce the vehicle that can move faster and faster by enhance the vehicle engine and increase the maximum speed of the vehicle. Due to this, the number of cases of accident happens on highway also increasing dramatically. So, there is a need to have a low-cost vehicle speed detector system.

However, the equipment to use this method is costly. So, finding other equipment to reduce the cost is necessary. Image processing technology can serve this well. Object recognition can use to identify and track the vehicle. The task of finding the target object in an image or video sequence can be done by object recognition. Image processing technology does not require any special hardware. It is based on the software component. With a typical video recorder and a computer is enough to create a vehicle speed detection system.

Video surveillance is a very popular research topic in computer vision applications that continuously tries to detect, recognize and track the targets. Every tracking method requires a detection method in every single frame. Object tracking is the process of following one or more objects that found on detection process using camera. Background is the most common method used for simple object tracker.

1.1 EXISTING SYSTEM

The government have placed traffic police, so that they can keep a track on the speed limit of a vehicle, whether a person is following traffic rule or not. This method is quiet time consuming and also requires a lot of man power which is not efficient.

Fig:1.1 Speed camera.

Currently to capture and detect the speed of the moving vehicle speed camera are used in various parts of country including India. It is used for recording fast-moving objects like photographic images onto a storage medium.

1.2 PROPOSED SYSTEM

In the proposed system, the speed of a vehicle is detected with the help of a camera, OpenCV software is used for this. It captures the image and give the speed of the vehicle. With the help of this technique it is very easy to trace down the over speeding vehicle. It will store the image of the Offenders in the folder who have crossed the speed limit. Hence this system is implemented to eliminate the manpower involved and make the system fully automated.

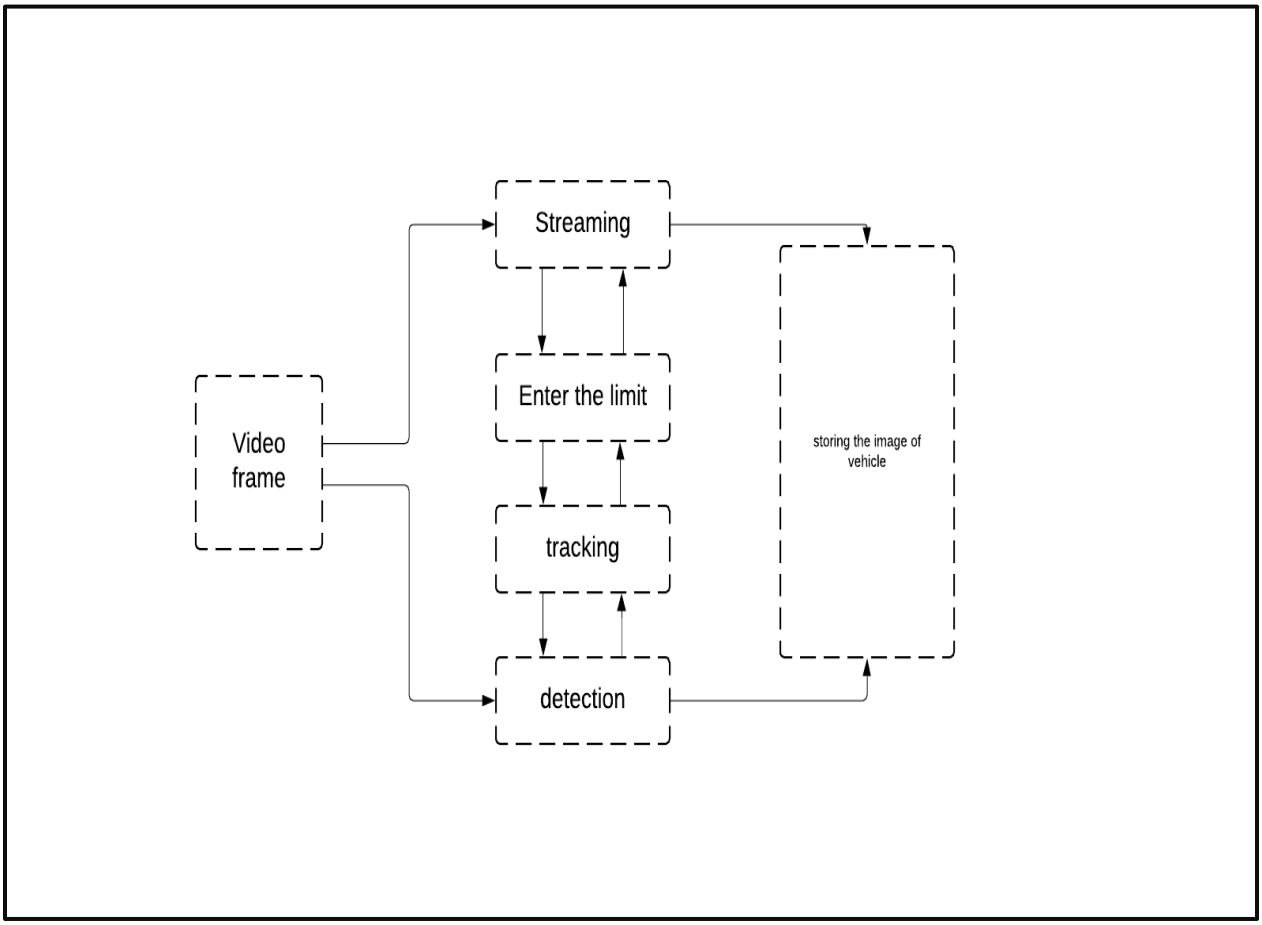


Fig :1.2 Working module of speed detection using open CV

Chap 2. PROPOSED SYSTEM

2.1FEATURES

1)Captures speed in real time:

This work as an advantage as speed of the vehicle moving on the road is captured as the vehicle passes the speed limit.

2) Image subtraction:

This software makes use of image subtraction which is also known as pixel subtraction whereby the digital numeric value of one pixel or whole image is subtracted from another image.

3)Performance:

This software is more efficient when it comes to the performance of the software.

4)Reduced manpower as cameras are involved:

With help of this technique human based work is reduced resulting in more accuracy with the help of camera.

5)Easy to use:

This is a user-friendly software that can be used by anyone. Only regular maintenance is required to ensure whether the software is working correctly.

6)Highly accurate:

The software developed is proven to be accurate for detecting and capturing the speed of vehicle.

Chap 3. TECHNIQUE USED

3.1 BACKGROUND SUBTRACTION

Background subtraction is a major preprocessing step in many vision-based applications. For example, consider the case of a visitor counter where a static camera takes the number of visitors entering or leaving the room, or a traffic camera extracting information about the vehicles etc. In all these cases, first you need to extract the person or vehicles alone. Technically, you need to extract the moving foreground from static background.

If you have an image of background alone, like an image of the room without visitors, image of the road without vehicles etc. it is an easy job. Just subtract the new image from the background. You get the foreground objects alone. But in most of the cases, you may not have such an image, so we need to extract the background from whatever images we have. It becomes more complicated when there are shadows of the vehicles. Since shadows also move, simple subtraction will mark that also as foreground. This project has implemented three such algorithms which are very easy to use.

Fig:3.1 Example of background subtraction.

In the above fig: 3.1, shows the example of background subtraction, where the left side of image is original image and right side of the image is the result that is obtained after background subtraction takes place.

3.1.1 BACKGROUND SUBTRACTOR_MOG2

It is also a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. One important feature of this algorithm is that it selects the appropriate number of gaussian distribution for each pixel. It provides better adaptability to varying scenes due illumination changes etc. It detects and marks shadows, but decreases the speed. Shadows will be marked in gray color.

Original Frame:

Original image

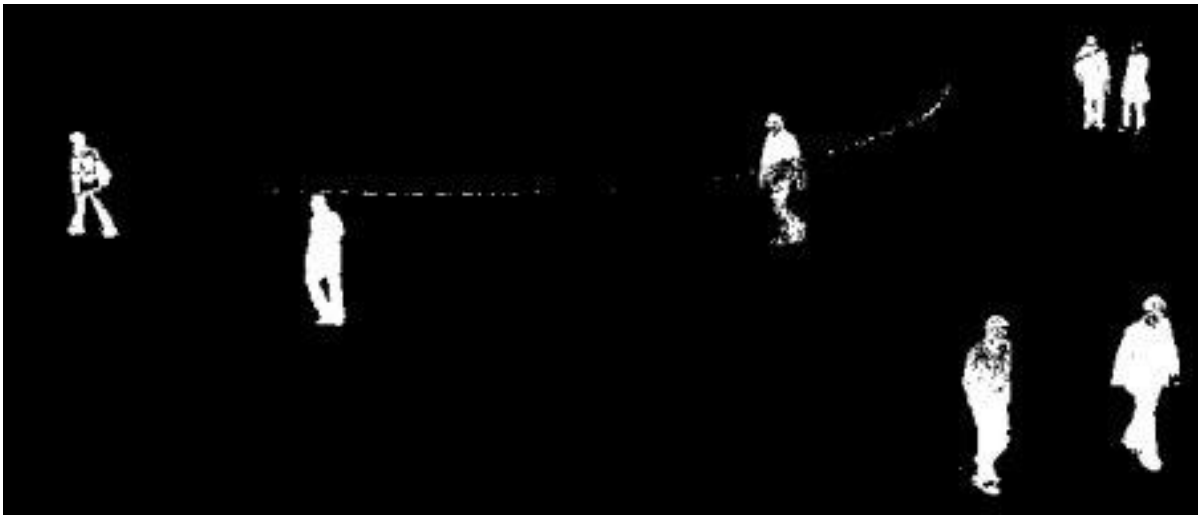
Gray color region shows shadow region

Fig:3.1.1 Example of background subtractor MOG2

3.1.2 BACKGROUND SUBTRACTOR_MOG

It is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. 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.createBackgroundSubtractorMOG().

Original image



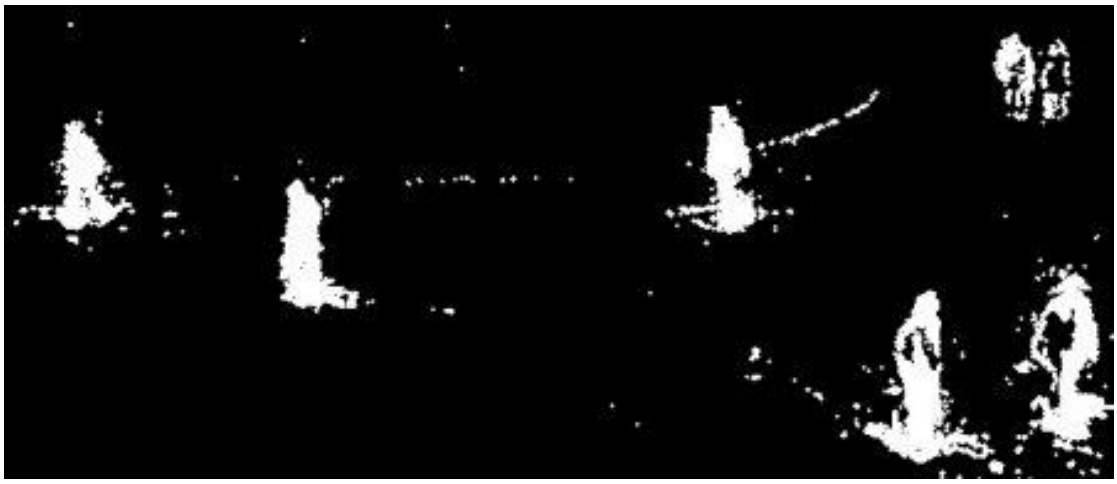
Result of background subtractor MOG2

Fig:3.1.2 Example of background subtractor MOG2

3.1.3 BACKGROUND SUBTRACTOR_GMG

This algorithm combines statistical background image estimation and per-pixel Bayesian segmentation. It uses first few (120 by default) frames for background modelling. It employs probabilistic foreground segmentation algorithm that identifies possible foreground objects using Bayesian inference. The estimates are adaptive; newer observations are more heavily weighted than old observations to accommodate variable illumination. Several morphological filtering operations like closing and opening are done to remove unwanted noise. You will get a black window during first few frames.

Original image



Noise is removed with morphological opening.

Fig:3.1.3 Example of background subtractor GMG

Create the background subtraction object:

```
method = 1 if method == 0:

bgSubtractor = cv2.bgsegm. createBackgroundSubtractorMOG ()

elif method == 1:

bgSubtractor = cv2.createBackgroundSubtractorMOG2()

else:

bgSubtractor = cv2.bgsegm. createBackgroundSubtractorGMG ()
```

Note: This coding is used in the project for implementing background subtraction.

3.2 MORPHING TECHNIQUES

3.2.1 MORPHOLOGICAL OPENING

In mathematical morphology, opening is the dilation of the erosion of a set A by a structuring element B = $A \bullet B = (A \oplus B) \ominus B,$

Together with closing, the opening serves in computer vision and image processing as a basic workhorse of morphological noise removal. Opening removes small objects from the foreground (usually taken as the bright pixels) of an image, placing them in the background, while closing removes small holes in the foreground, changing small islands of background into foreground. These techniques can also be used to find specific shapes in an image. Opening can be used to find things into which a specific structuring element can fit (edges, corners, ...).

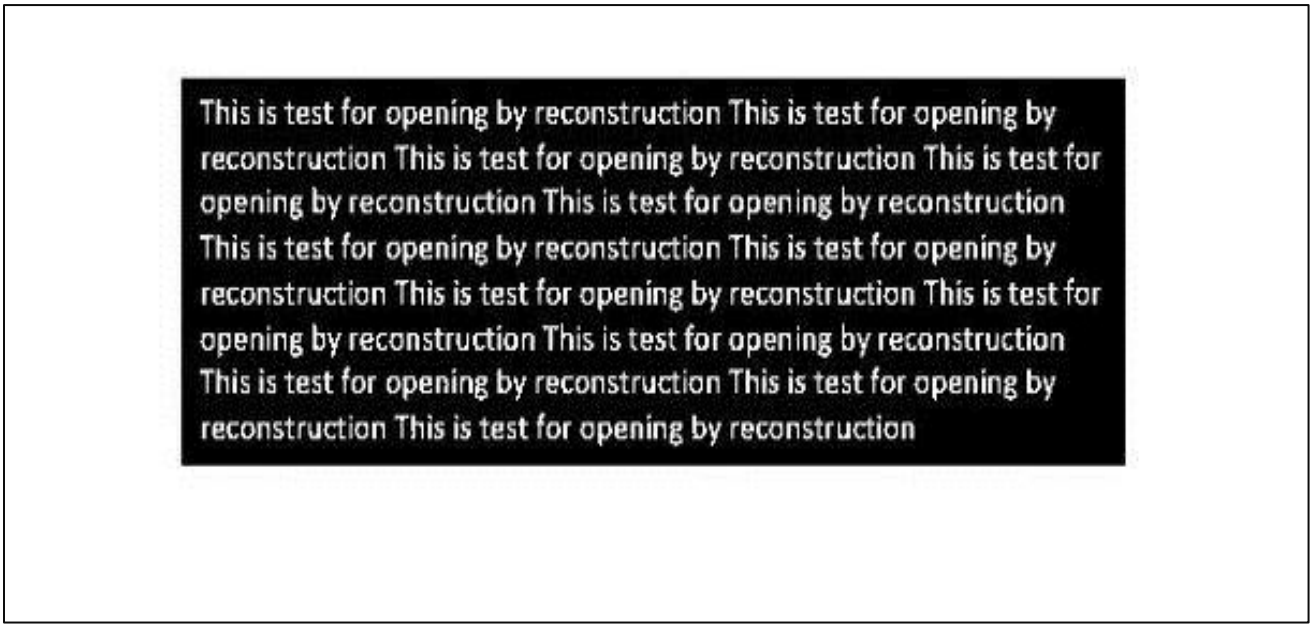
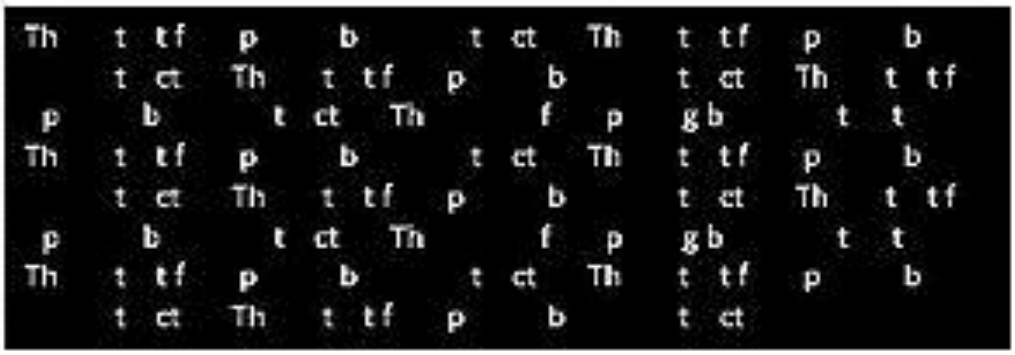


Fig:3.2.1 Original image for opening by reconstruction

Marker image



Result of opening by reconstruction

Fig:3.2.1 Example of Morphological Opening.

3.2.2 MORPHOLOGICAL CLOSING

In mathematical morphology, the **closing** of a set (binary image) *A* by a structuring element *B* is the erosion of the dilation of that set, In image processing, closing is, together with opening, the basic workhorse of morphological noise removal. Opening removes small objects, while closing removes small holes.

$$A \bullet B = (A \oplus B) \ominus B,$$

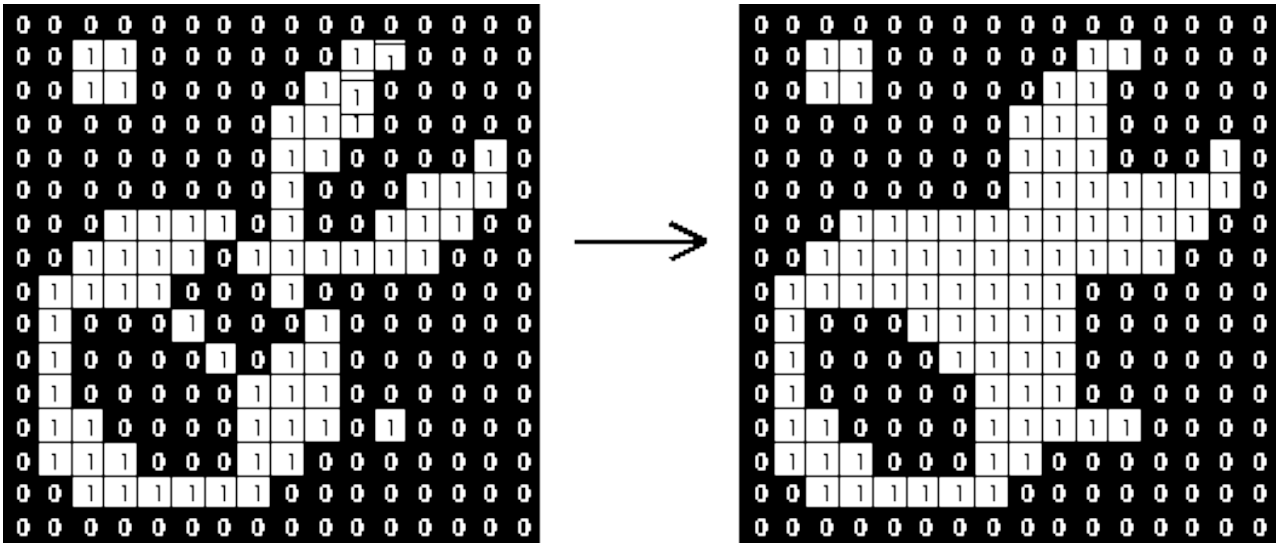


Fig:3.2.2. Example of morphological closing

3.3 IMAGE SEGMENTATION

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.[1][2] Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic.

Application:

Object detection

- Pedestrian detection
- Face detection
- Brake light detection
- Locate objects in satellite images (roads, forests, crops, etc.)

Traffic control systems

- Video surveillance
- Video Object Co-segmentation and action localization

a) Thresholding:

The simplest method of image segmentation is called the thresholding method. This method is based on a clip-level (or a threshold value) to turn a gray-scale image into a binary image.

The key of this method is to select the threshold value (or values when multiple-levels are selected). Several popular methods are used in industry including the maximum entropy method, balanced histogram thresholding, Otsu's method (maximum variance), and k-means clustering.

Some methods of thresholding used in application:

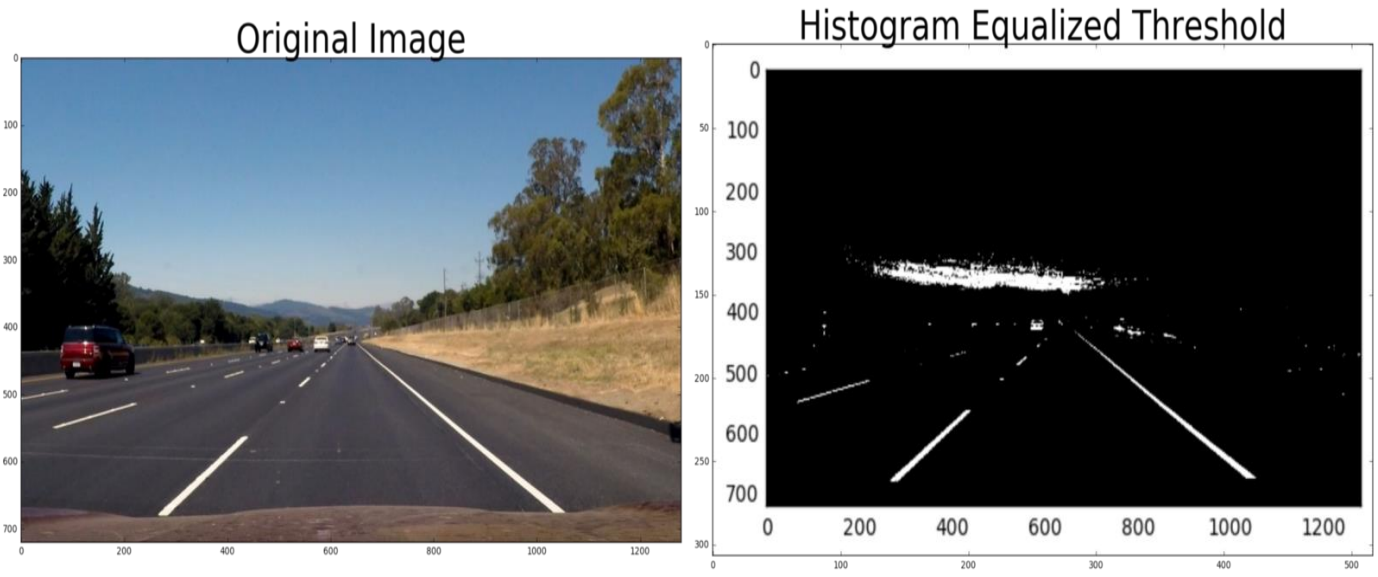


Fig: Example of Thresholding

In the above fig. the conversion of original image to histogram equalized threshold is shown.

b) Edge detection:

Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges.* Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction.

Fig: Example of edge detection.

The above fig. shows the actual image and the image that is obtained with edge detection.

Chap 4. SYSTEM ANALYSIS

4.1 MODULES

1. Road Tracking:

In this, road is divided into lanes to detect the vehicle is in which part. lane segmentation is used which detects the lane in two parts i.e.Lane_1 and Lane_2.

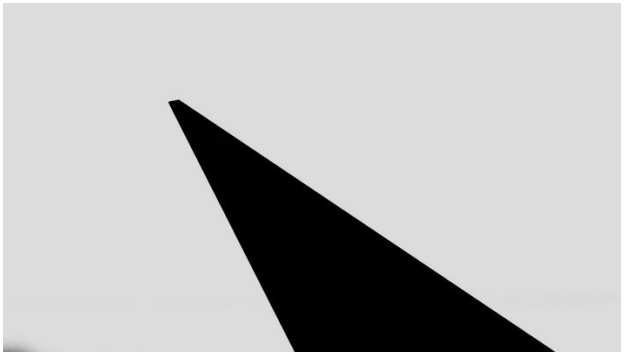


Fig lane_1

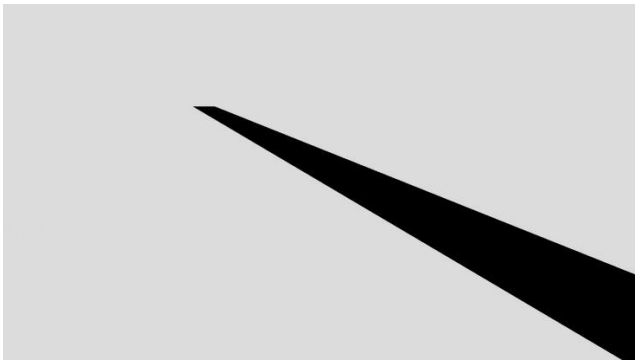


Fig. lane_2

2. Object Detection:

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

a) IMAGE ACQUISITION

The first stage of any vision system is the image acquisition stage. After the image has been obtained, various methods of processing can be applied. OpenCV gives the flexibility to capture image directly from a pre-recorded video stream, camera input feed, or a directory path.

Taking input from a directory path

mask1 = cv2.imread('m1.jpeg')

Capturing input from a video stream

cap = cv2.VideoCapture('test1.mp4')

This are some of the methods used for image acquisition used in the software. Below fig is the example of how image is acquired.

Fig: Example of Image Acquisition.

In the above fit the image is acquired from the road with the help of camera.

b) ROAD SEGMENTATION

OpenCV implemented a marker-based watershed algorithm where you specify which are all valley points are to be merged and which are not. It is an interactive image segmentation. What we do is to give different labels for our object we know. Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm.

Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

Create the kernel that will be used to remove the noise in the foreground

mask kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))

sure, background area

kernel_di = np. ones ((5, 1), np. uint8)

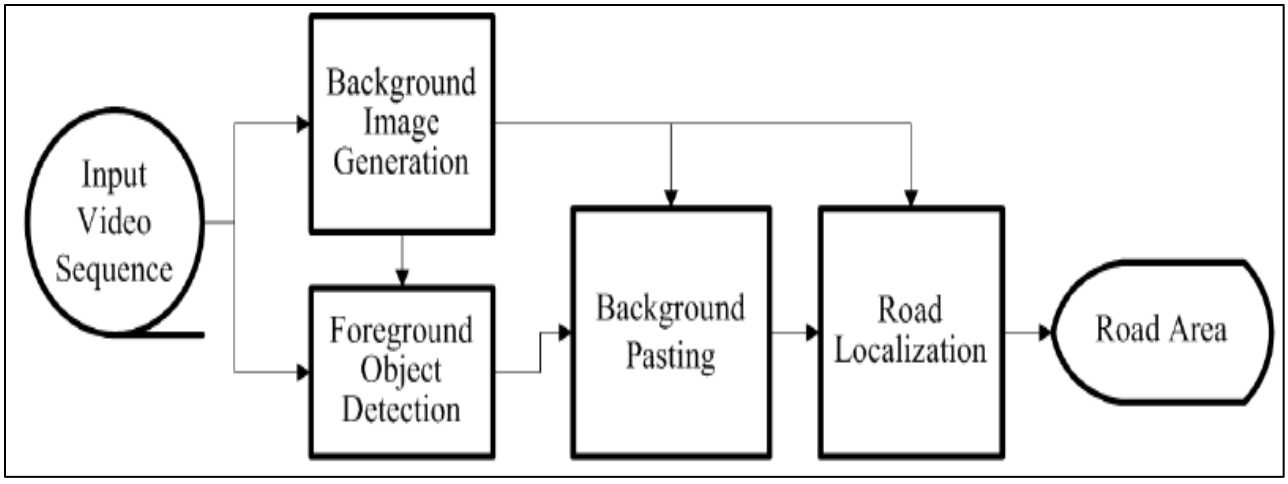


Fig.Working module of road segmentation

c) LOCATION DETECTION

Location detection is a computer technology related to computer vision and image processing that deals with detecting location of object.

d) SPEED CALCULATION

In this the speed of the moving object(vehicle) is calculated using time(). import numpy as np
import timeit
import datetime
import time
This library is used for calculating the speed of the moving vehicle. The formula used for calculating speed is = ((distance) / (seconds)).

e) IMAGE STORING

If the vehicle passes a certain limit of range, then the system captures the images and stores it in the folder. The images are stored as per the lanes i.e. Lane_1 and Lane_2 in the OFFENDER folder. The images can be viewed later also by the supervisor or the officer.

4.2 WORKING

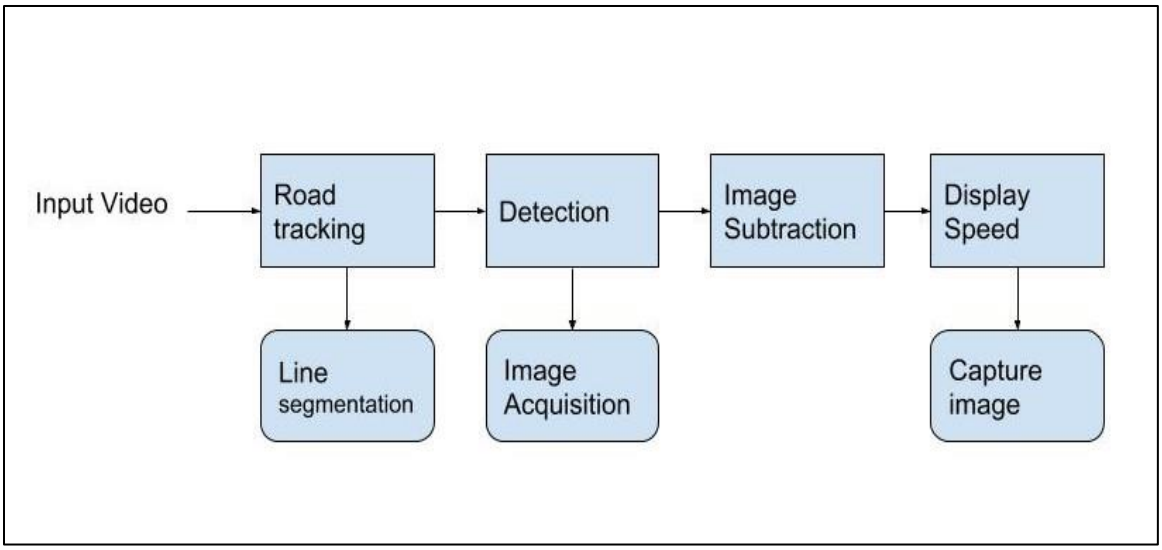


Fig :4.2 working of speed detection using open CV

This program uses Open cv to detect and capture car with help of a traffic camera, once a car passes through a certain distance its speed is calculated and displayed. This process takes place by road tracking which consist a technique known as lane segmentation.

Lane segmentation is a crucial issue in trajectory planning which classifies different lane and generate definite driving areas. Once the process is over detection of vehicle in short image acquisition takes place.

Image acquisition is a digital image processing technique, which is defined as an action of retrieving an image from some source usually some hardware source.

After the image is acquired image subtraction is also known as pixel subtraction whereby the digital numeric value of one pixel or whole image is subtracted from another image. After completion of all the process the speed of vehicle is displayed. If the speed of the vehicle is greater than the estimate limit then it will capture the image and it will store it in the folder.

1) cv2.threshold(mask1, 127, 255, cv2.THRESH_BINARY_INV)

If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black).

In BINARY_INV it will inverse the value i.e. 1=black, 0=while.

2) cv2.imread

To read an image, we simply call the imread function.

3) cvtColor

The function converts an input image from one color space to another.

4) Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc. also comes into play.

Fig. Original image

a. Erosion

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). all the pixels near boundary will be discarded depending upon the size of kernel. So, the thickness or size

of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises.

Result:

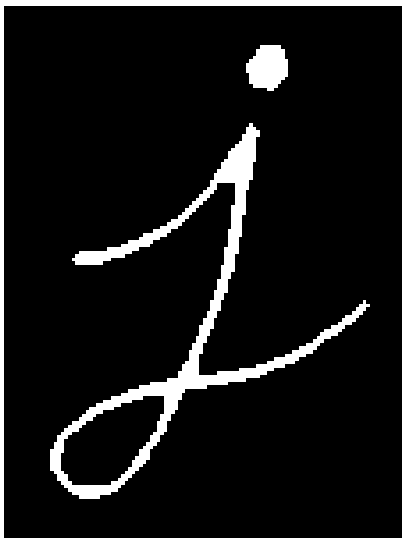


Fig: Result after implementation of erosion technique

b. Dilation

It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So, it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So, we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

Result:

Fig: Result after implementation of dilation technique.

c. Opening

Opening is just another name of erosion followed by dilation. It is useful in removing noise, as we explained above. Here we use the function, cv2.morphologyEx()

Result:

Fig: Result after implementation of opening technique.

d. 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 object.

Result:

Fig: Result after implementation of closing technique.

e. Morphological Gradient

It is the difference between dilation and erosion of an image. The result will look like the outline of the object.

Result:

Fig: Result after implementation of morphological gradient technique.

f. Median Blur

The Median blur operation is similar to the other averaging methods. Here, the central element of the image is replaced by the median of all the pixels in the kernel area. This operation processes the edges while removing the noise. You can perform this operation on an image using the median Blur () method of the imgproc class. (image before median blur operation)

Original image

Fig: Result after implementation of erosion technique

g. np. bitwise_and

The bitwise AND operation on the corresponding bits of binary representations of integers in input arrays is computed by np. bitwise_and () function.

h. cv2.imshow()

Use the function cv2.imshow() to display an image in a window. The window automatically fits to the image size. First argument is a window name which is a string. second argument is our image. You can create as many windows as you wish, but with different window names.

E.g. cv2.imshow("Lane_1", roi)

i. cv2.putText()

To put texts in images, you need specify following things.

-Text data that you want to write

- Position coordinates of where you want put it (i.e. bottom-left corner where data starts).

- Font type (Check cv2.putText() docs for supported fonts)

- Font Scale (specifies the size of font)

- Regular things like color, thickness, line Type etc. For better look, line Type = cv2.LINE_AA is recommended.

E.g. Font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(frame_og, str(int(speed)), (x, y), font, 2, (255, 255, 255), 8, cv2.LINE_AA)

cv2.putText(frame, str(int(speed)), (x, y), font, 2, (255, 255, 255), 8, cv2.LINE_AA)

Result:

Fig: cv2.putext()

The above fig shows the insertion of text in an image with the help of using cv2.putext().

4.3 SYSTEM REQUIREMENT

- **Software Requirement:**
  - Operating System: Windows 7,8,9,10, Linux
  - Memory: min-512MB, max-2GB
  - RAM: 3GB
  - Python, PyCharm, OpenCV

- **Hardware Requirement:**
  - Video Camera
  - Fairly capable processing system

4.4 FRONT END

PYTHON:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. The project is fully written in python. It uses the python editor PyCharm**.**

Features:

1) Easy to Learn and Use. Python is easy to learn and use.

2) Expressive Language. Python language is more expressive means that it is more understandable and readable.

3) Interpreted Language.

4) Cross-platform Language.

5) Free and Open Source.

6) Object-Oriented Language.

7) Extensible.

8) Large Standard Library**.**

PYCHARM:

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django. PyCharm is cross-platform, with Windows, macOS and Linux versions.

Features:

1) Coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes
2) Python refactoring: including rename, extract method, introduce variable, introduce constant, pull up, push down and others
3) Support for web frameworks: Django, web2py and Flask
4) Integrated Python debugger
5) Version control integration: unified user interface for Mercurial, Git, Subversion, Perforce and CVS with change lists and merge

OPEN_CV:

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itzel. The library is cross-platform and free for use under the open-source BSD license. It is a library of C programming language.it is used as it is much faster than the python. OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. All of the new developments and algorithms in OpenCV are now developed in the C++ interface.

Application:

- Segmentation and recognition
- Object detection
- Motion tracking
- Motion understanding
- Gesture recognition
- Face recognition system

Chap 5. PROJECT DESIGN

5.1 ALGORITHM

Step1: Start

Step 2: Declare two variable lane_1_1 = [],lane_1_2 = [] to capture speed limit of car

Step 3: Enter the speed limit

Step 4: Camera starts streaming to capture video of moving car using

```
cap = cv2.VideoCapture()
```

Step 5: Remove background in order to detect moving car, use

```
bgSubtractor = cv2.createBackgroundSubtractorMOG2()
```

Step 6: Create line to detect moving car crossing that lane, use

```
cv2.line(frame_og, (X coordinates), (Y coordinates), (R,G,B), Thikness)
```

Step 7: If

car crosses that line and speed limit entered by user

Then

capture speed of that car using

```
speed1 = ((distance) / (36.6 *(seconds1))) * 3600 * 90
```

Step 8: If

car moving in lane 1 crosses that line and speed limit

Then

save image of that car in folder using

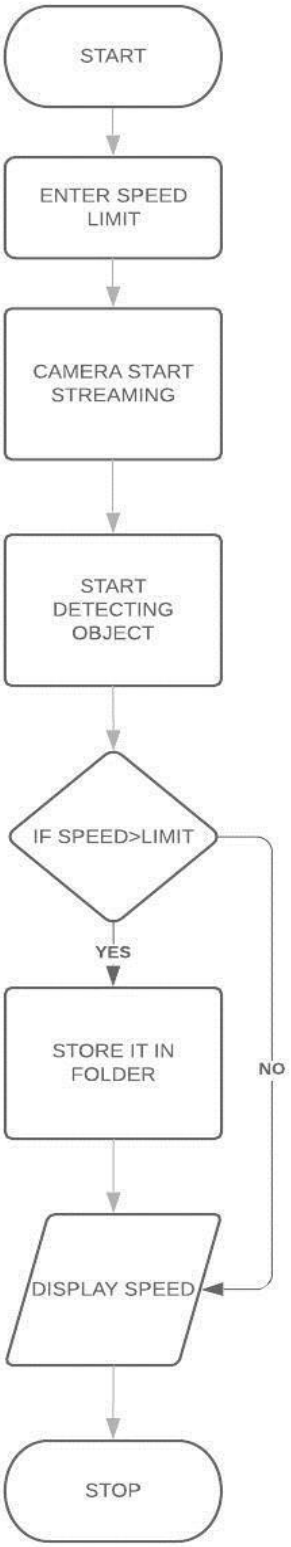
```
cv2.imwrite('Offenders/lane1/'+'Lane'+str(v)+'.jpeg',la)
```

else

```
cv2.imwrite('Offenders/lane2/'+str(v)+'.jpeg',li)
```

Step9: stop

5.2 FLOW CHART



5.3 USE CASE ANALYSIS

The use case analysis will be presented both specified in writing and graphically. From the objectives of the project as stated in the Introduction, the following use cases were deduced.

Table: Use case specification

Traffic Officer	Enter speed limit
	Detect vehicle speed
	Take vehicle picture
	Save generated data
	View save data
	Search saved data by date field

Fig:5.3 Use case analysis

The use cases in Table fig: 5.3 above will be illustrated graphically in Fig:5.3.1 below to allow for more insight into the relationships existing in the system.

This diagram shows the working once the video is been captured from the traffic camera. the Traffic officer is shown as the sole actor in the system. There is an include relationship between the Save generated data use case and the Detect vehicle speed, obtain speed of vehicle, Capture upload saved data, and the Take vehicle picture use cases because logically the later use cases need to be completed before the Save generated data use case can be executed. However, the include relationship is not direct, it is being implemented through the Capture data use case. First when the vehicles pass from certain area it is captured and the speed of the vehicles is calculated the calculated speed is then checked with the entered speed limit if the vehicles seem to be overspending the camera takes the picture, saves the generated data and the traffic user can view the saved generated data later on by searching save data in the data field or the folders. In this way each and every participant in the use case is interconnected with each other.

5.3.1 USE CASE DIAGRAM

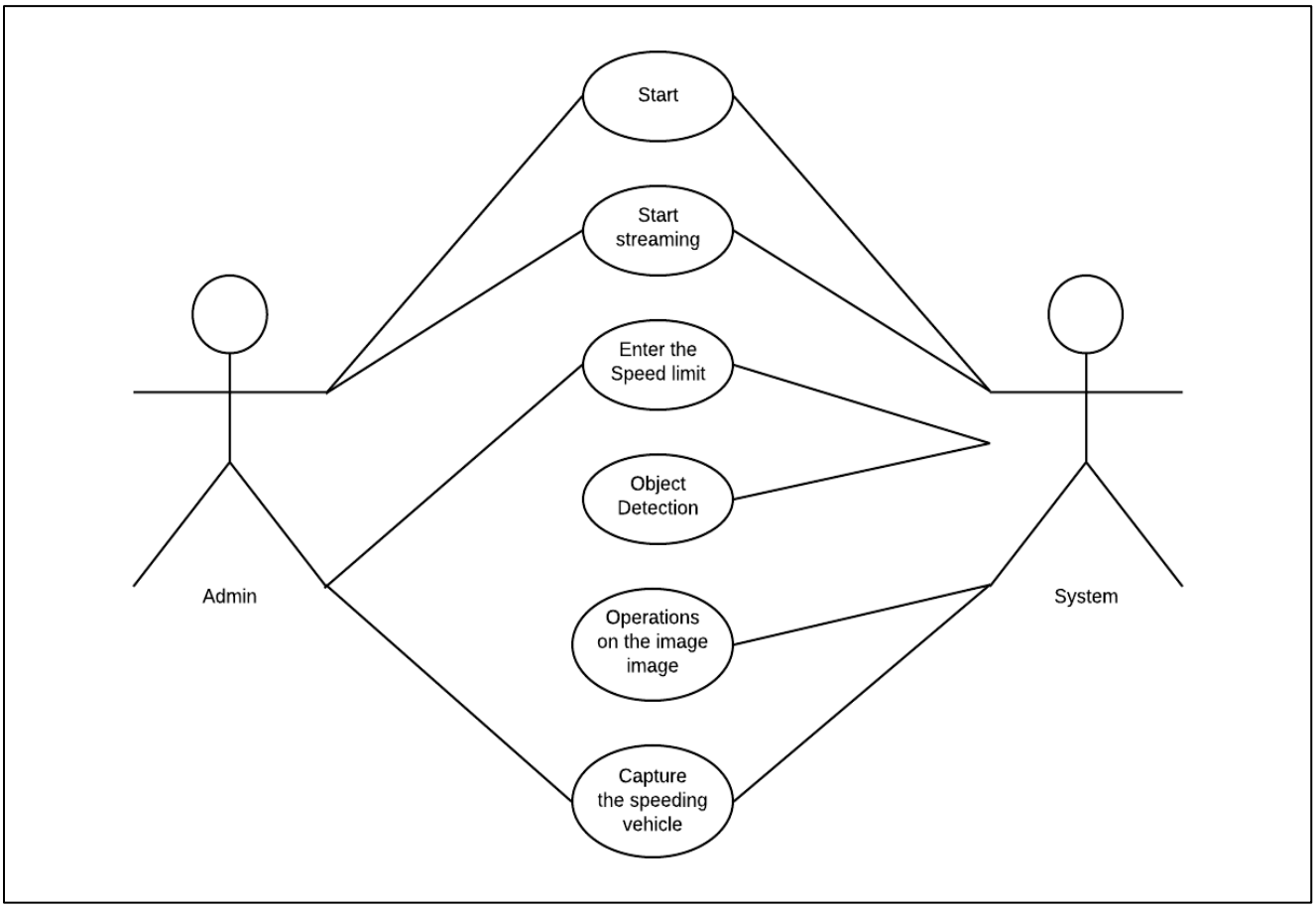


Fig:5.3.1 Use case digram

The above diagram shows the working of the application with the help of the use case. A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

The actors in the above diagram are the admin and The system.the relationship between the admin and the system is detected in the above diagram. The admin job is to start the camera feed by which the camera starts the streaming ,enter the speed limit,and monitor the captured speeding vehicles.the system or the software work is to monitor as well as start streaming,take the entered speed limit detect the moving object, perform operation on the image,I.e if the detected object is seen crossing the entered speed limit , to capture the speeding vehicle etc.

5.4 SEQUENCE DIAGRAM

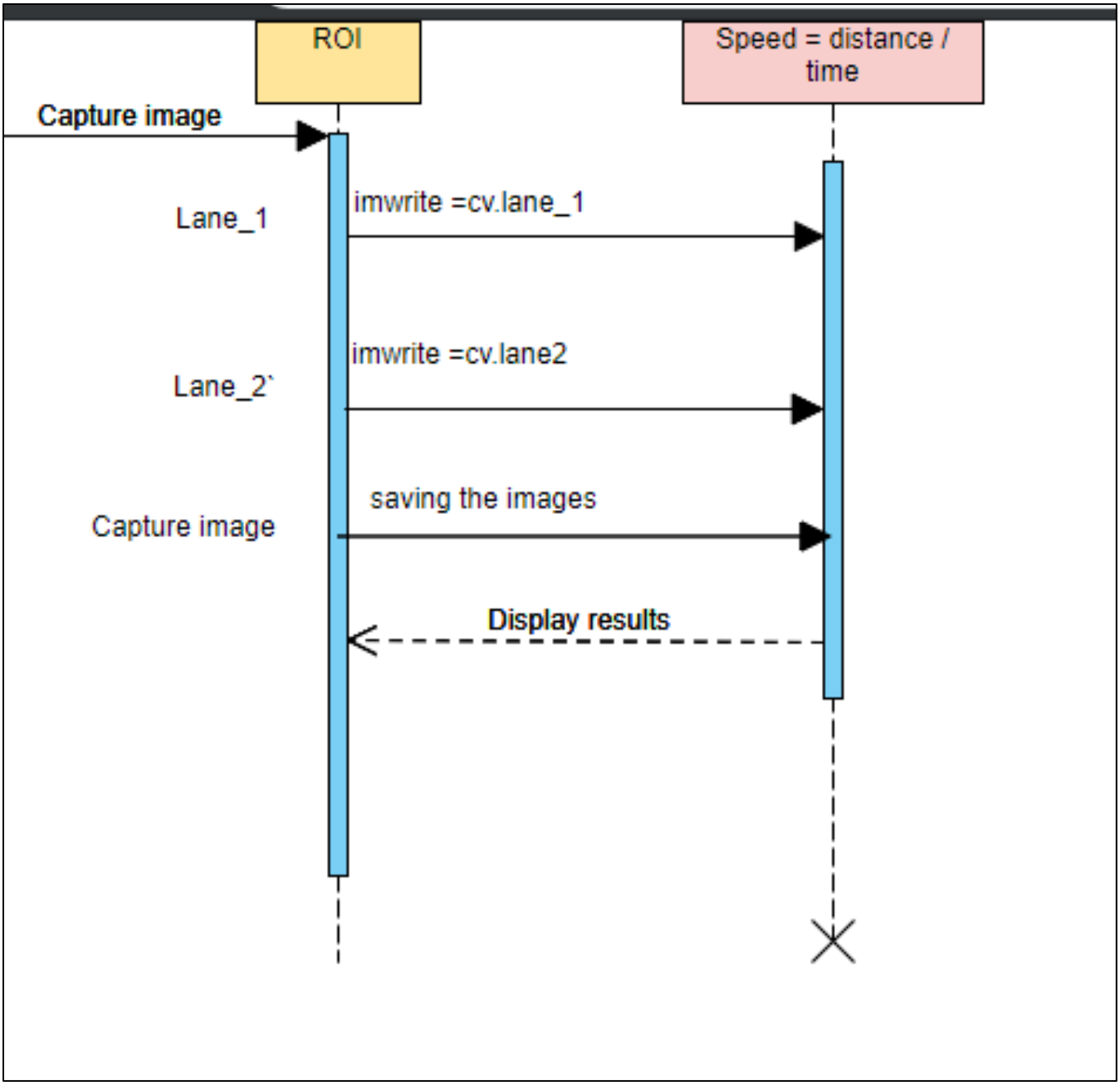


Fig:5.4 Sequence diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. The sequence diagram for the Detect vehicle speed use case is shown in Figure above.

The ROI stands for region of interest. The roads further classified into lane1 and lane2 once the car or vehicle passes through ROI the camera starts capturing the vehicle moving on the road the speed of vehicle is compared with the entered speed limit .the formula used for calculating the speed of vehicle is (speed =distance /time) .Once the vehicle is found overspending the vehicles image is captured and saved in the folder the folder consist of two files I.e. lane1 and lane2 the vehicle moving on lane 1 is saved in lane 1 folder and the lane 2 folder according to the location of the vehicle moving on the road. This event is arranged in an order by the sequence in which they occur.

Chap 6. EXPERIMENTAL RESULT

6.1 SCREEN SHOTS

Fig: 6.1.1

Here, the camera starts recording and detecting the vehicle moving on the road. The line shown in above fig 6.1 acts as a line that acts as point of detecting the vehicle as well as the speed of the vehicle passing across it.

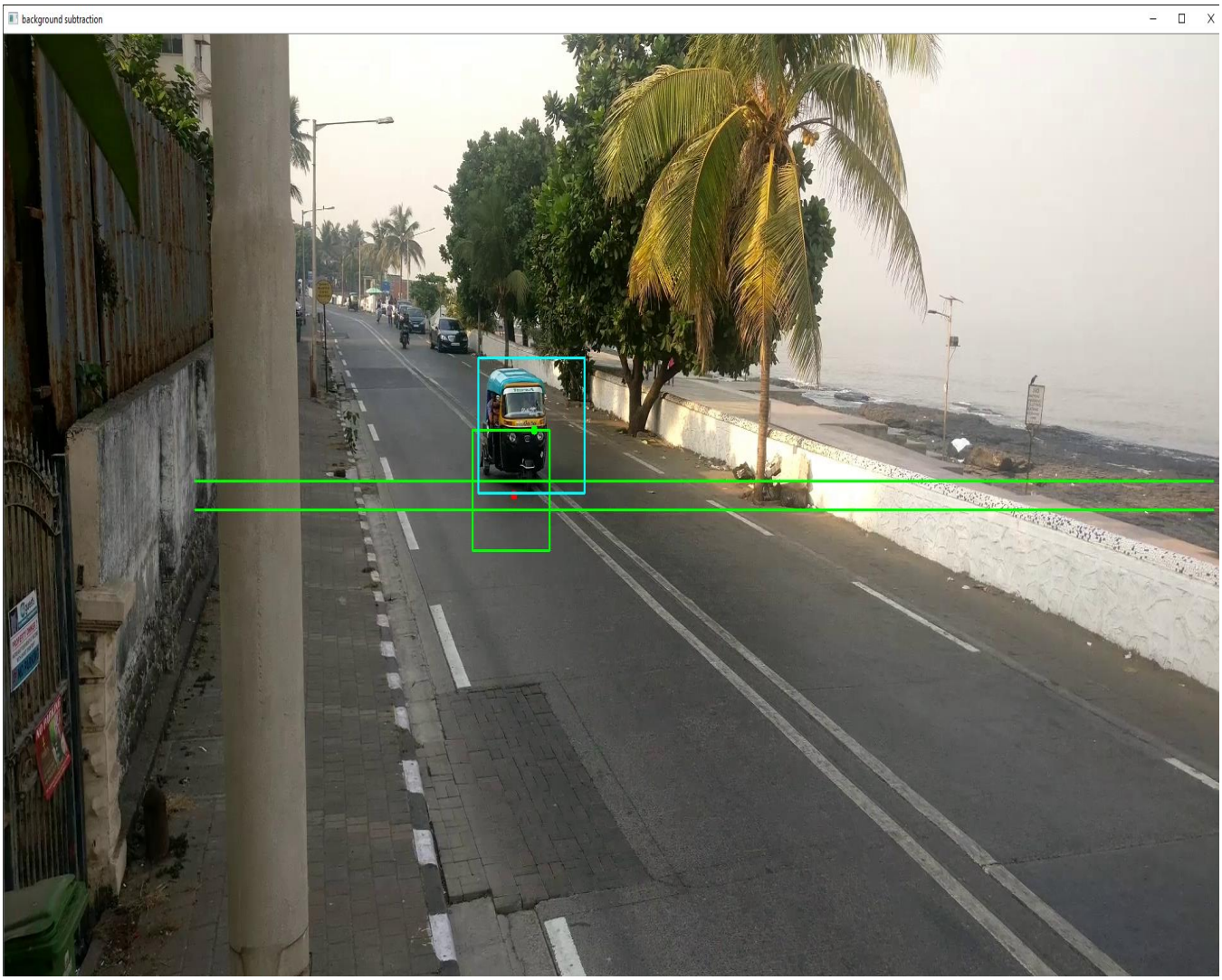


Fig: 6.1.2

In the above fig:6.1.2, the vehicle moving on the road id detected. The rectangle that surrounds the vehicle depicts the object (i.e. vehicle) moving. The rectangle in blue indicates that the vehicle is moving from right lane or right side of the road, and the rectangles with green color indicates the vehicle is moving from left. As the vehicle is moving in between of both lane both rectangles are seen.

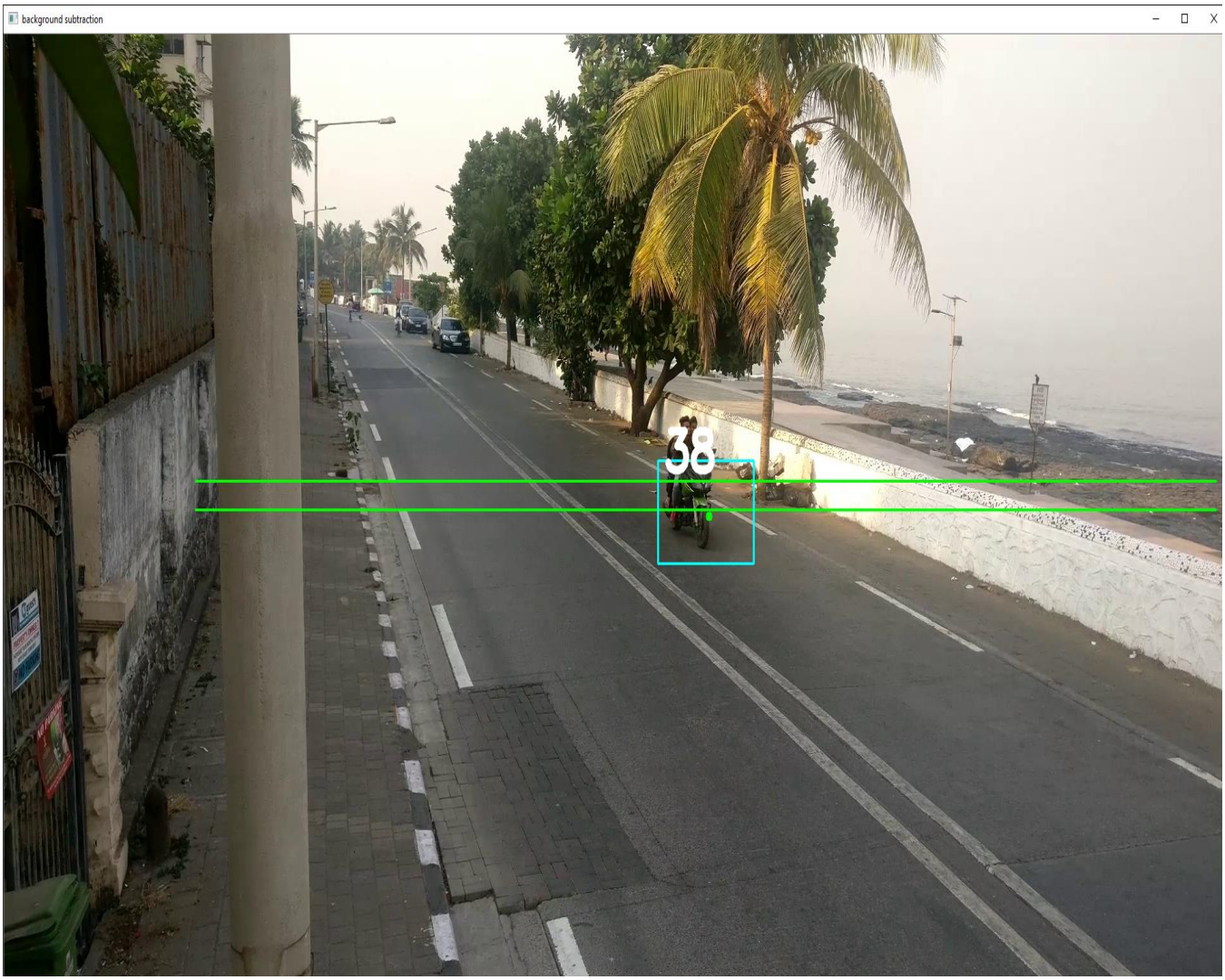
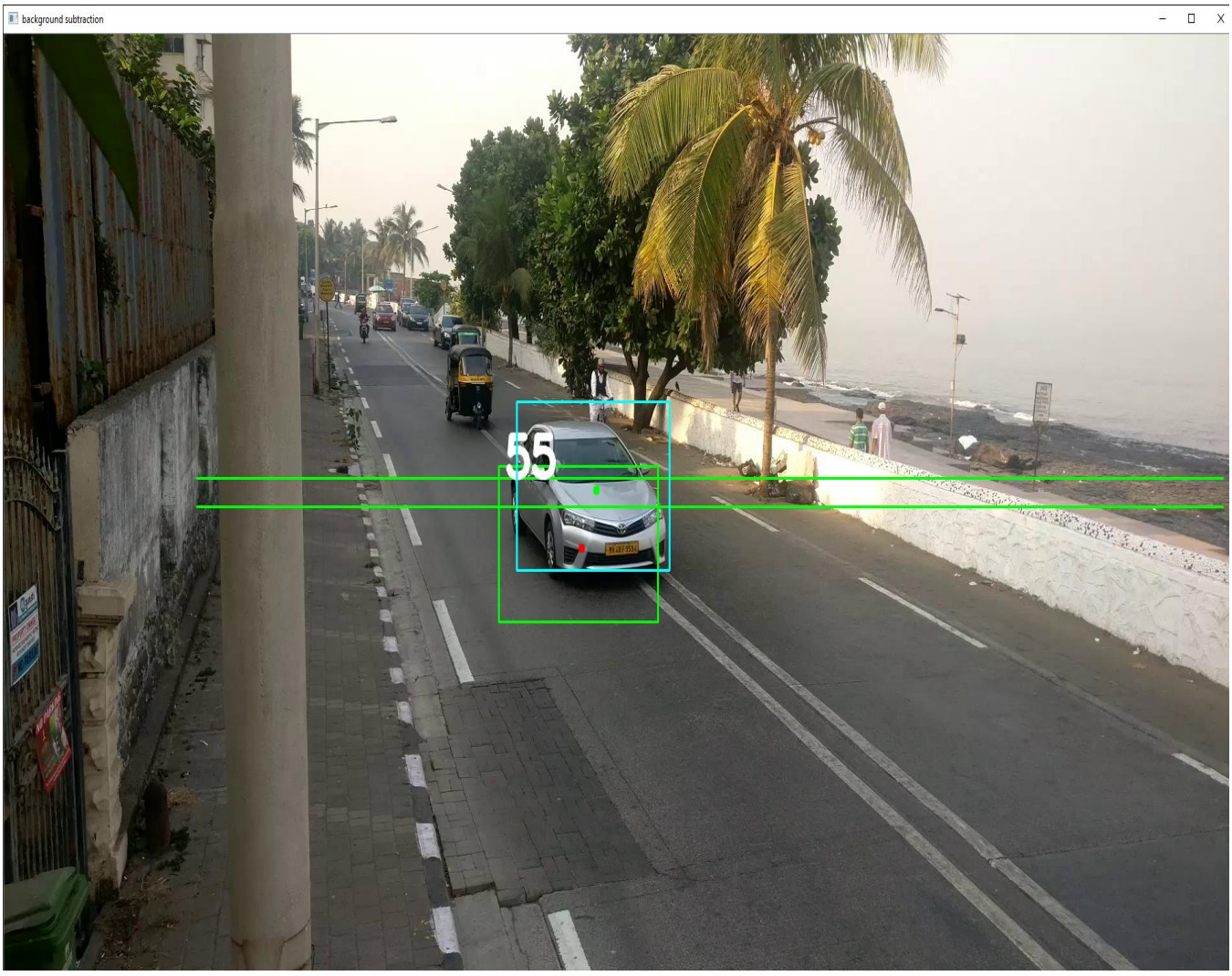


Fig: 6.1.3

In the above fig:6.1.3, The rectangle in blue indicates that the vehicle is moving from right lane or right side of the road. Once the vehicle passes the speed limit its speed is detected and displayed. The speed of above vehicle is 38km/h.



Fig; 6.1.4

In above fig:6.1.4, the vehicle is seen moving in between of two lanes. Hence two rectangles are seen. The vehicle is seen overspending the actual speed limit. The speed is 55km/h.

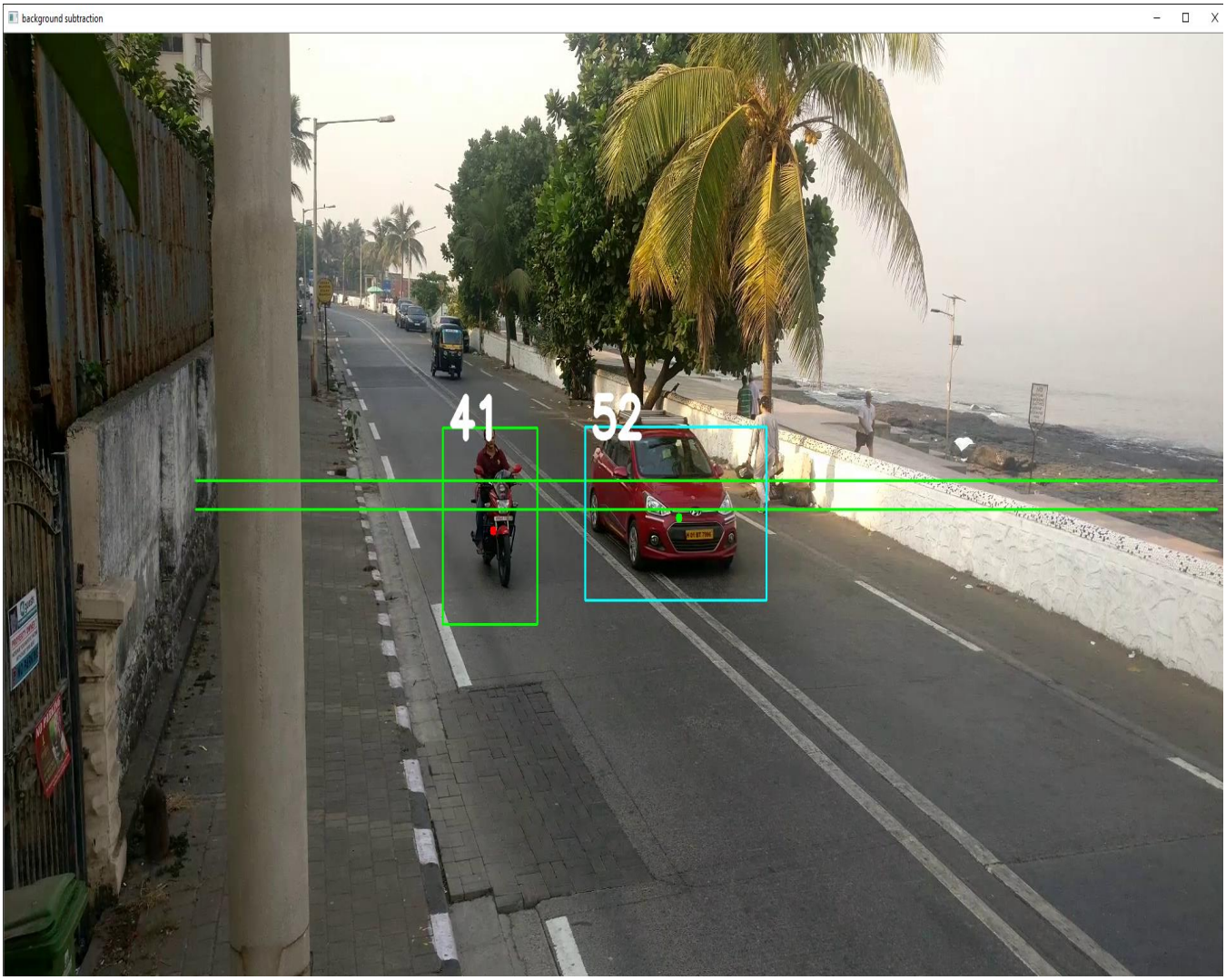


Fig: 6.1.5

In above fig:6.1.5, the vehicle on the right side of the lane is seen having speed 52km/h, and the vehicle moving on the left side of the lane is seen having speed 41km/h.

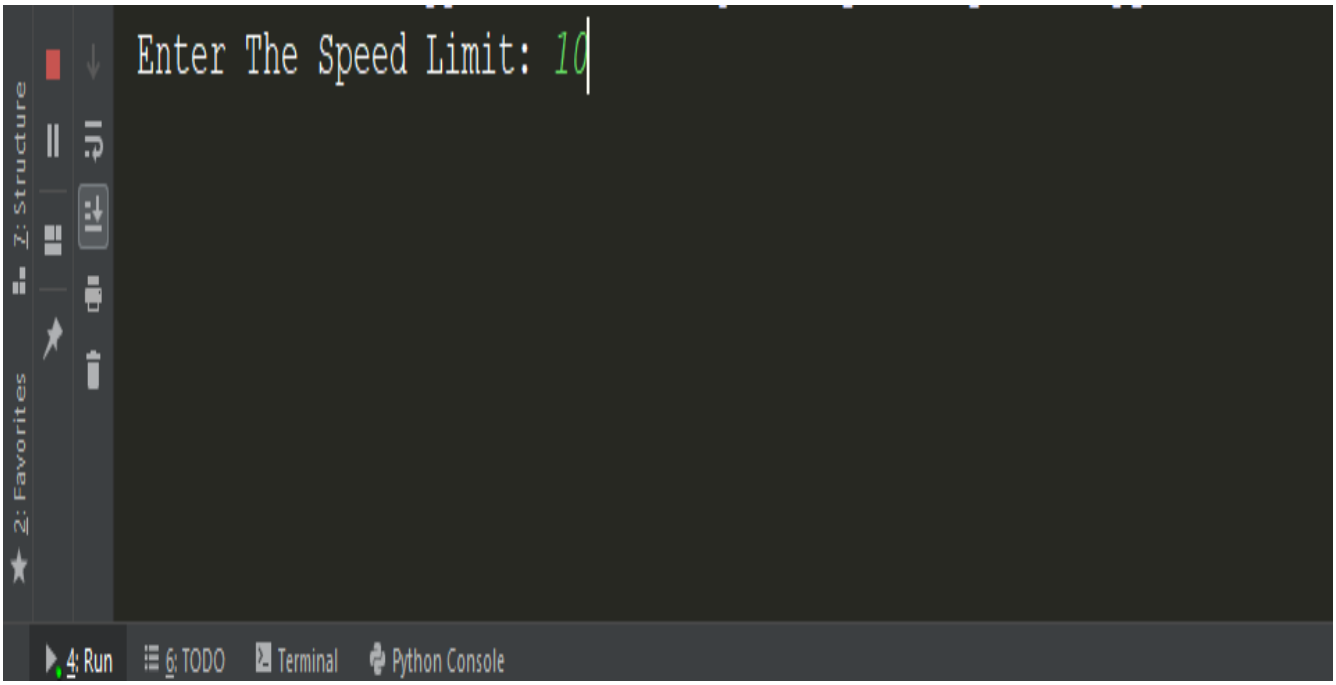


Fig: 6.1.6

In the above fig;6.1.6, is used to enter the speed limit of the vehicle.

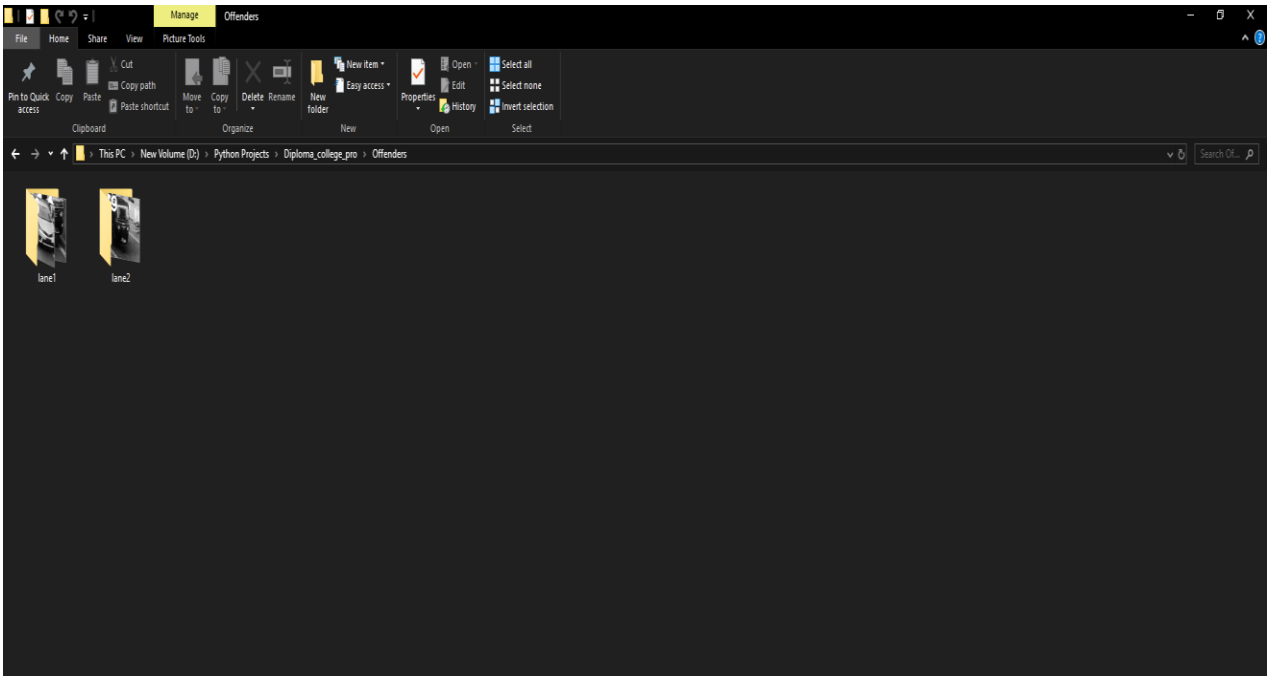


Fig: 6.1.7

Once the vehicle moving on the road is found overspending its image is captured and stored in the two respective folders i.e. lane1 and lane 2.

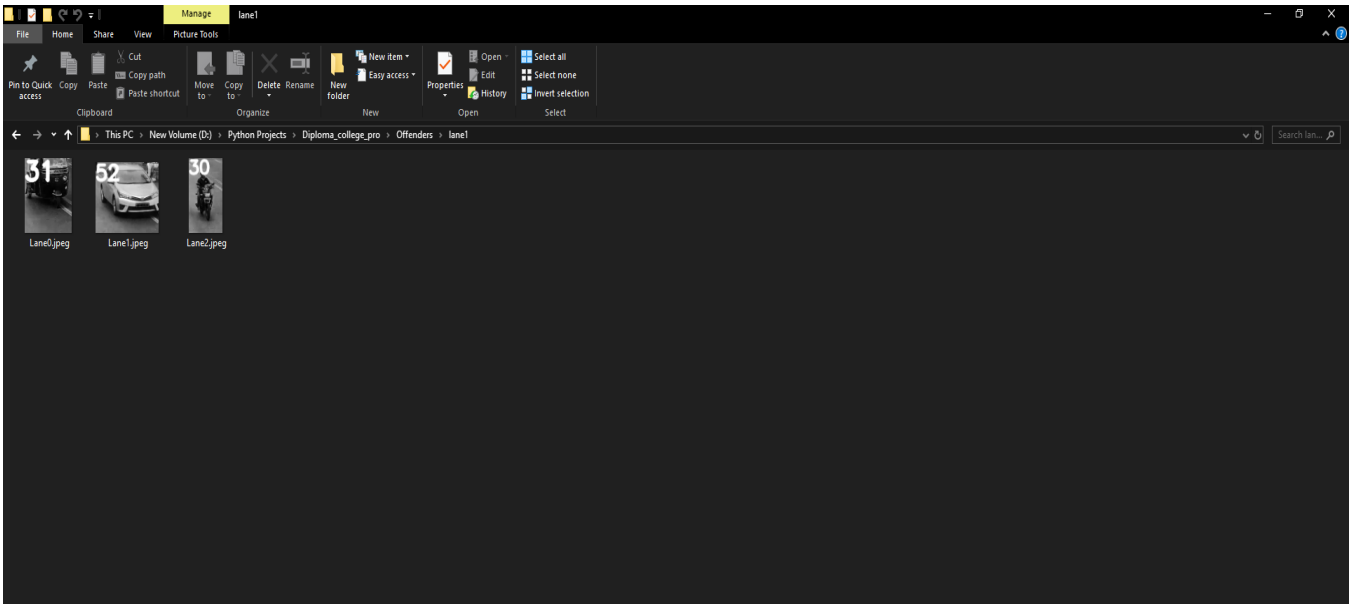


Fig. 6.1.8

In the above fig:6.1.8, the vehicle moving on the lane 1 which is found over speeding is captured and saved in the folder named as lane1.

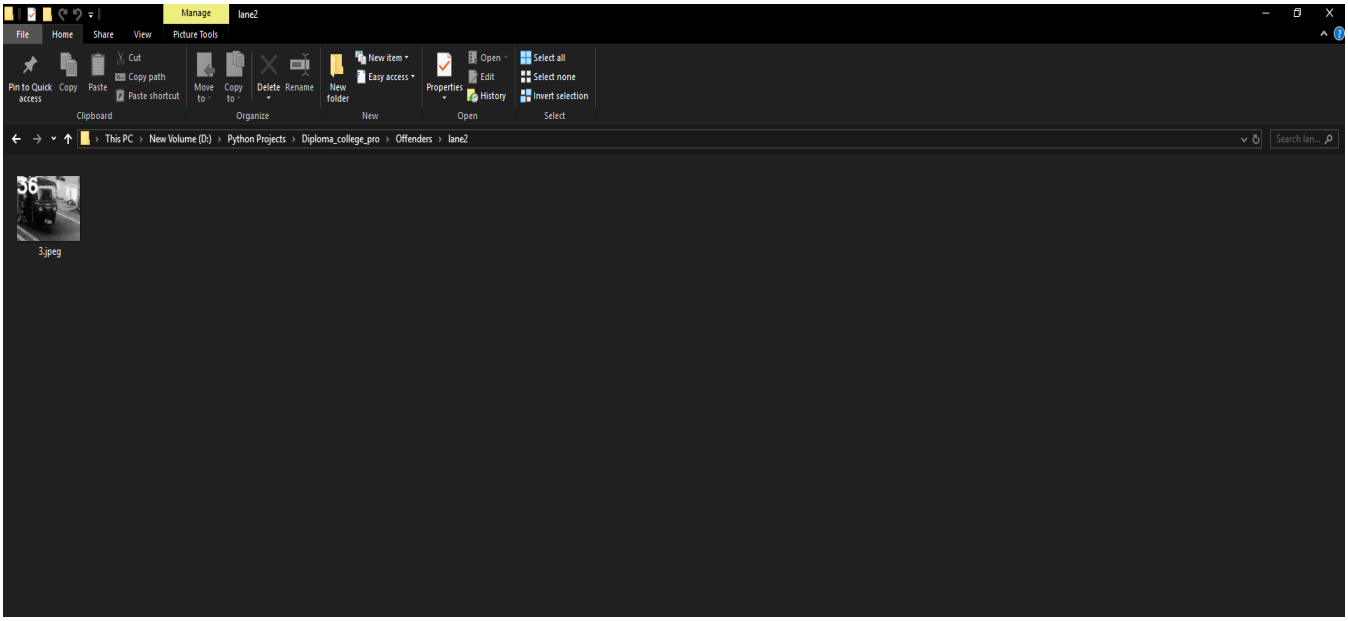


Fig. 6.1.9

In the above fig:6.1.9, the vehicle moving on the lane 2 which is found over speeding is captured and saved in the folder named as lane2.

6.2 CODE

```
import numpy as np
import cv2
import timeit
import datetime
import os
import time

speed_limit = int(input('Enter The Speed Limit: '))

# Initialize the video
cap = cv2.VideoCapture('test1.mp4')
lane_1_1 = []
lane_1_2 = []

mask1 = cv2.imread('m1.jpeg')
mask1 = cv2.cvtColor(mask1, cv2.COLOR_BGR2GRAY)
ret1, thresh_MASK_1 = cv2.threshold(mask1, 127, 255, cv2.THRESH_BINARY_INV)
mask2 = cv2.imread('m2.jpeg')
mask2 = cv2.cvtColor(mask2, cv2.COLOR_BGR2GRAY)
ret2, thresh_MASK_2 = cv2.threshold(mask2, 127, 255, cv2.THRESH_BINARY_INV)

# Create the background subtraction object
method = 1

if method == 0:
   bgSubtractor = cv2.bgsegm.createBackgroundSubtractorMOG()
elif method == 1:
   bgSubtractor = cv2.createBackgroundSubtractorMOG2()
else:
   bgSubtractor = cv2.bgsegm.createBackgroundSubtractorGMG()

# Create the kernel that will be used to remove the noise in the foreground mask
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
kernel_di = np.ones((5, 1), np.uint8)

# define variables
cnt = 0
cnt1 = 0
flag = True
flag1 = True
distance = 0.003
```

```
# Play until the user decides to stop
while True:
    start = timeit.default_timer()
    ret, frame = cap.read()
    frame_og = frame
    l, a, b = cv2.split(frame)
    clahe = cv2.createCLAHE(clipLimit=2, tileGridSize=(1, 1))
    frame = clahe.apply(l)
    cv2.line(frame_og, (300, 513), (1900, 513), (0, 255, 0), 2)
    cv2.line(frame_og, (300, 482), (1900, 482), (0, 255, 0), 2)
    if ret == True:
        foregroundMask = bgSubtractor.apply(frame)
        foregroundMask = cv2.morphologyEx(foregroundMask, cv2.MORPH_OPEN, kernel)
        foregroundMask = cv2.erode(foregroundMask, kernel, iterations=3)
        foregroundMask = cv2.morphologyEx(foregroundMask, cv2.MORPH_CLOSE,
kernel,iterations=6)
        foregroundMask = cv2.dilate(foregroundMask, kernel_di, iterations=7)
        foregroundMask = cv2.medianBlur(foregroundMask,5)
        thresh = cv2.threshold(foregroundMask, 25, 255, cv2.THRESH_BINARY)[1]
        thresh1 = np.bitwise_and(thresh, thresh_MASK_1)
        thresh2 = np.bitwise_and(thresh, thresh_MASK_2)
        im2, contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

        try:
            hierarchy = hierarchy[0]
        except:
            hierarchy = []
        for contour, hier in zip(contours, hierarchy):
            areas = [cv2.contourArea(c) for c in contours]
            max_index = np.argmax(areas)
            cnt = contours[max_index]
            (x, y, w, h) = cv2.boundingRect(cnt)
            cx = int((w / 2) + x)
            cy = int((h / 2) + y)
            if w > 10 and h > 10:
                cv2.rectangle(frame_og, (x - 10, y - 10), (x + w, y + h), (0, 255, 0), 2)
                cv2.circle(frame_og, (cx, cy), 5, (0, 0, 255), -1)

        if cy > 482 and w > 70 and h > 100:
            if flag is True and cy < 513:
                start_time = datetime.datetime.now()
```

```
            flag = False
        if cy > 513 and cy < 600:
            later = datetime.datetime.now()
            seconds = (later - start_time).total_seconds()
            if seconds <= 0.2:
                print("diff 0")
            else:
                print("seconds : " + str(seconds))
                if flag is False:
                    speed = ((distance) / (36.6 *(seconds))) * 3600 * 90
                    font = cv2.FONT_HERSHEY_SIMPLEX
                    cv2.putText(frame_og, str(int(speed)), (x, y), font, 2, (255, 255, 255), 8,
cv2.LINE_AA)
                    cv2.putText(frame, str(int(speed)), (x, y), font, 2, (255, 255, 255), 8, cv2.LINE_AA)
                    # if not os.path.exists(path):
                    #     os.makedirs(path)
                    if int(speed) > speed_limit and w > 70 and h > 100:
                        roi = frame[y-50:y + h, x:x + w]
                        cv2.imshow("Lane_1", roi)
                        lane_1_1.append(roi)
                        # write_name = 'corners_found' + str(cnt1) + '.jpg'
                        # cv2.imwrite(write_name, roi)
                        # cv2.imwrite(os.path.join(path, 'carimage_l2_' + str(cnt1)) + '.jpg', roi)
                        cnt += 1
                flag = True
                font = cv2.FONT_HERSHEY_SIMPLEX
                cv2.putText(frame, str(int(speed)), (x, y), font, 2, (255, 255, 255), 8, cv2.LINE_AA)

    im2, contours1, hierarchy1= cv2.findContours(thresh2, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    try:
        hierarchy1 = hierarchy1[0]
    except:
        hierarchy1 = []

    for contour1, hier1 in zip(contours1, hierarchy1):
        areas1 = [cv2.contourArea(c) for c in contours1]
        max_index1 = np.argmax(areas1)
        cnt1 = contours1[max_index1]
        (x1, y1, w1, h1) = cv2.boundingRect(cnt1)
        cx1 = int((w1 / 2) + x1)
        cy1 = int((h1 / 2) + y1)
```

```
        if w1 > 10 and h1 > 10:
            cv2.rectangle(frame_og, (x1 - 10, y1 - 10), (x1 + w1, y1 + h1), (255, 255, 0), 2)
            cv2.circle(frame_og, (cx1, cy1), 5, (0, 255, 0), -1)

    if cy1 > 482 and w1 > 70 and h1 > 100:
        if flag1 is True and cy1 < 513:
            start_time1 = datetime.datetime.now()
            flag1 = False
        if cy1 > 513 and cy1 < 600:
            later1 = datetime.datetime.now()
            seconds1 = (later1 - start_time1).total_seconds()
            if seconds1 <= 0.2:
                print("diff1 0")
            else:
                print("seconds1 : " + str(seconds1))
                if flag1 is False:
                    speed1 = ((distance) / (36.6 *(seconds1))) * 3600 * 90
                    speed1 = speed1 - 10
                    font = cv2.FONT_HERSHEY_SIMPLEX
                    cv2.putText(frame_og, str(int(speed1)), (x1, y1), font, 2, (255, 255, 255), 8,
cv2.LINE_AA)
                    cv2.putText(frame, str(int(speed1)), (x1, y1), font, 2, (255, 255, 255), 8,
cv2.LINE_AA)
                    # if not os.path.exists(path):
                    #     os.makedirs(path)
                    if int(speed1) > speed_limit and cy1 <= 720 and w1 > 70 and h1 > 100:
                        roi = frame[y1-50:y1 + h1, x1:x1 + w1]
                        cv2.imshow("Lane_2", roi)
                        lane_1_2.append(roi)
                        #cv2.imwrite(os.path.join('Offenders/', 'carimage_l2_' + str(cnt1)) + '.jpg', roi)
                        cnt1 += 1
                flag1 = True
                font = cv2.FONT_HERSHEY_SIMPLEX
                cv2.putText(frame_og, str(int(speed1)), (x1, y1), font, 2, (255, 255, 255), 8,
cv2.LINE_AA)
    #cv2.imshow('background subtraction', foregroundMask)
    #cv2.imshow('Sub',thresh)
    #cv2.imshow('Sub', thresh1)
    #cv2.imshow('Sub', frame)
    cv2.imshow('backgroundsubtraction', frame_og)
    stop = timeit.default_timer()
    time = stop-start
    print('One_frame = ',time)
```

```
        k = cv2.waitKey(1) & 0xff
        if k == ord('q'):
            break
    else:
        break
v = 0
u = 0
for la in lane_1_1:
    print('Saving')
    cv2.imwrite('Offenders/lane1/'+'Lane'+str(v)+'.jpeg',la)
    v+=1
for li in lane_1_2:
    print('Saving')
    cv2.imwrite('Offenders/lane2/'+str(v)+'.jpeg',li)
    u+=1
cap.release()
cv2.destroyAllWindows()
```

Chap 7. CONCLUSION

The designed speed detection system was capable of continuously monitoring the speed of the approaching vehicle. It worked well for the vehicle. The output was more accurate with no other moving objects in the surrounding. The value of speed of each passing vehicle was displayed in the LCD display. The detected speeds were proportional to the ground truth speeds. The developed system is very useful system to measure the low speeds accurately and without using expensive instruments. With development of such system, which is indeed helpful to the traffic law enforcement to maintain the safety and security of people on the road and avoiding the traffic rules to be violated. With a simple method of speed detection using a camera a useful software is developed.

Chap 8. FUTURE SCOPE

The speed detection using Open CV system is still a long way to go. However, with increasing development in field of artificial intelligence and machine learning the concept of adding advance technique in the software system is possible. Some of the suggestions towards extension and/or future related works are identified and are summarized below:

The software system will not only be able to detect the vehicle speed but also will be able to capture the vehicle plate (number) with help of machine Learning. With help of this, penalty on traffic rule violation can be enforced. Thus, with the current and growing awareness of the importance of security, trustworthy speed detection systems can be deployed in few years

9.REFERENCE

[1]. Lee, S., Gwak, J., Jeon, M.: Vehicle Model Recognition in Video. International Journal of Signal Processing, Image Processing and Pattern Recognition.

[2]. Yang, T., Kang, S.: Tracking for moving object

[3]. J. Pelegri, J. Alberola, V. Llario, "Vehicle Detection and Car Speed Monitoring, The IEEE Annual Conference in the Industrial Electronics Society (IECON 02), pp. 1693-1695, November 2002.

[4]. Z. Osman, S. Abou Chahine, "Speed Detection Scheme", The 14th International Conference on Microelectronics (ICM 2002), pp. 165-168, December 2002.

[5]. S. Pumrin, D.J. Dailey, "Roadside Camera Motion Detection for Automated Speed Measurement", The IEEE 5th International Conference on Intellegent Transportation Systems, pp. 147-151, September 2002.

[6]. https://www.wikipedia.org.

[7]. https://opencv.org/