VEHICLE SPEED MEASUREMENT ON URBAN ROADWAYS

A UG Project Phase – I report submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted By

G. INDIRA	**20UK5A0507**
K. SAINATH	**20UK5A0511**
A. ASHWINI	**19UK1A05N0**
MD. MUSEEF ANWAR	**18UK1A05F9**

Under the guidance of

Mrs. S. ANOOSHA

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE

Affiliated to JNTUH, HYDERABAD

BOLLIKUNTA, WARANGAL (T.S) – 506005

2019-2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE

WARANGAL

CERTIFICATE

This is to certify that the UG Phase - I Report entitled **“VEHICLE SPEED MEASUREMENT USING URBAN ROADWAYS”** is being submitted by **G.INDIRA(H.NO:20UK1A050), K.SAINATH (H.NO:20UK1A0511), A.ASHWINI (H.NO:19UK1A05N0), MD.MUSEEF ANWAR(H.NO:18UK1A05F9)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** to **Jawaharlal Nehru Technological University Hyderabad** during the academic year **2021-2022.**

Project Guide
Mrs. S. Anoosha
(Assistant Professor)

Head of the Department
Mr. Dr. R. Naveen Kumar
(Professor)

EXTERNAL

ACKNOWLEDGEMENT

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved **Dr. P. Prasad Rao**, Principal, Vaagdevi Engineering College for making us available all the required assistance and for his support and inspiration to carry out this UG Project Phase – I in the institute.

We extend our heartfelt thanks to **Dr. R. Naveen Kumar**, Head of the Department of CSE, Vaagdevi Engineering College for providing us necessary infrastructure and thereby giving us freedom to carry out the UG Project Phase – I.

We express heartfelt thanks to the Major Project Coordinator, **Dr. J. Srikanth**, Assistant Professor, Department of CSE for his constant support and giving necessary guidance for completion of this UG Project Phase – I.

We express heartfelt thanks to the guide, **Mrs. S. Anoosha**, Assistant Professor, Department of CSE for his constant support and giving necessary guidance for completion of this UG Project Phase – I.

Finally, we express our sincere thanks and gratitude to our family members, friends for their encouragement and outpouring their knowledge and experiencing throughout thesis.

G.INDIRA	**(20UK5A0507)**
K.SAINATH	**(20UK5A0511)**
A.ASHWINI	**(19UK1A05N0)**
MD. MUSEEF ANWAR	**(18UK1A05F9)**

ABSTRACT

In this paper, we propose a non-intrusive, video-based system for vehicle speed measurement in urban roadways. Our system uses an optimized motion detector and a novel text detector to efficiently locate vehicle license plates in image regions containing motion. Distinctive features are then selected on the license plate regions, tracked across multiple frames, and rectified for perspective distortion. Vehicle speed is measured by comparing the trajectories of the tracked features to known real world measures. The proposed system was tested on a data set containing approximately five hours of videos recorded in different weather conditions by a single low-cost camera, with associated ground truth speeds obtained by an inductive loop detector. Our data set is freely available for research purposes. The measured speeds have an average error of -0.5 km/h, staying inside the [-3,+2] km/h limit determined by regulatory authorities in several countries in over 96.0% of the cases. To the authors' knowledge, there are no other video-based systems able to achieve results comparable to those produced by an inductive loop detector. We also show that our license plate detector outperforms other two published state-of-the-art text detectors, as well as a well-known license plate detector, achieving a precision of 0.93 and a recall of 0.87.

Index Terms—vehicle speed measurement; license plate detection; feature tracking; vehicle motion detection.

TABLE OF CONTENTS

LIST OF CHAPTERS **PAGE NO**

LIST OF FIGURES PAGE NO:

1. INTRODUCTION

1.1. OVERVIEW

Rash driving is the cause of many road accidents all over the world. A total of 4,73,084 traffic accidents were reported during the last decade in India. The traffic population has increased considerably in India as there is no means to control or monitor the speed of vehicles running on roads. To overcome this problem and decrease the death rate due to accidents, the introduction of new and innovative speed enforcement technology is necessary. This project aims at designing a device that analyses the real-time video of vehicles passing on highways to estimate the speed of the vehicles passed and maintain a log of the speed of vehicles tracked at a particular timestamp.

- A web application is build to stream the live video of speed estimation of tracked vehicles.
- A Log of the speed of tracked vehicles with timestamp is stored in a CSV file.
- Analyzed frames are stored in the local disk in mp4 format.

1.2 PURPOSE

In this project, we make use of Computer Vision, python, Python Web Frame Works ,Open CV. The Department's current advice on speed limits contains a reference to detailed guidance on the "measurement and analysis of speed". This note provides some further advice on vehicle speed measurement for the purpose of determining speed limits. It discusses in detail techniques for carrying out measurement of speed using recently developed methods and for analysing results. There are also applications for improvement of alignments on all purpose roads such as at bends and short diversions. Whereas for new works and major improvements the Department's Highway Link Design Standard details the necessary techniques for geometric design using as a basis recent speed/flow/geometry relations, minor scheme design can better be based on the use of the measured 85 percentile vehicle speed of approach to the improvement section.

2. LITREATURE SURVEY

2.1 EXISTING PROBLEM

EXISTING PROBLEM APPROACHES OR METHOD TO SOLVE THE PROBLEM

The layout of major/minor junctions and accesses as described in Departmental Advice Note TA 20/81 is also, in a number of ways, dependent on a correct assessment of the 85 percentile vehicle speed. The advice herein should therefore be used as a basis. Additionally there are applications for the design of traffic signal installations. Here the position of the speed measurement is important, especially for new installations The methods described here in are Radar speed meter measurement. Measurement using vehicle detectors/timers such as inductive loops or noisy cables. This note is concerned with "when?", "where?", "how much?" and "how accurate?". Although the note generally applies to vehicles the sample can be confined to cars. Mention of this is made in the text at the appropriate points. Other publications such as Ref 4 provide useful information though of earlier date.

2.2 PROPOSED SOLUTION

This paper draws attention to the importance of vehicle speed measurement on urban roadways. The former is the instantaneous vehicle speed measured at a point as distinct from the latter which is measured over a length of road. Spot speeds are measured using devices such as radar speed meters or inductive loops. Journey speeds are measured by moving observer methods or by recording and matching registration numbers at times of passing. For determining speed limits the 85 percentile dry weather spot speed of cars is used as a yardstick. This is the speed only exceeded by 15% of the cars. When the 85 percentile spot speed has been arrived at, as described later in this Advice Note, it is used to determine the speed limit in the way described.

3. EXPERIMENTAL ANALYSIS

Whereas for speed limits the 85 percentile dry weather spot speed of cars is required, for improvement of alignments and major/minor junctions or accesses, and for new major/minor junctions or accesses on existing roads, the normal design methods are based on the 85 percentile wet weather journey speed of vehicles. The precise point at which the measurements are taken and the timing is important. A point just before the scheme length and a time of free flow are suitable. Measurements must be taken at both ends of the scheme so that traffic approaching from both directions is covered. If different values are obtained the higher speed value should be used in the design process. To get from the dry weather spot speed of vehicles measured to the wet weather journey speed used in design one of the following correction factors should be used - For AP Dual carriageways ... deduct 8kph For AP Single carriageways deduct 4kphp.Measurement error may also arise from the choice of sites, from the way in which the recording device is set up and used, and from the way in which the data are analyzed.

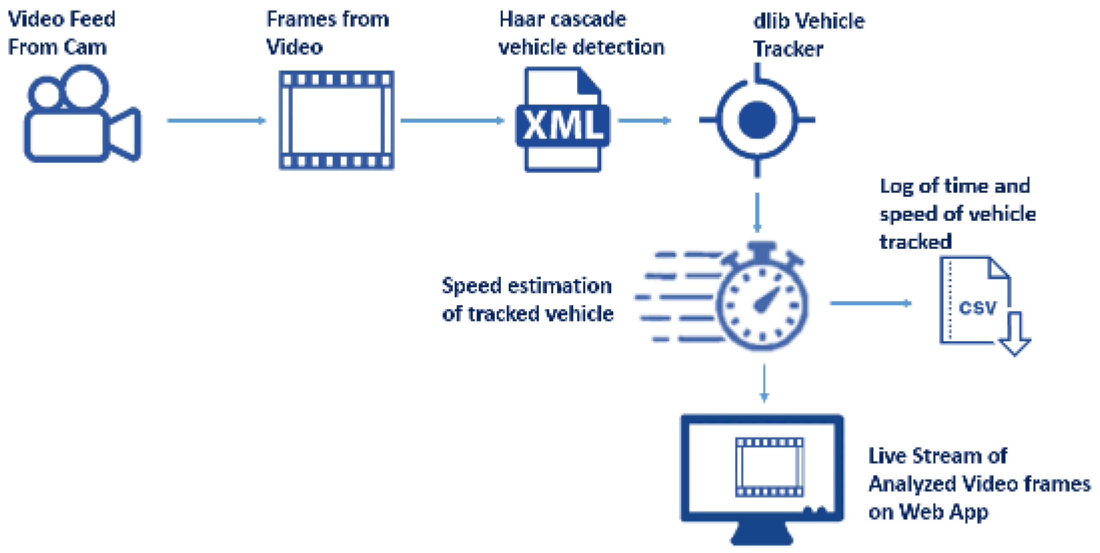

Video Feed From Cam
Frames from Video
Haar cascade vehicle detection
XML
dlib Vehicle Tracker
Log of time and speed of vehicle tracked
Speed estimation of tracked vehicle
CSV
Live Stream of Analyzed Video frames on Web App

Fig 1: Theoretical Analysis

Speeds vary from hour to hour, from day to day, from month to month, and from year to year, in a fairly systematic way. They have also been found to vary from one occasion to another more than would be expected from their variability on any one occasion. The total effect of all these variations, even when the times mentioned in paragraph above are excluded and only one year is considered, may produce a difference of more than 5kph between the highest and lowest levels of speed. It is essential, therefore, that more than one set of measurements be taken. At least two (and preferably more) recording periods at the site are required, at different times of day and on different days of the week. If measurements cannot be taken in different months, they should be taken in a month that is "neutral" as far as seasonal variation in traffic is concerned - late Spring and early Autumn are recommended, avoiding Bank Holidays - though this is less necessary for urban roads. Where measurements are required for modifying an existing traffic signal installation a different technique may be used with radar speedmeters or other methods which require an operator to be present. The sample should include only those vehicles that pass (at a point 150-200m back from the stop line) while a green signal is showing and no queue is present.

3.1 BLOCK DIAGRAM

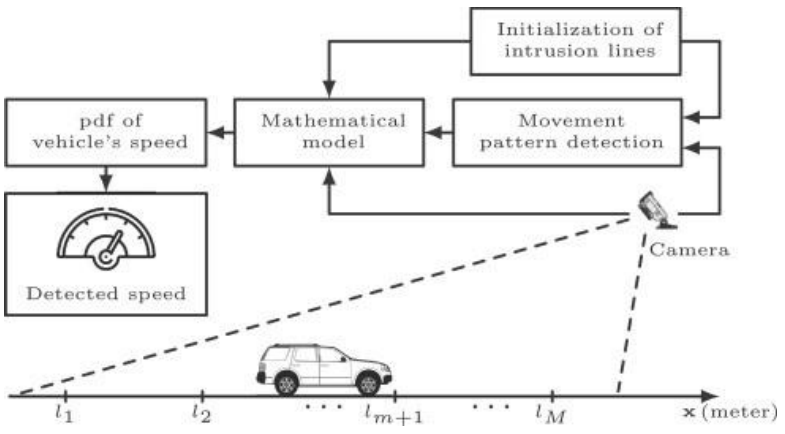


Fig 2: Block Diagram 1

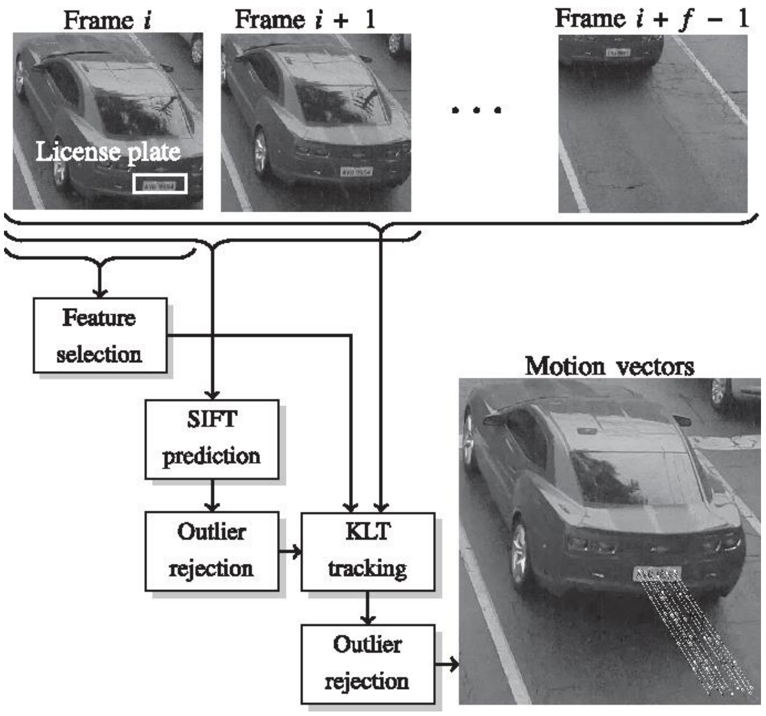


Fig 3: Block Diagram 2

3.2 HARDWARE AND SOFTWARE DESIGNING REQUIREMENTS OF THE PROJECT

- Use Python, Computer vision concepts
- work with Haar Cascade Classifiers
- work with Flask Web Framework
- work with vehicle tracking using pre-trained models
- implement Speed Measurement for tacked vehicles

This Project includes the following steps

1. Read the necessary libraries.
2. Initialize necessary variables.
3. Load the Haar cascade.
4. Capture the video.
5. Read the Frame.
6. Detect vehicles.

7. Assign a new ID to the vehicle tracked.
8. Find the locations of vehicles tracked.
9. Calculated pixels per meter by dividing the distance of the road in pixels to meters.
10. Calculate the speed.
11. Display the streaming video Web Application.
12. Save the speed of the vehicle and timestamp wt which vehicle is tracked into a CSV file.

4. EXPIREMENTAL INVESTIGATION

ANALYSIS OR THE INVESTIGATION WHILE WORKING ON THE SOLUTION

Most methods include a background/foreground segmentation step to detect image regions containing motion. Common approaches for this task include simple frame differences as well as statistic models based on medians gaussian distributions or other measures . Vehicle speeds are estimated by tracking image features or regions, including blobs. image patches, edges corners the license plate region or a combination of such features. Rectification for perspective distortion is also a step found in most methods, and may occur before or after feature tracking. Although the cited methods have some steps in common, they also have fundamental differences, not only on the way they measure vehicle speeds, but also on the type of scenario they can be used.

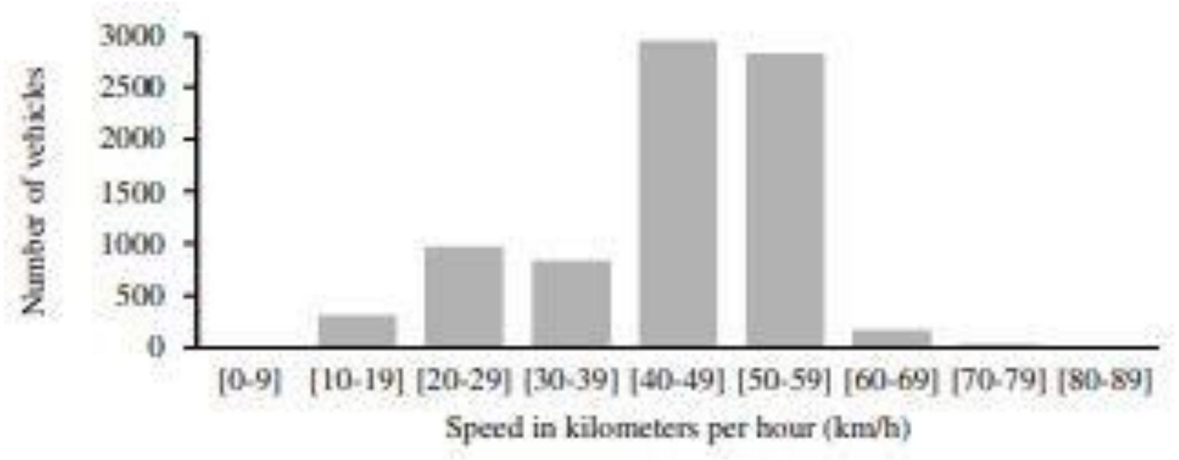


Fig 4: Speed in Kilometers per hour.

A proof-of-concept system was built for evaluating the proposed approach. Besides the cameras and physical infrastructure, we used a 2.2 GHz Intel Core i7 machine with 12 GB of RAM running Linux, with the algorithms implemented in C++. In the next sections we describe our dataset, and evaluate our system’s performance regarding motion detection, license plate detection, and speed measurement.

5. DESIGN

DIAGRAM SHOWING THE CONTROL FLOW OF THE SOLUTION

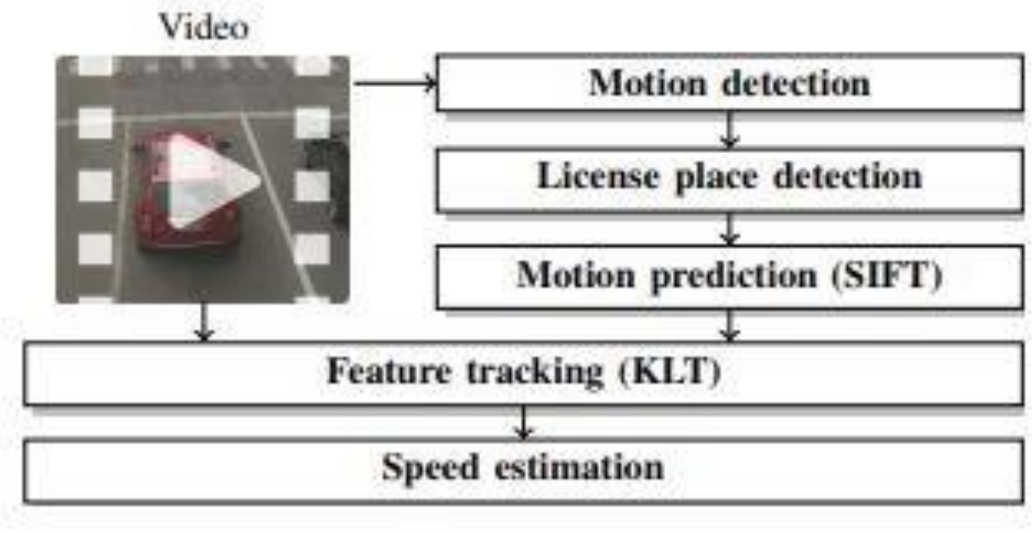


Fig 5: Control Flow of The Solution

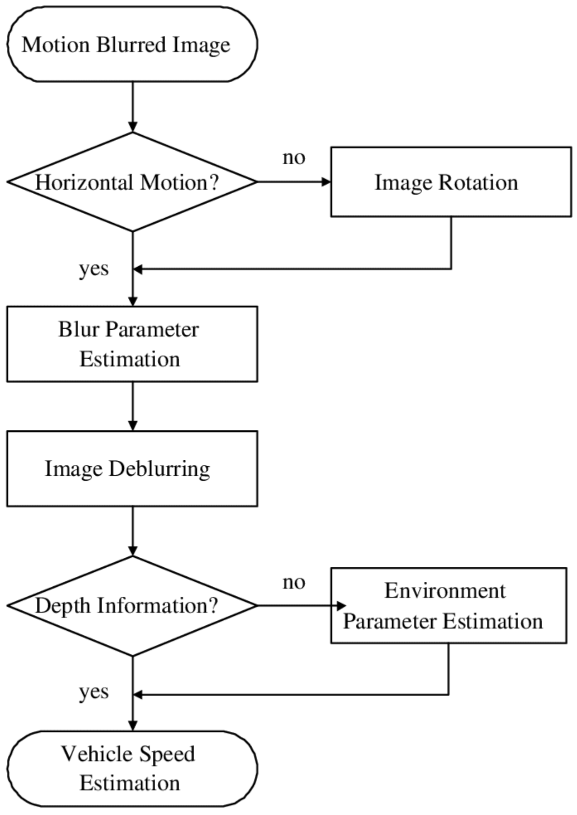


Fig 6: Flow Chart 2

Our system measures vehicle speeds based on the motion vectors obtained by the feature selection and tracking method. Each motion vector ~di can be associated with a measurement of a vehicle's instantaneous speed at a particular time, given in pixels per frame, in the image plane. The purpose of the speed measurement module is converting these measurements to kilometers per hour (km/h) in the real world. An important assumption of our system is that each street lane lies on a plane. That assumption makes it possible for us to map the motion vectors ~di , which are given in pixels in the image plane, to displacements ~vi , given in meters in the ground plane.

5.1 PREREQUISITES

To complete this project you should have some prerequisites which are listed Below

- Python imutils package-For Basic image processing like resize etc.
- Python OpenCV package- Loading videos and processing of frames.
- Any Python IDE, like Spyder / Pycharm/ Jupiter-For programming in python
- Keras
- Tensorflow
- dlib
- Flask- For building web application

5.2 INSTALL ANACONDA NAVIGATOR

Anaconda/ PyCharm IDE is Ideal to complete this project

To install Anaconda, please refer to Anaconda Installation Steps

To install PyCharm IDE, please refer to PyCharm IDEInstallation steps

Fig 7: Installation of Anaconda Navigator

5.3 PYTHON PACKAGES INSTALLATION

If you are using anaconda navigator Please follow the below steps to download Open cv and imutils

- open anaconda prompt as administrator
- Type "pip install opencv-python" and click enter
- Type "pip install imutils" and click enter
- Type "pip install keras" and click enter
- Type "pip install tensorflow" and click enter
- Type "pip install flask" and click enter

The above steps allow you to install OpenCV and imutils , keras ,flask and tensorflow in the anaconda environment.

If you are using anaconda navigator Please follow the below steps to download Open cv and imutils

- open anaconda prompt as administrator
- Type "pip install opencv-python" and click enter
- Type "pip install imutils" and click enter
- Type "pip install keras" and click enter
- Type "pip install tensorflow" and click enter
- Type "pip install flask" and click enter

The above steps allow you to install OpenCV and imutils , keras ,flask and tensorflow in the anaconda environment.

To install dlib you should have the following prerequisites:

- install Build tools
- install cmake

Once the above packages are installed then open command prompt/ anaconda prompt and install dlib with the following package

"**pip install dlib**"

Note: make sure you get no errors while downloading dlib

5.3.1 Install Build Tools

Learn How to Install Visual Studio Build Tools. If you get error that ..

Learn How to Install Visual Studio Build Tools. If you get error that says "error: Microsoft Visual C++ 14.0 is required.". Be it for Python or any ot..

https://www.youtube.com/watch?v=y188HoZbSDE

5.3.2 Install Cmake

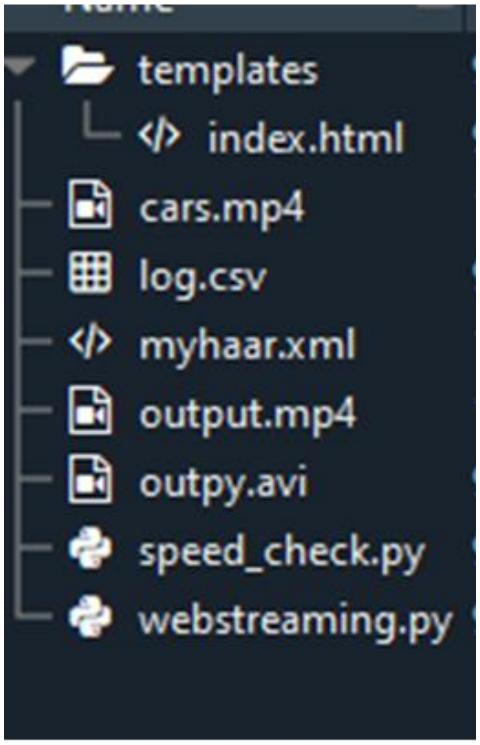
Installing CMake on Windows

Installing CMake on Windows

https://www.youtube.com/watch?v=sGLJMfCaWmw

6. PROJECT STRUCTURE

We are building a Flask Application so your project folder must contain the following subfolders



```
templates
└ index.html
cars.mp4
log.csv
myhaar.xml
output.mp4
outpy.avi
speed_check.py
webstreaming.py
```

Fig 8: Project Structure

- A temples subfolder where your HTML file is stored. (Web application file)
- A video file of vehicles passing on High way(you can use this video file or live stream of vehicles passing on highways.
- Haar cascade classifier file trained on car images.
- outpy.avi file which has the video of analysis of tracked vehicles and the respective speed at which the vehicle is traveling.
- A python script that implements speed estimation of passing vehicles on Highways.
- A webstreaming.py script implements live streaming of vehicle tracking and speed measurement.
- CSV file which has a log of time stamps and speed of vehicles tracked.

7. CODE SNIPPETS

Here we will be building the speed_check.py script which implements the speed estimation of a vehicle tracked

Create a python file using Spyder named speed_check.py

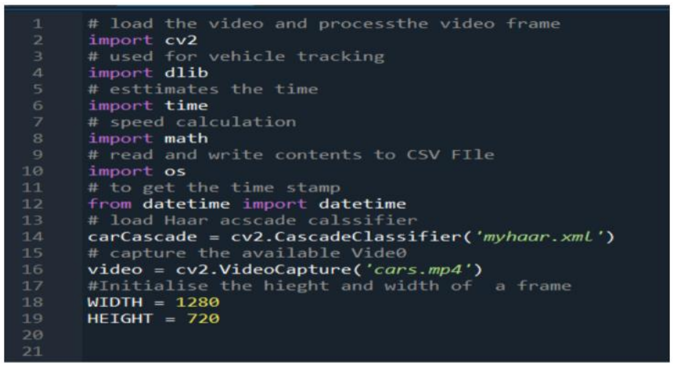
you can follow the youtube link given to know about the usage of Spyder IDE

This includes the following code snippets

- Read the necessary libraries.
- Initialize necessary variables.
- Load the Haar cascade.
- Capture the video.
- Read the Frame.
- Detect vehicles.
- Assign a new ID to the vehicle tracked.
- Find the locations of vehicles tracked.
- Calculated pixels per meter by dividing the distance of the road in pixels to meters.
- Calculate the speed.
- Display the streaming video Web Application.
- Save the speed of the vehicle and timestamp wt which vehicle is tracked into a CSV file.

7.1 LOAD NECESSARY FILES AND LIBRARIES:

let's begin the code by importing the libraries



```
# load the video and processthe video frame
import cv2
# used for vehicle tracking
import dlib
# esttimates the time
import time
# speed calculation
import math
# read and write contents to CSV FIle
import os
# to get the time stamp
from datetime import datetime
# load Haar acscade calssifier
carCascade = cv2.CascadeClassifier('myhaar.xml')
# capture the available Video
video = cv2.VideoCapture('cars.mp4')
#Initialise the hieght and width of  a frame
WIDTH = 1280
HEIGHT = 720
```

Fig 9: Loading Necessary Libraries

Import all the libraries which are showcased on the image. please refer to the comments for the usage of each library and each initialization

7.1.1 Read Frames Using Opencv

Join PyQt6 13 Hours Course in Udemy
https://www.udemy.com/course/pytho..

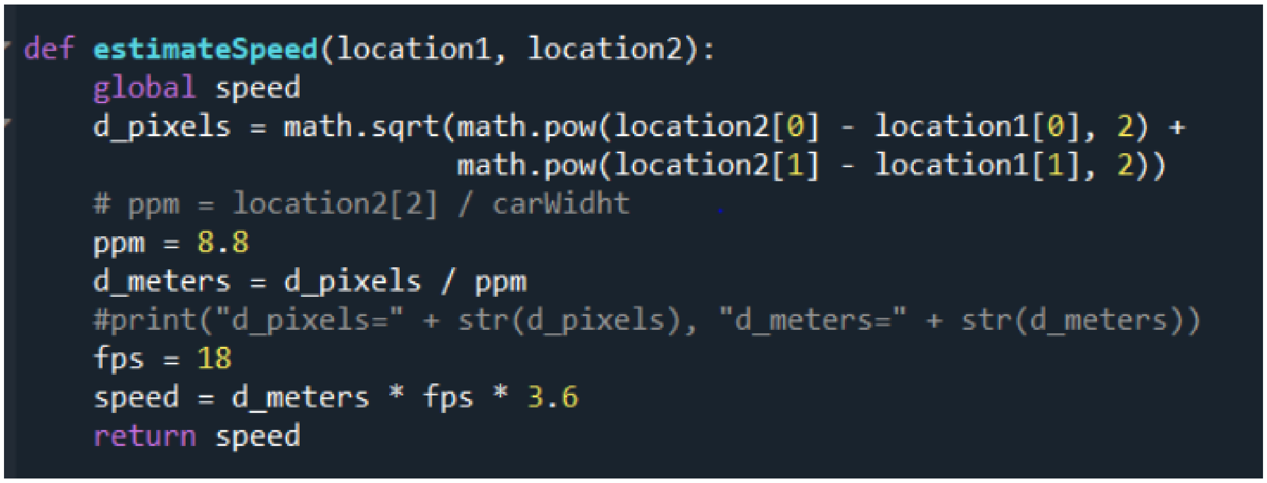
Join PyQt6 13 Hours Course in Udemy
https://www.udemy.com/course/python-gui-development-

https://www.youtube.com/watch?v=GymLwiQ2FNc

Fig 10: Downloading Opencv

7.2 SPEED ESTIMATION

- We are calculating the distance moved by the tracked vehicle in a second, in terms of pixels, so we need pixel per meter to calculate the distance traveled in meters.
- With distance traveled per second in meters, we will get the speed of the vehicle.



```
def estimateSpeed(location1, location2):
    global speed
    d_pixels = math.sqrt(math.pow(location2[0] - location1[0], 2) +
                         math.pow(location2[1] - location1[1], 2))
    # ppm = location2[2] / carWidht
    ppm = 8.8
    d_meters = d_pixels / ppm
    #print("d_pixels=" + str(d_pixels), "d_meters=" + str(d_meters))
    fps = 18
    speed = d_meters * fps * 3.6
    return speed
```

Fig 11: Speed Estimation

Here we have estimated the pixels per meter (ppm) manually. This ppm value will vary from road to road and have to be adjusted to be used on any other video.

to calculate ppm in meters :

• we need to know the actual width in meters of the road(you can use google to find the approximate width of the road in your country).

• we have taken the video frame and calculated the width of the road in pixels digitally.

• Now, we have the width of the road in meters from the real world and in pixels from our video frame.

• To map the distances between these two worlds, we have calculated pixels per meter by dividing the distance of the road in pixels to meters.

"d_pixels" gives the pixel distance traveled by the vehicle in one frame of our video processing.

To estimate speed in any standard unit first, we need to convert d_pixels to d_metres.

Now, we can calculate the speed(speed = d_meters * fps * 3.6). d_meters is the distance traveled in one frame. We have already calculated the average fps during video processing. So, to get the speed in m/s, just (d_metres * fps) will do. We have multiplied that estimated speed with 3.6 to convert it into km/hr.

7.3 VEHICLE TRACKING AND DETECTION

This activity Includes

• Vehicle Detection

o Use the Haar cascade classifier to identify vehicles.

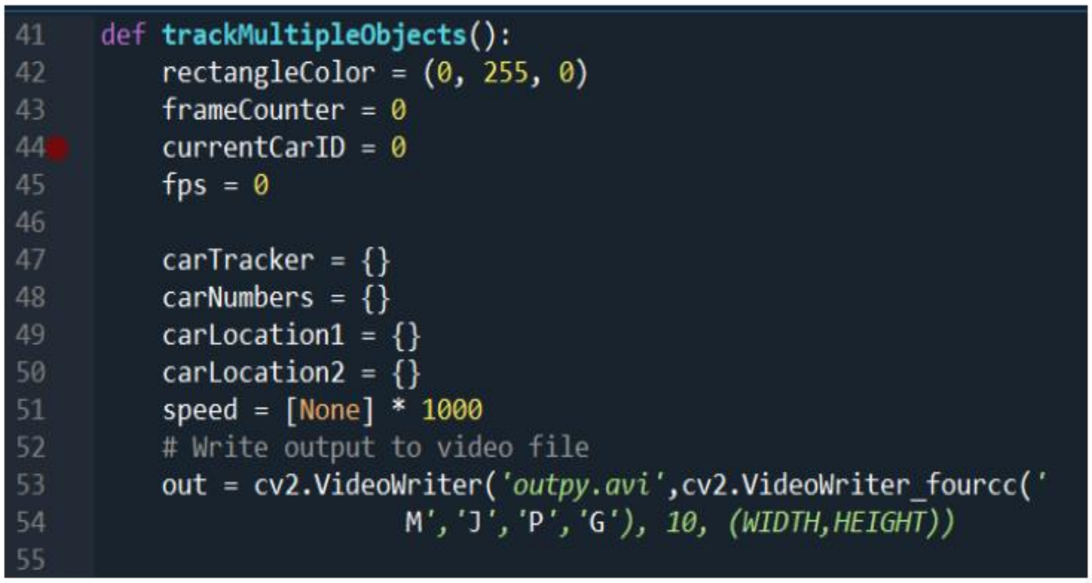
• Vehicle Tracking - (assigning IDs to vehicles)

o Use a correlation tracker from the dlib library.

• Speed Calculation

• Saving a Log in CSV file

Define A function to detect and track the vehicle :

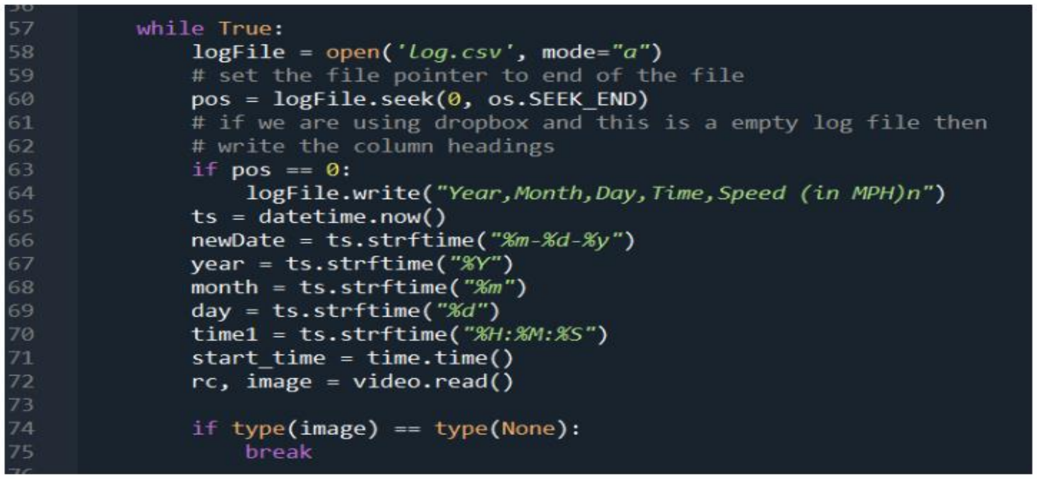


```python
def trackMultipleObjects():
    rectangleColor = (0, 255, 0)
    frameCounter = 0
    currentCarID = 0
    fps = 0

    carTracker = {}
    carNumbers = {}
    carLocation1 = {}
    carLocation2 = {}
    speed = [None] * 1000
    # Write output to video file
    out = cv2.VideoWriter('outpy.avi',cv2.VideoWriter_fourcc('
                    M','J','P','G'), 10, (WIDTH,HEIGHT))

```

Fig 12: Defining a Function to detect Vehicle

Define the Above parameters

- initialize the color of the bounding box which around the detected vehicle
- Initialize the frame count as 0
- initialize frames per second as 0
- create a "carTraker" dictionary which stores the data of the vehicle tracked
- create "carNumbers" dictionary
- Create two dictionaries to store the location of tracked cars
- create a variable named out to which the output video frames are stored in mp4 format, avi format



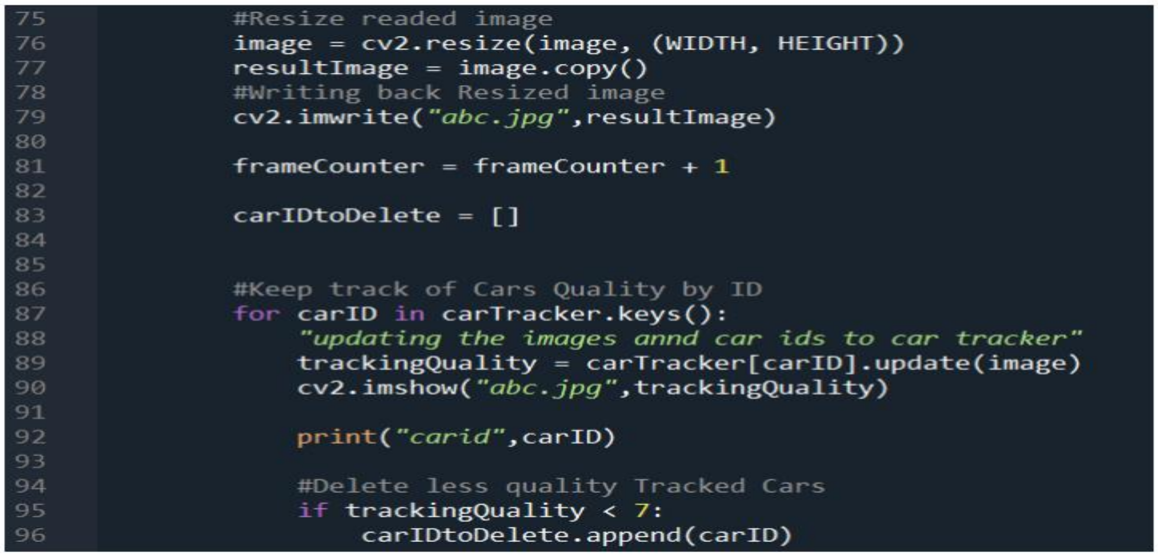
```python
    while True:
        logFile = open('log.csv', mode="a")
        # set the file pointer to end of the file
        pos = logFile.seek(0, os.SEEK_END)
        # if we are using dropbox and this is a empty log file then
        # write the column headings
        if pos == 0:
            logFile.write("Year,Month,Day,Time,Speed (in MPH)n")
        ts = datetime.now()
        newDate = ts.strftime("%m-%d-%y")
        year = ts.strftime("%Y")
        month = ts.strftime("%m")
        day = ts.strftime("%d")
        time1 = ts.strftime("%H:%M:%S")
        start_time = time.time()
        rc, image = video.read()

        if type(image) == type(None):
            break
```

Fig 13: Defining Code Snippet

The above code snippet implements the following aspects

- A CSV file is created to store the speed of vehicles with respect to timestamp.
- Using date-time library grab the present time and date.
- Grab year, month, and time from the read dateandtime.
- Read the frames from video.
- Check whether there is an image in the reading frame to perform vehicle tracking analysis, if not present real the loop.

7.4 DELETE THE PREVIOUSLY TRACKED CARIDS



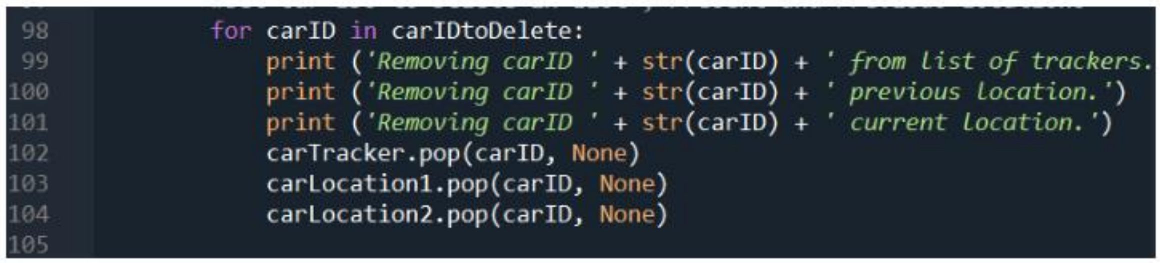
```python
            #Resize readed image
            image = cv2.resize(image, (WIDTH, HEIGHT))
            resultImage = image.copy()
            #Writing back Resized image
            cv2.imwrite("abc.jpg",resultImage)

            frameCounter = frameCounter + 1

            carIDtoDelete = []


            #Keep track of Cars Quality by ID
            for carID in carTracker.keys():
                "updating the images annd car ids to car tracker"
                trackingQuality = carTracker[carID].update(image)
                cv2.imshow("abc.jpg",trackingQuality)

                print("carid",carID)

                #Delete less quality Tracked Cars
                if trackingQuality < 7:
                    carIDtoDelete.append(carID)
```

Fig 14: Deleting the previously Tracked Data

let's depict the above code snippet

line 76 to 83:

- If there is an image, resize the image.
- Copy the same read image and save it.
- Increment the counter.
- create a list carIDtoDelete to store carID of tracked cars, and later to delete the car ID if the tracked object reaches the Nth frame.
- create an object tracker for each detected object to track the object as it moves around the frame.
- continue tracking until the N-th frame is reached.
- if the car stays in for less than 7 frames, then append the car id to the list created.



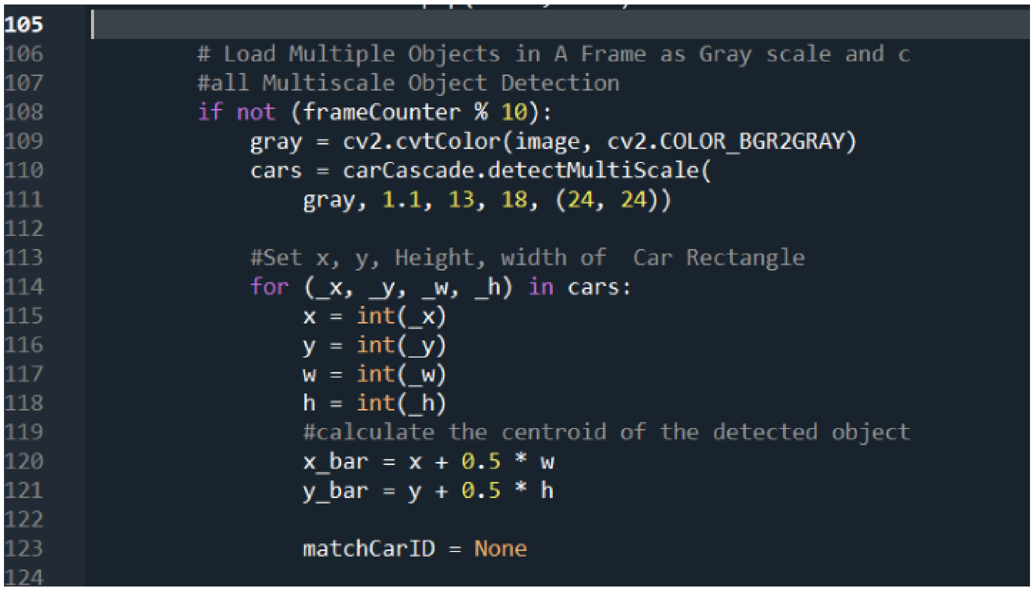
```
        for carID in carIDtoDelete:
            print ('Removing carID ' + str(carID) + ' from list of trackers.
            print ('Removing carID ' + str(carID) + ' previous location.')
            print ('Removing carID ' + str(carID) + ' current location.')
            carTracker.pop(carID, None)
            carLocation1.pop(carID, None)
            carLocation2.pop(carID, None)
```

Fig 15: Looping the Car Ids

Loop over the carIDtoDelete to delete the car IDS

7.5 DETECT THE CAR

Using har cascade classifier we detect the car and track the position of the car

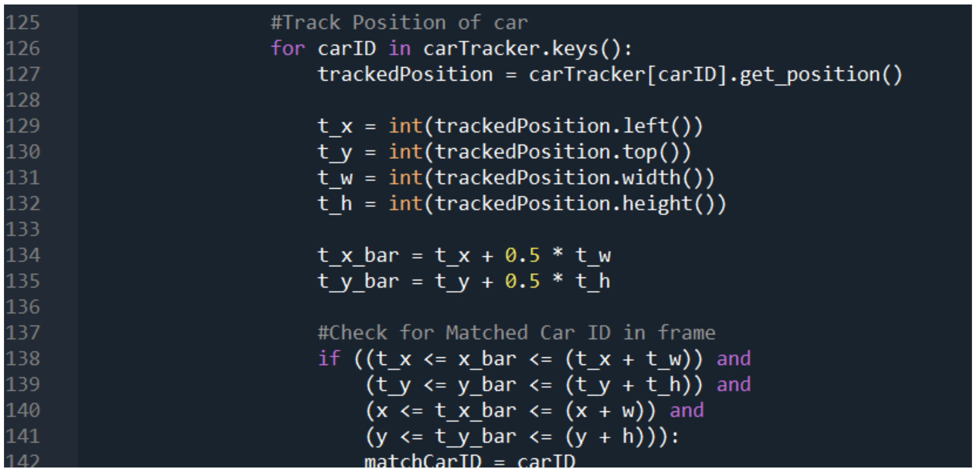


```
105
106         # Load Multiple Objects in A Frame as Gray scale and c
107         #all Multiscale Object Detection
108         if not (frameCounter % 10):
109             gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
110             cars = carCascade.detectMultiScale(
111                 gray, 1.1, 13, 18, (24, 24))
112
113             #Set x, y, Height, width of  Car Rectangle
114             for (_x, _y, _w, _h) in cars:
115                 x = int(_x)
116                 y = int(_y)
117                 w = int(_w)
118                 h = int(_h)
119                 #calculate the centroid of the detected object
120                 x_bar = x + 0.5 * w
121                 y_bar = y + 0.5 * h
122
123                 matchCarID = None
124
```

Fig 16: Detecting the Car

line 108 - 124

- For every 10th frame, we are detecting multiple cars using the haar cascade classifier function and grabbing all coordinates ((x,y)coordinates , length, and width of the car)of the detected cars.
- Now considering these coordinates to calculate the center point of the detected car.
- initially, we are declaring the matchcarID to none.



```
125                 #Track Position of car
126                 for carID in carTracker.keys():
127                     trackedPosition = carTracker[carID].get_position()
128
129                     t_x = int(trackedPosition.left())
130                     t_y = int(trackedPosition.top())
131                     t_w = int(trackedPosition.width())
132                     t_h = int(trackedPosition.height())
133
134                     t_x_bar = t_x + 0.5 * t_w
135                     t_y_bar = t_y + 0.5 * t_h
136
137                     #Check for Matched Car ID in frame
138                     if ((t_x <= x_bar <= (t_x + t_w)) and
139                         (t_y <= y_bar <= (t_y + t_h)) and
140                         (x <= t_x_bar <= (x + w)) and
141                         (y <= t_y_bar <= (y + h))):
142                         matchCarID = carID
```

Fig 17: Tracking the position of Car

- Now we are using dlib tracker to track the car, once the car is tracked the position of the tracked car is grabbed((x,y) coordinates, length, and width).

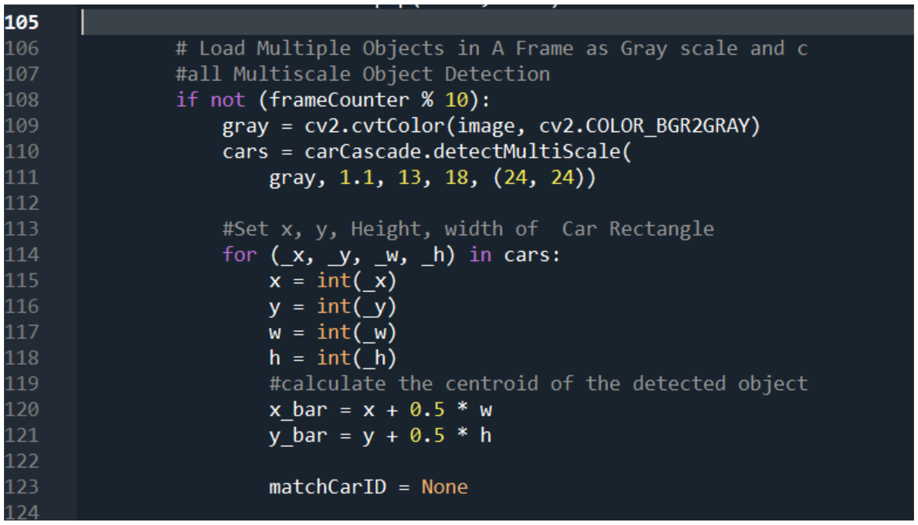
• Again Calculate the centroid of the tracked.

• Check whether the centroids of the detected car and tracked car are approximately the same.

• Then assign a carID to the car detected and tracked.]

7.6 MATCH CAR ID

Using har cascade classifier we detect the car and track the position of the car



```
105
106         # Load Multiple Objects in A Frame as Gray scale and c
107         #all Multiscale Object Detection
108         if not (frameCounter % 10):
109             gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
110             cars = carCascade.detectMultiScale(
111                 gray, 1.1, 13, 18, (24, 24))
112
113             #Set x, y, Height, width of  Car Rectangle
114             for (_x, _y, _w, _h) in cars:
115                 x = int(_x)
116                 y = int(_y)
117                 w = int(_w)
118                 h = int(_h)
119                 #calculate the centroid of the detected object
120                 x_bar = x + 0.5 * w
121                 y_bar = y + 0.5 * h
122
123                 matchCarID = None
124
```

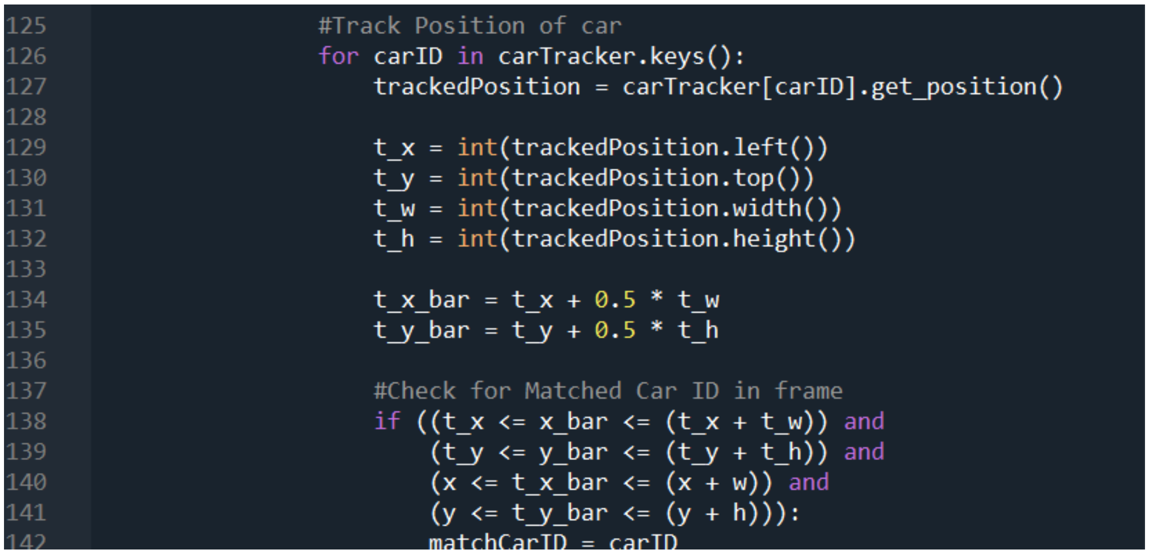
Fig 18: Matching the ID Card

line 108 - 124

• For every 10th frame, we are detecting multiple cars using the haar cascade classifier function and grabbing all coordinates ((x,y)coordinates , length, and width of the car)of the detected cars

• Now considering these coordinates to calculate the center point of the detected car

• initially, we are declaring the matchcarID to none



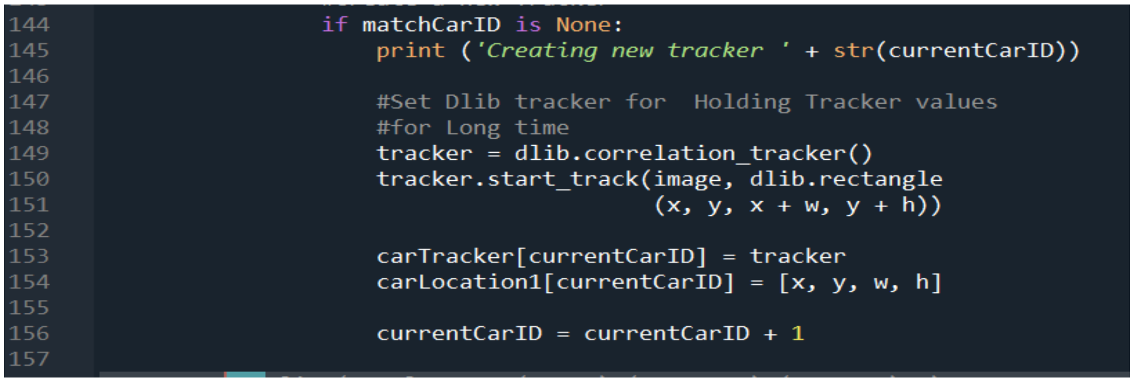
```python
#Track Position of car
for carID in carTracker.keys():
    trackedPosition = carTracker[carID].get_position()

    t_x = int(trackedPosition.left())
    t_y = int(trackedPosition.top())
    t_w = int(trackedPosition.width())
    t_h = int(trackedPosition.height())

    t_x_bar = t_x + 0.5 * t_w
    t_y_bar = t_y + 0.5 * t_h

    #Check for Matched Car ID in frame
    if ((t_x <= x_bar <= (t_x + t_w)) and
        (t_y <= y_bar <= (t_y + t_h)) and
        (x <= t_x_bar <= (x + w)) and
        (y <= t_y_bar <= (y + h))):
        matchCarID = carID
```

Fig 19: Declaring MatchCarID

- once the car is Detected the position of the tracked car is grabbed((x,y) coordinates, length, and width)
- Again Calculate the centroid of the position of the car
- check whether the centroids of the detected car and position of the car are approximately the same
- if it is same match the carIDs

7.7 ASSIGN CAR ID

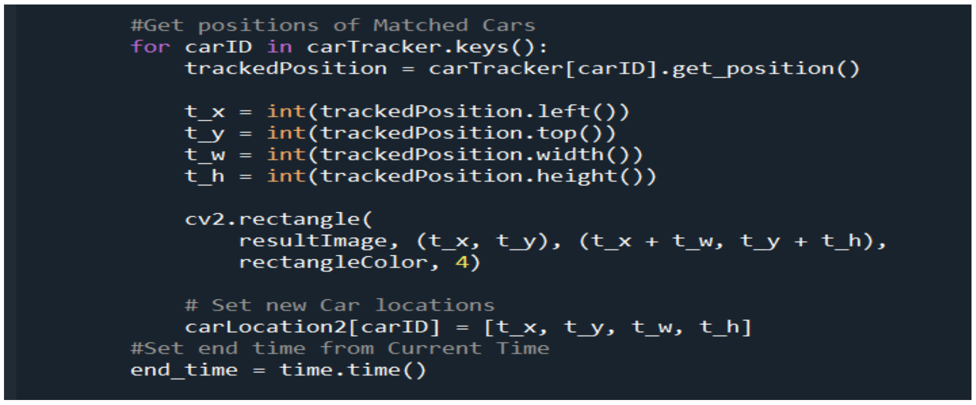
If the Car ID is Not matched then we have to track the detected car again.



```python
if matchCarID is None:
    print ('Creating new tracker ' + str(currentCarID))

    #Set Dlib tracker for  Holding Tracker values
    #for Long time
    tracker = dlib.correlation_tracker()
    tracker.start_track(image, dlib.rectangle
                        (x, y, x + w, y + h))

    carTracker[currentCarID] = tracker
    carLocation1[currentCarID] = [x, y, w, h]

    currentCarID = currentCarID + 1
```

Fig 20: Assigning Car ID

- Using dlib tracker track the car detected by using coordinates of the detected car.
- Assign a New Car id to Detected Car.

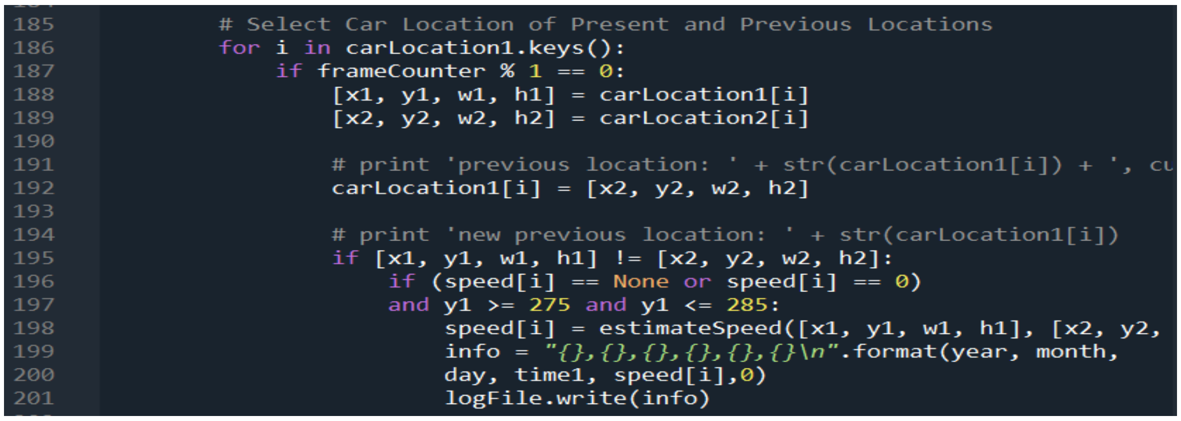


```python
#Get positions of Matched Cars
for carID in carTracker.keys():
    trackedPosition = carTracker[carID].get_position()

    t_x = int(trackedPosition.left())
    t_y = int(trackedPosition.top())
    t_w = int(trackedPosition.width())
    t_h = int(trackedPosition.height())

    cv2.rectangle(
        resultImage, (t_x, t_y), (t_x + t_w, t_y + t_h),
        rectangleColor, 4)

    # Set new Car locations
    carLocation2[carID] = [t_x, t_y, t_w, t_h]
#Set end time from Current Time
end_time = time.time()
```

Fig 21: Assigning New Car Id

Now get the positions of the tracked images and find the exact location at which the car is present.

7.8 LOAD DATA TO CSV



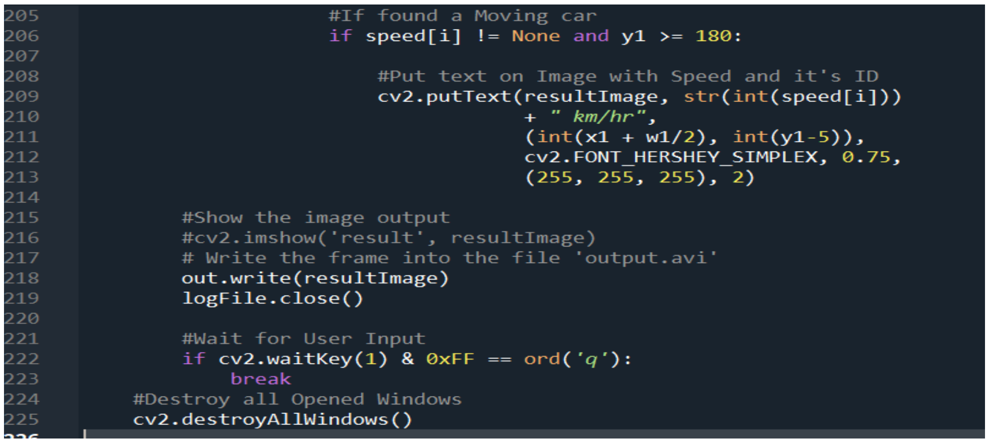
```python
        # Select Car Location of Present and Previous Locations
        for i in carLocation1.keys():
            if frameCounter % 1 == 0:
                [x1, y1, w1, h1] = carLocation1[i]
                [x2, y2, w2, h2] = carLocation2[i]

                # print 'previous location: ' + str(carLocation1[i]) + ', cu
                carLocation1[i] = [x2, y2, w2, h2]

                # print 'new previous location: ' + str(carLocation1[i])
                if [x1, y1, w1, h1] != [x2, y2, w2, h2]:
                    if (speed[i] == None or speed[i] == 0)
                    and y1 >= 275 and y1 <= 285:
                        speed[i] = estimateSpeed([x1, y1, w1, h1], [x2, y2,
                        info = "{},{},{},{},{},{}\n".format(year, month,
                        day, time1, speed[i],0)
                        logFile.write(info)
```

Fig 22: Loading Data to CSV

The above code says to get the location of the cars in the previous frame on the present frame in a span of 1 second

now give this location to the speed estimation function to calculate the distance based on these two locations.

Update the CSV file with cars speed and detected time



```
                    #If found a Moving car
                    if speed[i] != None and y1 >= 180:

                        #Put text on Image with Speed and it's ID
                        cv2.putText(resultImage, str(int(speed[i]))
                                    + " km/hr",
                                    (int(x1 + w1/2), int(y1-5)),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                                    (255, 255, 255), 2)

        #Show the image output
        #cv2.imshow('result', resultImage)
        # Write the frame into the file 'output.avi'
        out.write(resultImage)
        logFile.close()

        #Wait for User Input
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    #Destroy all Opened Windows
    cv2.destroyAllWindows()
```

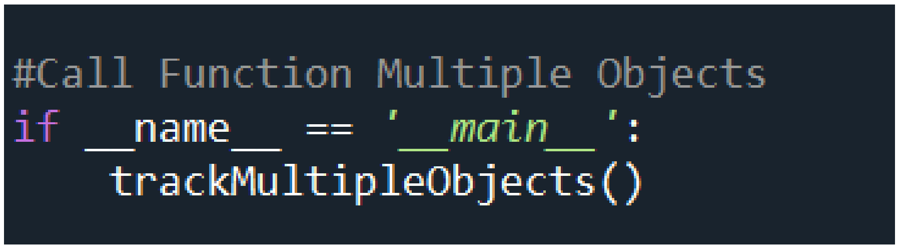
Fig 23: Updating the CSV files

Showcase the Output Video frame on the output window with text displayed on the window.

Press Q to stop video analysis.

7.9 VISUALIZE THE RESULT

Now run the code



```
#Call Function Multiple Objects
if __name__ == '__main__':
    trackMultipleObjects()
```

Fig 24: Visualizing the Result

- Save the file in the project folder
- Select the whole code and click SHIFT+Enter, a New window opens which visualizes the analysis
- you can also open CSV to see the log.

8. BUILD FLASK APPLICATION

In this section, we will be building a web application to stream the Speed estimation of vehicles traveling on the High way roads.

To complete this task you have to build the following files:

- HTML File
- webstreamingfile.py

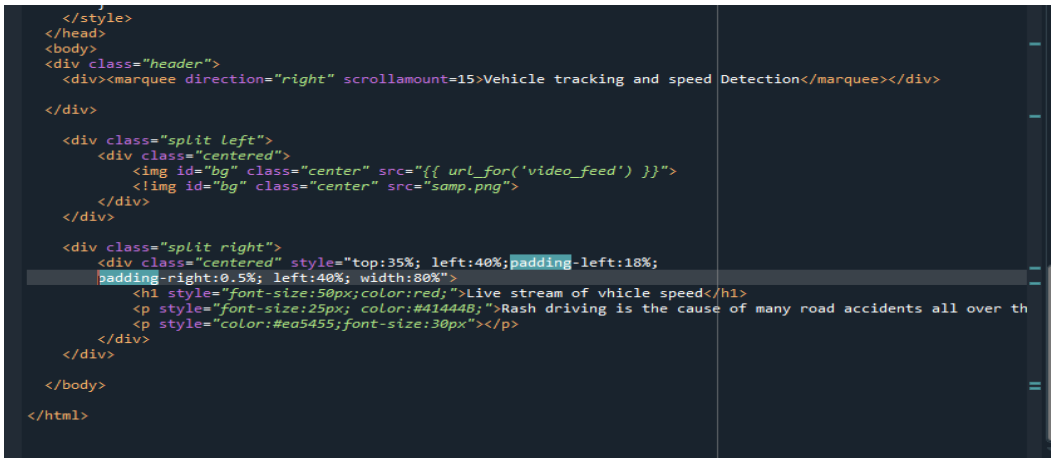
To know more about flask please watch below video

Flask: Web framework used for building Web applications.

8.1 BUILD HTML PAGE

In this section, we will be building an HTML which streams the video of vehicle detection and speed estimation in a local browser.

You can download the HTML page from the reference link.



```
    </style>
  </head>
  <body>
  <div class="header">
    <div><marquee direction="right" scrollamount=15>Vehicle tracking and speed Detection</marquee></div>

  </div>

    <div class="split left">
        <div class="centered">
            <img id="bg" class="center" src="{{ url_for('video_feed') }}">
            <!img id="bg" class="center" src="samp.png">
        </div>
    </div>

    <div class="split right">
        <div class="centered" style="top:35%; left:40%;padding-left:18%;
        padding-right:0.5%; left:40%; width:80%">
            <h1 style="font-size:50px;color:red;">Live stream of vhicle speed</h1>
            <p style="font-size:25px; color:#41444B;">Rash driving is the cause of many road accidents all over th
            <p style="color:#ea5455;font-size:30px"></p>
        </div>
    </div>

  </body>

</html>
```

Fig 25: Building HTML Page

src="{{ url_for('video_feed') }}" : we are fetching the URL from where your video strems on web browser

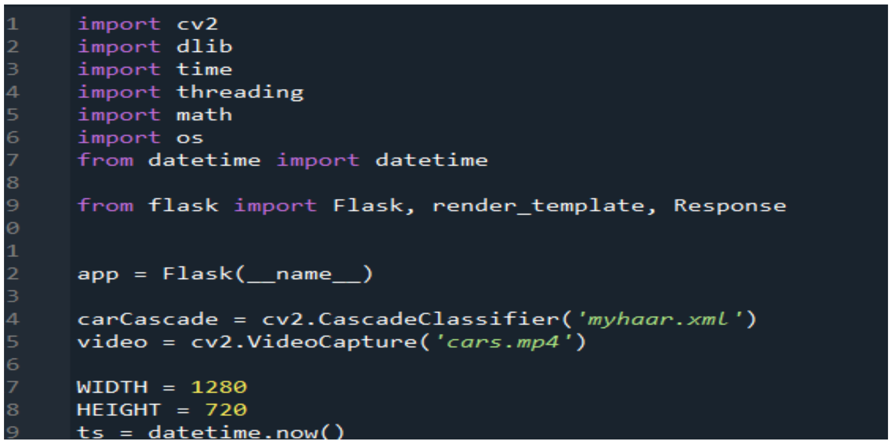
8.2 BUILD PYTHON CODE

In the section, you will be integration a python scrip to render the HTML page on to web browser to stream video

Create webstream.py file and follow the below instructions.

We will be using python for server-side scripting. Let’s see step by step process for writing backend code. The importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as an argument.

Load the necessary initializations and files needed for Vehicle detection and Speed Estimation

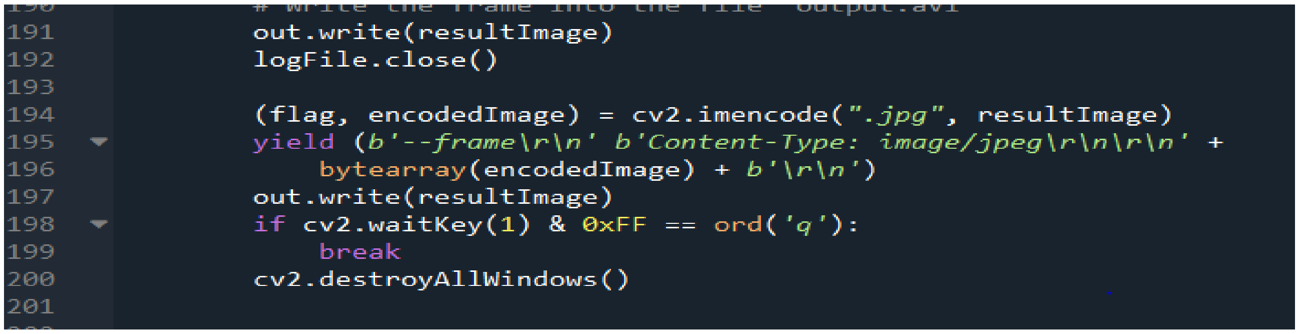


```
import cv2
import dlib
import time
import threading
import math
import os
from datetime import datetime

from flask import Flask, render_template, Response


app = Flask(__name__)

carCascade = cv2.CascadeClassifier('myhaar.xml')
video = cv2.VideoCapture('cars.mp4')

WIDTH = 1280
HEIGHT = 720
ts = datetime.now()
```

Fig 26: Import Necessary Libraries

Create two functions for speed estimation and track multiple objects as we did in the previous section

Add the below lines at the end of the track multiple objects function to encode the frames to stream on the webpage



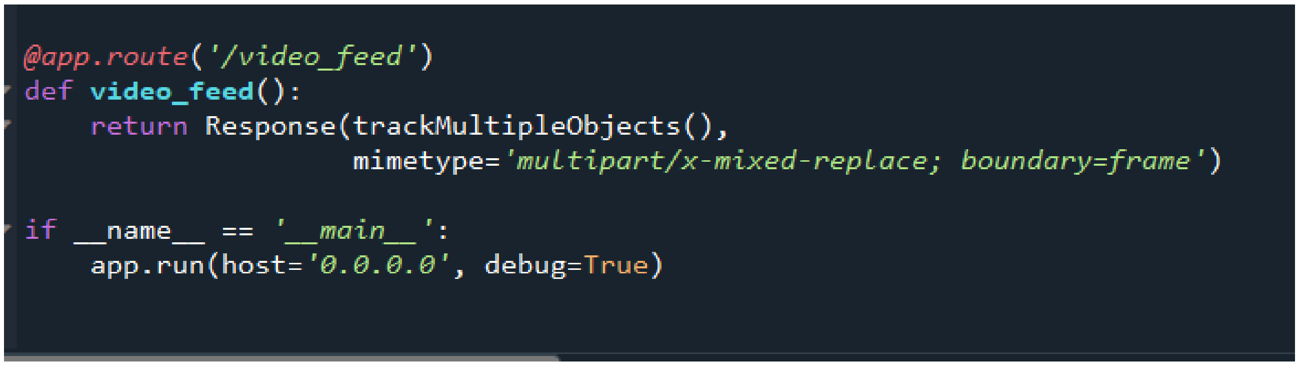
```
        # write the frame into the file 'output.avi'
        out.write(resultImage)
        logFile.close()

        (flag, encodedImage) = cv2.imencode(".jpg", resultImage)
        yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' +
            bytearray(encodedImage) + b'\r\n')
        out.write(resultImage)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        cv2.destroyAllWindows()
```

Fig 27: Tracking Multiple Objects

Here we will be using declared constructor to route to the HTML page which we have created earlier.

Route the encoded frames on to HTML Page created

Create a main function to run the flask app on the webserver



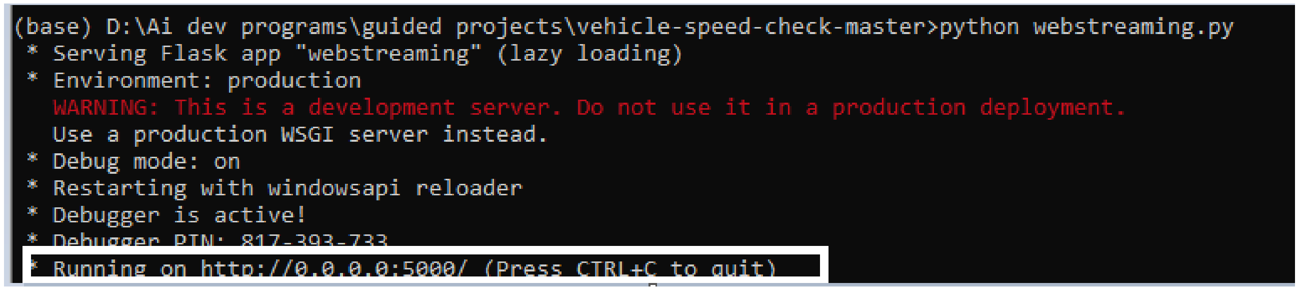
```
@app.route('/video_feed')
def video_feed():
    return Response(trackMultipleObjects(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

Fig 28: Main Function in Flask.

Save the file and let's run the Application

8.3 RUN FLASK APP

- Open anaconda prompt/command from the start menu
- Navigate to the folder where your webstreaming.py resides
- Now type "python webstreaming.py" command
- It will show the local host where your app is running.
- Navigate to the localhost on the browse where you can view your web page.

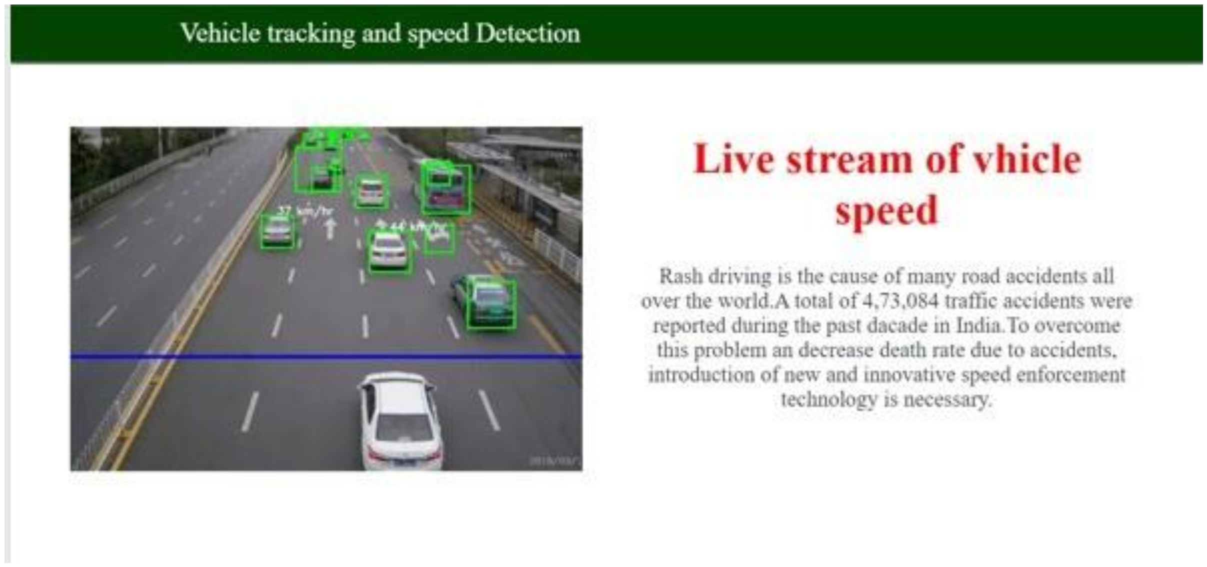


```
(base) D:\Ai dev programs\guided projects\vehicle-speed-check-master>python webstreaming.py
 * Serving Flask app "webstreaming" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 817-393-733
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Fig 29: Running Flask App through Local Host

9. RESULT

FINAL OUTPUT OF THE PROJECT


Vehicle tracking and speed Detection

Live stream of vhicle speed

Rash driving is the cause of many road accidents all over the world.A total of 4,73,084 traffic accidents were reported during the past dacade in India.To overcome this problem an decrease death rate due to accidents, introduction of new and innovative speed enforcement technology is necessary.

Fig 30: Final Output of Vehicle Speed Detection

10. CONCLUSION

The presented video-based speed measurement model provides the probability density function of a passing vehicle's speed based on its movement pattern vector. In this work, the proposed method was implemented using four intrusion lines, and the input frames were captured simultaneously by an off-the-shelf handheld camera and a smartphone device with frame rates of 50 fps and 30 fps, respectively. According to the experimental results, the average error rates of the speed measurement system at 50 fps and 30 fps were 1.77% and 2.17%, respectively, for vehicles travelling in the range of 70 km/h–100 km/h. The error rate decreased noticeably with an increase in the frame rate, as expected. In addition, the actual speeds of the vehicles were within the range of the estimated pdfs, increasing the confidence in the model. It was also observed that the ranges of obtained pdfs often decreased with the increase in the camera frame rate. Furthermore, the proposed model was employed to measure the speed of more than 670 vehicles, and the results were compared with the speed limits with respect to the vehicles categories passing through the measurement site. The experimental results demonstrated that the detected speeds and the speed limits were highly correlated. Theses correlations validate the proposed model since the majority of heavy vehicles (e.g., buses, trucks and trailers) are equipped with speed regulators forcing them to follow the speed limits. It can be concluded that the proposed video-based model performs with high precision and robustness when measuring the speeds of passing vehicles. Future work should focus on utilizing a depth map in order to initiate the intrusion lines automatically.

11. BIBLIOGRAPHY

[1] T. V. Mathew, “Intrusive and non-intrusive technologies,”Indian Institute of Technology Bombay, Tech. Rep., 2014.

[2] N. Buch, S. Velastin, and J. Orwell,“A review of computer vision techniques for the analysis of urban traffic,” IEEE Trans.on Intelligent Transportation Systems, vol. 12, no. 3, pp. 920–939, Sept 2011.

[3] J. Shi and C. Tomasi, “Good features to track,” in IEEE Int. Conf.on Computer Vision and Pattern Recognition (CVPR), 1994, pp. 593–600.

[4] B. D. Lucas and T. Kanade,“An Iterative Image Registration Technique with an Application to Stereo Vision,”Joint Conference on Artificial Intelligence, pp. 674–679, 1981.