



Image Analysis and Computer vision

VISUAL ANALYSIS OF MOVING VEHICLES

Supervisor: Professor Vincenzo Caglioti

Contributors: Ghodrat Rezaei 10695039 – Hammad Naseem 10754026

Erfan AliMohammadi 10819936

Jan 2022

Vehicle Flow Computation (F04):

Problem:

In any major traffic flow area ranging from highways/motorways to city centers, two main issues are the speed of the moving vehicles and the check on number of vehicles entering/exiting a certain area. Where high speed of the vehicles may lead to traffic accidents and loss of the life and property, on the other hand, unaccountable flow of vehicles in a certain area may lead to grid-lock traffic jams resulting in loss of valuable time and resources for the commuters.

The use of computer vision provides a very inexpensive way of detecting the speed of the cars while at the same time enables the system to monitor the car over speeding, making it easier and inexpensive to fine the offender as compared to the traditional method where the highway patrolling cars detect the over speeding cars and follow the particular car in order to fine it. In addition to this, the monitoring of flow of cars in and out of a certain region via counter, helps us to monitor and effectively predict the traffic flow situation in the city. Here, relevant authorities can monitor the flow of cars and based on those readings, can recommend the commuters to adopt an alternative route, allowing the diversification of traffic flow and avoiding the traffic jams.

State outline of the art:

Common Techniques used for Object Detection are classical computer vision techniques and Deep learning approaches. In classical computer Vision, the background removal technique is used to detect object. The idea of background removal is not a modern one; instead, it has existed in the field since the dawn of modern technology. Hence, there have been numerous techniques for background removal in the domain of computer vision. Nevertheless, classical computer vision techniques are constantly being overthrown by modern, artificial intelligence-based ones by lodging more innovative ideas giving more precise and accurate results. In modern approaches toward object detection using Artificial Neural Network, objects can be detected by higher accuracy with respect to classical computer vision techniques. However we used both methods to detect object.

Using cross ratio invariant principle, along with the position of the two reference points, vanishing point of the image and the known real-world values, we can estimate the distance of the car and the marker in the real world. By using this change in distance from the subsequent images, we can calculate the distance travelled and hence the speed of the individual vehicle.

Description of the Implementation:

The implementation of the computer vision and image analysis technique to tackle our problem statement is done in the following steps:

Image Preprocessing:

Initially we extracted a video taken from a street camera of a 3-lane street where lane markers are visible, and vehicles are traveling at high speed. As part of the image preprocessing part, firstly, the video was converted into the set of images with frequency of 30 frames per second which gives use with a total of 150 images during the span of random 15 seconds analysis from the input video. The conversion of video to images was done using the tool libraries [FFMPEG](#) and [Magick](#) on Linux. After converting the video into images, the image editing was carried out in which borders of the images were edited to keep the focus on the street and till the greatest of extent, subtract the other things happening in the video (like visibility of other streets, etc.) which can later be a cause of disturbance in our analysis leading to incorrect values.

Vehicle detection

Classical Computer Vision Method (background Removal):

In this Method, each image first was converted to gray scale. Having gray scale of each frame, each pixel value was subtracted from value of corresponding pixels in other images. In pixels with high values of differences between image frames, color of corresponding pixels leads to white color. Similarly for pixels with less difference of values, color will be more toward dark. After implementing such mask on images frame, we set threshold, upon which color can be white and lower values will be considered as dark pixel. This threshold value was defined 60 (pixel color value) in our methodologies and was set experimentally to cover larger areas which have meaningful discrepancies. One important that issue to be solved is to allocate to each vehicles a unique index which is identifiable among other object in different image frame. To solve this issue, greedy approach was implemented. Other innovative and new solution can be [marriage problem or maximum weighted bipartite matching](#). Marriage problem is tracking each unique object in sequence of image frames. Challenge of object detection appears when we have object with similar size and shape in different image frames. To solve this problem, global optimization algorithm can be implemented over whole sequence of image frame. This global optimization algorithm has sub-optimization part in each image frame. To obtain stable tracking results, we propose a tracking method based on global optimization. Particularly, we first detect each individual vehicle in each image frame by background removal method, then formulate the multiple objects tracking problem as a combinatorial optimization problem over a pair of consecutive frames and solve the problem by the greedy approach algorithm. The target would be allocating a unique index to each unique vehicle in image frames which it is appeared, so that overall score of allocations is maximum over the image frames or equivalently total dissimilarity of objects over images gets minimum.

Two sequential image frames are shown in figure 1 and 2, and corresponding graph algorithm and greed search are shown in figure 3 and 4.

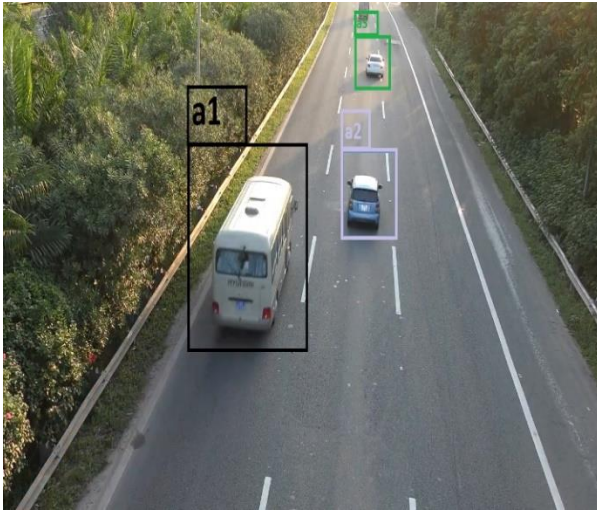


Figure 1

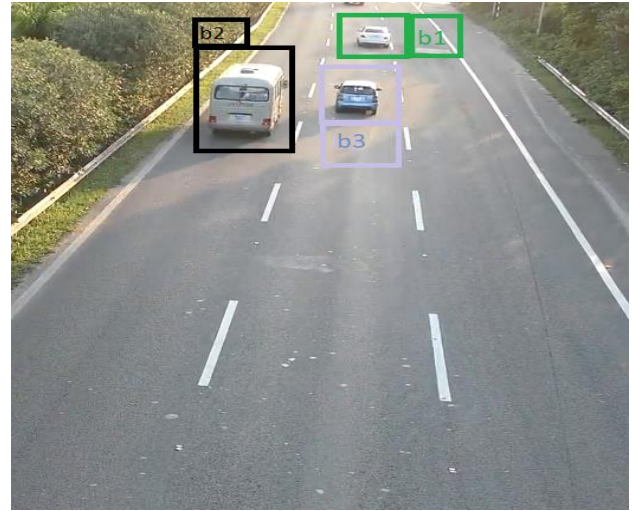


Figure 2

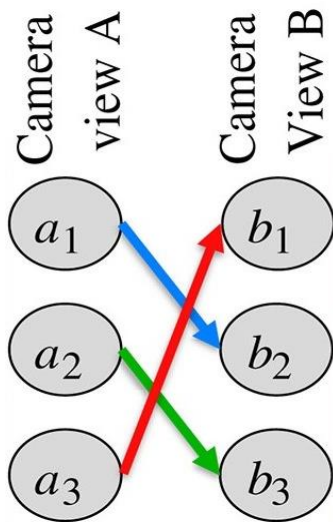


Figure 3

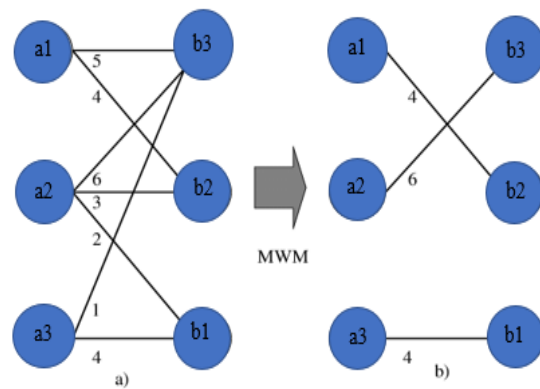


Figure 4

Deep Learning Methods:

Object detection using deep learning method is faster and accurate. In this project YOLO version 5 was implemented, which is now the most advanced object identification algorithm available. It is a

novel convolutional neural network (CNN) that detects objects in real-time with great accuracy. This approach uses a single neural network to process the entire picture, then separates it into parts and predicts bounding boxes and probabilities for each bounding box candidate (patch). These bounding boxes are weighted by the expected probability. So, we are dealing with both classification and regression problem; classification of object and assign to it a certain label which are preserved during whole image frame sequences and regression which certify the coordinates of the object (length and wide of bounding box and coordinates of upper right part of rectangular). Using a pre-trained model, we implement transfer learning and fine tuning to train whole multi-Layer-perceptions and potions of convolutional layers. To be mentioned that we did not use the label of objects, because it is not important in this project. To match certain objects in each image frame, greedy approach was implemented which was explained in previous method.

Speed Calculation

Once the vehicles is detected using the artificial intelligence, the phenomena of cross ratio is used in order to calculate the speed of the vehicle. The cross ratio of tuple points used the ratio of 4 points in image plane and 4 points in the planar view and use the ratio of those points which is always equal to -1 (figure 5).

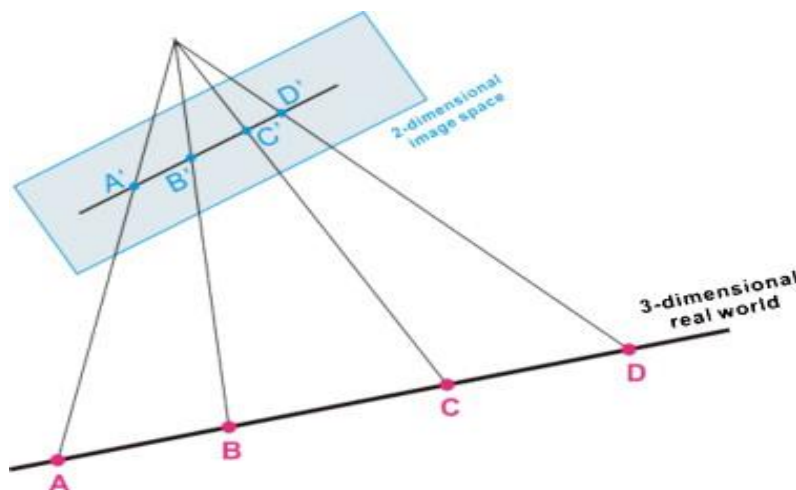


Figure 5

In the real-world planar view, point A represents the point at infinity, whereas points B and D represents the Forward Marker and Rear Marker respectively on the road and point C represents the real-world position of the car. Similarly, the corresponding values of these points are projected on the

2-dimensional image plane where the point A' represents the vanishing point of the image, B' and D' represents the position of the marker and C' represents the position of the center of the car in the image.

From the 3-Dimensional real world perspective, we have the point A located at infinity whereas the distance between the front and the rear marker (B-D) is known to use which is equal to 52m. The distance we are calculating is the distance of the car from the front marker.

On the other hand, in the image space, point A' which represents the vanishing point of the image is calculated by combining the parallel line features of the vehicle moving direction and the vehicle feature. The pair of parallel lines gives us the vanishing point. Here we assume that since the vehicle is travelling in the straight direction within our region of study, the vanishing point would remain the same from image to image. However, to reduce the error in this assumption to a significant level, we calculated the vanishing point using the average of multiple vanishing points of the images we are studying. Points B' and D' which represents the front marker and the rear marker respectively on the image plane, would also remain the same as the position of the street and hence the lane markers remain the same with respect to the camera with which the video is being taken. The point C', which represents the position of the vehicle on the image, is taken from the midpoint of the box position in which the vehicle is detected. The box of the image is extracted from the first part of the project related to the vehicle detection part. Using these values along with the relevant formulas given below, we were able to estimate the distance of the car from the front lane marker.

Once we know the distance travelled by a particular car from one image to the following one, along with the time taken which is given by the FPS frequency with which we converted the video to the image, we were able to estimate the speed of the vehicle (eq. 1).

$$(A B C D) = \frac{|AC*BD|}{|BC*AD|} \quad (\text{eq. 1})$$

$$(A B C D) = (A' B' C' D')$$

In our code, the following notations were used along with the final equation(eq.2);

$$d = \frac{|Va-MR|*|MF-Cc|}{|MF-MR|*|Va-Cc|} * 52 \text{ m} \quad (\text{eq.2})$$

d = Distance of the car from the front marker (m)

Va = Vanishing point of the image scene

MR = Marker rear in the image

MF = Marker front in the image

Cc = Center of the car in the image

Experimental Procedures:

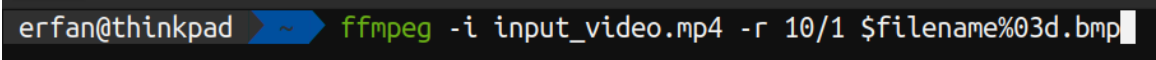
0. Assumptions

Due to unknown camera position with respect to the world reference plane in addition to unknown camera parameters, our evaluation has some limitations as few assumptions are considered in the methodology applied

- 1) Since the vanishing point is extracted using the features (parallel lines) of both the lane marker and vehicle features, it goes without saying that we assume that the vehicles are running perfectly in direction of the street whereas the results of vehicles changing the lanes may be affected by this assumption
- 2) Since for calculating the speed of the vehicles, we are using the average distance travelled in a certain time segment, the speed of the vehicle is the average speed hence we assume that the vehicle is traveling at a constant velocity.
- 3) Since the point at infinity of the image is taken to be fixed as the position of camera is fixed, the point cannot be completely accurate due to the limitation of quality of image and human error. However, to mitigate that effect, we did take the average of point at infinity of random images.

1. Conversion of Video to JPEG:

As very first step we converted a video of 30 minutes to images with frequency of 30 frames per second. The conversion of video to images was done using the tool libraries [FFMPEG](#) and [Image Magick](#) on Linux. The command below was executed on the Linux to convert the video to the images (figure 6). For the ease of experimentation, we are just considering the time slot of 15 seconds, hence the consecutive number of images we study are 150.



```
erfan@thinkpad ~$ ffmpeg -i input_video.mp4 -r 10/1 $filename%03d.bmp
```

Figure 6

After extracting the images, we can use them in our next parts of experimentation.

2. Vehicle Detection:

2.1. Background Removal:

As explained in previous parts, background removal is implemented over gray scale image, and then a threshold of 60 was defined to separate the image into two parts: white part mismatch pixels and dark parts for matched pixels. Whole procedure was implemented using Python which is shown in figure below (figure 7). To track each unique vehicle along image frames, a greedy approach method was used which is a well approximated approach of marriage problem or maximum weighted bipartite matching.

For Running the code, following code lines should be ran.

```
pip3 -r requirements.txt
python3 main.py
```

```
import os
import re
import cv2
import numpy as np
from os.path import isfile, join
import matplotlib.pyplot as plt

images_filename_list = os.listdir("video_frames/")
images_filename_list.sort(key=lambda f: int(re.sub("\D", "", f)))

images_list = []

for filename in images_filename_list:
    image = cv2.imread("video_frames/" + filename)
    images_list.append(image)

for i in range(len(images_list) - 1):
    grayscale_image_1 = cv2.cvtColor(images_list[i], cv2.COLOR_BGR2GRAY)
    grayscale_image_2 = cv2.cvtColor(images_list[i + 1], cv2.COLOR_BGR2GRAY)
    diff_image = cv2.absdiff(grayscale_image_2, grayscale_image_1)
    ret, thresholded_image = cv2.threshold(diff_image, 60, 255, cv2.THRESH_BINARY)

    dilation_kernel = np.ones((1,1), np.uint8)
    dilated_image = cv2.dilate(thresholded_image, dilation_kernel, iterations=1)

    plt.imshow(dilated_image, cmap="gray")
    plt.show()

    contours, hierarchy = cv2.findContours(thresholded_image.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

    objects_centroid_list = []

    for j, object_centroid in enumerate(contours):
        x, y, w, h = cv2.boundingRect(object_centroid)
        if cv2.contourArea(object_centroid) >= 100:
            objects_centroid_list.append(object_centroid)

    cv2.drawContours(images_list[i], objects_centroid_list, -1, (255, 0, 0), 2)
    # plt.imshow(images_list[i])
    plt.imshow(cv2.cvtColor(images_list[i], cv2.COLOR_BGR2RGB))
    plt.show()
```

Figure 7

Background Removal does not yield accurate detection of the objection. As shown in figure 7 and 8, all parts of vehicles are not detected and the detected parts are not consistent, meaning that the detection accuracy is not so high.

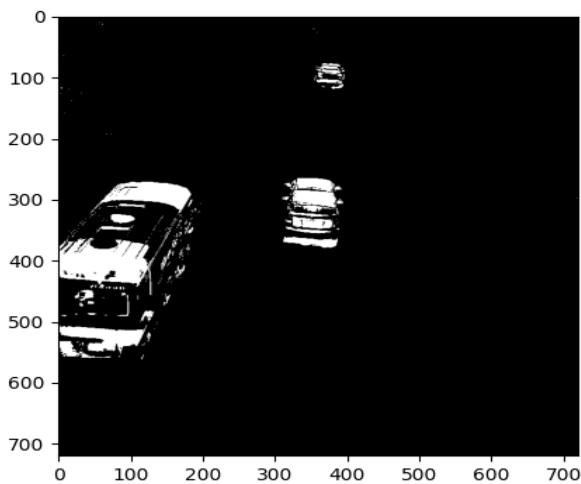


Figure 8

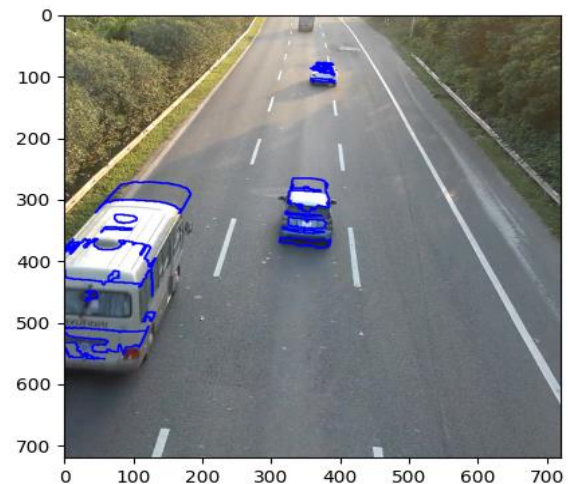


Figure 9

2.2. Deep Learning (YoloV5):

Using yolo for object Detection allows us to get the bounding box which is maximally matched with the object. As explained before, we use a pretrained model and did fine tuned it to the get the final network model for object detection. Input is image frame, and output is bounding box with defined circle and coordinates (figure 10). Function for bounding box of vehicles is shown in figure 11. Figures 12 and 13 are two examples of the output from the object detection by YoloV5.

```

# YOLOv5 by Ultralytics, GPL-3.0 license

import argparse
from cmath import rect
import sys
import time
from pathlib import Path

import cv2
import numpy as np
import torch
import torch.backends.cudnn as cudnn

FILE = Path(__file__).absolute()
sys.path.append(FILE.parents[0].as_posix()) # add yolov5/ to path

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, colorstr, is_ascii, non_max_suppression, \
    apply_classifier, scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path, save_one_box
from utils.plots import Annotator, colors
from utils.torch_utils import select_device, load_classifier, time_sync

MODEL_FILE_PATH = "yolo_model.pt"
INPUT_DIRECTORY_PATH = "video.mp4"

def center_of_rectangle(rectangle):
    return ((rectangle[0] + rectangle[2]) / 2.0, (rectangle[1] + rectangle[3]) / 2.0)

def is_point_inside_rectangle(point, rectangle):
    if point[0] < rectangle[0]:
        return False
    if point[1] < rectangle[1]:
        return False
    if point[0] > rectangle[2]:
        return False
    if point[1] < rectangle[3]:
        return False
    return True

```

Figure 10

```

def process_bounding_boxes(bounding_boxes_list, image):
    global points_list_by_time
    global total_objects
    image_width = image.shape[1]
    image_height = image.shape[0]
    bounding_boxes_list = preprocess_bounding_boxes_list(bounding_boxes_list, image_width, image_height)
    points_list_by_time = preprocess_points_list_by_time(points_list_by_time, image_width, image_height)
    cv2.rectangle(image, (0, int(0.5 * image_height)), (image_width, image_height), (0, 255, 255), 5)
    cv2.rectangle(image, (0, int(0.5 * image_height)), (int(0.3 * image_width), int(0.5 * image_height) + int(0.06 * image_height)), (255, 255, 255), -1)
    cv2.putText(img=image, text="Number of cars: " + str(len(bounding_boxes_list)),
                org=(int(0.025 * image_width), int(0.5 * image_height) + int(0.045 * image_height)), fontFace=cv2.FONT_HERSHEY_TRIPLEX,
                fontScale=image_width / 1280, color=(0, 0, 0), thickness=int(2 * image_width / 1280 + 0.5))
    updated_lists_dict = {}
    points_list = []
    for i in range(len(bounding_boxes_list)):
        points_list.append(center_of_rectangle(bounding_boxes_list[i]))
    for i in range(len(points_list)):
        min_distance_index = -1
        for j in range(len(points_list_by_time)):
            if near_points(points_list[i], points_list_by_time[j][-1], 50, 50, 50):
                if min_distance_index < 0 or distance(points_list[i], points_list_by_time[j][-1]) < distance(points_list[i], points_list_by_time[min_distance_index][-1]):
                    min_distance_index = j
        if min_distance_index != -1:
            updated_lists_dict[min_distance_index] = True
            points_list_by_time[min_distance_index].append(points_list[i])
            added_to_points_list_by_time = True
        else:
            total_objects += 1
            points_list_by_time.append([points_list[i]])
            updated_lists_dict[len(points_list_by_time) - 1] = True
    points_list_by_time = remove_unupdated_points(points_list_by_time, updated_lists_dict)
    print("")
    # print("total cars until now:")
    # print(total_objects)
    for i in range(len(points_list_by_time)):
        print("speed:")
        print(calculate_speed(start_point=np.concatenate((points_list_by_time[i][0], np.array([1]))),
                             end_point=np.concatenate((points_list_by_time[i][-1], np.array([1]))),
                             vanishing_point=np.array([762.764, 4.05, 1]),
                             rear_marker=np.array([569.25, 676.25, 1]),
                             front_marker=np.array([715.8028, 159.6927, 1]),
                             markers_distance=52,
                             time_delta=len(points_list_by_time[i]) * 1 / 30.0))

```

Figure 11

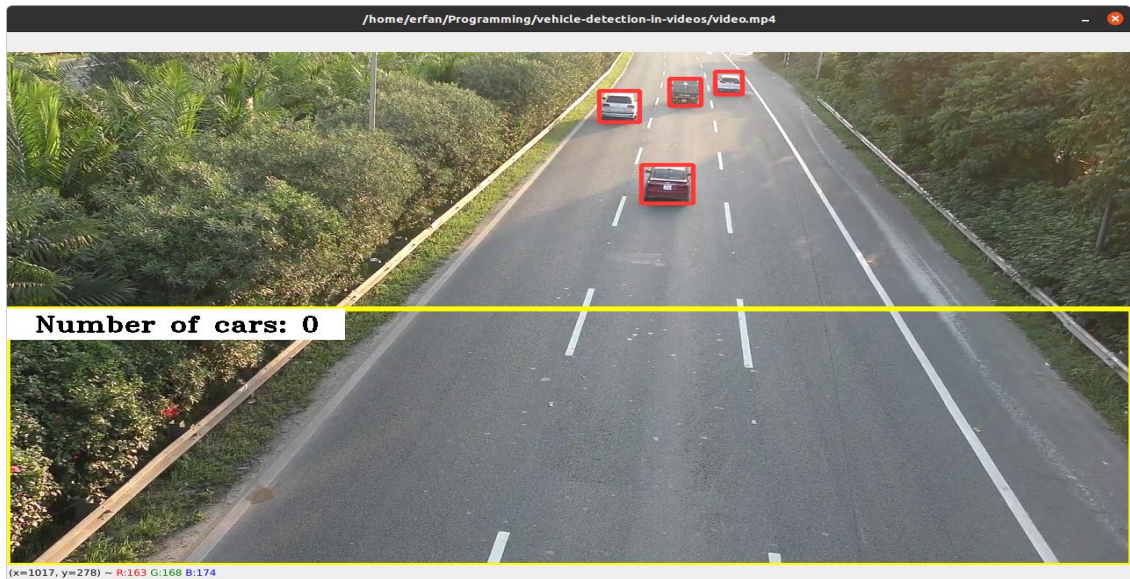


Figure 12

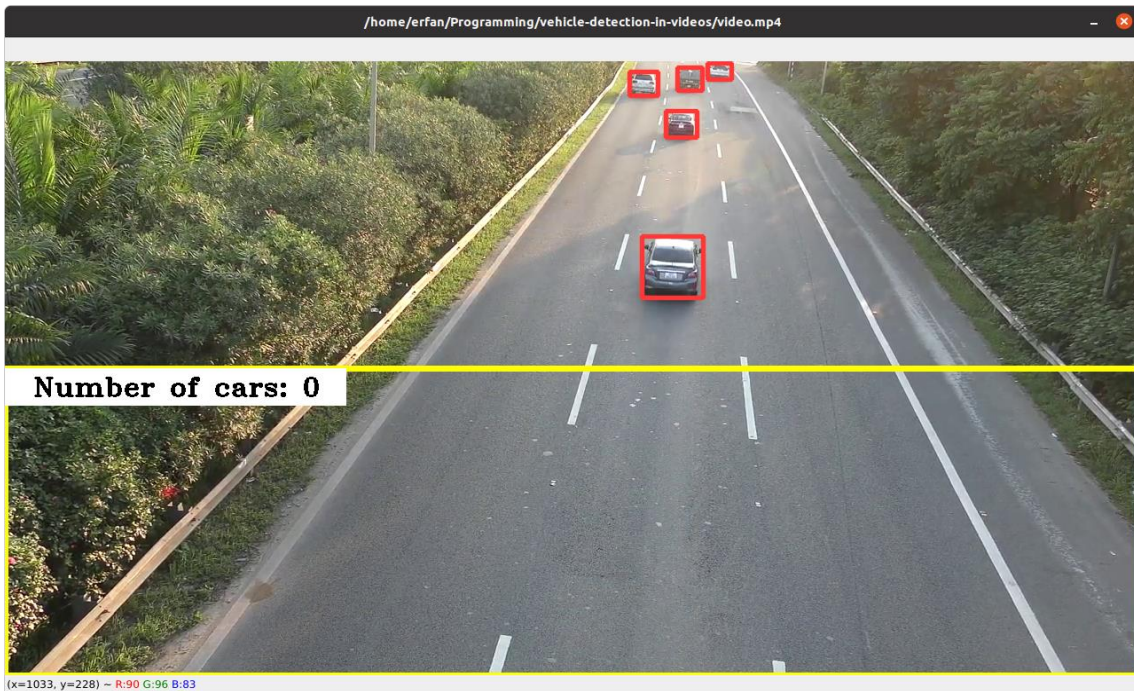


Figure 13

3. Vehicle Counting:

Vehicle counting is done inside function `Process_bounding_box` in figure 11 and is implemented when each unique vehicle is crossing the yellow marker which is shown in figure 14 and 15.

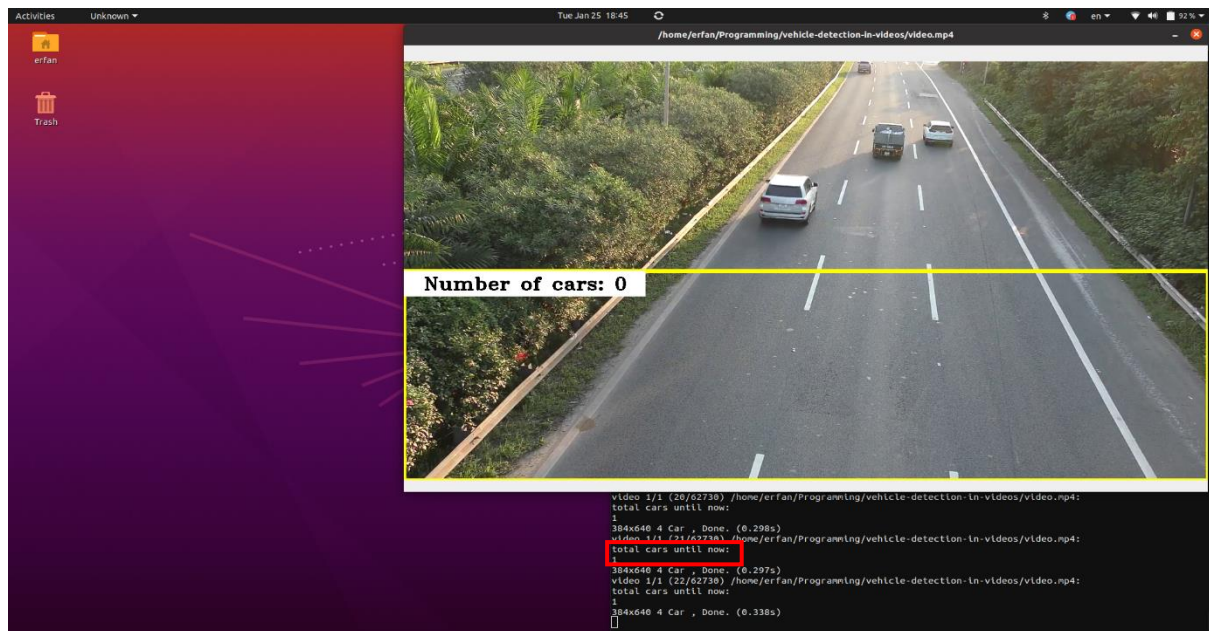


Figure 14

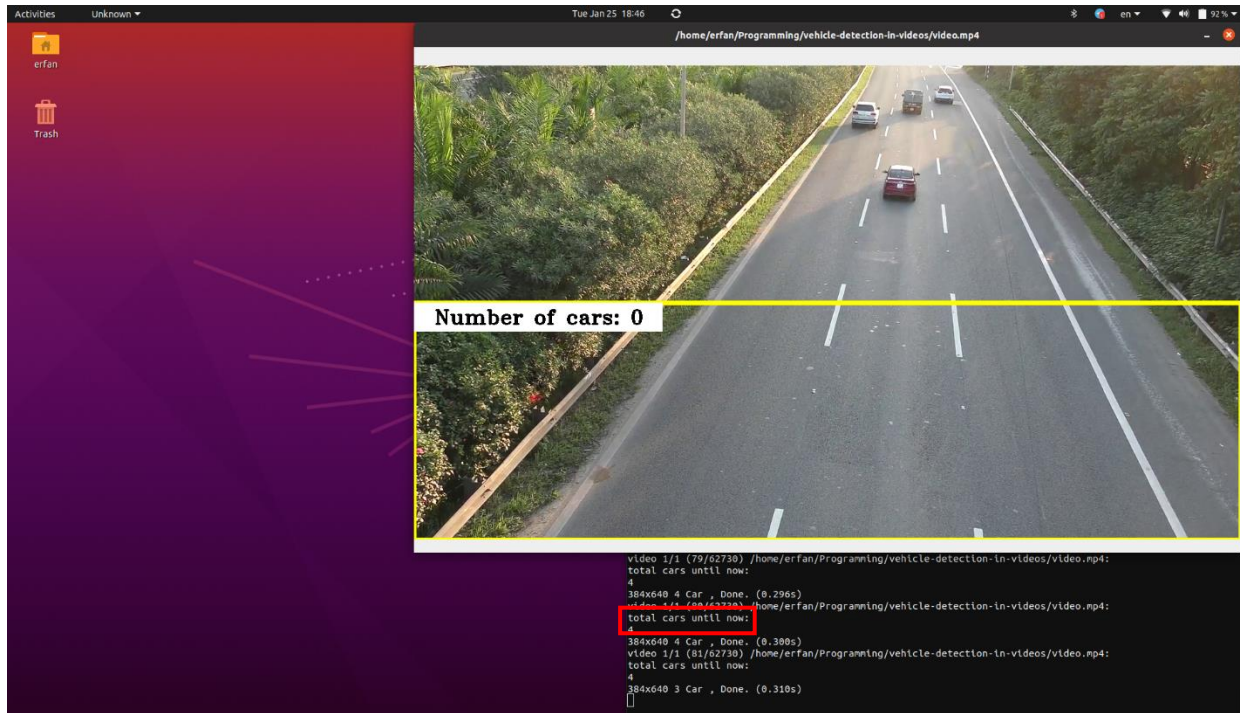


Figure 15

4. Average of vanishing point:

In order minimize the error of the vanishing point we estimated for our experimentation, we calculated vanishing points of random images and then took the average of it. This way any human error, or error due to selection of the parallel lines is minimized.

In figure 14 below, we constructed a pair of parallel lines, one using the vehicle feature and the other using the street feature.

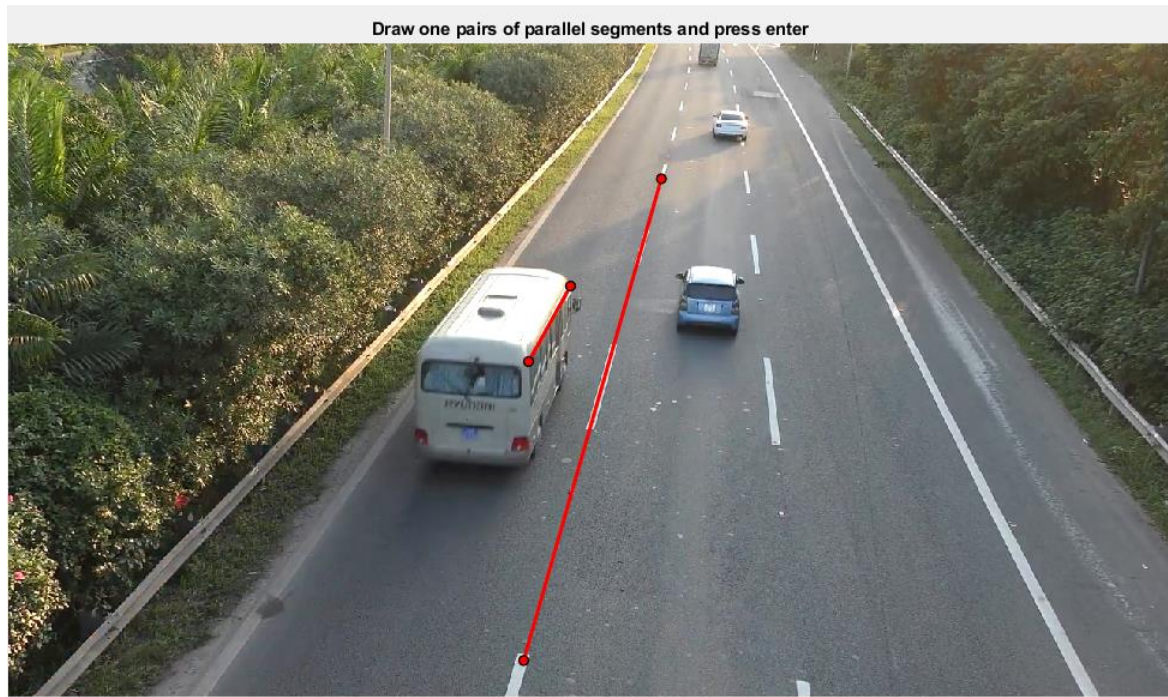


Figure 16

Following the construction of parallel lines, we can estimate the vanishing point of the image with the help of the intersection point of these two parallel lines. In the figure 15 below, we can see that the vanishing point is located at the top of the image, that is in the direction of vehicle movement. In addition to this, here we can also see the location of the markers located on the street which is taken as the reference point. The front marker is labelled as 'b' and the rear marker is labelled as 'a'. The distance between these two points is 52m which was calculated using the standard distance of the markers. The two marker points and the vanishing point is considered fixed for all the images; hence it is given as input to the system.



Figure 17

5. Speed Estimation

Using the vehicle detection ability, we can track the vehicles from one image to the subsequent ones. This feature, combined with the cross ratio of tuple points, where the four points of the image plane is compared with the real-world points (details explained in the methodology section above), we can estimate the distance of the detected vehicle and the reference markers. Since a particular vehicle is detected until it crosses the second marker on the street, we can get the time taken by the vehicle to travel the specified distance (between the two labelled markers). This allows us to estimate the average speed of that vehicle. Python code for mentioned procedure is shown in figure 12.


```

def calculate_speed(start_point, end_point, vanishing_point, rear_marker, front_marker, markers_distance=52, time_delta):
    vanishing_point = np.array([400, 50, 1])
    rear_marker = np.array([500, 500, 1])
    front_marker = np.array([500, 300, 1])

    vanishing_point_start_point_norm = np.linalg.norm(vanishing_point - start_point)
    front_marker_start_point_norm = np.linalg.norm(front_marker - start_point)
    vanishing_point_rear_marker_norm = np.linalg.norm(vanishing_point - rear_marker)
    front_marker_rear_marker_norm = np.linalg.norm(front_marker - rear_marker)

    start_point_distance = (vanishing_point_rear_marker_norm / front_marker_rear_marker_norm) * markers_distance / (vanishing_point_start_point_norm / front_marker_start_point_norm)

    vanishing_point_end_point_norm = np.linalg.norm(vanishing_point - end_point)
    front_marker_end_point_norm = np.linalg.norm(front_marker - end_point)
    vanishing_point_rear_marker_norm = np.linalg.norm(vanishing_point - rear_marker)
    front_marker_rear_marker_norm = np.linalg.norm(front_marker - rear_marker)

    end_point_distance = (vanishing_point_rear_marker_norm / front_marker_rear_marker_norm) * markers_distance / (vanishing_point_end_point_norm / front_marker_end_point_norm)

    return abs(end_point_distance - start_point_distance) / time_delta

```

Figure 18

In the following Figures (17,18), the average speed of each car is visible which is calculated using the time taken by the car to cross the two markers whose distance is pre-defined in the previous part.

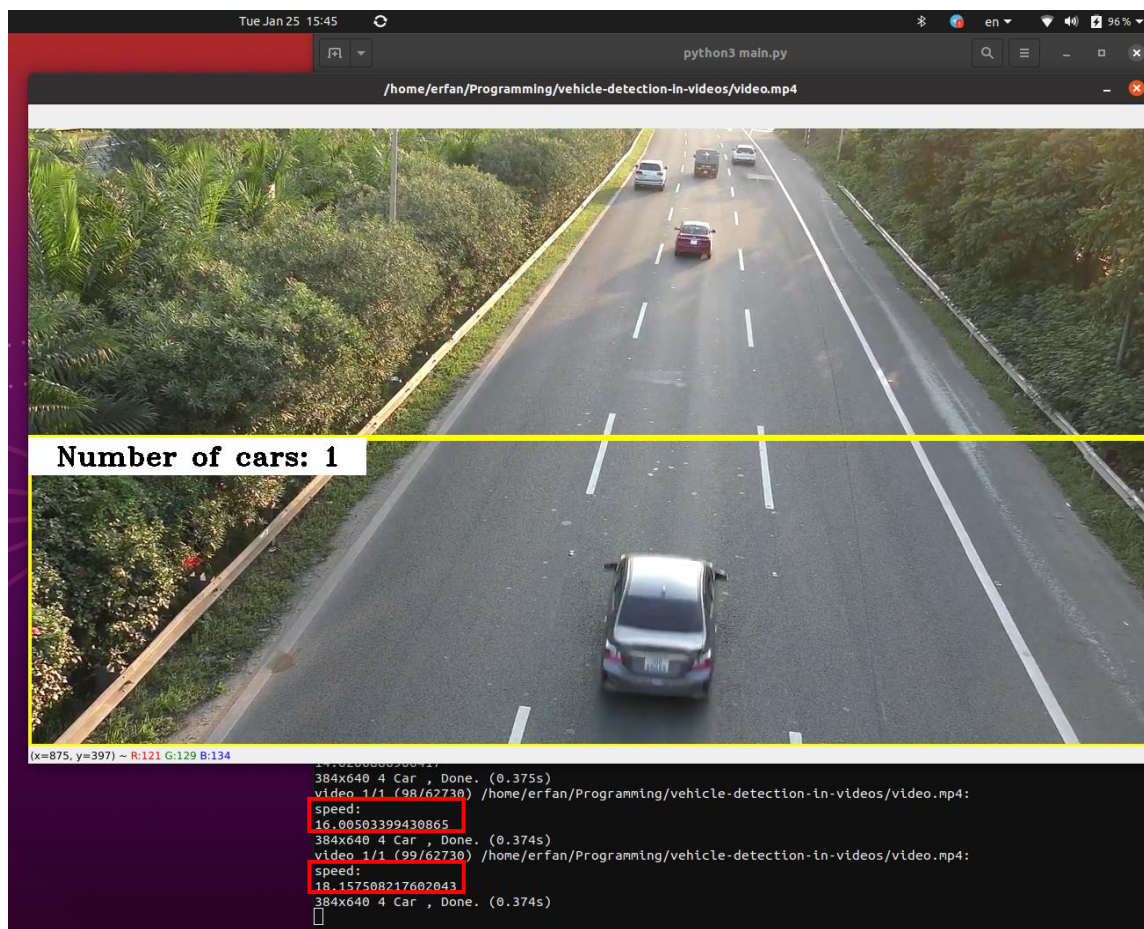


Figure 19

As can be seen in figure 17 and 18, average speed of each car has is shown in red rectangle and the unit is m/s. The speed values make sense in real-world. As an example, speed of 20 (m/s) is equal to 72(km/h) and in highway it is standard speed for car.

Conclusion

The results of the average speed estimated of the cars along with the counter value, is somehow acceptable. Furthermore, while implementing the cross-ratio algorithm, the distance between the front marker and the card was also checked with the estimated value of the real-world distance using the values of distance between the markers, which were also acceptable. This concludes that the methodologies implemented in this experimentation which includes the conversion of the video to images, followed by vehicle detection, estimation of vanishing points and tracking of the vehicle using the greedy approach some what gives an acceptable set of results.

However, due to the initial assumptions we considered, there may be some errors, specially for cars changing lane in between the region of study as it would no longer hold on to the assumption that the car is parallel to the street in the real world. Furthermore, there might be cars which are accelerating or decelerating in the region between two markers, resulting in incorrect average speed estimation.

Improvements can be brought to the experiment in many ways. Firstly, a better-quality camera with higher resolution would enable better estimation of the vanishing point as it would reduce the error while drawing the parallel lines. Secondly, we can use an alternative way of vehicle tracking known as marriage problem instead of greedy approach which would be more intrinsic but will give better results. Thirdly, smaller, and multiple segments of regions can be chosen to study instead of just one as in our case. This would increase the mathematical complexity of the program but would give a better estimation of speed specially for vehicle who are accelerating / decelerating while driving. Lastly, the camera intrinsic parameters along with the position of camera with respect to world coordinates can be easily calculated in the real world for situations like these. Once they are calculated, the camera geometry can be put to use which would also give a better result in terms of image projection and rectification which then can be further processed to estimate our desired parameters.

References:

_ A. Hamano, K. Fujisaki, S. Uchida and O. Shiku, "Stable Marriage Algorithm for Tracking Intracellular Objects," 2013 First International Symposium on Computing and Networking, 2013, pp. 305-307, doi: 10.1109/CANDAR.2013.53.

_ <https://www.analyticsvidhya.com/blog/2021/12/how-to-use-yolo-v5-object-detection-algorithm-for-custom-object-detection-an-example-use-case/>

_ https://labelbox.com/product/objectdetection?utm_source=google&utm_medium=cpc&utm_term=object%20detection&utm_content=509038652366&utm_campaign=12606718007&gclid=CjwKCAiA3L6PBhBvEiwAINIJ9OibUmZ6mw3hVMVqc9yLxra8qNEN0jyUdHx61qY6WG-VvX-ii4eCyhoC3GYQAvD_BwE

_ T.W. Wong, C.H. Tao, Y.K. Cheng, K.H. Wong, C.N. Tam, Application of cross-ratio in traffic accident reconstruction, Forensic Science International, Volume 235, 2014, Pages 19-23, ISSN 0379-0738,

<https://doi.org/10.1016/j.forsciint.2013.11.012>.

(<https://www.sciencedirect.com/science/article/pii/S0379073813005112>)