Vehicle Tracking and Counting System

Introduction

A Python based Vehicle Tracking and Counting System using OpenCV 4, Tensorflow YOLOv3 and DeepSORT. This software aims to track southward moving vehicles and count the number of vehicles exiting the road. This report documents my approach and methodology in designing this program.

Problem Statement and Scope

The defined problem statements for this software are

1. To detect and track vehicles that are moving south and draw a bounding box around them
2. To count the number of vehicles exiting the southbound lanes

The defined scope for this project is vehicle counting for cameras that are stationary, or with minimal camera motions.

Observations and Considerations

The following are some observations on the video that needs to be taken into consideration as for the design of this software.

Observations

1. Vehicles are moving north on the leftmost lanes, while vehicles are moving south on the right lanes.
Implication: Software must be able to differentiate south-moving vehicles from north-moving vehicles.

2. Video is taken from an angled view instead of a front view. Therefore, vehicles at the first lane are often being occluded by vehicles of subsequent lanes.
Implication: Software must not lose track of vehicles due to occlusion.

3. Video footage is most likely handheld as it is unstable.
Implication: Software must be able to handle these movements.

4. From camera angle, there seem to be 2 exits for vehicles. At the bottom and at the far right. (To include picture)

5. Features on the top region of frames seem to be non-moveable in the world coordinate: Such as buildings, and trees. Perhaps these features can be used to stabilize frames.

Observations from Frame

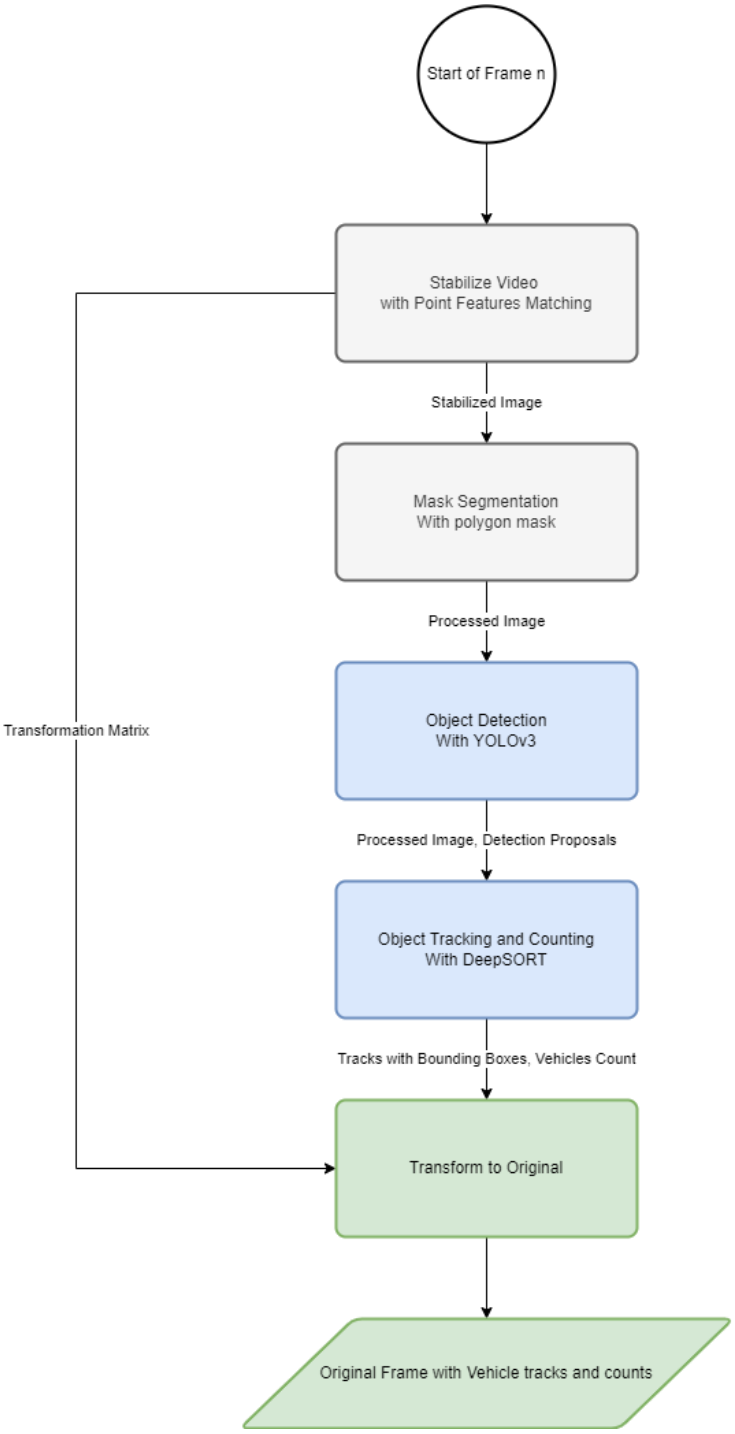
<u>Concluded Technical Requirements</u>

1. Software must conduct object detection on vehicles
2. Software must differentiate south-moving vehicles from north-moving vehicles
3. Software must track vehicles even with occlusions, that is to identify the same vehicle in multiple frames
4. Software must reduce effects due to camera motions
5. Software must successfully count vehicles moving out of the frame (not losing track)

Designed Pipeline and Technologies

Overall Pipeline

To perform tracking and counting of vehicles at the southbound lane, the pipeline for each frame is shown below:



Technologies Used

Point Features Matching with OpenCV

OpenCV provides algorithms to perform point feature matching to compute transformation matrix between 2 frames caused my camera motion.

1. goodFeaturesToTrack is a fast feature detector that searches an image for points that are good for tracking. For our use case, the camera motion is relatively small. Therefore, having a fast feature detector is sufficient to ensure good accuracy with fast detection.
2. Lucas-Kanade Optical Flow can be used to track points identified by goodFeaturesToTrack in subsequent frames.

With point features obtained from a reference frame and tracked in subsequent frames, a transformation matrix can be computed to stabilize all frames to a reference.

Object Detection with YOLOv3 on Tensorflow 2

YOLOv3, also known as You Only Look Once, is a fast object detection model that predicts multiple bounding boxes and classes simultaneously. It detection speed is faster as compared to other object detection model. Moreover, the choice of using the Tensorflow 2 version of YOLOv3 provides the user flexibility to run the program either on a CPU or GPU.

Multiple Object Tracking with DeepSORT

DeepSort is an object tracking framework which is effective in tracking objects, even under occlusion. DeepSort utilizes these methods to track objects effectively, even under occlusion.

1. Kalman Filter: Using current and previous detections to predict the current status while recording errors probabilities. DeepSort then uses Hungarian Algorithm, combinatorial optimization algorithm that incorporates distance metrics with Kalman Filter's uncertainties, to associate detections with tracks
2. Appearance Feature Vector: An appearance metric of an object is used to associate detection with tracks. The authors of deep sort trained a classifier over a dataset and removed its' classification layer. This model, thus, provides an appearance feature vector for images. Therefore, DeepSort 'remembers' each object by its appearance feature, which can be obtained through training a classification neural network, without the classification layer.

With DeepSort, vehicles can be tracked effectively even under occlusions, preventing tracks to be lost.

Noted: I obtained this library from TensorFlow-2.x-YOLOv3 repository by *pythonlessons*.
Repo in github: https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3

Functions Design

Preprocessing Class [VideoPreprocessor]

The purpose of this preprocessing class is to narrow our tracking and counting scope to the southbound lanes region of interests.

Video Stabilizer

The purpose of this step is to remove any camera motion in the video frames. This is so that we will be able to crop the southbound lane without any shift in the region of interest that we are cropping. This function aims to obtain a 3x3 transformation vector, ***H,*** to represent camera motion. The inverse of this vector**, H^{-1}**, can be utilized to stabilize every frame of the video, using the stationary point features of the first frames such as trees and buildings. This is performed in 3 steps.

1. Stationary ROI Crop

From observation (5), we saw that features at the top region of a frame are stationary in world coordinates. Thus, we can conclude that the motion of these point features of the image is due to camera motion since they are stationary in the real world. Under the assumption that cameras are usually positioned upright, the top region of an image frame will tend to consist of background scenes such as the sky, buildings, or trees, unless the camera is facing directly on the ground. With that, we define a function to crop an image from the top given the amount of percentage we want to crop from the top. For our case, we crop 20% of both the reference image and subsequent frames from the top to extract an image consisting only of the background. For detailed information of code implementation, please refer to the *cropStabilizeRoi* method under *VideoPreprocessor* class in *VehicleCounter.py* python script.

2. Point Features Extraction

After we cropped out the region of interest, the next step is to extract point features from our reference frame. This can be done using goodFeaturesToTrack method in the OpenCV library to obtain feature points, $\mathbf{P(}\boldsymbol{X, Y}\mathbf{)_0}$. Then, these points are saved and will be utilized to perform tracking on subsequent frames. For code implementation, please refer to *registerOriginal* method under *VideoPreprocessor* class in *VehicleCounter.py* python script.

3. Features Tracking and stabilization with Optical Flow

After we obtain points $\mathbf{P(}\boldsymbol{X, Y}\mathbf{)_0}$ from the reference frame, optical flow is utilized to track these features in the subsequent frames to obtain $\mathbf{P(X,Y)_n}$, where n refers to the current frame number. This can be done using the OpenCV method, *calcOpticalFlowPyrLK*. After we obtained $\boldsymbol{P(X, Y)_n}$, we utilize OpenCV's method, estimateAffinePartial2D, using $\boldsymbol{P(X,Y)_0}$ as source points, and $\boldsymbol{P(X, Y)_n}$ as target points to obtain the transformation matrix, H. The relationship between $\boldsymbol{P(X, Y)_0}$ and $\boldsymbol{P(X, Y)_n}$ is given as followed:

$$\boldsymbol{P(X, Y)_n = H_n . P(X,Y)_0}$$

Thus, $\boldsymbol{P(X, Y)_0 = H^{-1} P'(X, Y)_n}$

Thus, the inverse of the transformation matrix, H-1, can be used to transform the current frame to the reference frame, using stationary points as references points of transformation. The transformation matrix, H, is also saved to transform detections back to the original frame in the final steps. For code

implementation, please refer to *stabilize_frame* method under *VideoPreprocessor* class in *VehicleCounter.py* python script.

Stable Frame

Original Frame

Southbound Land Masking

After stabilizing the frames, we can now apply a filter mask to only filter the southbound lane. This is so that we will be able to detect and track vehicles only in this region and ignore those the are moving northwards on the left, as well as those vehicles parked at the right in the parallel parking carpark. With video stabilization, the polygon points to segment southbound lane in the first frame can be assumed to be the same points for all other frames. Polygon points can be obtained using image applications such as paint.exe.

Vehicle Counting Class [VehicleCounter]

After we have pre-processed the frames, we can now conduct our tracking and counting of vehicles. This class provides the methods required to detect, track and count vehicles.

Vehicle Detection

For vehicle detection, the YOLOv3 Tensorflow API is utilized. This model can detect 80 different classes of objects including vehicles such as cars, trucks and buses, and provides an output in the form of (x, y, w, h, class, score). As DeepSort requires both the image and the detection bounding box as input in the form of (x, y, w, h), the output of YOLOv3 will be fed directly to the DeepSort tracker. As DeepSort

requires high accuracy for detection proposals, we must ensure that the detections of YOLOv3 must be of high confidence scores. This is to prevent false positives from being registered and tracked throughout the video, which might hinder the counting system.

There are 3 main parameters of YOLOv3 to be adjusted to ensure detections of high confidence scores.

1. Input Size: By default, YOLOv3 input size is set to (416, 416) – multiples of 32. With a higher input size, the model can increase its accuracy to detect objects, especially smaller objects in the frame. However, this will result in a drop of speed. I have set the default input size to be (832, 832) to ensure high accuracy.
2. Confidence Threshold [0-1]: The minimum confidence scores of each detection. If confidence score is lower than threshold, detection will be removed. I have set this threshold to be 0.7.
3. Non-Maximum Suppression Threshold [0-1]: Maximum Intersection Over Union (IOU) overlaps for bounding box of same class. Detections with high overlaps will be removed. I have set the default value to 0.2.

After obtaining detections, we filter only classes that belong to car, truck and buses and motorbikes. The rest of the detections are deleted. For code implementation, please refer to *vehicleDetection* method under *VehicleCounter* class in *VehicleCounter.py* python script.

Vehicle Tracking

After obtaining detections from YOLOv3, we performing object tracking with DeepSORT to assign a unique tracking ID to every vehicle detected. A pretrained model, mars-small128.pb, is utilized to determine appearance features detection for each object. This model is trained on the MARS dataset, which mostly comprises of different positions of a single person. However, this model works sufficiently well for vehicles as well.

There are 2 main parameters for the DeepSORT tracker.

1. Maximum Cosine Distance: This is the maximum distance metric for the appearance vectors of respective tracks. If the cosine distance between 2 detections is low, it means that the tracker will recognize these 2 detections as of higher likelihood to be the same object. Thus, if the cosine distance is lower than this maximum value, the tracker will consider the current track to be unassociated with any detections in this frame. If this parameter is set too high, there will be more likelihood of ID switching between detections. Thus, I have set this parameter to a low value of 0.2 to prevent switching of IDs.

2. Maximum Age: The maximum number of frames that a track is allowed to be disassociated with any detections. Thus, a track will be deleted if the tracker is unable to determine any detection with this track after the set maximum age number of frames has passed. I have set this parameter to a low value of 7 since we want the tracker to ‘forget’ as soon as possible once a vehicle exited the frame.

Vehicle Counting

After tracking is performed for each vehicle, we can now count the number of vehicles exiting the frame. We first determine the criteria to determine if a vehicle left the frame.

The first step is to determine the region of exits. As observed from the image, there appears to be two exits. Thus, two lines equations can be determined to check if vehicles have exited the frame. The image below shows the exit lines and its respective line equations.

A matrix representing these lines can be written as $[[a_1, b_1, c_1], [a_2, b_2, c_2], \ldots, [a_n, b_n, c_n]]$, where a, b and c are coefficients of the line equation: $ax + by + c = 0$. We let this matrix be **L**. The image below illustrates these exit lines and the exit regions.

The second step is to check if a vehicle falls above of below the exit region. We use the bottom right point of each bounding box as a determining factor. We let this point be **B** (**X**, **Y, 1**), represented as a column vector. By using matrix dot product, we can check if a vehicle falls above or below the exit line (for the case of the right lane, we can check if a vehicle falls on the right side or the left side of the exit line). The rule is thus defined as followed:

*If any elements of the dot product of **L** and **B** is less than 0, we can conclude that the object is in the exit region. Else, the object is at the non-exit region.*

However, there are cases where false detections may occur on objects below the exit regions. To prevent this, we also took into consideration the first position of a track once it is detected. This position must fall within the non-exit region. We let this point be **A**(**X**, **Y**, **1**). Using the same mathematical formula as above, we can determine if the original position of a track falls inside or outside the exit region.

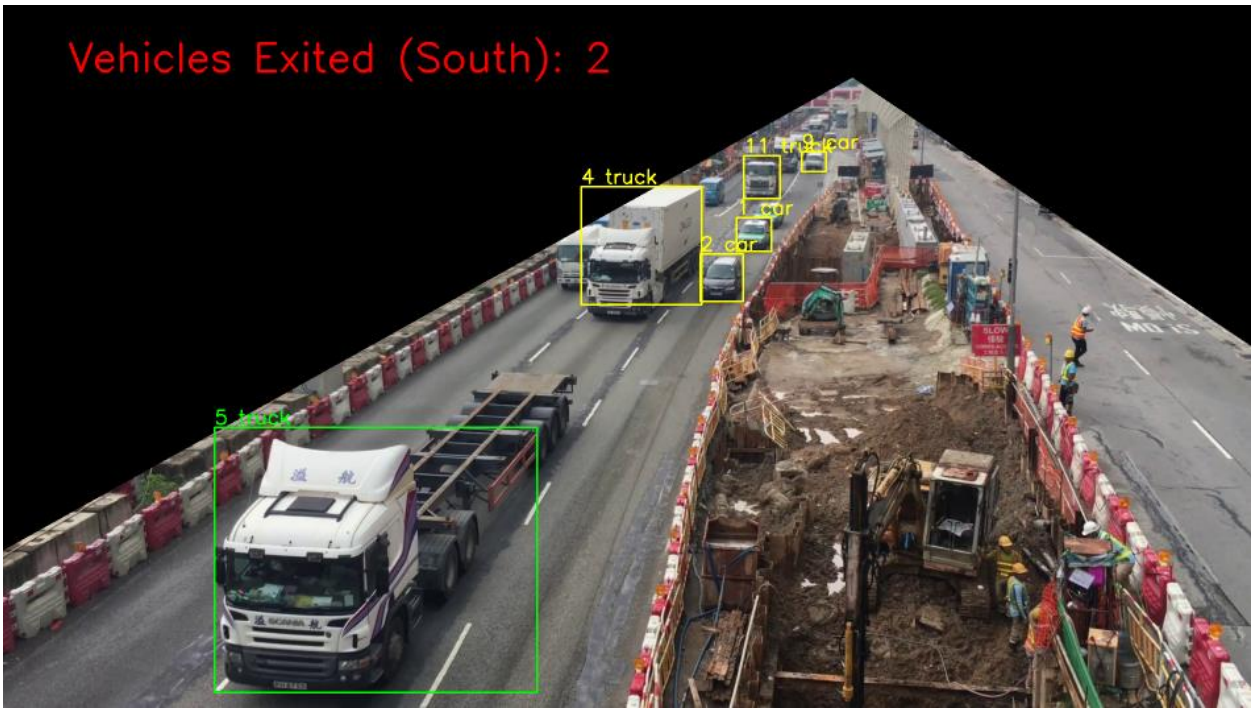
There is also a factor to be considered – that is there is already vehicle at the exit region when the program has just started. Thus, an interval can be implemented to include vehicles that are in exit region into count. However, after the interval passed, a vehicle must start at a non-exit region and end in an exit region.

Thus, the second rule is added:

A vehicle track is determined to have exited the region only if the track starts in a non-exit region and ends in an exit region, after a given amount of interval has passed since the start of program.

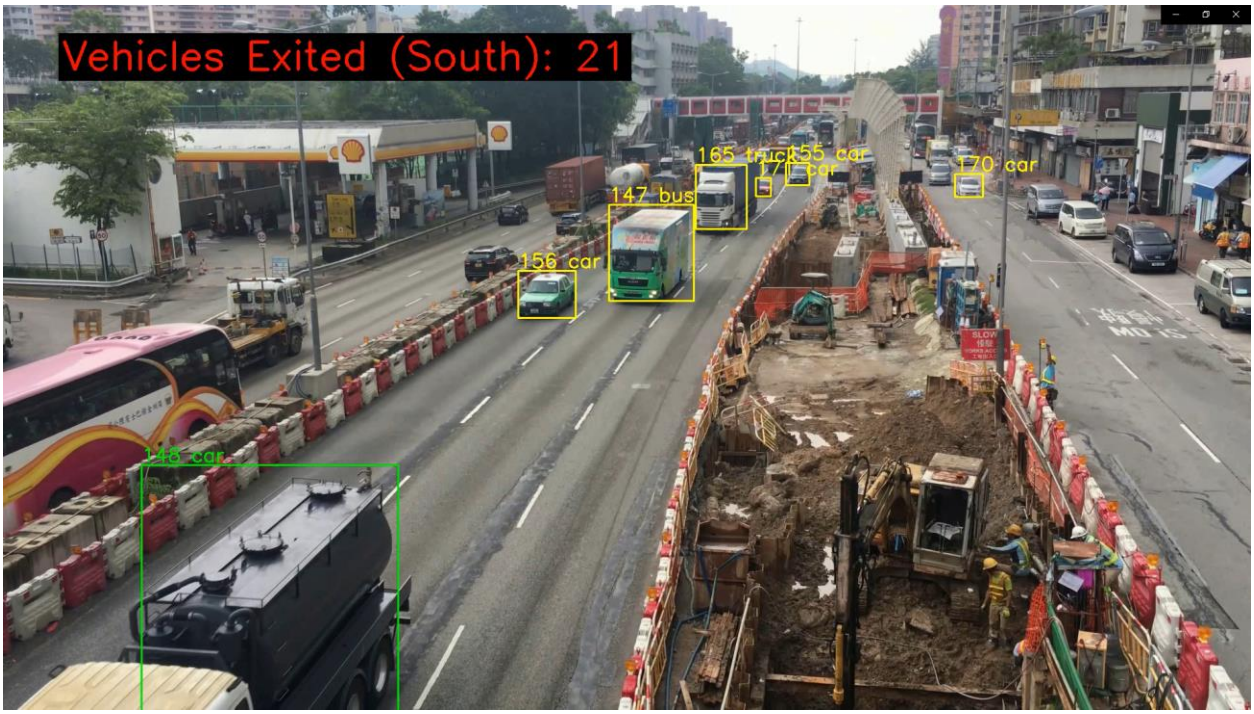
With that, we add these color codes to vehicles' bounding box and text:
Yellow – Vehicle is detected but has not yet exited the lane.
Green – Vehicle has exited the lane and is counted.
Red – Vehicle is detected only within detected region and is not counted. (Possibility of false positive).

The result of this class is as shown below:



Post-Process

After implementing the object tracker and counting system, we can now transform our bounding boxes back into its original frames. This can be done by taking the dot product of the transformation matrix, H, obtained earlier, with the coordinates of every bounding box (in the form of (x1, y1, x2, y2)).



Results

Since our scope does not cover deep learning models' accuracy, I measure my results according to how many vehicles are successfully counted for each track. There are a total of 32 successful tracks and all 32 tracks are counted.

However, there are 2 false negatives of new tracks that were detected after the exit line. This is due to poor tracking accuracy as the tracker loses its track on the vehicle and assigned a new ID when it approaches the exit line.

Overall, the tracking system performs well and has potential for significant improvement, given more time and data for model training for object detection and tracking.

Challenges and Possible Improvements

ID Switching for Visually Similar Objects travelling the same direction
One challenge that I foresaw is that ID switching may occur between 2 vehicles that are visually similar and are moving in the same direction. Since DeepSort relies heavily on appearance and motion estimation, such cases may potentially cause ID switching and may even cause double counting if the vehicles are near the exit regions.

Suggestion for improvements: To include an additional depth camera to obtain depth information of every object. Since it is scientifically impossible for two vehicles to share the same position in 3D space, depth data can be incorporated to the Kalman Filter of DeepSORT to give a much precise prediction of each object's location. Depth data can also be incorporated into an additional space for appearance vectors.

Cameras' Positioning
There were 2 major problems for this video input with regards to camera positioning.

1. Camera Motion exists which requires program to stabilize cameras
2. Camera is held at a certain angle facing the traffic, which caused vehicles of the first lane to be occluded

Suggestion for improvement: To facilitate better tracking and counting, the camera should be fixed and positioned directly in front of the traffic flow, instead of the current setup. This will greatly improve tracking and reduce causes of false negative detections and loss of tracks.

Use cases in other locations
Since every different location has different region of interests and exit regions, one must always change the polygon points and exit line equations in different places.
Suggestion for improvement: To reduce the tediousness of finding line equations and polygon points, an OpenCV UI can be implemented which allows user to select polygon points, and points of exit line equations, an automatically generate and incorporate these equations into the program.