

Abstract

Traffic Management Model includes the detection of the type of vehicles in ROI(region of interest) region in this model a specific id is provided to each vehicle detected and then respective data is extracted.

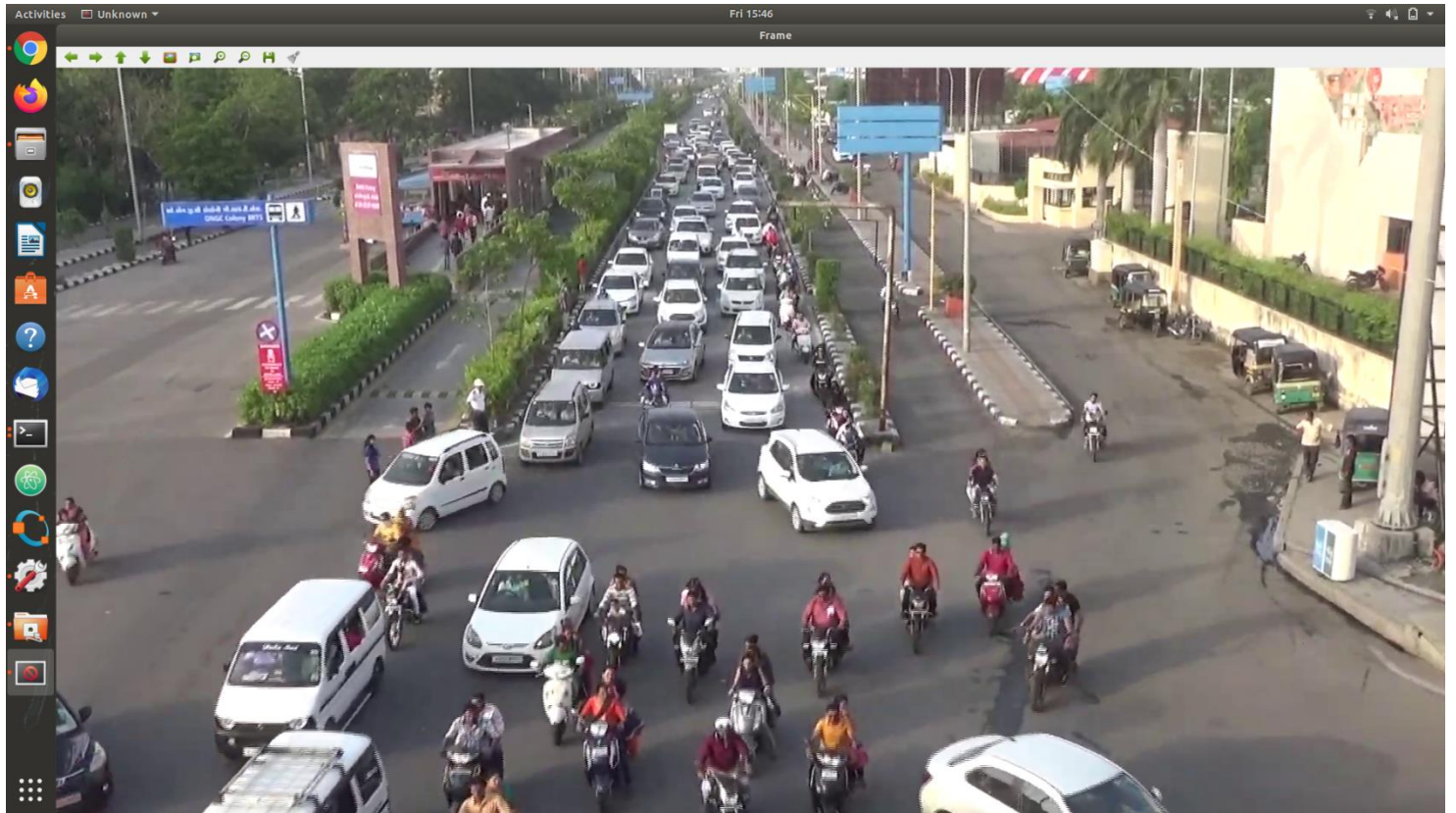
Respective data includes the centroid coordinates of the frame at which the vehicle is detected at the first place. The whole video clip is divided into several frames so for each frame the separate data is recorded that gives the trajectory data within the ROI region for that region the whole tracking coordinates are recorded for each vehicle so we can extract the trajectory data of any car identified in the ROI region.

Introduction

Traffic Management Model self trained Model includes OpenCV and ML(machine learning) algorithms. The model is trained for detection of vehicles using ***YOLO object detection model*** all algorithms and codes are written using **PYTHON**. The model contains algorithm which is highly trained for detection of vehicles with maximum accuracy, model also includes the counting of vehicles i.e the number of vehicles detected also get counted. This model can be practically used for traffic management in the cities to solve the problem of Traffic in big cities. As there are cameras

already installed for monitoring of traffic in many cities so, on data from these cameras we can apply our model which can help in detection, tracking and also help in minimizing accidents. This model can also be used at the round circles where there is TRAFFIC LIGHTS, which are controlling the traffic so by analyzing the amount of traffic on a road we can customize the traffic lights according to the traffic i.e for the lane with maximum traffic the timing for the green light could be increased to initiate easy flow of traffic and for limited or relatively less traffic the timing of green light could be customized accordingly. This helps to minimize the accumulation of traffic.

The aim of this project to achieve good traffic monitoring system. This project was a great learning opportunity for us to apply our engineering knowledge. This report addresses the difficulties our group faced throughout the project.



The Initial input

Algorithms Used:

1.DEFINING ROI:

In this model we define a region, we give the specified coordinates and within that coordinates a bounding box is constructed in which the detection and tracking

and trajectory data is detected within this region. Within this region the the following algorithms are applied.



ROI

2.YOLO OBJECT DETECTION:

YOLO stands for You Only Look Once is a system for **detecting objects** on the Pascal VOC 2012 dataset. It can detect the 20 Pascal Object classes(person,bird, cat, cow, dog, horse, sheep,aeroplane, bicycle, boat, bus, car, motorbike, train,bottle, chair, dining table, potted plant, sofa, tv/monitor).

In this Project we used YOLO MODEL for detection of CAR, BUSES, MOTORCYCLE and TRUCK.

How YOLO Works:

All prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detected.

In YOLO Model it use a totally different approach. It apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

This model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

In this Model we give the video input as .mp4 file for detection and further processing. This model can also work in REAL TIME. To do this we need to compile DARKNET with CUDA and OPENCV. **Darknet** is a framework to train neural networks, it is open source and written in C/CUDA and serves as the basis for **YOLO**. ... **Darknet** is used as the framework for training **YOLO**, meaning it sets the architecture of the network.

YOLO model is trained by machine learning.

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: The ability to learn.

How ML works?

- *Gathering past data in any form suitable for processing. The better the quality of data, the more suitable it will be for modeling*
- *Data Processing – Sometimes, the data collected is in the raw form and it needs to be pre-processed. Example: Some tuples may have missing values for certain attributes, and, in this case, it has to be filled with suitable values in order to perform machine learning or any form of data mining. Missing values for numerical attributes such as the price of the house may be replaced with the mean value of the attribute whereas missing values for categorical attributes may be replaced with the attribute with the highest mode. This invariably depends on the types of filters we use. If data is in the form of text or images then converting it to numerical form will be required, be it a list or array or matrix. Simply, Data is to be made relevant and consistent. It is to be converted into a format understandable by the machine*
- *Divide the input data into training, cross-validation and test sets. The ratio between the respective sets must be 6:2:2*

- *Building models with suitable algorithms and techniques on the training set.*
- *Testing our conceptualized model with data which was not fed to the model at the time of training and evaluating its performance using metrics such as F1 score, precision and recall.*

Thats how YOLO MODEL is trained for detection of vehicles.

3.ALLOTING SPECIFIC VEHICLE ID :

After detection of type of vehicle we give specific id to each model the specific id is given by ***Blob Detection***. This is a computer vision trained model.

What is Blob Detection?

In computer vision, blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other. The most common method for blob detection is convolution.

Given some property of interest expressed as a function of position on the image, there are two main classes of blob detectors:

- (i) differential methods, which are based on derivatives of the function with respect to position.*
- (ii) methods based on local extrema, which are based on finding the local maxima and minima of the function.*

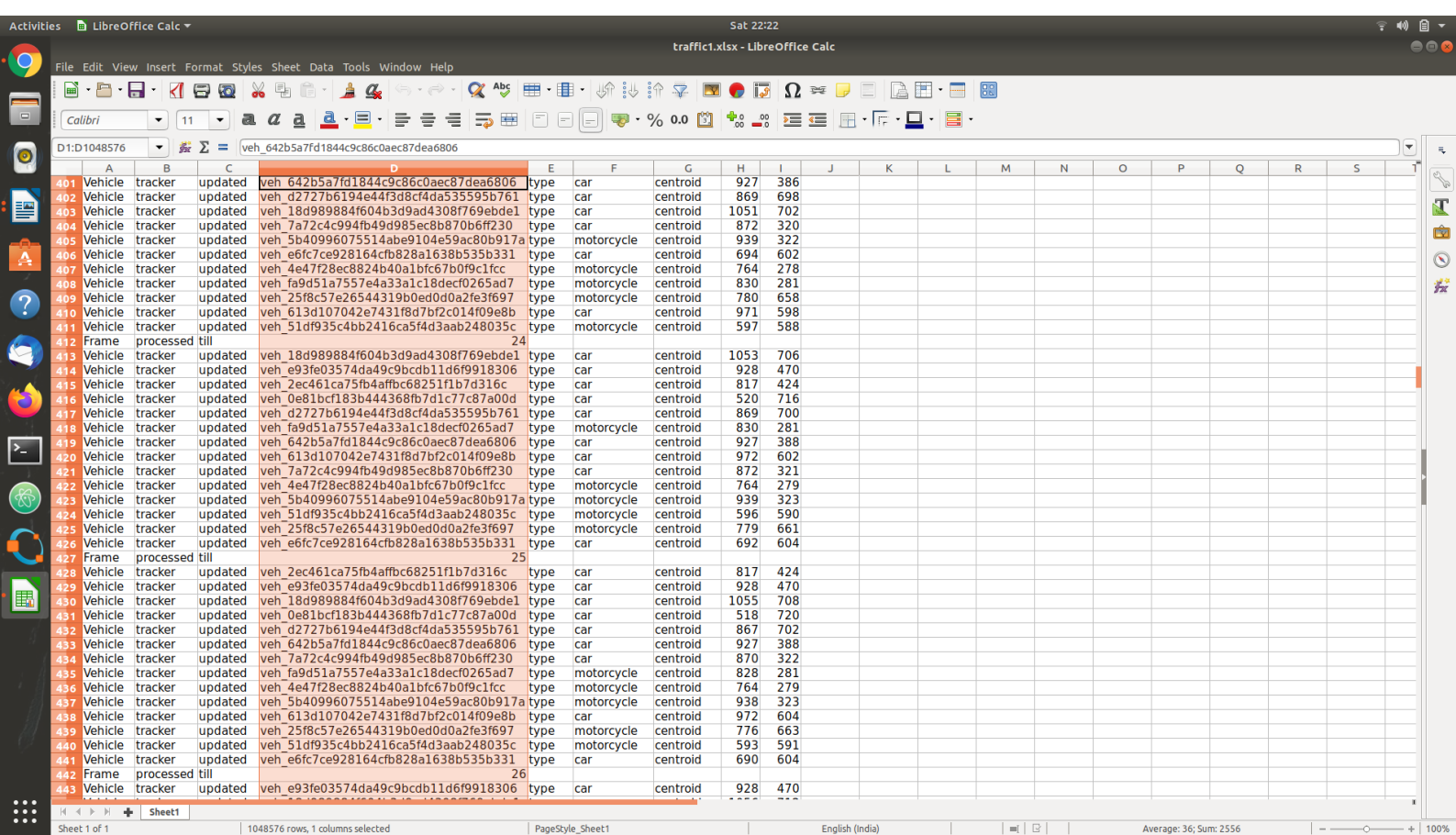
With the more recent terminology used in the field, these detectors can also be referred to as interest point operators, or alternatively interest region operators .

There are several motivations for studying and developing blob detectors. One main reason is to provide complementary information about regions, which is not obtained from edge detectors or corner detectors. In early work in the area, blob detection was used to obtain regions of interest for further processing. These regions could signal the presence of objects or parts of objects in the image domain with application to object recognition and/or object tracking.

So, By Blob detection method specific id is given to the particular vehicle detected there is a hex digit which is auto generated sequentially as the vehicles are detected.

In every frame it is important to specify same id to same vehicle i.e_ for all frames when the vehicle is detected at the very first time the assigned id must not change for

next frames if that happens It is difficult to get Trajectory data.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
401	Vehicle	tracker	updated	veh_642b5a7fd1844c9c86c0aec87dea6806	type	car	centroid	927	386										
402	Vehicle	tracker	updated	veh_d2727b6194e44f3d8cf4da535595b761	type	car	centroid	869	698										
403	Vehicle	tracker	updated	veh_18d989884f604b3d9ad4308f769ebde1	type	car	centroid	1051	702										
404	Vehicle	tracker	updated	veh_7a72c4c994fb49d985ec8b870b6ff230	type	car	centroid	872	320										
405	Vehicle	tracker	updated	veh_5b40996075514abe9104e59ac80b917a	type	motorcycle	centroid	939	322										
406	Vehicle	tracker	updated	veh_e6fc7ce928164cfb828a1638b535b331	type	car	centroid	694	602										
407	Vehicle	tracker	updated	veh_4e47f28ec8824b40a1bfc67b0f9c1fcc	type	motorcycle	centroid	764	278										
408	Vehicle	tracker	updated	veh_fa9d51a7557e4a33a1c18decf0265ad7	type	motorcycle	centroid	830	281										
409	Vehicle	tracker	updated	veh_25f8c57e26544319b0ed0d0a2fe3f697	type	motorcycle	centroid	780	658										
410	Vehicle	tracker	updated	veh_613d107042e7431f8d7bf2c014f09e8b	type	car	centroid	971	598										
411	Vehicle	tracker	updated	veh_51df935c4bb2416ca5f4d3aab248035c	type	motorcycle	centroid	597	588										
412	Frame	processed	till	24															
413	Vehicle	tracker	updated	veh_18d989884f604b3d9ad4308f769ebde1	type	car	centroid	1053	706										
414	Vehicle	tracker	updated	veh_e93fe03574da49c9bcdb11d6f9918306	type	car	centroid	928	470										
415	Vehicle	tracker	updated	veh_2ec461ca75fb4affbc68251f1b7d316c	type	car	centroid	817	424										
416	Vehicle	tracker	updated	veh_0e81bfc183b444368fb7d1c77c87a00d	type	car	centroid	520	716										
417	Vehicle	tracker	updated	veh_d2727b6194e44f3d8cf4da535595b761	type	car	centroid	869	700										
418	Vehicle	tracker	updated	veh_fa9d51a7557e4a33a1c18decf0265ad7	type	motorcycle	centroid	830	281										
419	Vehicle	tracker	updated	veh_642b5a7fd1844c9c86c0aec87dea6806	type	car	centroid	927	388										
420	Vehicle	tracker	updated	veh_613d107042e7431f8d7bf2c014f09e8b	type	car	centroid	972	602										
421	Vehicle	tracker	updated	veh_7a72c4c994fb49d985ec8b870b6ff230	type	car	centroid	872	321										
422	Vehicle	tracker	updated	veh_4e47f28ec8824b40a1bfc67b0f9c1fcc	type	motorcycle	centroid	764	279										
423	Vehicle	tracker	updated	veh_5b40996075514abe9104e59ac80b917a	type	motorcycle	centroid	939	323										
424	Vehicle	tracker	updated	veh_51df935c4bb2416ca5f4d3aab248035c	type	motorcycle	centroid	596	590										
425	Vehicle	tracker	updated	veh_25f8c57e26544319b0ed0d0a2fe3f697	type	motorcycle	centroid	779	661										
426	Vehicle	tracker	updated	veh_e6fc7ce928164cfb828a1638b535b331	type	car	centroid	692	604										
427	Frame	processed	till	25															
428	Vehicle	tracker	updated	veh_2ec461ca75fb4affbc68251f1b7d316c	type	car	centroid	817	424										
429	Vehicle	tracker	updated	veh_e93fe03574da49c9bcdb11d6f9918306	type	car	centroid	928	470										
430	Vehicle	tracker	updated	veh_18d989884f604b3d9ad4308f769ebde1	type	car	centroid	1055	708										
431	Vehicle	tracker	updated	veh_0e81bfc183b444368fb7d1c77c87a00d	type	car	centroid	518	720										
432	Vehicle	tracker	updated	veh_d2727b6194e44f3d8cf4da535595b761	type	car	centroid	867	702										
433	Vehicle	tracker	updated	veh_642b5a7fd1844c9c86c0aec87dea6806	type	car	centroid	927	388										
434	Vehicle	tracker	updated	veh_7a72c4c994fb49d985ec8b870b6ff230	type	car	centroid	870	322										
435	Vehicle	tracker	updated	veh_fa9d51a7557e4a33a1c18decf0265ad7	type	motorcycle	centroid	828	281										
436	Vehicle	tracker	updated	veh_4e47f28ec8824b40a1bfc67b0f9c1fcc	type	motorcycle	centroid	764	279										
437	Vehicle	tracker	updated	veh_5b40996075514abe9104e59ac80b917a	type	motorcycle	centroid	938	323										
438	Vehicle	tracker	updated	veh_613d107042e7431f8d7bf2c014f09e8b	type	car	centroid	972	604										
439	Vehicle	tracker	updated	veh_25f8c57e26544319b0ed0d0a2fe3f697	type	motorcycle	centroid	776	663										
440	Vehicle	tracker	updated	veh_51df935c4bb2416ca5f4d3aab248035c	type	motorcycle	centroid	593	591										
441	Vehicle	tracker	updated	veh_e6fc7ce928164cfb828a1638b535b331	type	car	centroid	690	604										
442	Frame	processed	till	26															
443	Vehicle	tracker	updated	veh_e93fe03574da49c9bcdb11d6f9918306	type	car	centroid	928	470										

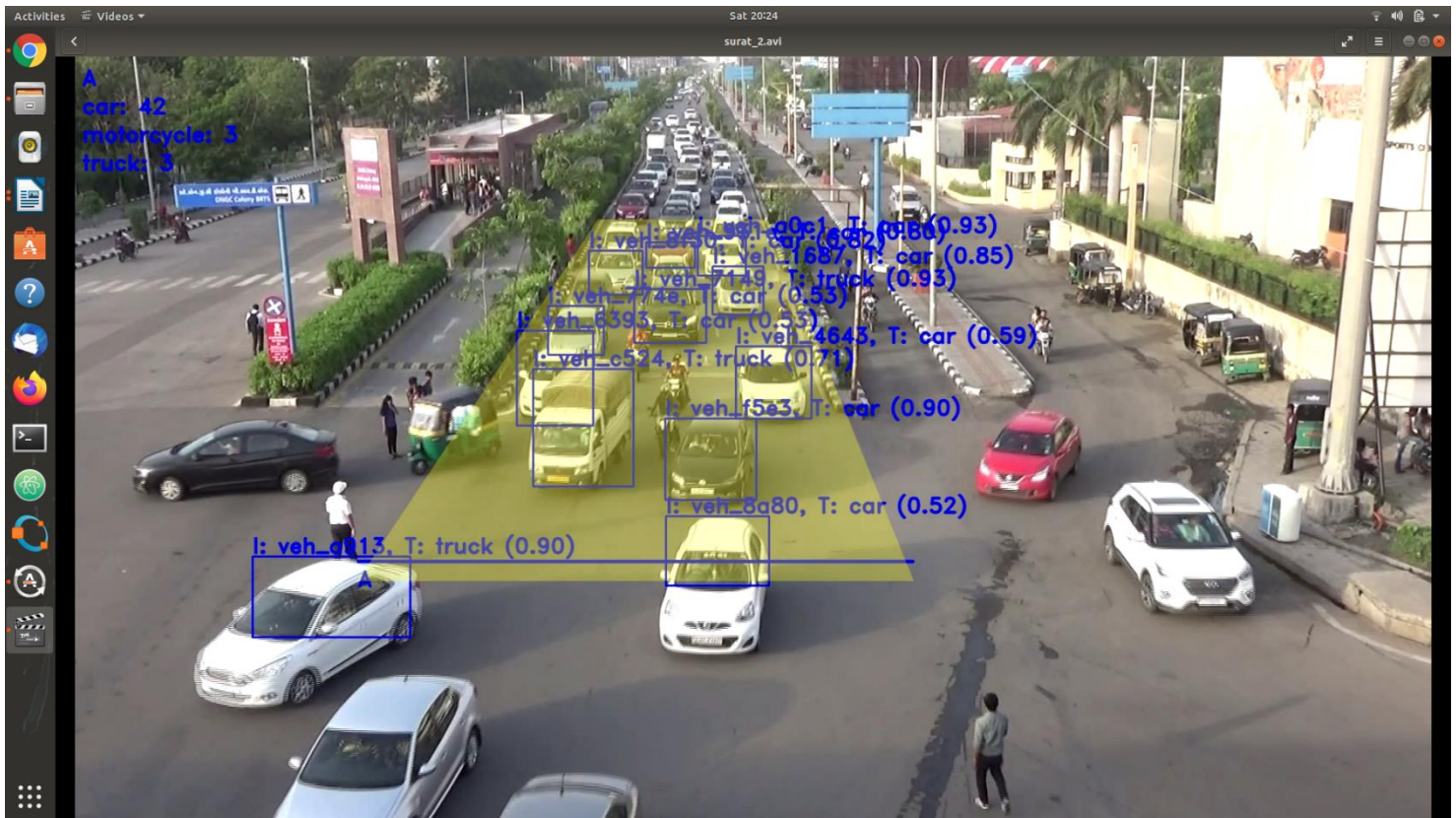
Specific Vehicle Id given to each vehicle in different frames.

4.COUNTING ALGORITHM :

In this algorithm a line is defined when a vehicle crosses this line the count appends that gives the count of vehicles detected.The vehicle counting system is made

up of three main components: a detector, tracker and counter. The detector identifies vehicles in a given frame of video and returns a list of bounding boxes around the vehicles to the tracker. The tracker uses the bounding boxes to track the vehicles in subsequent frames. The detector is also used to update trackers periodically to ensure that they are still tracking the vehicles correctly. The counter draws a counting lines across the road. When a vehicle crosses the line, the vehicle count is incremented. This line can also be defined lane-wise i.e the whole road track could be divided into several lanes that gives the count of specific lane so that will be more efficient to monitor.

COUNTING OF VEHICLES ON TOP LEFT CORNER



5.CENTROID COORDINATES :

When the Vehicle is detected using YOLO model at that time there is a bounding box i.e. the detection takes place within the bounding box so, we need to get the coordinates of the centroid. An algorithm for **centroid-based tracking** of moving objects. A set of data **centroids** is tracked using intra-frame and inter-frame recursions. **Centroids** are computed as weighted sums of the edge points belonging to the object boundary.

Sat 22:26
traffic1.xlsx - LibreOffice Calc

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
401	Vehicle	tracker	updated	veh_642b5a7fd1844c9c86c0aec87dea6806	type	car	centroid	927	386										
402	Vehicle	tracker	updated	veh_d2727b6194e443d8cf4da535595b761	type	car	centroid	869	698										
403	Vehicle	tracker	updated	veh_18d989884f604b3d9ad4308f769ebde1	type	car	centroid	1051	702										
404	Vehicle	tracker	updated	veh_7a72c4c994fb49d985ec8b870b6ff230	type	car	centroid	872	320										
405	Vehicle	tracker	updated	veh_5b40996075514ab9104e59ac80b917a	type	motorcycle	centroid	939	322										
406	Vehicle	tracker	updated	veh_e6fc7ce928164cfb828a1638b535b331	type	car	centroid	694	602										
407	Vehicle	tracker	updated	veh_4e47f28ec8824b40a1bfc67b0f9c1fcc	type	motorcycle	centroid	764	278										
408	Vehicle	tracker	updated	veh_fa9d51a7557e4a33a1c18decf0265ad7	type	motorcycle	centroid	830	281										
409	Vehicle	tracker	updated	veh_25f8c57e26544319b0ed00a2fe3f697	type	motorcycle	centroid	780	658										
410	Vehicle	tracker	updated	veh_613d107042e7431f8d7bf2c014f09e8b	type	car	centroid	971	598										
411	Vehicle	tracker	updated	veh_51d9f35c4bb2416ca5f4d3aab248035c	type	motorcycle	centroid	597	588										
412	Frame	processed	till	24															
413	Vehicle	tracker	updated	veh_18d989884f604b3d9ad4308f769ebde1	type	car	centroid	1053	706										
414	Vehicle	tracker	updated	veh_e93fe03574da49c9bcdb11d6f9918306	type	car	centroid	928	470										
415	Vehicle	tracker	updated	veh_2ec461ca75fb4afbc68251f1b7d316c	type	car	centroid	817	424										
416	Vehicle	tracker	updated	veh_0e81bcf183b444368fb7d1c77c87a00d	type	car	centroid	520	716										
417	Vehicle	tracker	updated	veh_d2727b6194e443d8cf4da535595b761	type	car	centroid	869	700										
418	Vehicle	tracker	updated	veh_fa9d51a7557e4a33a1c18decf0265ad7	type	motorcycle	centroid	830	281										
419	Vehicle	tracker	updated	veh_642b5a7fd1844c9c86c0aec87dea6806	type	car	centroid	927	388										
420	Vehicle	tracker	updated	veh_613d107042e7431f8d7bf2c014f09e8b	type	car	centroid	972	602										
421	Vehicle	tracker	updated	veh_7a72c4c994fb49d985ec8b870b6ff230	type	car	centroid	872	321										
422	Vehicle	tracker	updated	veh_4e47f28ec8824b40a1bfc67b0f9c1fcc	type	motorcycle	centroid	764	279										
423	Vehicle	tracker	updated	veh_5b40996075514ab9104e59ac80b917a	type	motorcycle	centroid	939	323										
424	Vehicle	tracker	updated	veh_51d9f35c4bb2416ca5f4d3aab248035c	type	motorcycle	centroid	596	590										
425	Vehicle	tracker	updated	veh_25f8c57e26544319b0ed00a2fe3f697	type	motorcycle	centroid	779	661										
426	Vehicle	tracker	updated	veh_e6fc7ce928164cfb828a1638b535b331	type	car	centroid	692	604										
427	Frame	processed	till	25															
428	Vehicle	tracker	updated	veh_2ec461ca75fb4afbc68251f1b7d316c	type	car	centroid	817	424										
429	Vehicle	tracker	updated	veh_e93fe03574da49c9bcdb11d6f9918306	type	car	centroid	928	470										
430	Vehicle	tracker	updated	veh_18d989884f604b3d9ad4308f769ebde1	type	car	centroid	1055	708										
431	Vehicle	tracker	updated	veh_0e81bcf183b444368fb7d1c77c87a00d	type	car	centroid	518	720										
432	Vehicle	tracker	updated	veh_d2727b6194e443d8cf4da535595b761	type	car	centroid	867	702										
433	Vehicle	tracker	updated	veh_642b5a7fd1844c9c86c0aec87dea6806	type	car	centroid	927	388										
434	Vehicle	tracker	updated	veh_7a72c4c994fb49d985ec8b870b6ff230	type	car	centroid	870	322										
435	Vehicle	tracker	updated	veh_fa9d51a7557e4a33a1c18decf0265ad7	type	motorcycle	centroid	828	281										
436	Vehicle	tracker	updated	veh_4e47f28ec8824b40a1bfc67b0f9c1fcc	type	motorcycle	centroid	764	279										
437	Vehicle	tracker	updated	veh_5b40996075514ab9104e59ac80b917a	type	motorcycle	centroid	938	323										
438	Vehicle	tracker	updated	veh_613d107042e7431f8d7bf2c014f09e8b	type	car	centroid	972	604										
439	Vehicle	tracker	updated	veh_25f8c57e26544319b0ed00a2fe3f697	type	motorcycle	centroid	776	663										
440	Vehicle	tracker	updated	veh_51d9f35c4bb2416ca5f4d3aab248035c	type	motorcycle	centroid	593	591										
441	Vehicle	tracker	updated	veh_e6fc7ce928164cfb828a1638b535b331	type	car	centroid	690	604										
442	Frame	processed	till	26															
443	Vehicle	tracker	updated	veh_e93fe03574da49c9bcdb11d6f9918306	type	car	centroid	928	470										

Sheet 1 of 1 | 1048576 rows, 3 columns selected | PageStyle_Sheet1 | English (India) | Average: 666.129144851658; Sum: 1526768 | 100%

Coordinates of centroid of every vehicle detected in every frame.

6.TRACKING:

The model we used here for tracking is **KCF – Model it stands for TRACKING WITH CORRELATION FILTERS**. An object to be tracked is usually selected by a Rectangular bounding box . The filter is trained on-line and updated successively with every frame in order the tracker adapts to moderate target changes.

KCF tracker tries to calculate the motion of these points by looking at the direction of change in the next frame. In every consecutive frame, it try to look for the same

set of points in the neighborhood. Once the new positions of these points are identified, we can move the bounding box over the new set of points. There is mathematics involved in making the search faster and more efficient

In the TRAFFIC MANAGEMENT MODEL during the video processing the complete video is divided into various frames for eg: if during tracking there is a frame1 containing car with vehicle id1 then that particular car should be detected every next time for many next frames so for each frame the vehicle id should remain same that gives the tracking info of specific car detected with specific vehicle id.

7.FINAL OUTPUT OF THE PROCESSED VIDEO:

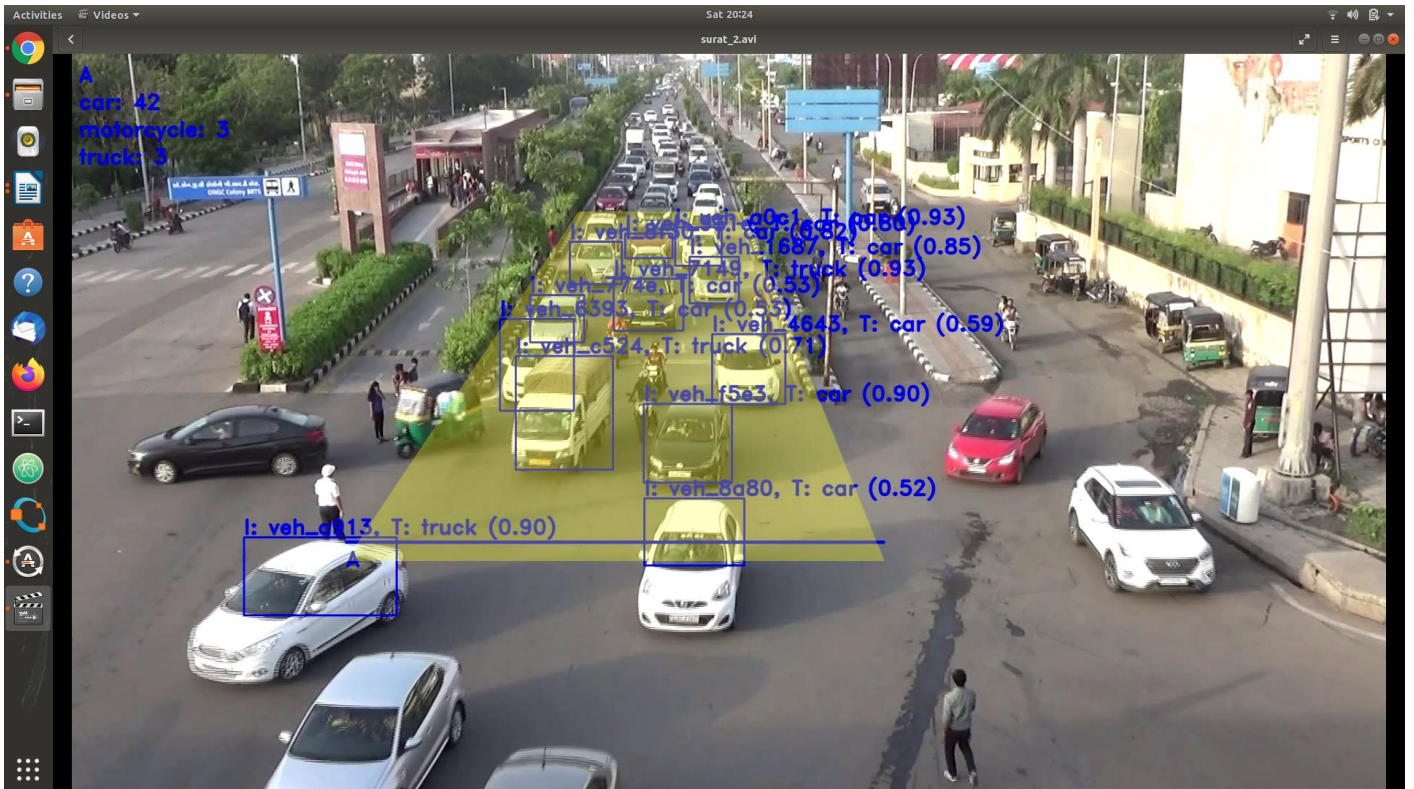


Image contains ROI , the type of vehicle detected in the ROI with % accuracy and on the top left corner the count of each type of vehicle is recorded. And tracking details are recorded in .CSV file.

**Why tracking is needed
although there is
detection?**

Tracking is faster than Detection:

Usually tracking algorithms are faster than detection algorithms. The reason is simple. When you are tracking an object that was detected in the previous frame, you know a lot about the appearance of the object. You also know the location in the previous frame and the direction and speed of its motion. So in the next frame, you can use all this information to predict the location of the object in the next frame and do a small search around the expected location of the object to accurately locate the object. A good tracking algorithm will use all information it has about the object up to that point while a detection algorithm always starts from scratch. Therefore, while designing an efficient system usually an object detection is run on every n^{th} frame while the tracking algorithm is employed in the $n-1$ frames in between. It is true that tracking benefits from the extra information it has, but you can also lose track of an object when they go behind an obstacle for an extended period of time or if they move so fast that the tracking algorithm cannot catch up. It is also common for tracking algorithms to accumulate errors and the bounding box tracking the object slowly drifts away from the object it is tracking. To fix these problems with tracking algorithms, a detection algorithm is run every so often. Detection algorithms are trained on a large number of examples of the object. They, therefore, have more knowledge about the general class of the object.

On the other hand, tracking algorithms know more about the specific instance of the class they are tracking.

Tracking can help when detection fails:

If you are running a vehicle detector on a video and the car gets occluded by any other vehicle, the vehicle detector will most likely fail. A good tracking algorithm, on the other hand, will handle some level of occlusion.

□ Tracking preserves identity: The output of object detection is an array of rectangles that contain the object. However, there is no identity attached to the object. Let us take a dot on the detected object's bounding box. In the first frame, a particular dot might be represented by the rectangle at location 10 in the array and in the second frame, it could be at location 17. While using detection on a frame we have no idea which rectangle corresponds to which object. On the other hand, tracking provides a way to literally connect the dots!

Progress:

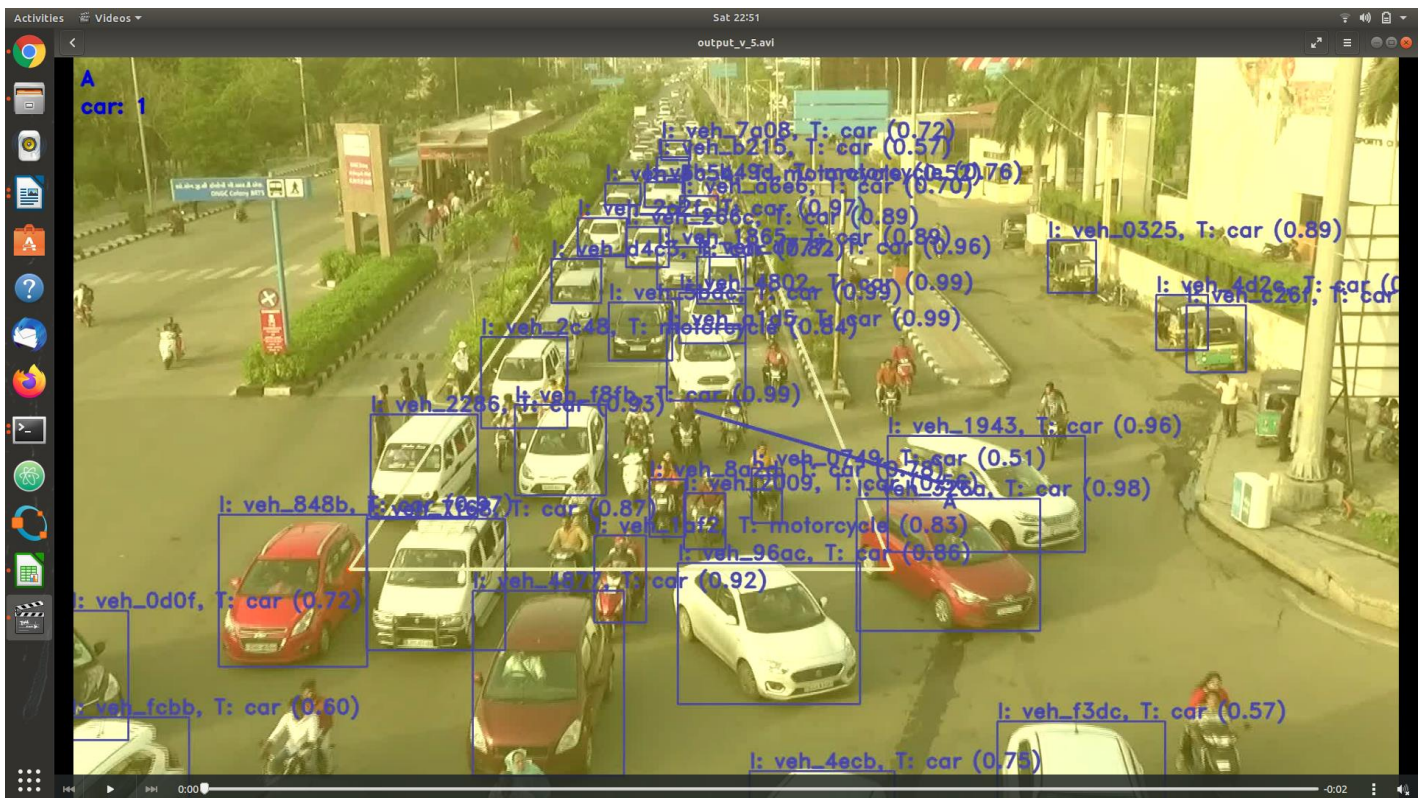
This model is completely written in Python Language so to write all codes one must learn python completely.

At initial stages the ROI was only implemented no detection and tracking is coming out as a result as shown below:



As shown no detection is there only ROI is coming out. So this is rectified later after studying about different ml trained detection models.

After defining the ROI region detection in specific ROI region is not coming out the whole frame detection is coming as output as shown below:



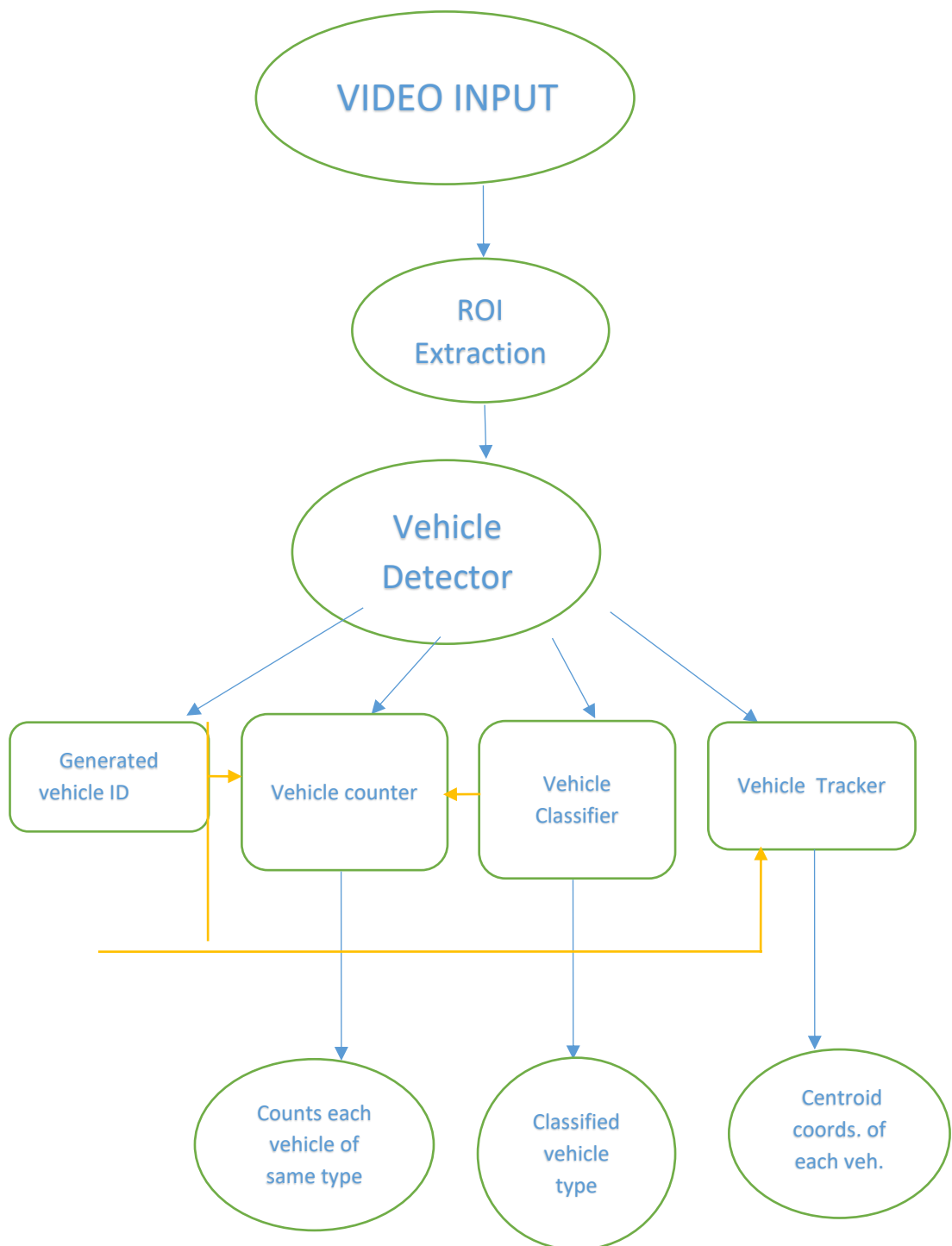
As Shown the whole area is detected not specifically the ROI region, hence it is not appropriate.

After putting the above mentioned algorithms in the ROI loop it is possible to detect within the ROI region only. That gives the specific count and monitoring of the vehicles.

After getting this data there are difficulties faced during writing the processed data into .csv file but that problem is solved by putting the mentioned requirements in the loop of ROI and then by using pandas (In computer programming, pandas is a software library written for Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series).

That gives the final processed video with all requirements fulfilled.

Codeblock Architecture



```

10     import ast
11     import os
12     import time
13     import cv2
14     import numpy as np
15
16     from util.image import take_screenshot
17     from util.logger import get_logger
18     from util.debugger import mouse_callback
19     from VehicleCounter import VehicleCounter
20
21     logger = get_logger()
22
23     # capture traffic scene video
24     is_cam = ast.literal_eval(os.getenv('IS_CAM'))
25     video = int(os.getenv('VIDEO')) if is_cam else os.getenv('VIDEO')
26     cap = cv2.VideoCapture(video)
27     if not cap.isOpened():
28         raise Exception('Invalid video source {0}'.format(video))
29     ret, frame = cap.read()
30     f_height, f_width, _ = frame.shape
31
32     detection_interval = int(os.getenv('DI'))
33     mcdf = int(os.getenv('MCDF'))
34     mctf = int(os.getenv('MCTF'))
35     detector = os.getenv('DETECTOR')
36     tracker = os.getenv('TRACKER')

```

First step is to import all the required packages, here we import some basic libraries like ast (helps Python applications to process trees), os (provides a portable way of using operating system dependent functionality), time (functions for getting local time), cv2 (computer vision and machine learning software library), numpy (package for scientific computing).

After that call user defined functions for reading the frames one by one by parsing the arguments, extraction of region of interest and getting maximum consecutive detection failures (MCDF) and maximum consecutive tracking failures (MCTF) detector options provided, YoloV3:

- YOLO is extremely fast
- YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.
- YOLO learns generalizable representations of objects so that when trained on natural images and tested on artwork, the algorithm outperforms other top detection methods.

Haar-cascade detector:

(iii) Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones.

(iv) Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

Positive images – These images contain the images which we want our classifier to identify.

Negative Images – Images of everything else, which do not contain the object we want to detect.

Detectron2:

- Detectron2 is Facebook's new vision library that allows us to easily use and create object detection, instance segmentation, keypoint detection and panoptic segmentation models
- Detectron2 is flexible and extensible, and able to provide fast training on single or multiple GPU servers.

Tensorflow-object-detection model:

- Perform detection using models created with the Tensorflow Object Detection API.
- Tensorflow's Object Detection API is a powerful tool which enables everyone to create their own powerful Image Classifiers.

Tracking is done by mainly one of this two algorithm:

KCF:

- KFC stands for **Kernelized Correlation Filters**. This tracker builds on the ideas presented in the previous two trackers. This tracker utilizes that fact that the multiple positive samples used in the MIL tracker have large overlapping regions. This overlapping data leads to some nice mathematical properties that is exploited by this tracker to make tracking faster and more accurate at the same time.

CSRT:

- In the Discriminative Correlation Filter with Channel and Spatial Reliability (DCF-CSR), we use the spatial reliability map for adjusting the filter support to the part of the selected region from the frame for tracking. This ensures enlarging and localization of the selected region and improved tracking of the non-rectangular regions or objects. It uses only 2 standard features (HoGs and Colornames). It also operates at a comparatively lower fps (25 fps) but gives higher accuracy.

```

37 # create detection region of interest polygon
38 use_droi = ast.literal_eval(os.getenv('USE_DROI'))
39 droi = ast.literal_eval(os.getenv('DROI')) \
40     if use_droi \
41     else [(0, 0), (f_width, 0), (f_width, f_height), (0, f_height)]
42 show_droi = ast.literal_eval(os.getenv('SHOW_DROI'))
43 counting_lines = ast.literal_eval(os.getenv('COUNTING_LINES'))
44 show_counts = ast.literal_eval(os.getenv('SHOW_COUNTS'))
45
46 vehicle_counter = VehicleCounter(frame, detector, tracker, droi, show_droi, mcdf,
47                                 mctf, detection_interval, counting_lines, show_counts)
48
49 record = ast.literal_eval(os.getenv('RECORD'))
50 headless = ast.literal_eval(os.getenv('HEADLESS'))
51
52 if record:
53     # initialize video object to record counting
54     output_video = cv2.VideoWriter(os.getenv('OUTPUT_VIDEO_PATH'), \
55                                   cv2.VideoWriter_fourcc(*'MJPG'), \
56                                   30, \
57                                   (f_width, f_height))
58
59 | logger.info('Processing started.', extra={
60     'meta': {
61         'label': 'START_PROCESS',
62         'counter_config': {
63             'di': detection_interval,
64             'mcdf': mcdf,
65             'mctf': mctf,
66             'detector': detector,
67             'tracker': tracker,
68             'use_droi': use_droi,
69             'droi': droi,
70             'show_droi': show_droi,
71             'counting_lines': counting_lines
72         },
73     },
74 })
75
76 if not headless:
77     # capture mouse events in the debug window
78     cv2.namedWindow('Debug')
79     cv2.setMouseCallback('Debug', mouse_callback, {'frame_width': f_width, 'frame_height': f_height})

```

In this step, Extraction of ROI is done by getting the coordinates of counting lines and ROI coordinates. By parsing the arguments of Frame, Detector, Tracker, ROI, MCDF, MCTF, Counter lines to the vehicle_counter function. Defining the path of the output file is going to be saved, and parsing all the information of tracker and detector and other parameters to the log file is done. The event of Detection and tracking and counting is shown by the 'Debug' named window.

```

84 # main loop
85 while is_cam or cap.get(cv2.CAP_PROP_POS_FRAMES) + 1 < cap.get(cv2.CAP_PROP_FRAME_COUNT):
86     k = cv2.waitKey(1) & 0xFF
87     if k == ord('p'): # pause/play loop if 'p' key is pressed
88         is_paused = False if is_paused else True
89         # logger.info('Loop paused/played.', extra={'meta': {'label': 'PAUSE_PLAY_LOOP', 'is_paused': is_paused}})
90     if k == ord('s') and output_frame is not None: # save frame if 's' key is pressed
91         take_screenshot(output_frame)
92     if k == ord('q'): # end video loop if 'q' key is pressed
93         # logger.info('Loop stopped.', extra={'meta': {'label': 'STOP_LOOP'}})
94         break
95
96     if is_paused:
97         time.sleep(0.5)
98         continue
99
100     _timer = cv2.getTickCount() # set timer to calculate processing frame rate
101
102     if ret:
103         vehicle_counter.count(frame)
104         output_frame = vehicle_counter.visualize()
105
106         if record:
107             output_video.write(output_frame)
108
109         if not headless:
110             debug_window_size = ast.literal_eval(os.getenv('DEBUG_WINDOW_SIZE'))
111             resized_frame = cv2.resize(output_frame, debug_window_size)
112             cv2.imshow('Debug', resized_frame)
113             # cv2.imshow('ROI', frame)
114
115     processing_frame_rate = round(cv2.getTickFrequency() / (cv2.getTickCount() - _timer), 2)
116     frames_processed = round(cap.get(cv2.CAP_PROP_POS_FRAMES))
117     frames_count = round(cap.get(cv2.CAP_PROP_FRAME_COUNT))
118     logger.info('Frame processed till.', extra={
119         'meta': {
120             'label': 'FRAME_PROCESS',
121             'frames_processed': frames_processed,
122             'frame_rate': processing_frame_rate,
123             'frames_left': frames_count - frames_processed,
124             'percentage_processed': round((frames_processed / frames_count) * 100, 2),
125         },
126     })

```

The main loop started by capturing each frame of the video one by one and processing on it. Here are some user-friendly options like play and pause, taking screenshots. Then some information about how many frames are processed, frame rate and other by using the data fetched by local system by using time library.

```

128         ret, frame = cap.read()
129
130     # end capture, close window, close log file and video object if any
131     cap.release()
132     if not headless:
133         cv2.destroyAllWindows()
134     if record:
135         output_video.release()
136     # logger.info('Processing ended.', extra={'meta': {'label': 'END_PROCESS'}})
137
138
139 if __name__ == '__main__':
140     from dotenv import load_dotenv
141     load_dotenv()
142
143     from util.logger import init_logger
144     init_logger()
145
146     run()

```

It is just the closure of all the windows, and saving the output files and the log data.