CSCI 69500 Project Report

Highway Lane Detection and Distance Per Pixel Detection based on CCTV with Unknown Aiming Direction

Joint Transportation Research Program - Incident Detection

Anup Atul Mulay,

anmulay@iupui.edu

Submitted to the department of Computer & Information Science

For the fulfilment of the requirement of M.S. Project

Project Advisors

Dr. Stanley Chien

schien@iupui.edu

Dr. Snehasis Mukhopadhyay

smukhopa@cs.iupui.edu

Dr. Jiang Zheng

jzhenge@iupui.edu

Date: January 25, 2021

Table of Contents

1. Introduction

The United States has continuously growing road highway networks. It is beneficial to have intelligent highway surveillance systems for efficient traffic management. The State of Indiana has a funded Joint Transportation Research Program (JTRP) for the Department of Transportation (INDOT) and academia to focus on the technological impact on the efficient operation of transportation systems throughout Indiana state. There are hundreds of CCTV cameras installed on Indiana highways for road incident surveillance. To monitor these road incidents such as congestions, the operators at the Indiana Department of Transport need to control CCTV cameras manually. Due to fewer operators than the number of cameras, most of the incidents could not be well monitored. Therefore, intelligent traffic monitoring by traffic status identification of each lane on the road is required.

2. Project Overview

The newer advancements in computer vision and Deep Learning have enabled us to perform efficient object detection and tracking from the available video data feed. So, this project aims to identify highway incidents such as congestion through constant CCTV footage surveillance. With the help of object detection and tracking, we can identify the number of lanes on the road, their exact lane center locations with respect to the video frame, and the lane direction. Once we identify the lanes, we compute the number of vehicles passing through them and the average speed of vehicles passing through the respective lanes projected per hour. With the projected number of vehicles and speed, we can filter normal speed, slow speed, and congestion on the road.

2.1 Description of project modules

The project is divided into the following modules (Refer Fig. 1):

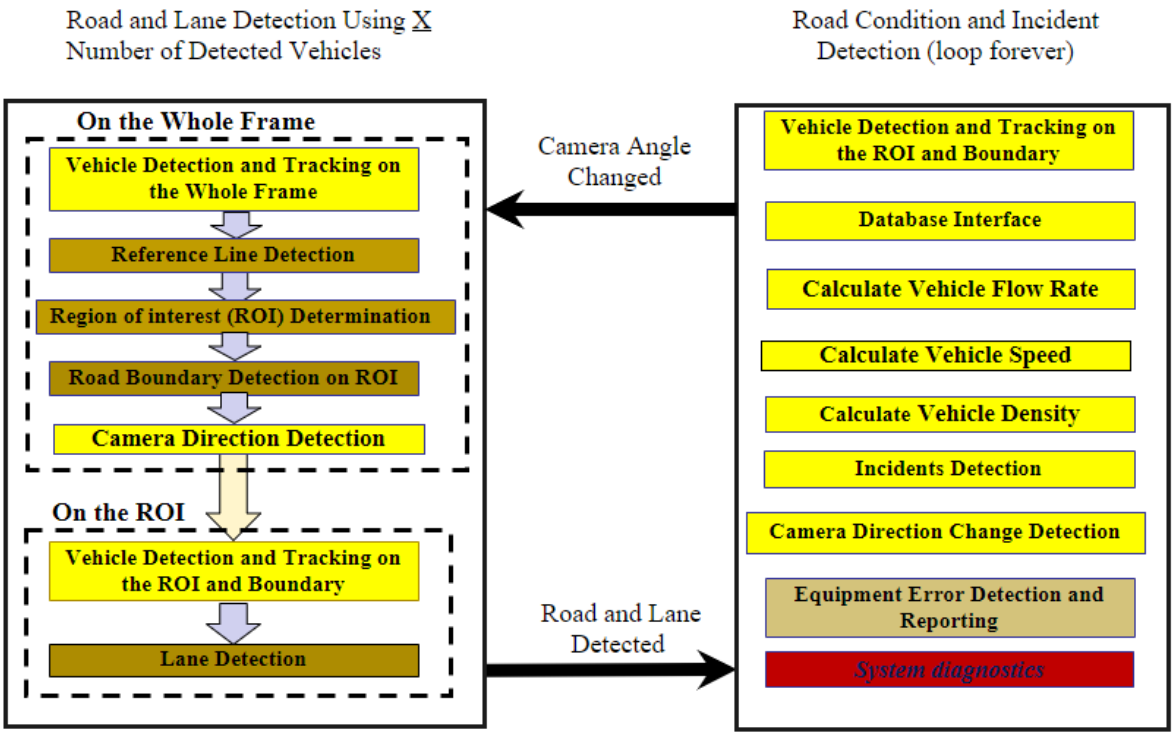


Fig 1. Project Overview

Detection of Baseline and Region of Interest: As the video is a data of 2-D array frames. So, the baseline is the best possible row on the frame where the probability of observing the vehicles is the most, i.e., almost all the vehicles cross that line on the frame voting only once. The region of interest is a region around the baseline where we are interested in tracking the vehicle. It is set to be 100 pixels up and 200 pixels down with respect to the baseline.

Road Boundary Detection: It is an image segmentation-based method for extracting the road boundary. So, we get the segment of the frames from the video explicitly belonging to the highway, excluding other segments such as road sideway trees, sky, etc.

Object/Vehicle Detection: YOLO_v4 based object identification technique inputs the extracted frame from the video and performs the high-level feature extraction using convolution neural networks. CNN can classify extracted high-level features into various classes such as 'Car', 'Truck', 'Aeroplane,' 'Bus,' etc. From these features, we are more interested in the detection of cars. On the video frame, we get the bounding box around the detected car and its dimensions such as starting pixel coordinates, height, the width of the bounding box, etc.

Object/Vehicle Tracking: Deep SORT (Deep Simple Online and Real-Time Tracking) based multi-object tracking technique enables us to identify the same vehicle's coordinates along with the consecutive frames of the video. We track each detected vehicle's position within the Region of Interest (ROI), where the vehicle intersects the baseline at a specific coordinate with respect to the frame.

Lane Center and Direction Detection: Object Detection and Tracking based road lane center and direction detection technique provide the number of lanes present on the video frame and their lane center coordinates (with respect to frame) and associated direction. These directions are annotated +1 for vehicles appearing to approach the camera and -1 for the vehicles moving away from the camera.

Flow Rate and Speed Calculation: Once we have the lane center coordinates, we can predict the number of vehicles passing through those lanes per hour. Also, we know that the white lane markings done on roads are 10ft in length along the lane's direction. For the frame, we compute the meter per pixel, which predicts the average speed of vehicles passing through the lane per hour.

Congestion Detection: From the above module, we have the flowrate and speed values for each of the road's lanes. So, through observation, we set the threshold on those values to classify the traffic scenarios into "Normal Speed," "Slow Speed," and "Congestion" traffic statuses. The flowrate and speed values update periodically to identify the road traffic incident. So, the necessary measurements need to be taken.

2.2 Objectives of this project

A**.** The prime objective of the project is to automate the real-time traffic surveillance system to record the traffic congestion incidents and take necessary actions such as

- Alert the travellers
- Alert the congestion control traffic officers
- Provide a detour

B. Along with the real-time tracking of congestion incidents the historical analysis of congestion tracking can provide following information:

- Time and duration of congestion incidents for the give road
- Prediction of road's congestion status for the give time instance

All the above objectives are not possible without strong technical analysis with help of computer vision and deep learning of data feed provided by CCTV cameras. From the technical perspective to perform the efficient image analysis, the goals are

- Effectively choose the baseline, region of interest and road boundary
- Implement an efficient vehicle detection and tracking algorithm
- Precisely use vehicle detection and tracking information for lane center and direction
- detection
- Using lane centers and directions generate the flow rate and speed for each lane
- Efficiently threshold flow rate and speed for relevant traffic congestion

From the above technical objectives, I contributed for following objectives.

- Improve Road Lane Center Detection and Lane Direction Detection
    - Lane Finding Algorithm
    - Lane Filtering Algorithm
- Implementation of Incident Detection Module
- Implementation of Meter Per Pixel Module

Let us have a look at each of the contributions one by one.

3. Lane Center and Direction Detection

The lane center and direction detection can use different methods. The approach used in this project depends on the information provided by vehicle detection and vehicle tracking modules.
Most of the proposed lane detection systems have the settings where the camera is fitted in the car focusing the forward movement on the road.

Fig 2. Camera positioning for available lane detection system

In our case, the camera fitted on the surveillance or light poles capturing the entire view of road traffic movements. So, we cannot rely on the above proposed methods. Therefore, we make the use of information provided by vehicle detection and tracking modules.

Fig 3. CCTV camera positioning in this project

3.1 Find lane centers

The whole idea in lane finding is based on the assumption that most vehicles are driven within a lane and cross lane occasionally. So, a lane center in the x-direction is where many cars pass through. The vehicle detection module detects the position of vehicle for the given frame and draws the bounding box around it. So, the detection of a vehicle provides the starting pixel coordinates, width, and height of bounding box. The vehicle tracking module tracks the trace of movement of each vehicle for subsequent frames. The baseline is a reference line where the vehicles are most observed. We record the intersection i.e., crossing, coordinates of centroid point of vehicle's bounding box and the baseline and analysis where on the baseline the vehicle pass.

Therefore, as a result we have intersection information for a certain number of vehicles (in our case initially 250 vehicles and after improvement 100 vehicles – observed values). Since a large vehicle like a truck has a large bounding box which covers more than one lane, its center location does not indicate lane well. Therefore, among these vehicles we are interested in the information of passenger (or small) cars. So, we discarded the vehicles with bounding box width greater than median bounding box width of all observed vehicles.

3.1.1. Using mean filter to find lane centers

To visualise where every car passes the baseline, we use a histogram representation where the x-axis represents the x-axis coordinates of the baseline where car intersects it, and y-axis represents the number of vehicles passing/crossing from the corresponding x-axis coordinate. Therefore, the representation results into a histogram plot. As you can see in Fig 5a each intersection point represents the spike in the histogram. But we are interested in the cumulative effect of these intersections which effectively represent the road lanes. Therefore, we used a mean filter with widow size of 5 to smooth the histogram. The mean filter smoothening works as follows (Fig 4):

For the spikes with window size equal to 5:

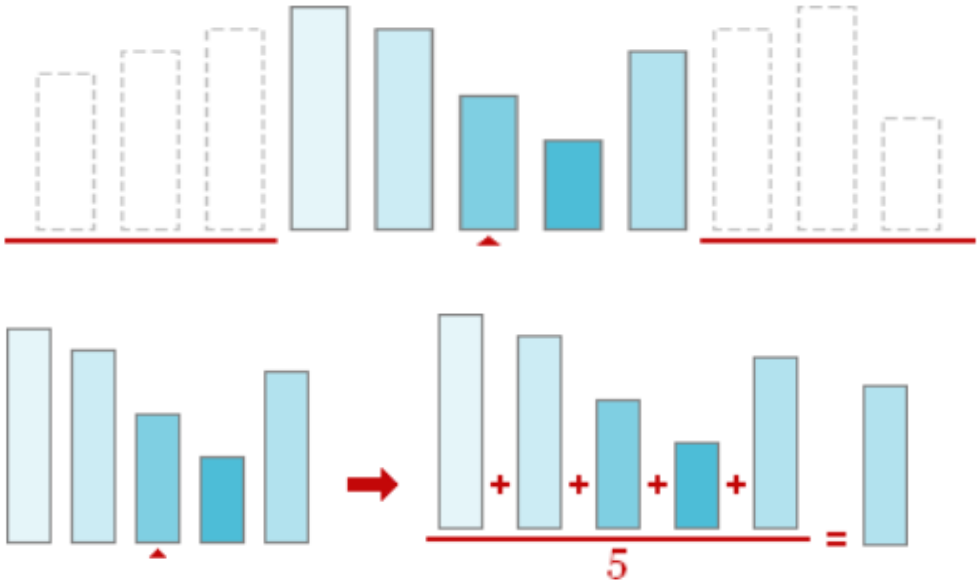


Fig 4. Mean filter with window size of 5 for smoothening

Let x be the current pixel location then (x-1) and (x-2) will be the previous pixel and (x+1) and (x+2) will be the consecutive pixels.

There the equation of mean filter smoothening becomes:

$$\frac{(x-2)+(x-1)+(x)+(x+1)+(x+2)}{5} = X'$$

For boundary conditions, i.e., the first and last pixel the padding is performed to balance the above equation. Because for the first spike there are no previous pixel and for the last spike there are no consecutive pixels available.

Here, each pixel (in overlapping fashion), its adjacent pixels are considered which falls within the given window size, are then averaged. Such averaging over each of the pixels gives the smoothened representation of pixel value resulting into a wave like combination of peaks and valleys. We perform the above smoothening technique multiple times until we get the representation of peaks and valleys as probable lane centers and white lane markings, respectively.

As mentioned above, Fig 5a shows the original histogram. Applying mean filter for the first time we get a step signal like representation of spikes (Figure 5b). So, to generate the wave like combination of peaks and valleys (probable lane centers and white lane marking segments) iterative mean filtering should be applied.

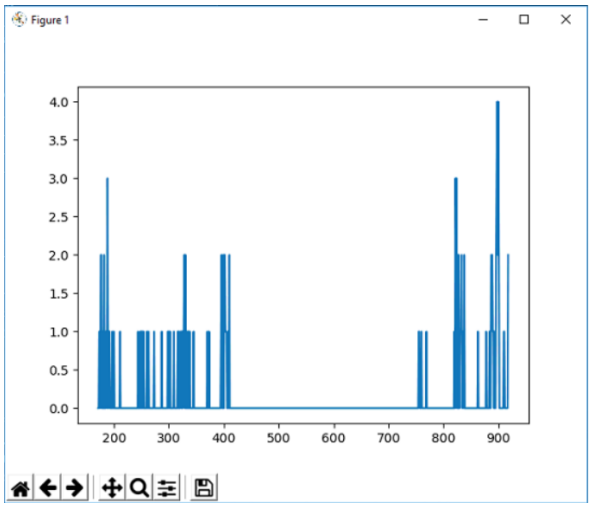


Fig 5a

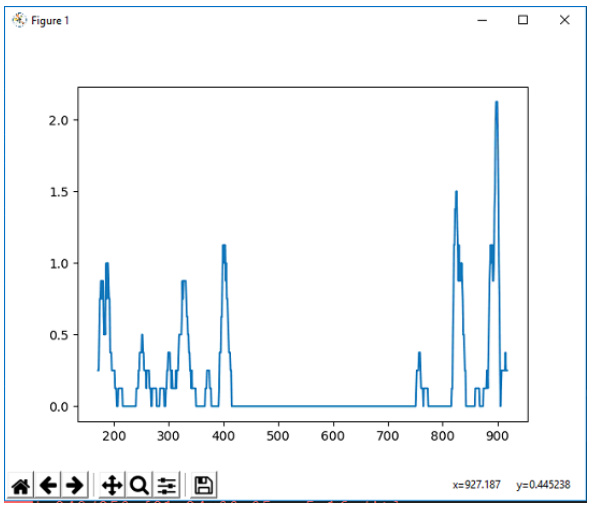


Fig 5b

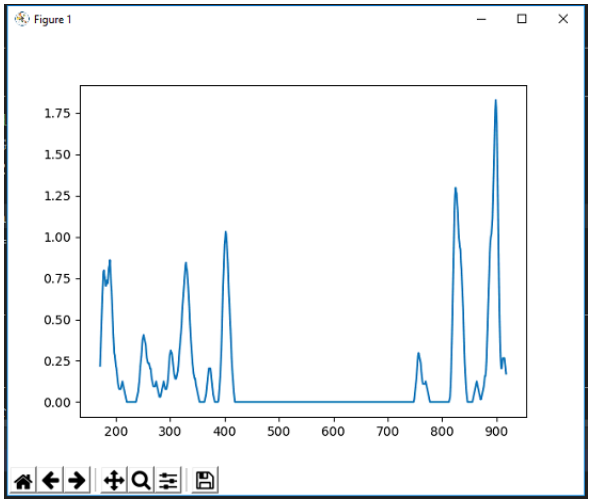


Fig 5c

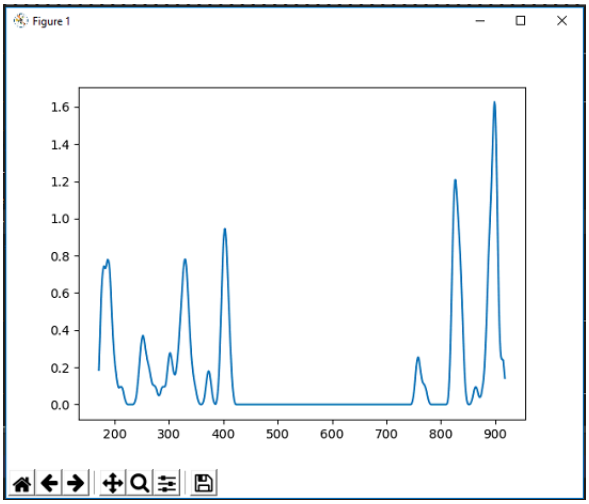


Fig 5d

Applying mean filter, the second time on Fig 5b we get spikes representation shown in Fig 5c. As we can see in Fig 5c still the peaks are not distinct, there are irregular local minima and maxima; their representation is not near to the probable lane centres positions with respect to give video frame therefore applying mean filter 3rd time gives Fig 5d.

Fig 5d distinctly smoothens the abnormalities observed in Fig 5c but there are still minute sharp peaks. If these minute sharp peaks get captured as lane centers then it does not match with the expected number of lanes on the frame. So further smoothening i.e., 4th iteration gives clear distinct representation of peaks and valleys closely associated to expected results as in Fig 5e (eliminating the minute sharp peak noises).

The 5th iteration of smoothening from Fig 5e to Fig 5f is to strengthen the detected peaks and valley representation to perform the smoothening of any unhandled unexpected local minima peak scenarios.

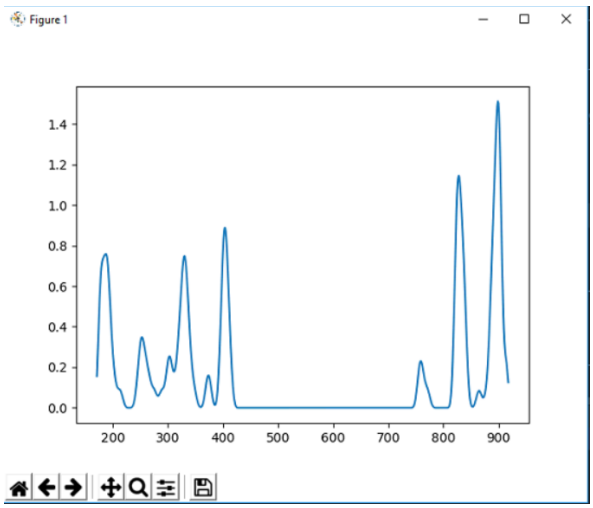


Fig 5e

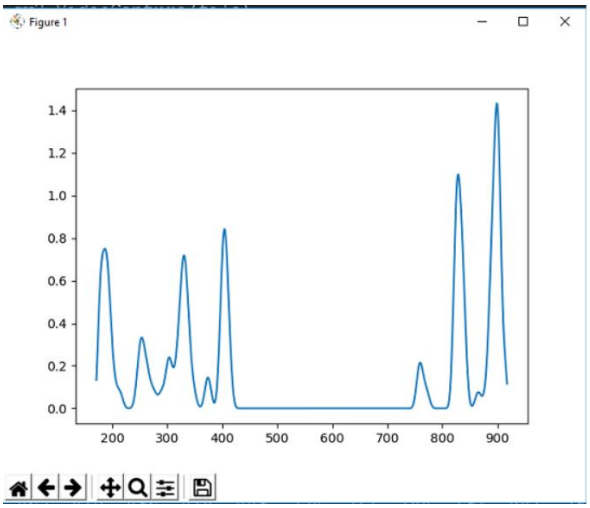


Fig 5f

Fig 5. Visualisation of intersection of cars and baseline (See each row left right)

Similar explanation as Fig 5a – 5f is valid for following histograms (Fig 6a – 6f). Fig 6a shows the original histogram. Applying mean filter for the first time we get a step signal like representation of spikes (Figure 6). So, to generate the wave like combination of peaks and valleys (probable lane

centers and white lane marking segments) iterative mean filtering should be applied. Applying mean filter, the second time on Fig 6b we get spikes representation shown in Fig 6c. As we can see in Fig 6c still the peaks are not distinct, there are irregular local minima and maxima; their representation is not near to the probable lane centres positions with respect to give video frame therefore applying mean filter 3rd time gives Fig 6d. Fig 6d distinctly smoothens the abnormalities observed in Fig 6c but there are still minute sharp peaks. If these minute sharp peaks get captured as lane centers then it does not match with the expected number of lanes on the frame. So further smoothening i.e., 4th iteration gives clear distinct representation of peaks and valleys closely associated to expected results as in Fig 6e (eliminating the minute sharp peak noises). The 5th iteration of smoothening from Fig 6e to Fig 6f is to strengthen the detected peaks and valley representation to perform the smoothening of any unhandled unexpected local minima peak scenarios. Fig 6g shows the actual x-axis pixel coordinates of lane centers along the baseline. Here in Figure 6h, on the left of the divider one lane is missing than expected.

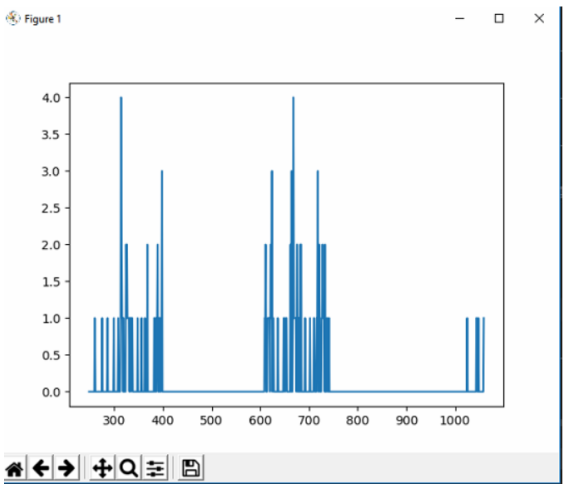


Fig 6a

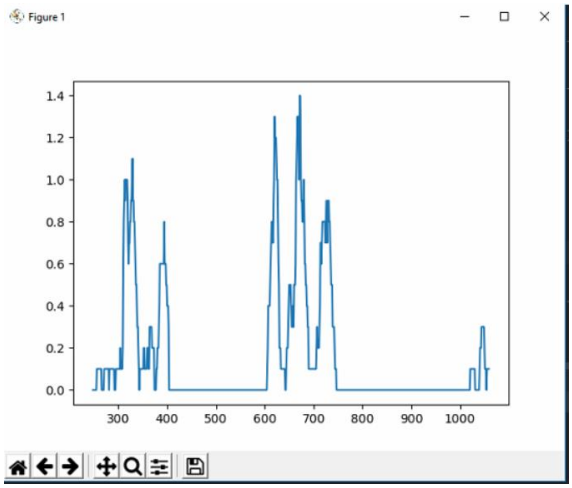


Fig 6b

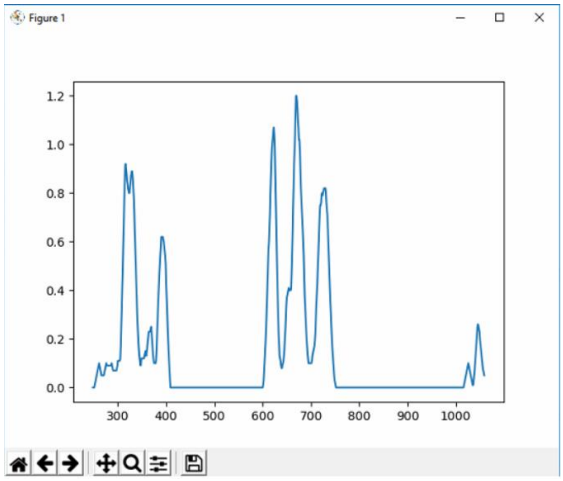


Fig 6c

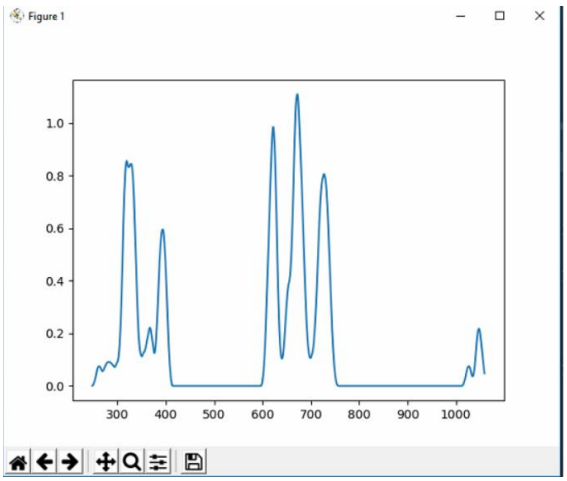


Fig 6d

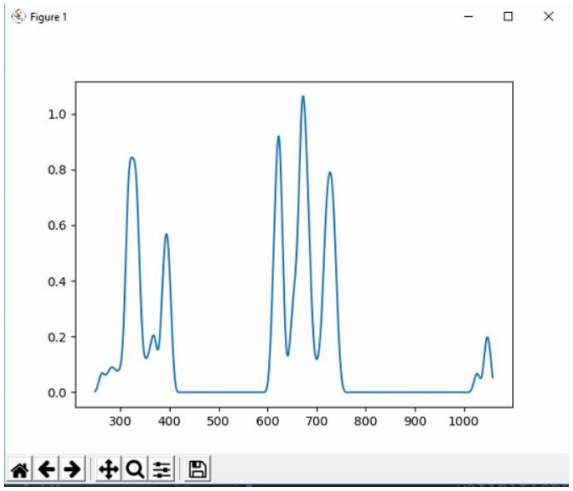


Fig 6e

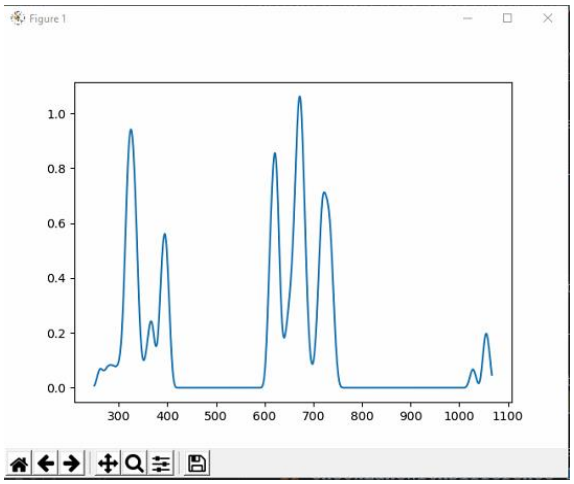


Fig 6f

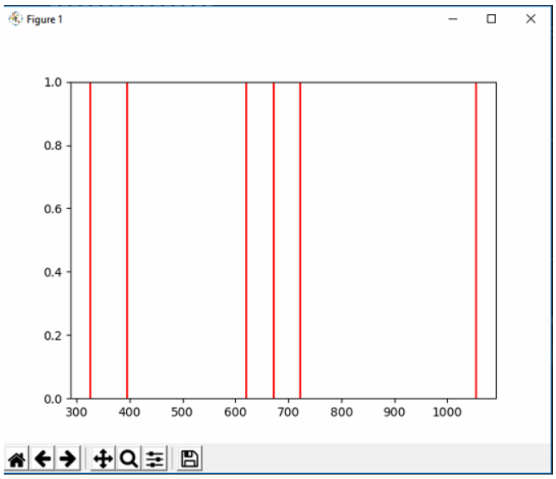


Fig 6g

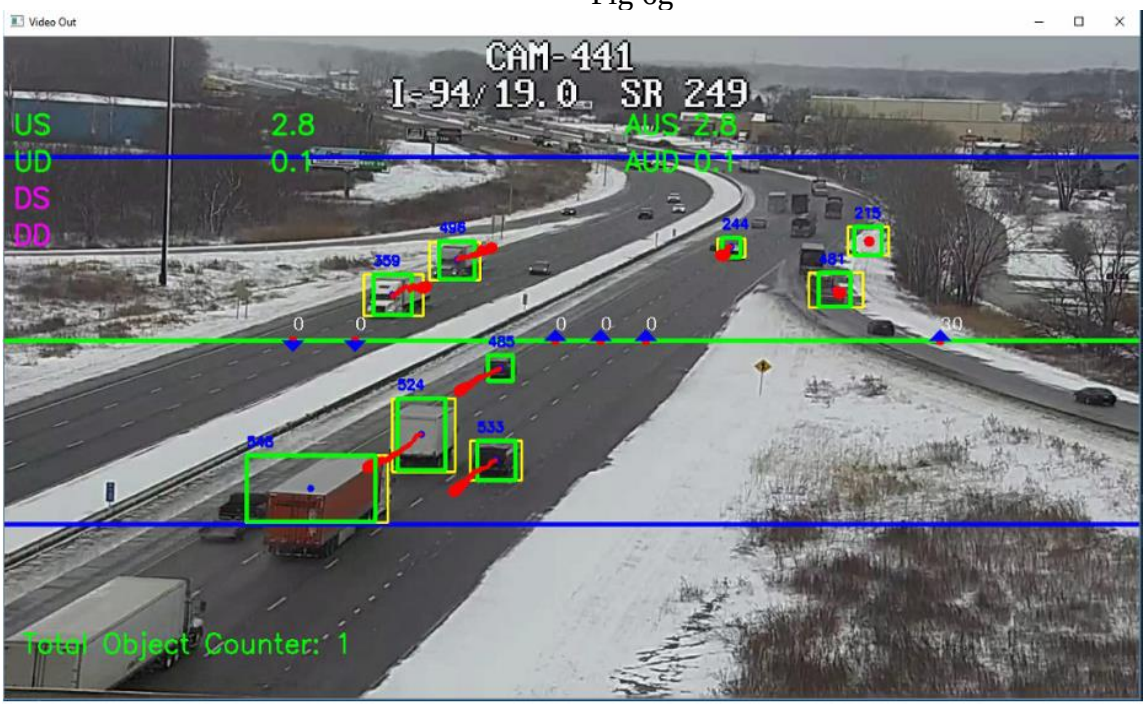


Fig 6h

How to determine the number of iterations is a question. When the iteration number is small, we get a lot of false peaks hence extra lanes. When the iteration number is big, we lost some true peaks hence let some lanes not detected. Figure 7 shows the result of more than 5 iterations. Therefore, we restrict smoothening to 5 iterations, so we do not lose true peaks. Now we have the Figure 5f and 6f representation of peaks which are probable lane centers. Among these probable lane centers we need to identify the actual lane centre peaks for the given frame. In the next section, we describe how to use the car width information to remove the remaining false peaks.

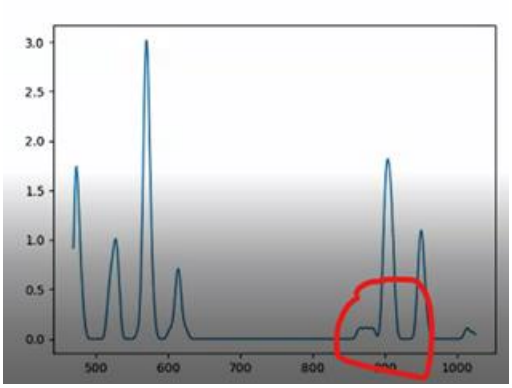


Fig 7. Flattening effect of Over smoothening

3.1.2 Removing peaks that are not a lane centers using lane width information

We do know the high peaks are the real lane centers. We do not know if the small peaks are lane centers. Here we use the lane width to eliminate the peaks that are not the lane centers. If a small peak is too close to a lane center peak, it is a false lane center.

The next question is how to find the lane width. Through the observation of videos of many cameras, we learned that the bounding box of small cars are within one lane width. So, our next step is to find the bounding box width of small vehicles at the reference line.

For the given camera scenarios, we see truck, trailers, cars on the road. The bigger heavy vehicles' bounding box often has a horizontal width greater than the horizontal lane width. Therefore, for avoiding the problem, we should consider vehicles smaller than equal to median width of all the detected vehicles. The expected lane centers must be at an approximate median width distance from each other. But as the vehicles move away from the camera their bounding box size goes on decreasing and for the vehicles approaching the camera their bounding box width goes on increasing. So, we only get the bounding box width on the baseline.

From the statistical averaging of vehicles' bounding box width, we observed that the distance between the adjacent lane centers over a horizontal line about 1.34 * median_car_wdith. Therefore, for all the computations of lane filtering algorithm we used the modified median_lane_width = 1.34 * median_car_width.

Observing the videos, we can show following facts:

A. Most of vehicles drive within a lane

B. The width of the box around the small car is about the 1.34* the width of a lane

C. If lanes 1 and 3 have a large number of cars, lane 2 should have no less than half of cars either through lane 1 or lane 3.

D. The width of each lane on an image is changing from right to left

E. Some cars change lanes

Following lane finding algorithm use these facts.

3.2. Detailed algorithms for lane finding method

3.2.1. Lane finding algorithm using mean filter

As explained in the section 3.1 the following lane finding algorithm uses the mean filter to generate the probable lane center peaks (figure 5f and 6f). The lane filtering algorithm removes the peaks that are not lane centers using lane width information.

Lane_Finding(Frame, Sampling_rate, baseline, cross_baseline_list)

lane_centers = []
lane_directions = []
Id_list = []
Intersection_list = []
Trusted_intersection = []
Width_list = []
Direction_list = []
Trusted_direction_list = []
Lane_Detected = False

Definition of the variables:

1. **lane_centers**: list to store the final coordinates of lane centers.
2. **lane_directions**: list to store the associated final lane directions to corresponding lane centers.
3. **Id_List**: list to stores the IDs of detected vehicles (given by tracking module)
4. **Width_list**: list to store the bounding box widths (in pixels) for the detected vehicles.
5. **Intersection_list**: list to store x-axis intersection coordinate of vehicle's centroid point and Baseline.
6. **Trusted_intersection**: list to store intersection point of vehicle and baseline for the vehicles having tracking trace greater than 20 pixels.
7. **Direction_list**: list to store associated direction to corresponding detected vehicle.
8. **Trusted_direction_list**: list to store associated direction to corresponding detected vehicles having tracking trace greater than 20 pixels

```
If not Lane_Detected
{
        If cross_baseline_list is not None and length(cross_baseline_list) >=
        sampling_rate
        {
                - Compute Trusted_cross_baseline_list
                - Compute Id_list, Intersection_list, Trusted_intersection_list,
                  width_list, direction_list, Trusted_direction_list
                - Calculate Median Car Width from the vehicle width_list
                - Filter intersection_list and direction_list as corresponding
                  vehicle_wdith <= median_car_width
                - Generate Histogram of intersection_list as x-axis and y-axis the
                  scaled number of vehicles passing through x
                - Get Histogram peaks (Probable Lane_centers) and Adjacent Valleys
                - median_lane_width = median_car_width * 1.34
                - Final Lane_centers = CALL False Peak_Filtering Algorithm
                - From Detected lanes and Trusted_intersection_list find the index of
                  lane required to find its associated direction Trusted_direction_list
                - Find all the associated directions
                - Send Lane_Centers and Lane_directions to other modules in required
                  format
                - Send Lane_Centers and Lane_directions to display on video frame
        }
        else
        {
                PASS
        }
}

return lane_centers, lane_directions
```

3.2.2. False peak removing using lane width information.

Definitions:

- Up_Road- On a road, vehicles moving in up direction (-1: cars moving away from camera)
- Down_Road- On a road, vehicles moving in down direction (+1: cars moving towards camera)
- Edge lane - The lane has only one adjacent lane on Up_Road or Down_Road
- Inner lane - The lane has two adjacent lane on both sides on Up_Road or Down_Road
- Sampling_rate - User defined number of vehicles under observation
- Number_of_vehicles_crossing_baseline - Number of vehicles crossed baseline at time t
- Baseline_intersection_points - Points of intersection of centroid of cars' bounding box with baseline
- Direction_points - Directions associated to intersection points (See calculation of direction_points)

Lane Filtering Algorithm Pseudo Code:

- Find highest peak from the Histogram as the first found lane

Repeat Following Algorithm Until all the probable lane centers are checked
{
Find the height of nextPeak
if(distance(nextPeak) and each of the known lanes >= 1.2 * median_lane_width)
- THIS IS A NEW LANE
elseif(distance(nextPeak) and any of its adjacent lane < 0.75 * median_lane_width)
- THIS PEAK IS A NOISE NOT A LANE
else
{ if(height(nextPeak) < 0.5 lower_height(adacent_lane) //(between 0.7* Median_lane_width and 1.2 * median_lane_width) <FACT C>)
- THIS PEAK IS A NOISE NOT A LANE
elseif(height(adjacent_valley) < 0.02 and Ratio (AbsDiff(height(nextPeak), height(adjacent_valley)), height(nextPeak)) >1.01)
- THIS IS A LANE
elseif (AbsDiff(height(nextPeak), height(adjacent_valley)) < 0.70 *height(nextPeak))
- THIS PEAK IS NOISE NOT A LANE
else
- THIS IS A LANE
}

Explanation:

- **nextPeak: nextPeak is the next highest peak not processed yet (either a lane center or a noise).**

- **if(distance(nextPeak) and each of the known lanes >= 1.2 * median_lane_width)**: The distance between this peak and each of the known lanes is greater than or equal to 1.2*D then **it is a new lane**.

- **elseif(distance(nextPeak) and any of its adjacent lane < 0.75 * median_lane_width):** The distance between this peak and any known adjacent lane is less than 0.7*D then the **Peak is a noise**.
- **else:** (there are 1 lane on 1 side or 2 lanes on both sides of the peak between 0.7D and 1.2D)

- **if(height(nextPeak) < 0.5 lower_height(adacent_lane) //(between 0.7*Median_lane_width and 1.2 * median_lane_width):** If (the height of this peak is less than 0.5 of the car count in the adjacent lower count lane (between 0.7D and 1.2D)) then **its noise.**

- **elseif(height(adjacent_valley) < 0.02 and Ratio (AbsDiff(height(nextPeak), height(adjacent_valley)), height(nextPeak)) >1.01):** If Considered valley Height is < 0.02 and ratio of difference between peak under consideration and nearest valley and height of peak is not > 0.97 then **it is a lane**. (It is based on observed data)

- **elseif (AbsDiff(height(nextPeak), height(adjacent_valley)) < 0.70 *height(nextPeak))**: The difference between the height this peak and the valley between this peak and closest lane is small then the **peak is a noise.**

- **else:** The **peak is a lane center**.

For step-by-step execution of these steps please refer Appendix B on page no 52.

3.3 Testing results for lane detection

Example 1 Lane Detection Under the Sunshine in the Morning: Using all the above data we compute the lane center coordinates and associated directions for the given video as follows:

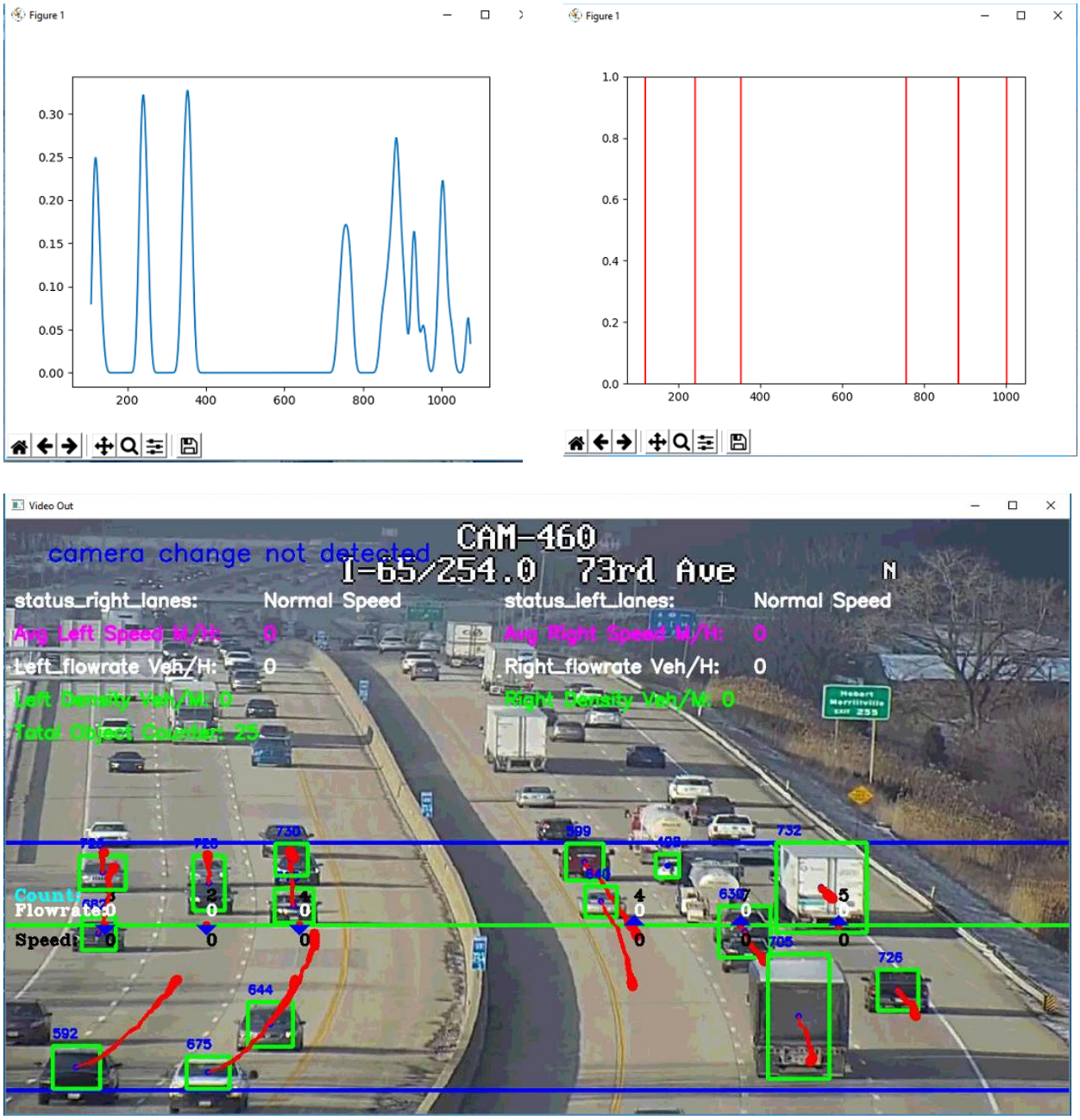


Fig 8. Result of lane finding algorithm for mathematical explanation with baseline at 491

The mathematical explaining is according to the **lane filtering algorithm on the Page No. 15**.
For Detailed explanation of every step follow **Appendix A & B. Page No.47 - 56**
So, the Final Lane_Centers are [119.0, 241.0, 353.0, 756.0, 884.0, 1002.0]
Therefore, final Lane Directions [1, 1, 1, -1, -1, -1]

+1 ⇒ Vehicles moving downwards in the frame i.e., approaching the camera
-1 ⇒ Vehicles moving upwards in the frame i.e., moving away from the camera

Example 2 Lane Detection in a Cloudy Day:

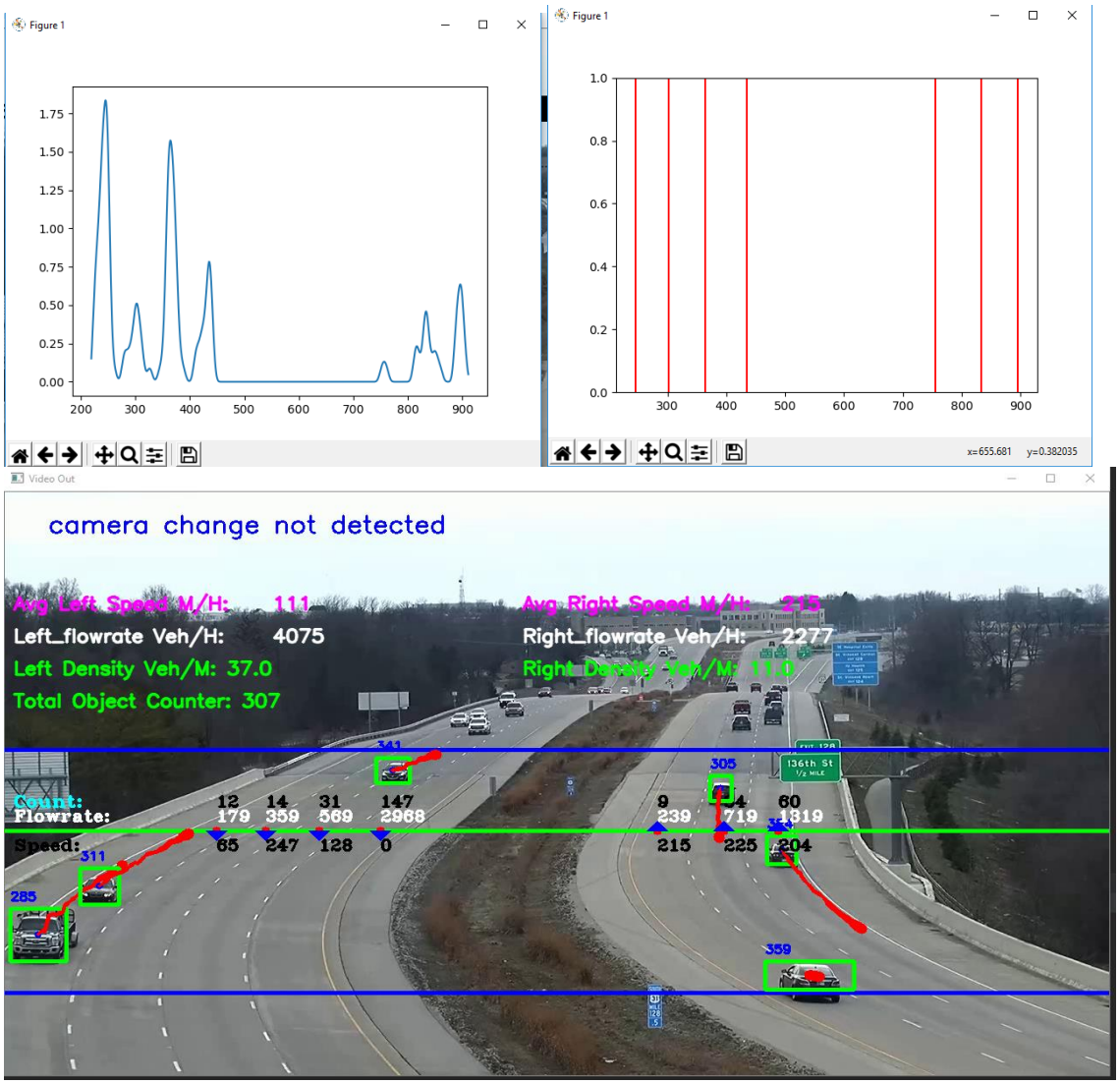


Fig 9. Other results -1: lane finding algorithm with baseline at 418

The mathematical explaining is according to the **lane filtering algorithm on the Page No. 15**.

The Detailed explanation of every step follows **like Appendix A. Page No.47 - 52**

So, the Final Lane_Centers are [240, 309, 361, 437, 760, 832, 891]
Therefore, final Lane Directions [1, 1, 1, 1, -1, -1, -1]

+1 ⇒ Vehicles moving downwards in the frame i.e., approaching the camera
-1 ⇒ Vehicles moving upwards in the frame i.e., moving away from the camera

Example 3 Lane Detection Under the Sun in Day Time:

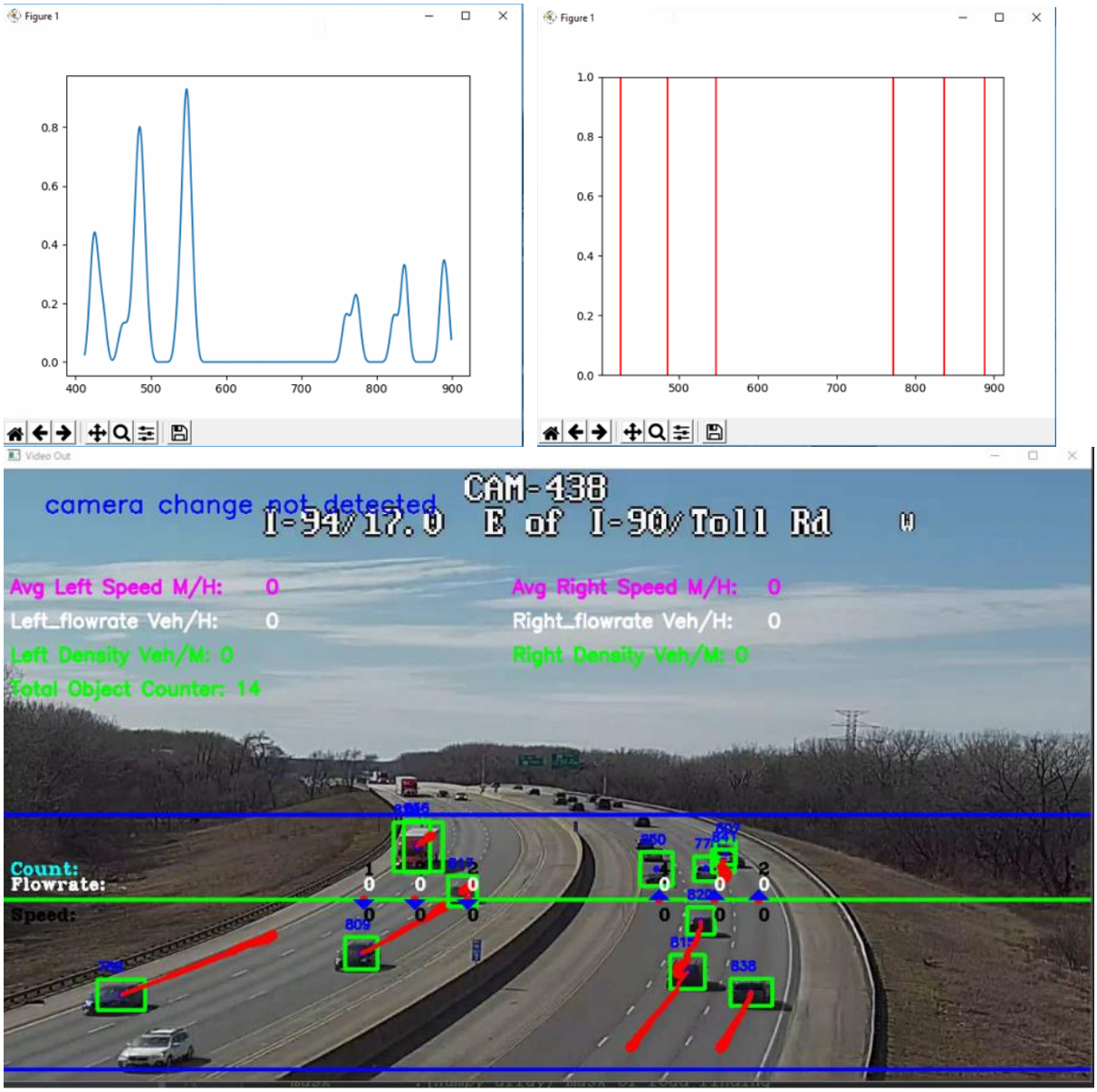


Fig 10. Other results -2: lane finding algorithm with baseline at 506

The mathematical explaining is according to the **lane filtering algorithm on the Page No. 15.**

The Detailed explanation of every step follows **like Appendix A. Page No.47 - 52**

So, the Final Lane_Centers are [240, 309, 361, 437, 760, 832, 891]
Therefore, final Lane Directions [1, 1, 1, 1, -1, -1, -1]

+1 ⇒ Vehicles moving downwards in the frame i.e., approaching the camera
-1 ⇒ Vehicles moving upwards in the frame i.e., moving away from the camera

3.4. Idea for lane direction assignment

To assign the direction to each detected lane center we consider the direction of car object passing from that detected lane center coordinate or the x-axis coordinate which is nearest to that lane center. The car object gets its associated direction as follows.

The video is a formulation of frames where n number of frames are flashed sequentially per second. Vehicle detection module detects the car on the current frame and draws the bounding box around it with parameters as bounding box starts coordinate (upper left x, y), width and height of bounding box.

Vehicle detection module detects the same car on the subsequent frames and vehicle tracking module tracks the centroid point coordinates of that car for those subsequent frames. As the car moves in the subsequent frames the tracking module can calculate the change of position and associate the direction to the car object.

The associated directions are:

+1: for cars moving downwards i.e., cars moving towards the camera
-1: for cars moving upwards i.e., cars moving away from the camera

Each car should get their associated direction via above mentioned method. So, we expected that each car with correct assignment of direction. But we faced the following problem.

Direction Assignment is as follows:

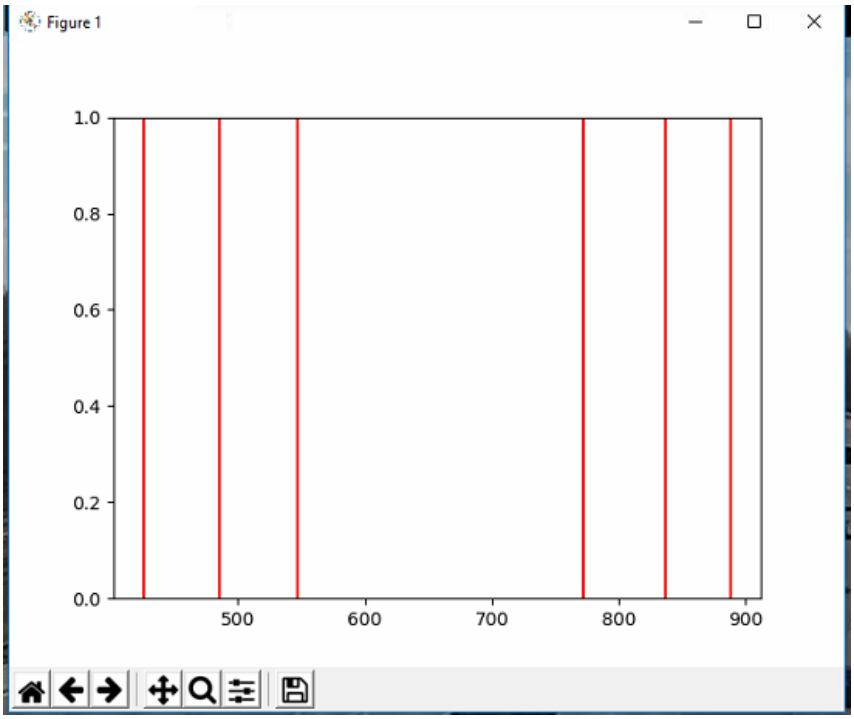


Fig 11a Lane Center Coordinates

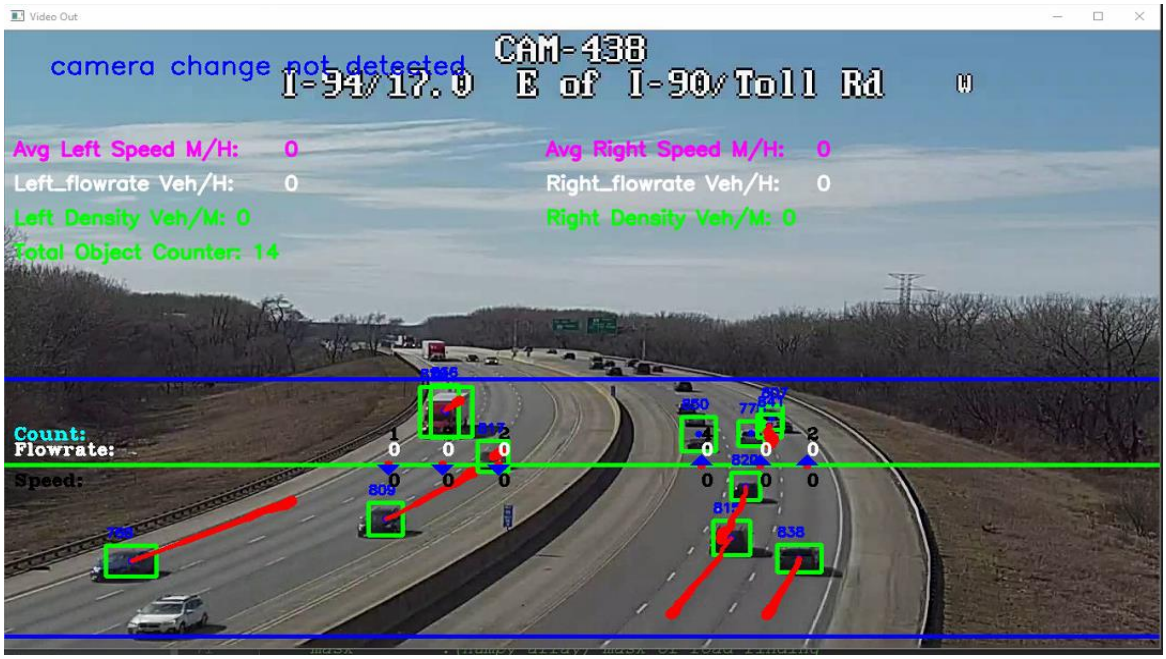


Fig 11b. Direction Assignment to Lane Centers

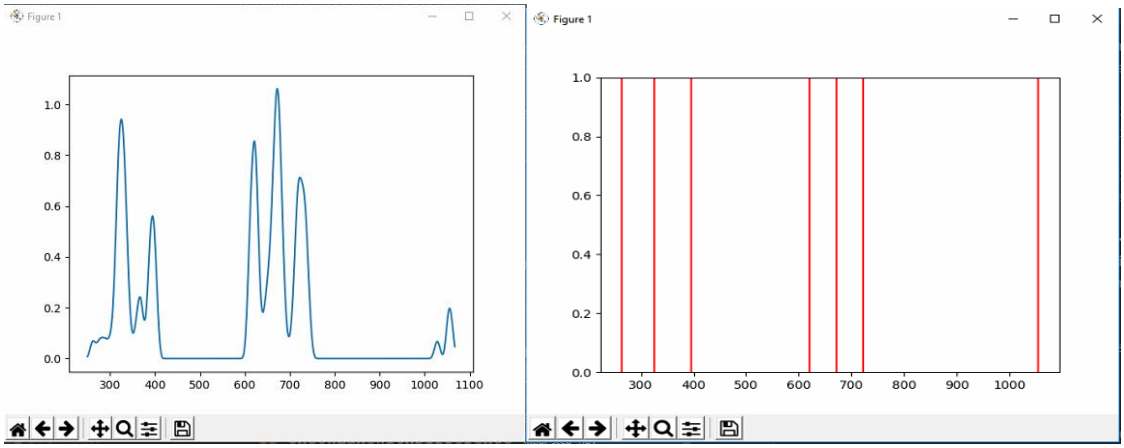
Lane Centers [425, 485, 547, 772, 837, 889]
Lane Directions [1, 1, 1, -1, -1, -1]

On the baseline the coordinates 425, 485, 547, 772, 837 and 889 are the x-axis coordinates of lane centers.

The vehicles passing through coordinates 425, 485 and 547 or to the nearest coordinate of these respective coordinates are in downward direction i.e., they are approaching the camera. So, the direction is +1.

The vehicles passing through coordinates 772, 837 and 889 or to the nearest coordinate of these respective coordinates are in upward direction i.e., they are moving away from the camera. So, the direction is -1.

3.5 Techniques for obtaining correct lane direction



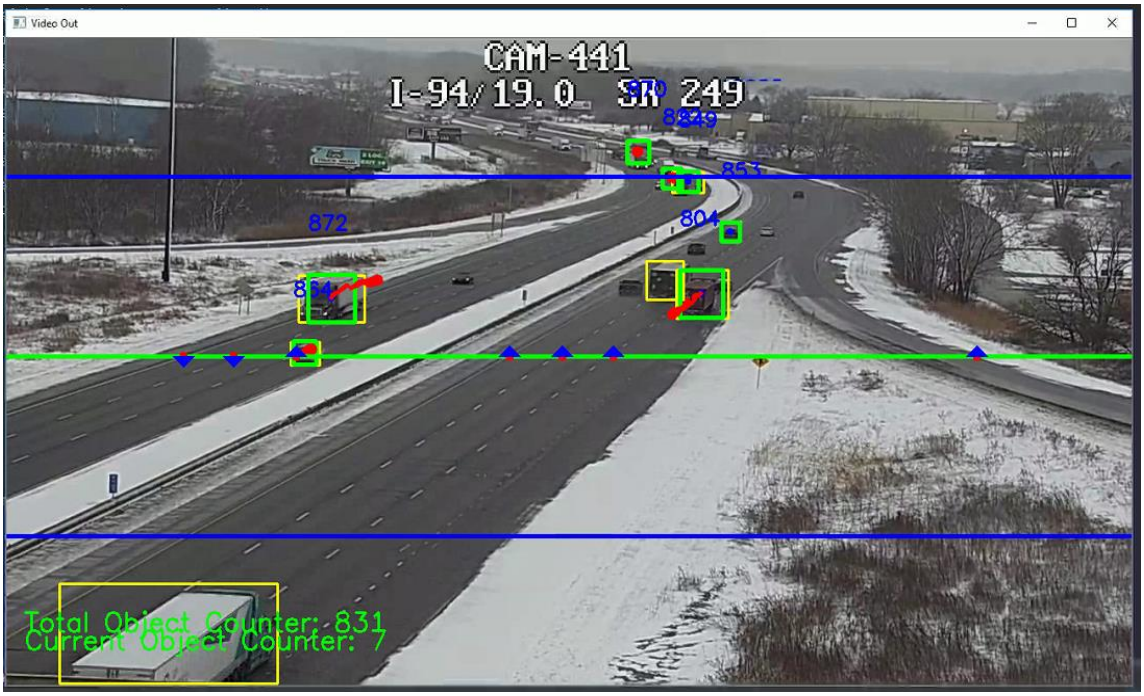


Fig 12. Lane with wrong direction

Here in Figure 10; the lane on the left side of the divider has been assigned the wrong direction. On the left side of the divider the cars are approaching the camera, so all the lanes need to be assigned +1 i.e., downward direction but one lane has upward direction; we observed the reason for this in the tracking trace and direction provided by vehicle tracking module.

As the video is a formulation of frames where n number of frames are flash sequentially per second. We observed that for some car objects passing from the histogram peak coordinate or the nearest closest coordinate has the tracking trace shorter than 5 pixels. For such shorter traces; for the consecutive frames; after vehicle detection the computed vehicle's bounding box centroid point is not progressed in the expected forward direction i.e., in our case the centroid point should move in the downward direction. But it is either at the same pixel coordinate or the one pixel coordinate upwards. And for direction calculation we use the trace start and trace

end to determine the direction for the vehicle. In that case vehicle gets the wrong associated direction than expected.

To resolve the above issues, we find the lane direction with only those vehicle objects having tracking traces greater than 20 pixels.

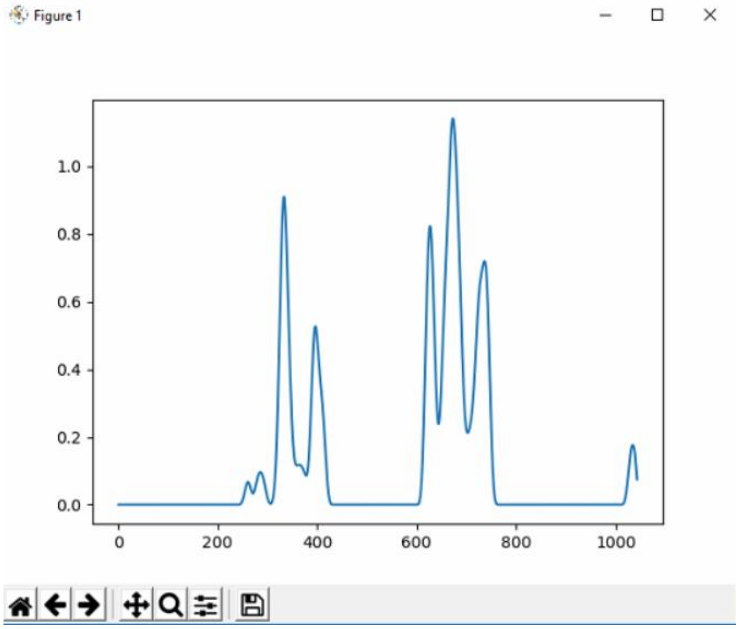
The structure of cross_baseline_list is as follows:

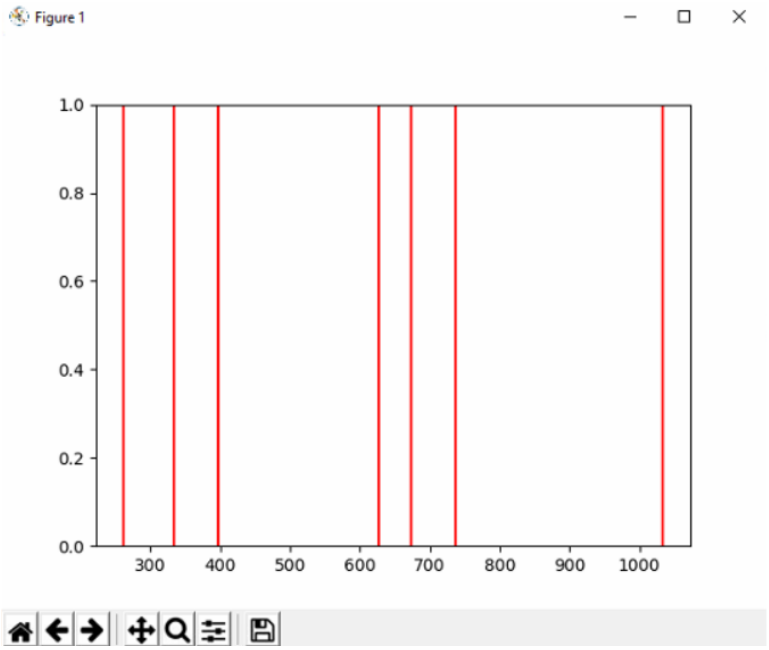
Cross_Baseline_list:

[(VehicleID, X-axisPointOfIntersectionWithBaseline, VehicleBoundingBoxWidth, VehicleDirection, TrustedDirectionFlag), (....), (....),.....]

For Trusted Vehicle Object ⇒ TrustedDirectionFlag ⇒ 1
For Untrusted Vehicle Object ⇒ TrustedDirectionFlag ⇒ 0

Result:





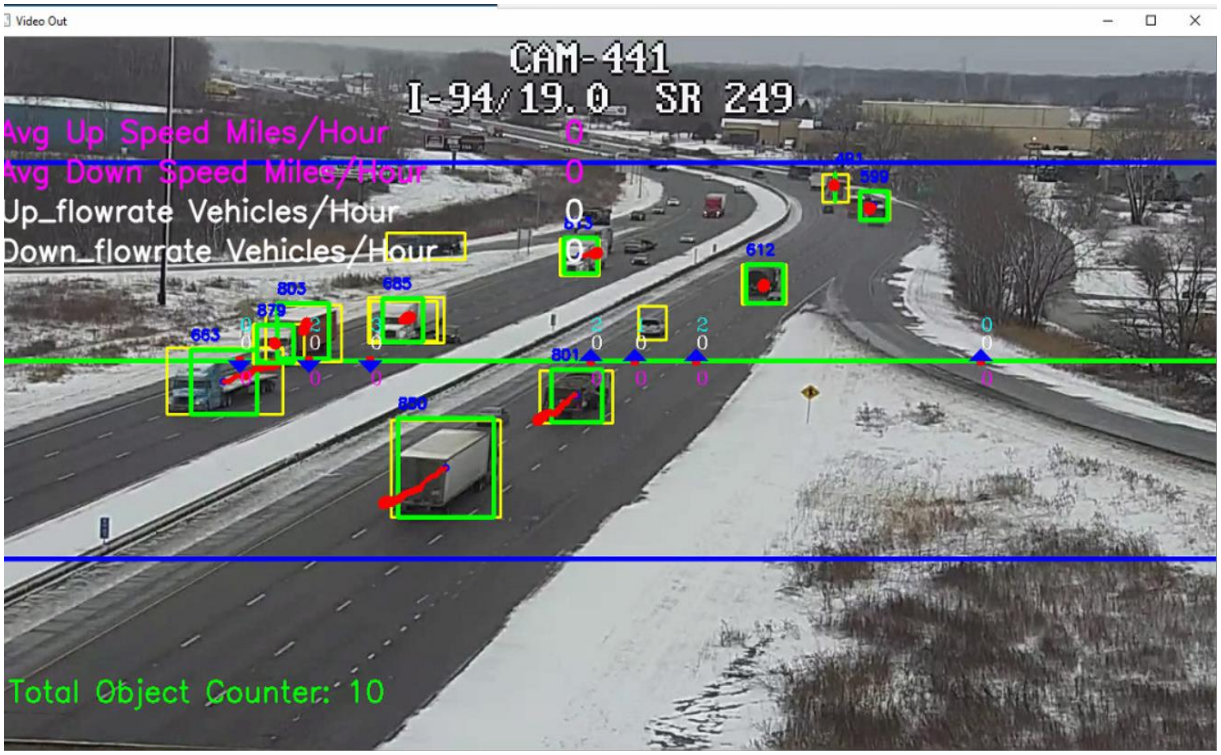


Fig 13. Result for correct lane direction

3.6 Discussion and limitations of proposed lane detection technique

This lane detection and lane direction module depends on vehicle detection and vehicle tracking module. So, the above algorithm fails to detect the lanes in following scenarios:

- Poor vehicle detection and consequently no tracking information available.

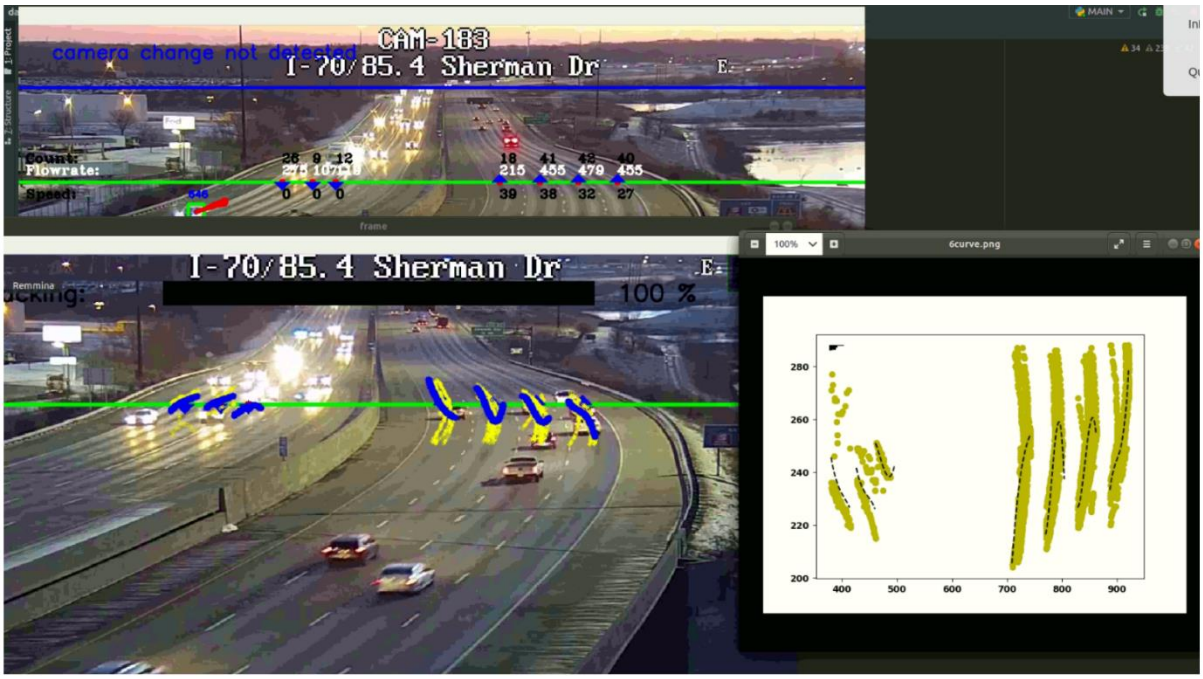


Fig 14. Insufficient Information for Lane Detection

Here in Figure 14; as we can see from vehicle detection and tracking Road curve fitting plot; there is a clear vehicle detection and tracking on right side of divider, so we are getting distinct 4 lanes which is matching the expected number lanes.

But on the left side we can see that cars are not getting detected in the leftmost lane. So, there is no tracking information available. And as a result, we can see the detected lanes are 3 but expected are 4 lanes. So, in such cases we can try to improve the accuracy of vehicle detection and tracking information.

Also, if we detect very minute level of tracking from these non-identified lanes; we can increase the number of vehicles to be considered i.e., 200 or 250 or 300 instead of 100; so there will be more tracking information of detected vehicles, which may improve the detection of previously unidentified lane center.

Other issues might be with the camera equipment:

- Camera angle is out of focus (Camera not facing the road): Change the camera focus / angle facing towards the road
- Camera with blocked focus due to rainy or snowy conditions

4. Idea of Incident Detection Module

The incident detection module identifies the road traffic congestion status as “Normal Speed”, “Slow Speed” or “Congestion”. Once we have the inputs from lane finding, and flow rate module, we apply the thresholding on the values of flow rate (projected number of vehicles passing from the lane per hour) to identify congestion incident statuses. These statuses are updated periodically and stored for historical analysis.

Incident detection module can be explained using following state Machine Diagram:
event0: Avg flowrate increase (within threshold flowrate > A & flowrate < B)
event1: Avg flowrate increase (Dense vehicles: flowrate > C)
event2: Avg flowrate decreases (Dense to moderate vehicles flowrate > A & flowrate < B)
event3: Avg flowrate Decreases (Moderate to spare vehicles flowrate < D)
event4: Sudden decrease in Avg flowrate (Huge Jumps)
event5: Sudden increase in Avg flowrate (Huge Jumps)

A, B, C and D: Average flowrate thresholds in vehicles/hr (Average flowrate for each side of road)

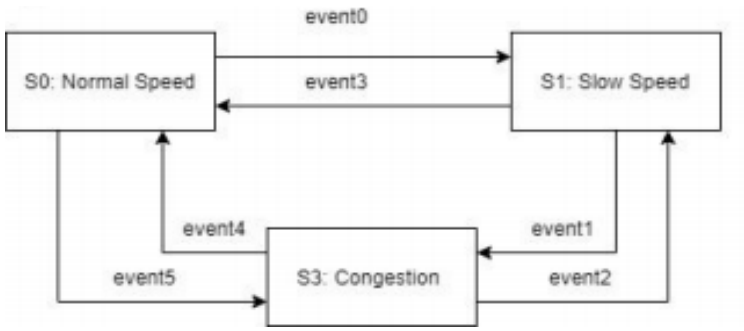


Fig 15. State machine diagram for Incident Detection

The flowrate module provides the following inputs for the Incident detection module.

<Flow Rate: number of vehicles passing through a lane per hour>
- Number of lanes on the left side of the divider
- Number of lanes on the right side of the divider
- Total flow rate of the lanes on the left of the divider
- Total flow rate of the lanes on the right of the divider

The incident detection module then computes the average flow rates of vehicles passing through all lanes on the left of the divider and on the right of the divider, respectively.

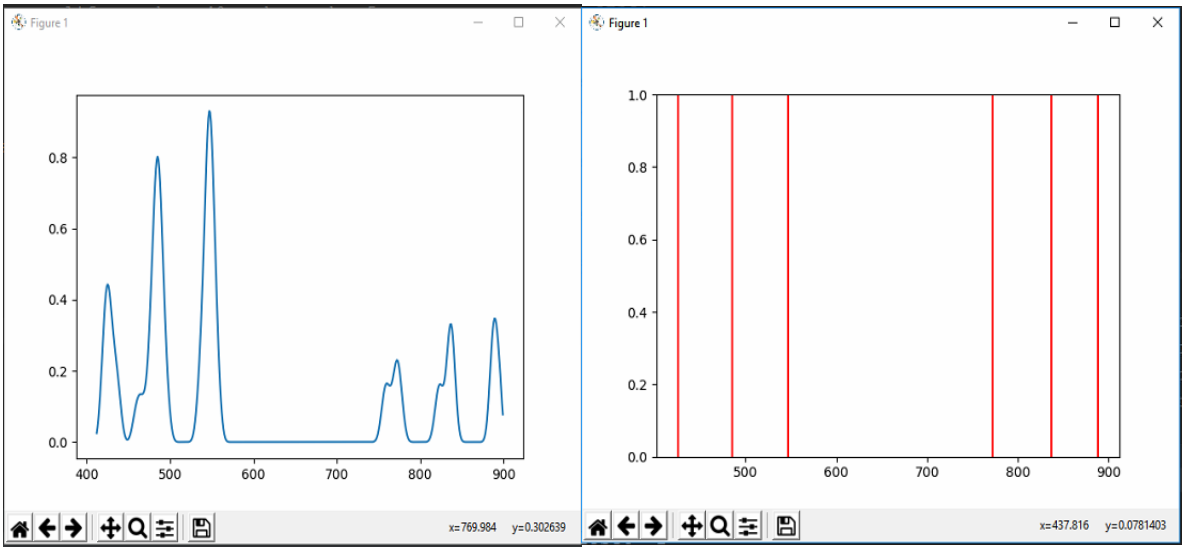
From the observation following are the thresholds set for incident statues:

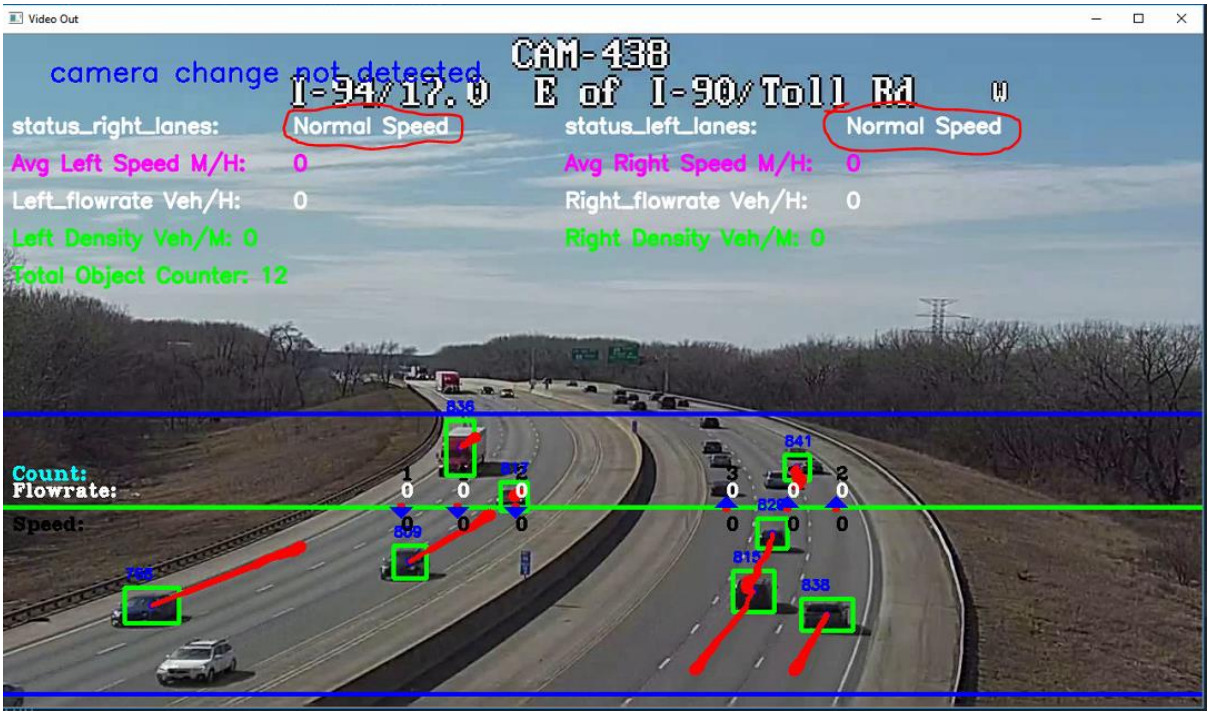
if 0< avgFlowRate < 2000
    “Normal Speed”
else if 2000<= avgFlowRate < 2500
    “Slow Speed”
else
    “Congestion”

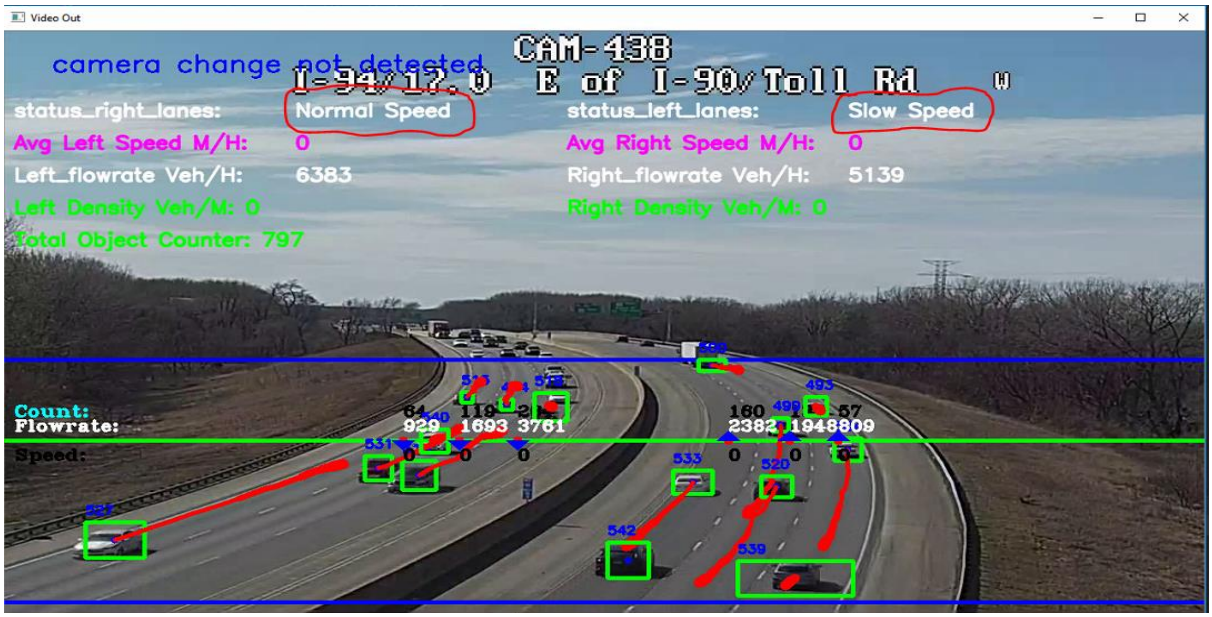
Although we have the average flowrates to decide the statues of congestion; it is not a reliable measure. Because the flowrates depend on the number of vehicles passing through the lanes for one hour. So, as number of cars increases the flowrate increases, and the too many cars with jam, the flow rate decreases. Therefore, along with flowrate we need another metric to tune the filtering conditions of congestion.

In real world on road scenarios the speed limits play a measure road to align and monitor the traffic. So, from the video we can predict the speed associated with the lane centers; that means the cumulative speed of vehicles passing from the lane centers. We can use the average vehicle speed of lane centers on left side and right side to threshold the congestion statues.
The speed calculation module is still under development. It is discussed in detail in next section.

4.1. Results of incident detection: using average flowrate of left and right-side lanes of road divider:




Video Out
camera change not detected
CAM-438
I-94/17.0 E of I-90/Toll Rd W
status_right_lanes: Normal Speed
status_left_lanes: Normal Speed
Avg Left Speed M/H: 0
Avg Right Speed M/H: 0
Left_flowrate Veh/H: 0
Right_flowrate Veh/H: 0
Left Density Veh/M: 0
Right Density Veh/M: 0
Total Object Counter: 12
Count:
Flowrate:
Speed:


Video Out
camera change not detected
CAM-438
I-94/17.0 E of I-90/Toll Rd W
status_right_lanes: Normal Speed
status_left_lanes: Slow Speed
Avg Left Speed M/H: 0
Avg Right Speed M/H: 0
Left_flowrate Veh/H: 6383
Right_flowrate Veh/H: 5139
Left Density Veh/M: 0
Right Density Veh/M: 0
Total Object Counter: 797
Count:
Flowrate:
Speed:

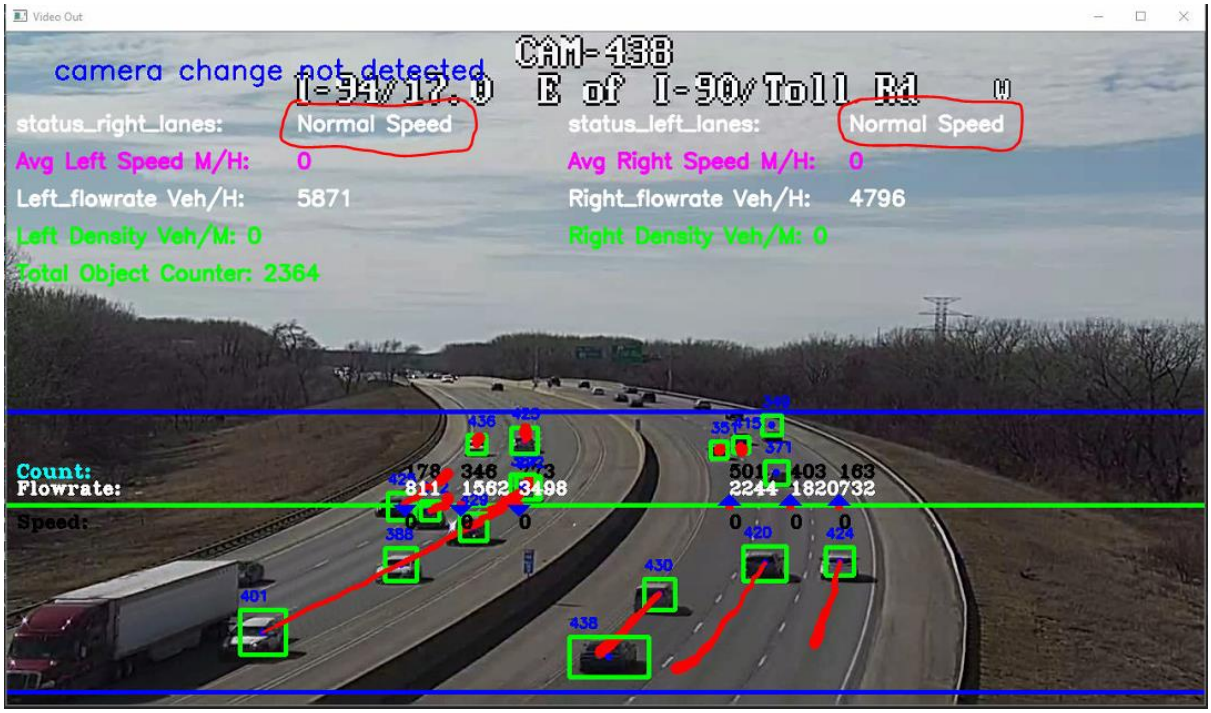


Fig 16. Congestion detection results

Fig 16. Shows the status of congestion for the left and right side of lanes, respectively.

This module is still under development; so, a closed observation on the thresholds of average flow rates and its integration of average speed will result in robustness of the module. The calculation of average speed depends on the module 5 where we compute the meter per pixel information. The meter per pixel module gives the information of number of pixels representing one-meter distance from which vehicles speed can be computed. Module 5 discuss it in detail.

4.2. Database Schema for storing the incident results

DB: congestion_status_current: Stores the current congestion status for a single camera

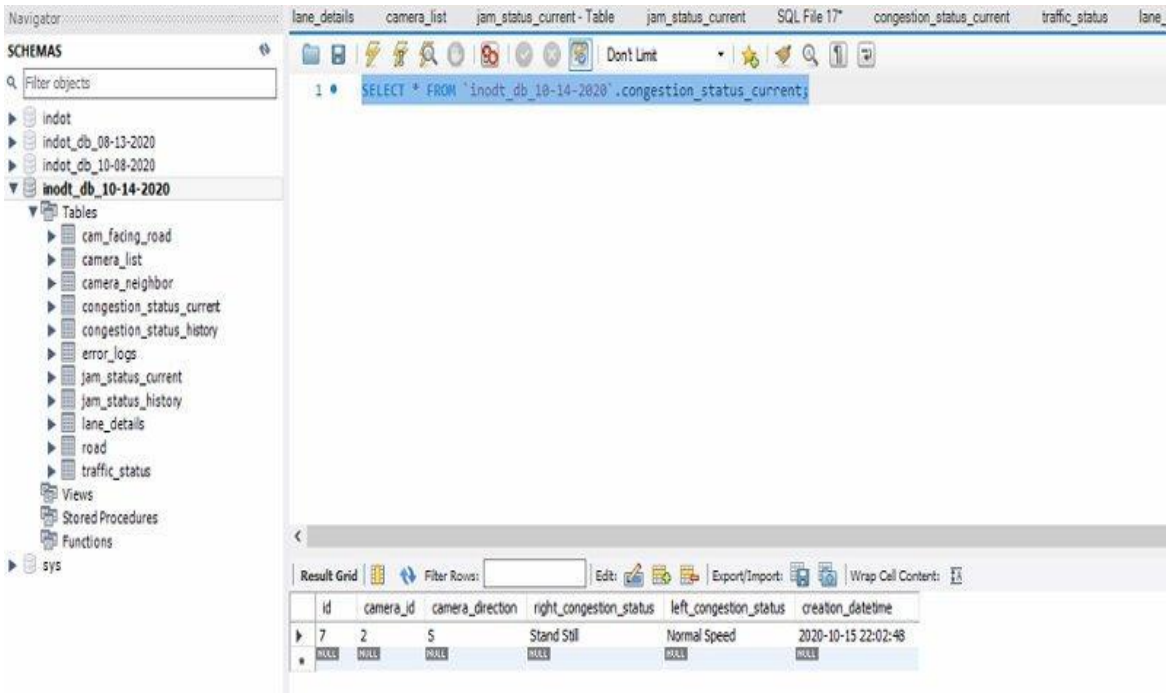


Fig 17. DB table to store current congestion status

DB: congestion_status_history: Stores the historical congestion statues for a single camera

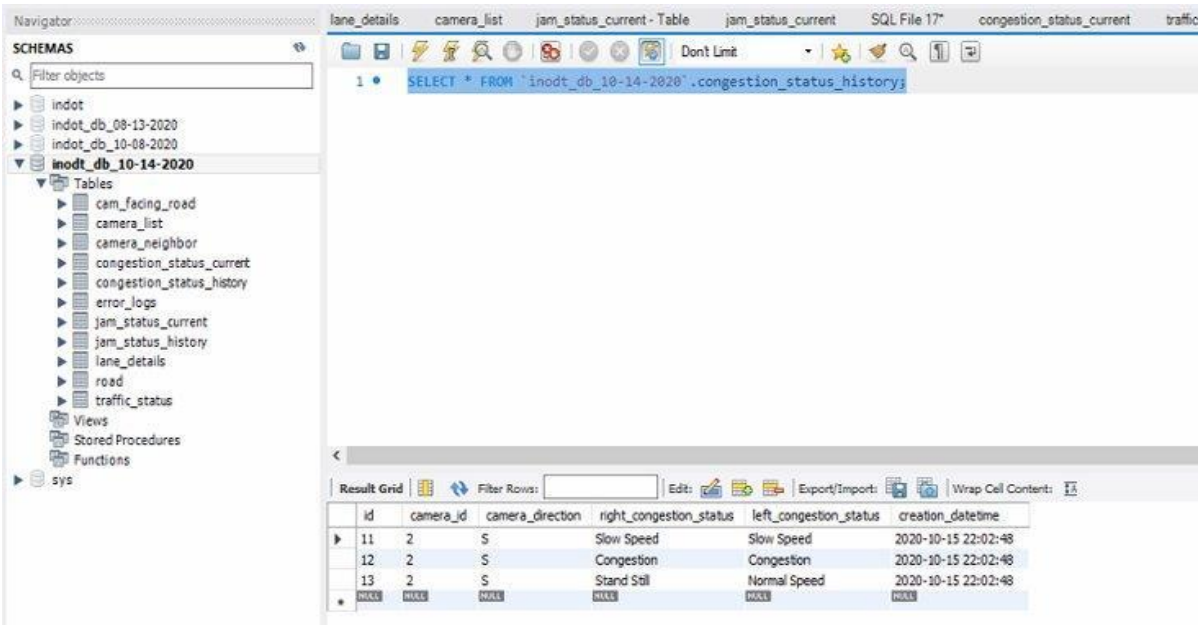


Fig 18. DB table to store historical congestion status

5. Idea of Meter Per Pixel Module: Pixels for white lane marking

Generally, the speed information of vehicles can be computed using radar systems, where we have all the information about angle of camera and the height of camera from the ground surface.

But in our case, we do not have this information. Since the standard length of the white lane markings painted on the road is 10 feet and they are repeated every 40 feet, we try to use this info to compute the number of pixels representing per meter at the baseline.

Once we have the information of number of meters per pixel we can use it to compute the speed in terms of pixels per second, and then miles/hr.

Input: input_frame, baseline location (in y-direction), lane_centers coordinates (in x-direction), lane_directions.

Output: Number of Pixels the white lane marking spans on the left of the divider
Number of Pixels the white lane marking spans on the right of the divider

5.1 Algorithm for white lane marking detection

1. As per lane_directions find the lanes on the left and right of the divider respectively
2. From the adjacent lane_center coordinates on the baseline, compute the approximate location of lane marking on the video frame.

Fig 19. Step 2 of white lane marking detection algorithm

3. Since we need to observe the white segments from the image. Masking is a technique to emphasis the white segments from input image rest of the colours get transformed to black. So, we have the coloured image into black and white image for clear distinctions of white (255,255,255) and black (0,0,0) pixels (as the lane markings are white their pixel values are (255,255,255)).

4. For each of the above approximated lane marking on the baseline, find the closest start of the white lane marking pixel using following non-maximum suppression technique

Pixel (i-1, j-1)	Pixel (i-1, j)	Pixel (i-1, j+1)
Pixel (i, j-1)	**Pixel (i, j)**	Pixel (i, j+1)
Pixel (i+1, j-1)	Pixel (i+1, j)	Pixel (i+1, j+1)

Around the pixel (i, j) find the white pixel in one-pixel distance in 8 directions, up, up-left, upright, left, and right, bottom, bottom left and bottom right. If white pixel is not found, then increase the level to 2 and so on. Continue this process for distance equal to 20.

5. Once the white pixel is found, follow the consecutive white pixels along the height of frame/baseline around that white pixel to find the exact number of white pixels the lane marking spans.

6. For each of white lane marking segments on the left and right side of the road divider we will get the number of pixels representing the white lane marking segment at the baseline. These number of pixels should be approximately equal in Y-direction.

7. Return the detected number of Pixels for corresponding white lane marking for the lanes on left and right side of the divider to flow rate module to predict the speed of the vehicle.

5.2. Results of Meter Per Pixel Module: Pixels for white lane marking

Example 1. Morning Sunny day

The frame under consideration for meter per pixel computation is as follows:

Fig 20a. Meter_Per_Pixel Result Image – 1

As per the step 3 of above algorithm in section 5.1 the mask of image is as follows: We can see here the white segments from the image are distinctly visible with rest of the colours turned black. (Figure 20b).

Fig 20b. Meter_Per_Pixel Result Image - 1 White Mask

Detected pixels for white lane markings: As we can see the white lane marking pixels are detected on left and right side of the road

Fig 20c. Meter_Per_Pixel Result Image - 1 white lane marking pixels [5,4,9,2 left to right]

##
Baseline Complete 491
Lane Centers are: [119, 241, 353, 756, 884, 1002]
Lane Center Directions: [1, 1, 1, -1, -1, -1]
left_lane_markings = [180,297]
right_lane_markings = [820,943]
lanePoint = 180
Number of Pixels on left for lane marking 5
lanePoint = 297
Number of Pixels on left for lane marking 4
lanePoint = 820
Number of Pixels on right for lane marking 9
lanePoint = 943
Number of Pixels on right for lane marking 2

Example 2. Cloudy day

The frame under consideration for meter per pixel computation is as follows:

Fig 21a. Meter_Per_Pixel Result Image - 2

Baseline computed at Y-coordinate: 418

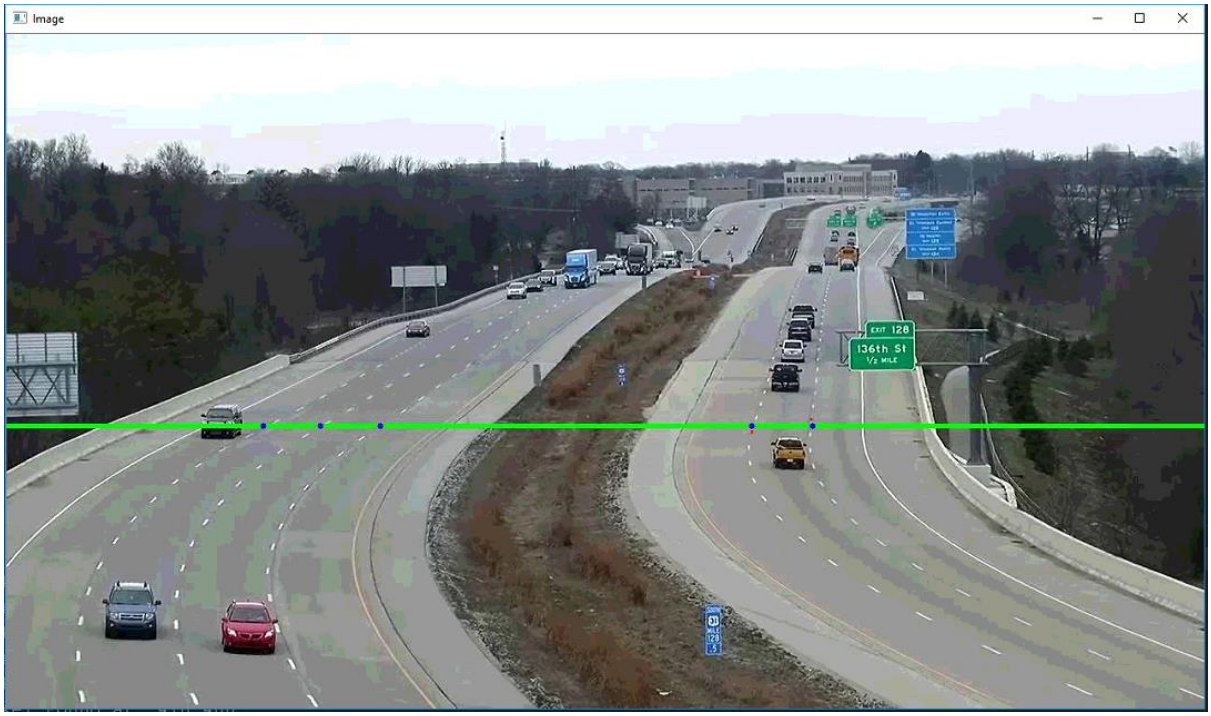


Fig 21b. Meter_Per_Pixel Result Image - 2 Step 2 of white lane marking detection algorithm

As per the step 3 of above algorithm in section 5.1 the mask of image is as follows: Detected pixels for white lane markings: We can see here the white segments from the image are distinctly visible with rest of the colours turned black. (Figure 21c.)

Fig 21c. Meter_Per_Pixel Result Image - 2 White Mask

Detected pixels for white lane markings: As we can see the white lane marking pixels are detected on left and right side of the road.

Fig 21d. Meter_Per_Pixel Result Image - 2 white lane marking pixels [2,3,4,5,4 left to right]

Baseline Complete 418
lane_centers = [240, 309, 361, 437, 760, 832, 891]
lane_directions = [1, 1, 1, 1, -1, -1, -1]
left_lane_markings = [274,335,399]
right_lane_markings = [796,861]
lanePoint = 274
Number of Pixels on left for lane marking 2
lanePoint = 335
Number of Pixels on left for lane marking 3
lanePoint = 399
Number of Pixels on left for lane marking 4
lanePoint = 796
Number of Pixels on right for lane marking 5
lanePoint = 861
Number of Pixels on right for lane marking 4

5.3 Problem with Generation for White Mask

We know that colours can be denoted within the range of 0(Black) – 255(White)
So, the cv2.inRange() takes 3 Parameters

1. Image
2. Lower Bound of Colour Saturation
3. Higher Bound of Colour Saturation

mask_white = cv2.inRange(gray_image, 170, 255) ==> gives the white mask for the colours within the range 170 to 255
Video 5 with cv2.inRange(gray_image, 170, 255)

Fig 22a. Poor Mask when range of colours is 170 - 255

Video 5 with cv2.inRange(gray_image, 125, 255)

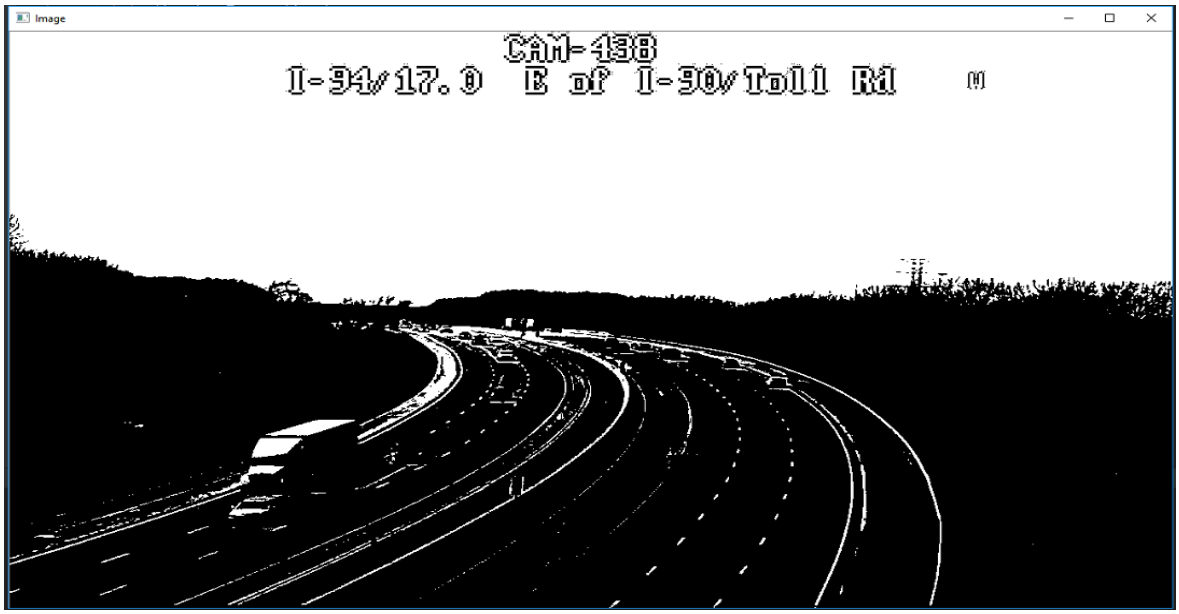


Fig 22b. Frame Mask when range of colours is 125 – 255

But for other videos 125 to 255 range does not work well:

Video 1 with cv2.inRange(gray_image, 125, 255)

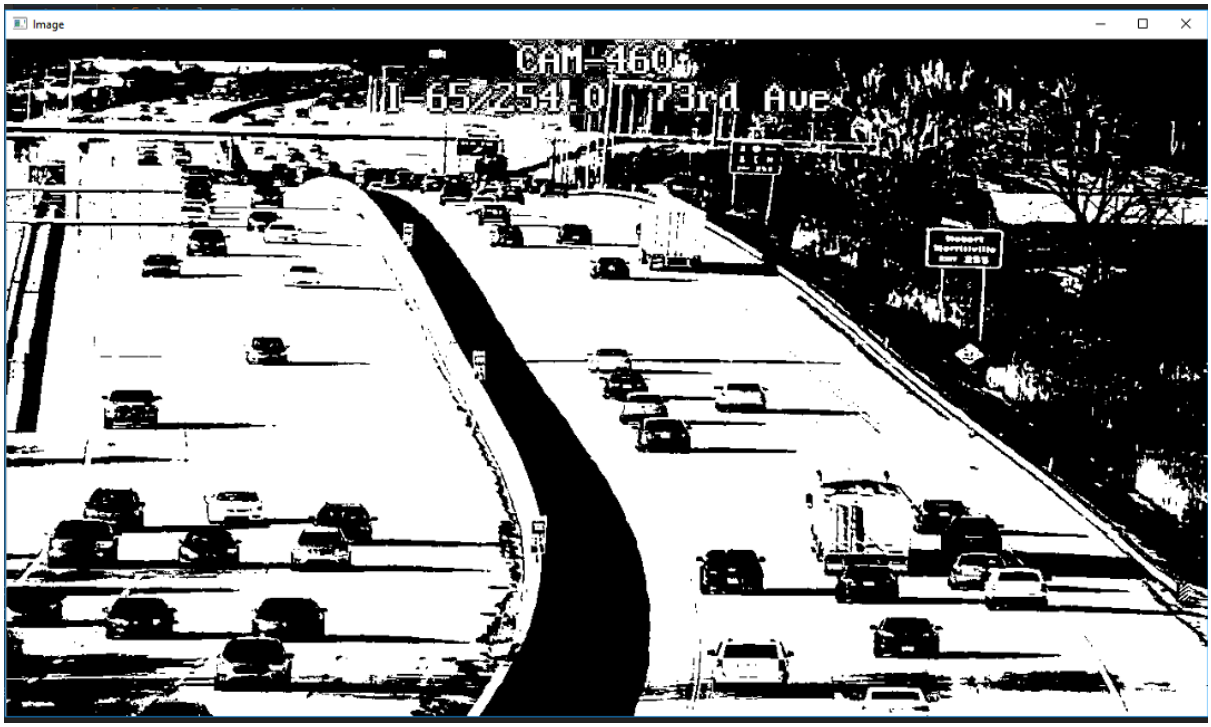


Fig 23a. Frame Mask (Over whitened) when range of colours is 125 - 255

Video 1 with cv2.inRange(gray_image, 170, 255)

Fig 23b. Frame Mask (As Required) when range of colours is 170 - 255

Figure 22a, 22b and 23a, 23b shows that we cannot predefine the saturation values in inRange function to generate the white mask.

5.4 Adaptive generation of white mask: for clear distinction of white and black pixels on the frame

1. It is clear from the above problem description that we cannot set one range of values for cv2.inRange(image, start_color_range, end_color_range)

2. Also, we need not consider the entire image for the mask generation. To narrow down the analysis we can consider the cropped image within the region of interest i.e., 100 pixels above and 200 pixels below the baseline.

3. Here, for the range of values we generate the white mask. So, for an image the pixels belonging to that range of values becomes white and rest becomes black. That is how the mask is generated.

4. Once a mask is generated, we can take the mean of all the pixel values for the mask, i.e., the sum of all the (white + black) pixels divided by total number of pixels.

5. For the adaptive generation of mask, we can use following approach.
   I. Start with a default range of values (100-255)
   II. Iteratively increase the range by a constant factor i.e. (110-255); (120-255); …., (210-255) and generate the corresponding white mask.
   III. For each mask generate the mean of the white mask image
   IV. Plot all the mean values associated to different ranges
   V. Observe the pattern if ANY

Results:

Original Image: Image under consideration

Fig 24a: Original Image

Cropped Image: We are interested in the movement of vehicles within the Region of Interest around the baseline. There we can crop the image at baseline (100 Pixels Up and 200 Pixels Down: Region of Interest) for further processing.

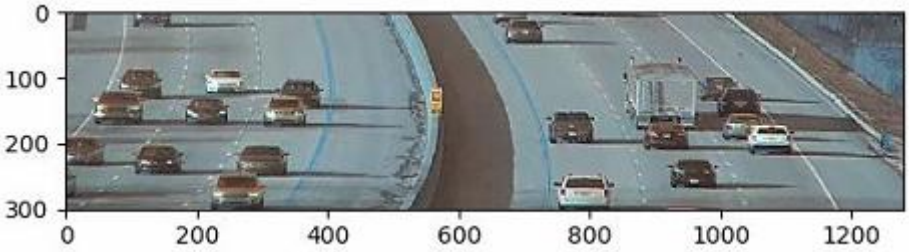


Fig 24b: Cropped Image

cv2.inRange(image,100,255)

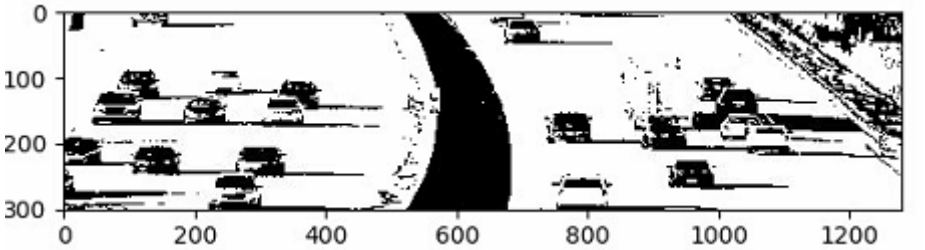


Fig 24c: inRange (100,255)

cv2.inRange(image,110,255)

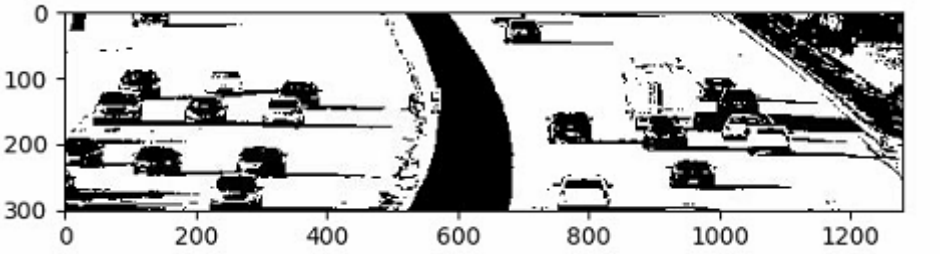


Fig 24d: inRange (110,255)

cv2.inRange(image,120,255)

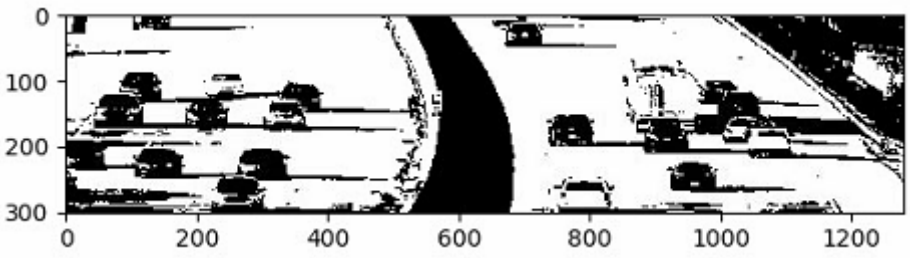


Fig 24e: inRange (120,255)

cv2.inRange(image,130,255)

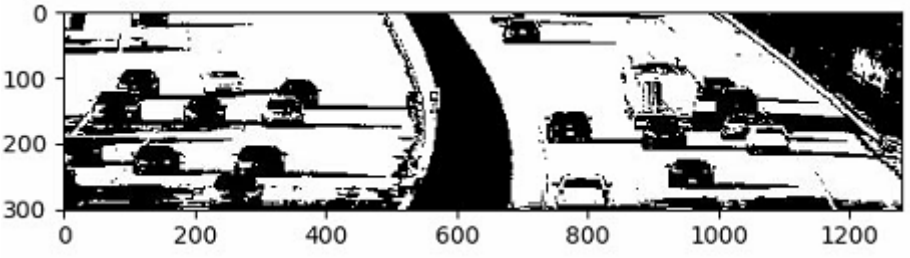


Fig 24f: inRange (130,255)

cv2.inRange(image,140,255)

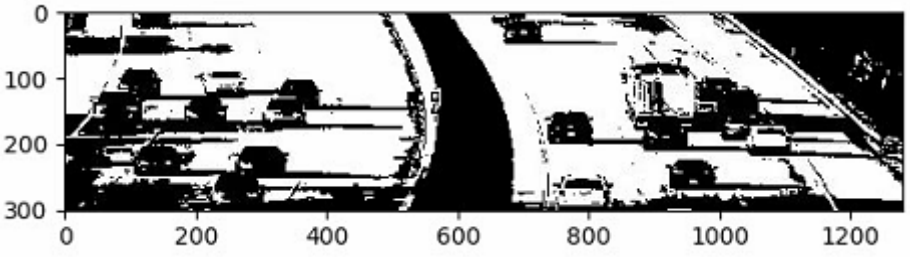


Fig 24g: inRange (140,255)

cv2.inRange(image,150,255)

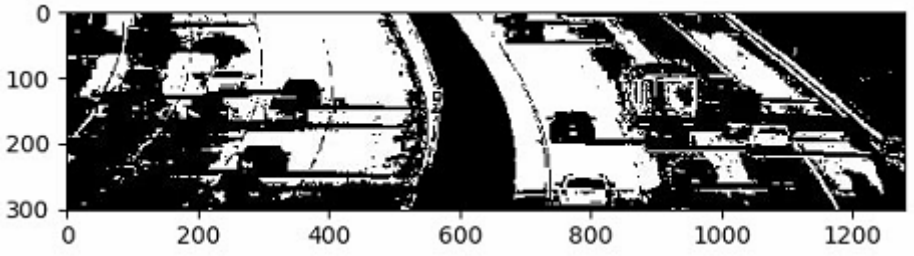


Fig 24h: inRange (150,255)

cv2.inRange(image,160,255)

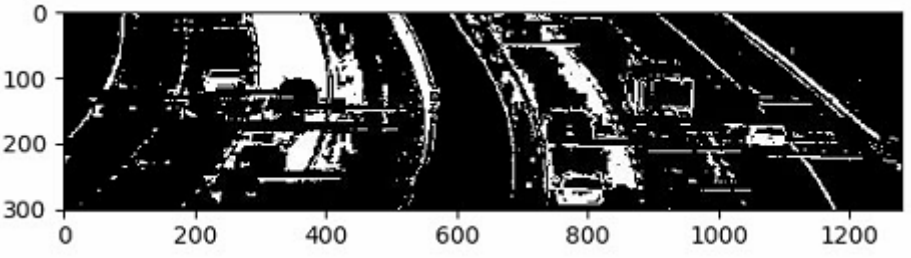


Fig 24i: inRange (160,255)

cv2.inRange(image,170,255)

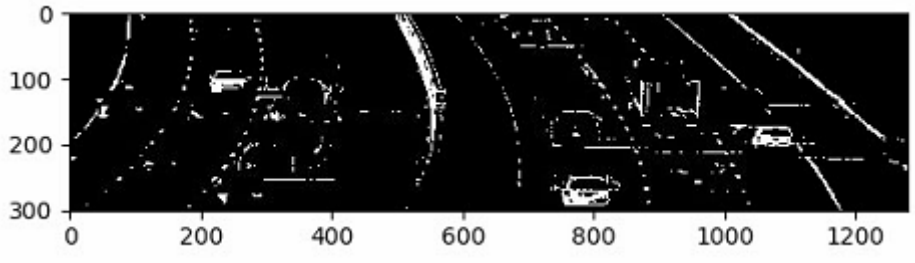


Fig 24j: inRange (170,255)

cv2.inRange(image,180,255)

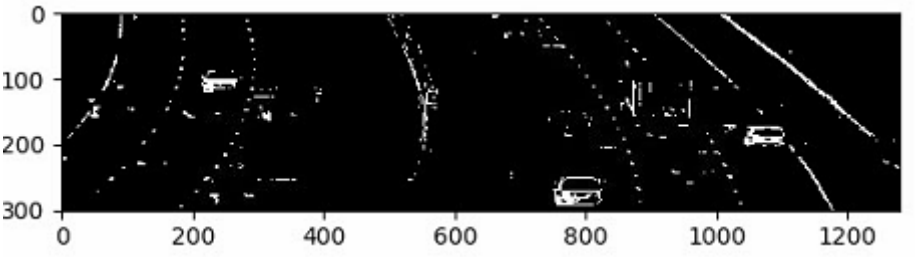


Fig 24k: inRange (180,255)

cv2.inRange(image,190,255)

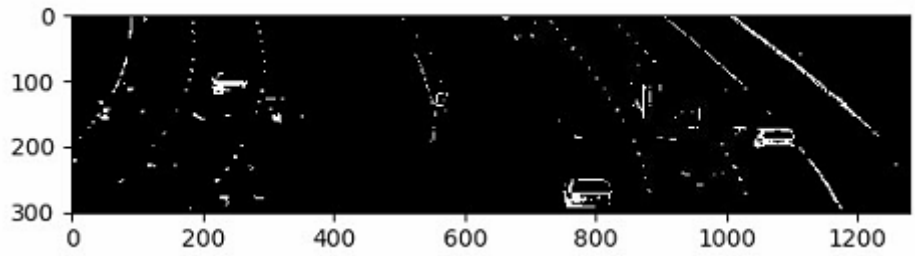


Fig 24l: inRange (190,255)

cv2.inRange(image,200,255)

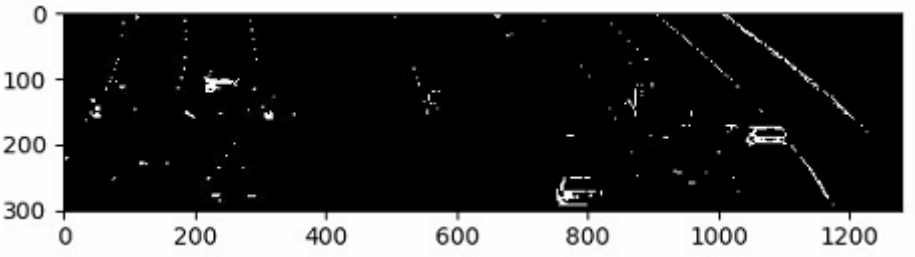


Fig 24m: inRange (200,255)

cv2.inRange(image,210,255)

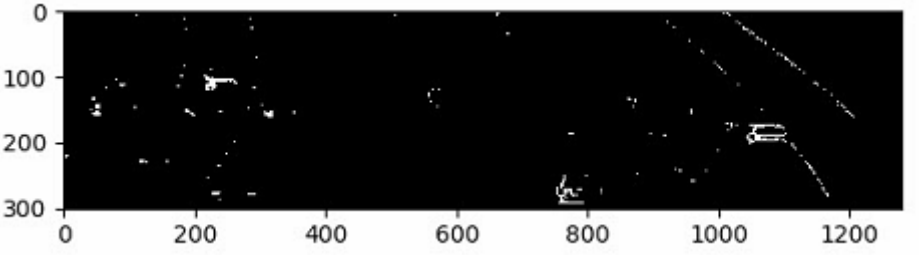


Fig 24n: inRange (210,255)

Figure 24a – 24n: White Mask with different range of colour saturation combinations

Plotting Mean of Pixels for all the above ranges:

For above each white mask combination the histogram in slot is a mean of all the pixels in that cropped white mask.

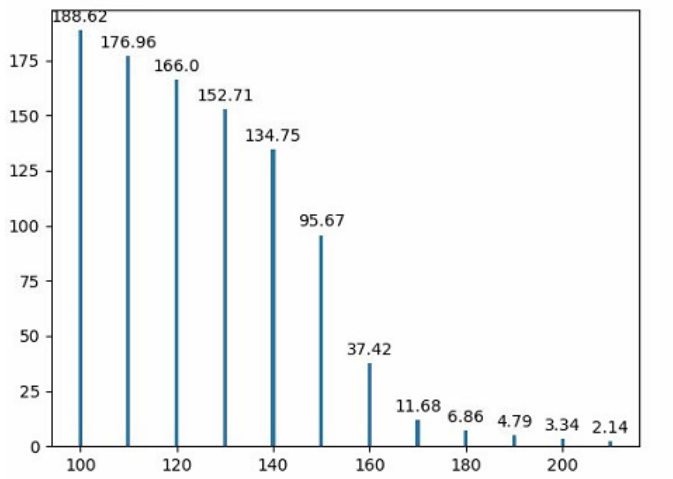


Fig 24o: Mean plot for all the above ranges

Observed Pattern:

As the mean drops less than 30 the distinct white lane markings are visible:

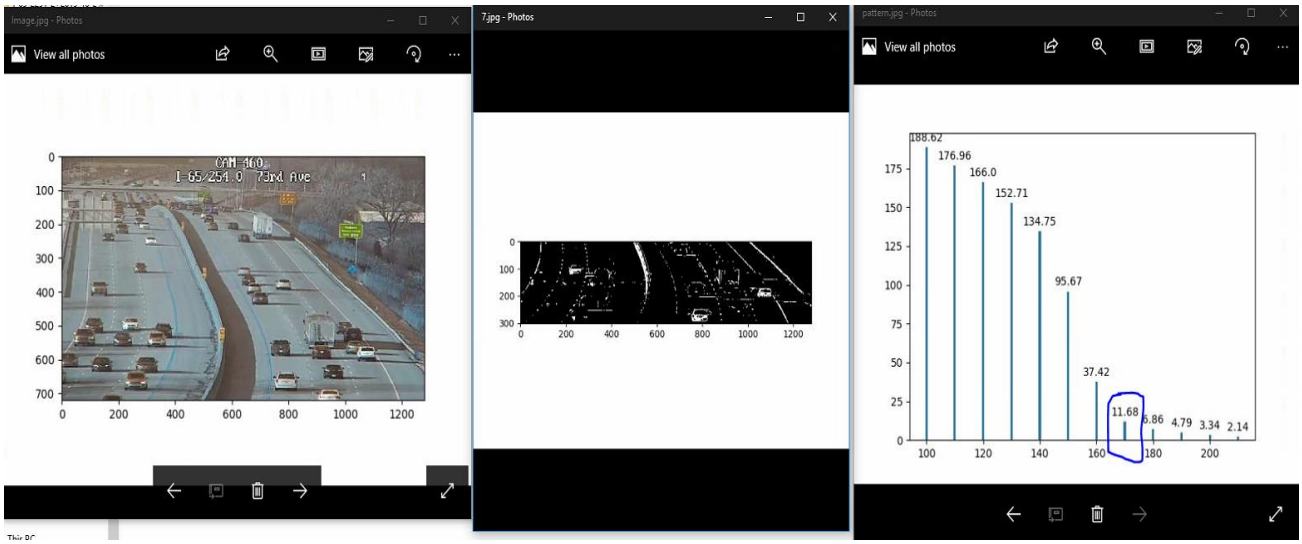


Fig 25: When Mean of Pixels in the cropped image less than 30 we get required white mask

When Mean of Pixels in the cropped image less than 30 we get required white mask i.e., we get the distinct white segments on the cropped image to be used for white lane segment pixels detection.

This is how we can set the white mask adaptively. Once we get the required white mask, we can implement the meter per pixel module to calculate the number of pixels spanned by white lane marking segment.

6. Meters Per Pixel Module: Pixels between consecutive white lane markings

-The meters per pixel module results the number of pixels the white lane marking segment spans on both side of the divider, respectively.

-From the results of meter per pixel module we can observe that the values of number of pixels the white lane segment spans on both sides of the divider is in single digits (<=9).

-For the calculation of speed module, we need these number of pixel values to be significantly large enough and reliable.

-Also, sometimes due to poor white mask these white lane segments might not be continuous resulting into inaccurate results.

-To avoid this, we need larger values to represent by pixels. As we know the in United States the distance between the two white segments is constant.

-So, for the meter per pixel module, we need to calculate the number of pixels spanned over the length between two white segments.

6.1 Results: Pixels between consecutive white lane makings

The frame under consideration for meter per pixel computation is as follows:

Fig 26a. Original frame for Meter Per Pixel Approach 2

Baseline computed at Y-coordinate: 418

Fig 26b: Baseline drawn on the original image

As per the step 3 of above algorithm in section 5.1 the mask of image is as follows: Distinct white segments

Fig 26c: White Mask set using the automatic white mask generation method

Detected pixels: from end of a white lane segment to start of next white segment:

Fig 26d: Detected Pixels between the consecutive white lane markings

In Figure 26d, from left to right the pixels are:

Number of Pixels Detected on Left:
Number of Pixels between these points 20
Number of Pixels between these points 17
Number of Pixels between these points 17

Number of Pixels Detected on Right:
Number of Pixels between these points 10
Number of Pixels between these points 11

7. Summary and learnings from the project

Through this project I inculcated real-time problem-solving skills. It helped me to improve the robustness of the lane detection module. The day-to-day scenario where the vehicles remain in the lane during most of the travel time except for changing the lanes for overtake; reflected in the histogram generation technique and I could solve them for predicting the probable lane centers through lane filtering algorithm.
The lane detection module requires details of vehicle objects, the meter per pixel module requires the generated lane center coordinates and the congestion detection module requires the flow rate and speed results, this modularised interdependent approach taught me to draft the requirements and provide module results into designated format for further processing. The developed lane filtering algorithm, searching technique for white lane marking pixels and threshold definition for congestion module gave me an opportunity to apply the learnt course work knowledge from different perspectives.

8. Acknowledgement

I would like to thank Dr. Stanley Chien, Dr. Snehasis Mukhopadhyay and Dr. Jiang Zheng for providing me this data drive software implementation opportunity.

Appendix A. Detailed data output from for the Lane finding algorithm

Example 1. Output of the program execution of the Lane_Finding algorithm:

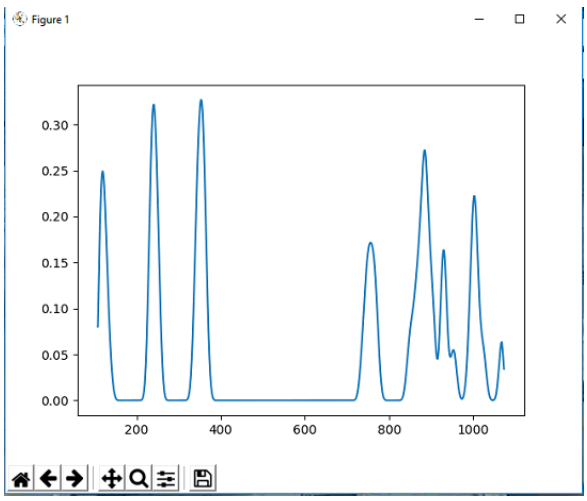


Fig A

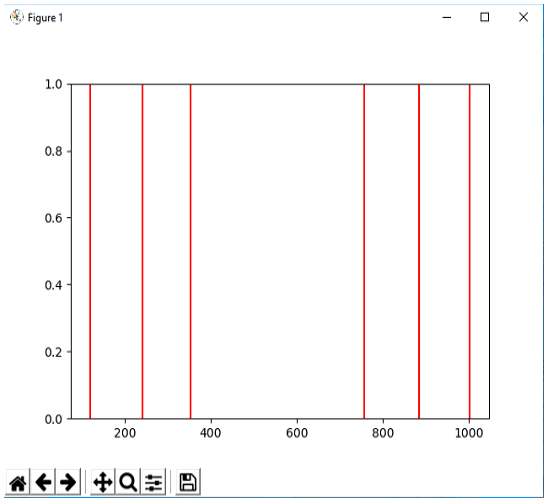


Fig B

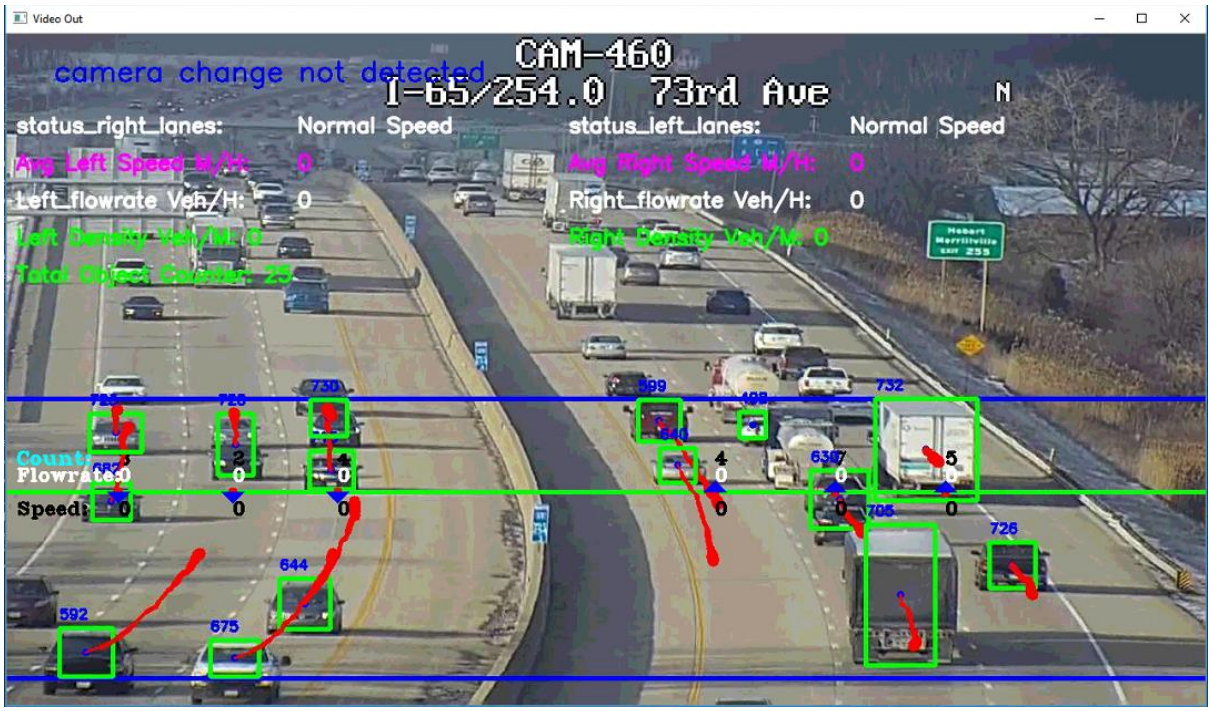


Fig C

Module Inputs: Input_Frame, baseline, sampling_rate, cross_baseline_list

Input_Frame: Video frame under consideration
Baseline: Y-coordinate on Input_Frame where probability of observing the vehicles is the most
Sampling_rate: Number of Vehicles under consideration
cross_baseline_list: Information of vehicles under consideration when they cross the baseline

Module Outputs: Lane_Centers and Lane_Directions

Follow the algorithm steps.

Content of Cross baseline list is as follows: It is the information of vehicles under consideration (100) when they cross the baseline

Cross Baseline List [**(3, 107, 37, 1, 0)**, (5, 741, 56, 1, 0), (10, 256, 67, 1, 0), (12, 889, 61, 1, 0), (20, 757, 54, -1, 1), (13, 860, 44, 1, 0), (9, 900, 45, -1, 1), (11, 340, 44, 1, 1), (7, 759, 62, 1, 0), (6, 753, 53, -1, 1), (34, 119, 53, 1, 1), (8, 884, 36, -1, 1), (22, 237, 43, 1, 1), (56, 359, 58, 1, 1), (26, 357, 35, 1, 0), (35, 248, 37, 1, 1), (66, 109, 38, 1, 1), (40, 110, 40, 1, 1), (57, 759, 42, -1, 1), (79, 247, 37, 1, 1), (52, 1020, 51, -1, 1), (89, 123, 51, 1, 1), (60, 903, 60, -1, 1), (96, 348, 46, 1, 1), (74, 1004, 42, -1, 1), (81, 904, 45, -1, 1), (118, 1070, 17, 1, 0), (102, 930, 3, -1, 1), (146, 1062, 95, 1, 0), (148, 919, 132, 1, 0), (135, 934, 70, 1, 0), (122, 1023, 41, -1, 1), (166, 917, 80, 1, 0), (151, 749, 47, -1, 1), (182, 239, 37, 1, 1), (209, 883, 40, 1, 0), (218, 871, 68, 1, 0), (188, 347, 49, 1, 1), (219, 741, 60, -1, 1), (215, 122, 36, 1, 1), (185, 739, 36, 1, 0), (177, 887, 46, -1, 1), (201, 891, 35, -1, 1), (293, 133, 42, 1, 0), (236, 764, 43, -1, 1), (320, 230, 106, 1, 0), (330, 871, 38, -1, 1), (345, 216, 110, 1, 0), (266, 344, 44, 1, 1), (244, 929, 6, -1, 1), (340, 357, 45, 1, 1), (276, 770, 47, -1, 1), (338, 251, 35, 1, 1), (363, 897, 101, 1, 0), (346, 122, 43, 1, 1), (348, 930, 2, 1, 0), (395, 879, 54, 1, 0), (362, 348, 37, 1, 1), (357, 115, 35, 1, 1), (387, 953, 33, 1, 0), (364, 241, 34, 1, 1), (398, 761, 60, -1, 1), (339, 1008, 47, -1, 1), (408, 911, 111, 1, 0), (409, 754, 77, 1, 0), (385, 357, 55, 1, 1), (365, 745, 53, 1, 0), (420, 234, 58, 1, 1), (394, 233, 47, 1, 0), (354, 1023, 51, -1, 1), (433, 364, 38, 1, 0), (372, 882, 43, -1, 1), (418, 354, 47, 1, 1), (448, 224, 72, 1, 0), (376, 998, 55, -1, 1), (468, 357, 71, 1, 0), (396, 884, 66, -1, 1), (520, 872, 58, 1, 0), (437, 360, 47, 1, 1), (538, 849, 47, 1, 0), (465, 119, 50, 1, 1), (426, 993, 46, -1, 1), (517, 349, 48, 1, 0), (512, 235, 39, 1, 1), (549, 1014, 59, -1, 1), (618, 999, 42, 1, 0), (547, 351, 49, 1, 1), (502, 751, 37, -1, 1), (438, 1014, 54, -1, 1), (584, 122, 38, 1, 1), (592, 238, 52, 1, 1), (595, 1004, 43, 1, 0), (552, 760, 48, -1, 1), (675, 368, 48, 1, 0), (644, 366, 61, 1, 0), (622, 998, 53, 1, 0), (578, 1074, 3, -1, 1), (627, 120, 53, 1, 1), (694, 917, 77, -1, 1), (697, 870, 36, -1, 1)]

Length of Cross Baseline List is100

Here,
the first element is **(3, 107, 37, 1, 0)**

VehicleID 3 crossed baseline at pixel 107.
The vehicle 3's bounding box width is 37 Pixels, and the moving direction is away from camera, and the TrustedFlag is set 0 i.e., it is not a trustable vehicle object for further computations.

You can observe the last element of tuple object in the list is either 0 or 1 that indicates whether this vehicle object can be trusted for further computation or not.

It is decided on if the vehicle object has a tracking trace greater than 20 Pixels, then it is a trusted vehicle object.

From Cross Baseline List we compute the Trusted Cross Baseline List

The Trusted Cross Baseline List is a list of vehicle objects with trusted direction.
Each vehicle object is in the tuple object format with 5 attributes as follows:

(VehicleID, x-axis intersection coordinate with baseline, bounding box width, direction (moving closer / away from camera), trustedFlag).

TrustedFlag is set to one for trusted vehicle object.

- **Compute Trusted_cross_baseline_list**
- **Compute Id_list, Intersection_list, Trusted_intersection_list, width_list, direction_list, Trusted_direction_list**
- **Calculate Median Width from the vehicle width_list**
- **Filter intersection_list and direction_list as corresponding vehicle_wdith <= median_width**

The content of Trusted_cross_baseline_list is:

Trusted Cross Baseline List [**(20, 757, 54, -1, 1)**, (9, 900, 45, -1, 1), (11, 340, 44, 1, 1), (6, 753, 53, -1, 1), (34, 119, 53, 1, 1), (8, 884, 36, -1, 1), (22, 237, 43, 1, 1), (56, 359, 58, 1, 1), (35, 248, 37, 1, 1), (66, 109, 38, 1, 1), (40, 110, 40, 1, 1), (57, 759, 42, -1, 1), (79, 247, 37, 1, 1), (52, 1020, 51, -1, 1), (89, 123, 51, 1, 1), (60, 903, 60, -1, 1), (96, 348, 46, 1, 1), (74, 1004, 42, -1, 1), (81, 904, 45, -1, 1), (102, 930, 3, -1, 1), (122, 1023, 41, -1, 1), (151, 749, 47, -1, 1), (182, 239, 37, 1, 1), (188, 347, 49, 1, 1), (219, 741, 60, -1, 1), (215, 122, 36, 1, 1), (177, 887, 46, -1, 1), (201, 891, 35, -1, 1), (236, 764, 43, -1, 1), (330, 871, 38, -1, 1), (266, 344, 44, 1, 1), (244, 929, 6, -1, 1), (340, 357, 45, 1, 1), (276, 770, 47, -1, 1), (338, 251, 35, 1, 1), (346, 122, 43, 1, 1), (362, 348, 37, 1, 1), (357, 115, 35, 1, 1), (364, 241, 34, 1, 1), (398, 761, 60, -1, 1), (339, 1008, 47, -1, 1), (385, 357, 55, 1, 1), (420, 234, 58, 1, 1), (354, 1023, 51, -1, 1), (372, 882, 43, -1, 1), (418, 354, 47, 1, 1), (376, 998, 55, -1, 1), (396, 884, 66, -1, 1), (437, 360, 47, 1, 1), (465, 119, 50, 1, 1), (426, 993, 46, -1, 1), (512, 235, 39, 1, 1), (549, 1014, 59, -1, 1), (547, 351, 49, 1, 1), (502, 751, 37, -1, 1), (438, 1014, 54, -1, 1), (584, 122, 38, 1, 1), (592, 238, 52, 1, 1), (552, 760, 48, -1, 1), (578, 1074, 3, -1, 1), (627, 120, 53, 1, 1), (694, 917, 77, -1, 1), (697, 870, 36, -1, 1)]
Length of Trusted Cross Baseline List is 63

The Trusted Cross Baseline List is a list of vehicle objects with trusted direction.

We identify the given vehicle object as a trusted vehicle object as per the consideration of tracking trace associated to it.

If the vehicle object has a tracking trace greater than 20 Pixels, then it is a trusted vehicle object.

Each vehicle object is in the tuple object format with 5 attributes as follows:

(VehicleID, x-axis intersection coordinate with baseline, bounding box width, direction (moving closer / away from camera), trustedFlag).

TrustedFlag is set to one for trusted vehicle object.

For example,
the first element is **(20, 757, 54, -1, 1)**

VehicleID 20 crossed baseline at pixel 757.

The vehicle 20's bounding box width is 54 Pixels, and the moving direction is away from camera, and the TrustedFlag is set 1 i.e., it is a trustable vehicle object for further computations.

The list of IDs of all vehicles under consideration (100) is as follows:

ID List [3, 5, 10, 12, 20, 13, 9, 11, 7, 6, 34, 8, 22, 56, 26, 35, 66, 40, 57, 79, 52, 89, 60, 96, 74, 81, 118, 102, 146, 148, 135, 122, 166, 151, 182, 209, 218, 188, 219, 215, 185, 177, 201, 293, 236, 320, 330, 345, 266, 244, 340, 276, 338, 363, 346, 348, 395, 362, 357, 387, 364, 398, 339, 408, 409, 385, 365, 420, 394, 354, 433, 372, 418, 448, 376, 468, 396, 520, 437, 538, 465, 426, 517, 512, 549, 618, 547, 502, 438, 584, 592, 595, 552, 675, 644, 622, 578, 627, 694, 697]

x-axes coordinate of point of intersection of vehicles with the baseline: All the vehicle objects under consideration (100) intersect with the baseline at the following respective x-axis coordinate.

intersection_list [107, 741, 256, 889, 757, 860, 900, 340, 759, 753, 119, 884, 237, 359, 357, 248, 109, 110, 759, 247, 1020, 123, 903, 348, 1004, 904, 1070, 930, 1062, 919, 934, 1023, 917, 749, 239, 883, 871, 347, 741, 122, 739, 887, 891, 133, 764, 230, 871, 216, 344, 929, 357, 770, 251, 897, 122, 930, 879, 348, 115, 953, 241, 761, 1008, 911, 754, 357, 745, 234, 233, 1023, 364, 882, 354, 224, 998, 357, 884, 872, 360, 849, 119, 993, 349, 235, 1014, 999, 351, 751, 1014, 122, 238, 1004, 760, 368, 366, 998, 1074, 120, 917, 870]
Length of Intersection List is100

x-axes coordinate of point of intersection of trusted vehicles with the baseline: All the trusted vehicle objects intersect with the baseline at the following respective x-axis coordinate.

Trusted Intersection List [757, 900, 340, 753, 119, 884, 237, 359, 248, 109, 110, 759, 247, 1020, 123, 903, 348, 1004, 904, 930, 1023, 749, 239, 347, 741, 122, 887, 891, 764, 871, 344, 929, 357, 770, 251, 122, 348, 115, 241, 761, 1008, 357, 234, 1023, 882, 354, 998, 884, 360, 119, 993, 235, 1014, 351, 751, 1014, 122, 238, 760, 1074, 120, 917, 870]
Length of Trusted Intersection List is 63

Bounding Box Width List of Vehicles under consideration (100): From this list of bounding boxes, we compute the median width of vehicle object's bounding box to eliminate vehicles spanning their bounding box greater than horizontal lane width.

width_list [37, 56, 67, 61, 54, 44, 45, 44, 62, 53, 53, 36, 43, 58, 35, 37, 38, 40, 42, 37, 51, 51, 60, 46, 42, 45, 17, 3, 95, 132, 70, 41, 80, 47, 37, 40, 68, 49, 60, 36, 36, 46, 35, 42, 43, 106, 38, 110, 44, 6, 45, 47, 35, 101, 43, 2, 54, 37, 35, 33, 34, 60, 47, 111, 77, 55, 53, 58, 47, 51, 38, 43, 47, 72, 55, 71, 66, 58, 47, 47, 50, 46, 48, 39, 59, 42, 49, 37, 54, 38, 52, 43, 48, 48, 61, 53, 3, 53, 77, 36]
Median Width = 47

Direction List of Vehicles under consideration (100):
direction_list [1, 1, 1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, 1, -1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1,

1, 1, 1, 1, 1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1, 1, -1, 1, 1, -1, -1, 1, 1, 1, -1, 1, 1, 1, -1, 1, -1, -1]
Length of Direction List is 100

Trusted Direction List of all trusted vehicles:
[-1, -1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, 1, -1, -1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, 1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1]
Length of Trusted Direction List is 63

From these intersection list, directions list we compute the final intersection and direction list eliminating the vehicles spanning their bounding box greater than horizontal lane width.

Final Intersection point list of vehicles with their bounding box width less than equal to median width
Intersection Points List [107, 860, 900, 340, 884, 237, 357, 248, 109, 110, 759, 247, 348, 1004, 904, 1070, 930, 1023, 749, 239, 883, 122, 739, 887, 891, 133, 764, 871, 344, 929, 357, 770, 251, 122, 930, 348, 115, 953, 241, 1008, 233, 364, 882, 354, 360, 849, 993, 235, 999, 751, 122, 1004, 1074, 870]
Length of Final Intersection List is 54

Final Trusted Direction list of vehicles with their bounding box width less than equal to median width
Direction Points List [1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 1, -1, 1, 1, 1, -1, 1, 1, -1, 1, 1, -1, -1]
Length of Final Direction List is 54

This final intersection and direction list is used to find the lane centers and trusted intersection list, direction lists are used to associate direction to the respective lane centers.

Appendix B: Detailed example output of the lane filtering algorithm

The x-coordinates of all the peaks in Fig A. are:
Printing Peaks X
119.0, 241.0, 353.0, 756.0, 884.0, 930.0, 953.0, 1002.0, 1068.0

Scaled height of Peaks:

These are the x-axis coordinate points of all the probable lane centre peaks.
Peak Dictionary {'peak_heights': array([0.24925024, 0.32183594, 0.32708273, 0.17162886, 0.27213119, 0.16356309, 0.05486461, 0.22228983, 0.06337123])}

Height of Local Mínimas in Fig. A.

Local Minimas [0.04503542356148052, 0.047127928420189875, 0.0015647217341090709, 0.00012418426461183104]
Local Minimas X Values [915.0, 946.0, 973.0, 1047.0]

median_width = 1.34 * median_width ⇒ 1.34 * 47 ⇒ 62.98

Lane_Filtering Algorithm: Each peak passed through lane filtering algorithm in the descending order of height.

Follow the lane filtering algorithm step-by-step from Page No. 20

The highest peak is at 353.0

Next Peak 241.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
353 - 241 = 112 > 75.576
}
Adding 241.0 As a Lane
Next Peak 241.0 is at distance >= 1.2*median_width from known lanes : **So it's A Lane**
Lanes [241.0, 353.0]

Next Peak 884.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
| 241 - 884 | = 643 > 75.576
| 353 - 884 | = 531 > 75.576
}
Adding 884.0 As a Lane
Next Peak 884.0 is at distance >= 1.2*median_width from known lanes : **So it's A Lane**
Lanes [241.0, 353.0, 884.0]

Next Peak 119.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
| 241 - 119 | = 122 > 75.576
| 353 - 119 | = 234 > 75.576
| 884 - 119 | = 765 > 75.576
}
Adding 119.0 As a Lane
Next Peak 119.0 is at distance >= 1.2*median_width from known lanes : **So it's A Lane**
Lanes [119.0, 241.0, 353.0, 884.0]

Next Peak 1002.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
| 119 - 1002 | = 883 > 75.576
| 241 - 1002 | = 761 > 75.576

| 353 - 1002 | = 649 > 75.576
| 884 - 1002 | = 118 > 75.576
}
Adding 1002.0 As a Lane
Next Peak 1002.0 is at distance >= 1.2*median_width from known lanes : **So it's A Lane**
Lanes [119.0, 241.0, 353.0, 884.0, 1002.0]

Next Peak 756.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
| 119 - 756 | = 637 > 75.576
| 241 - 756 | = 515 > 75.576
| 353 - 756 | = 403 > 75.576
| 884 - 756 | = 128 > 75.576
| 1002 - 756 | = 246 > 75.576
}
Adding 756.0 As a Lane
Next Peak 756.0 is at distance >= 1.2*median_width from known lanes : **So it's A Lane**
Lanes [119.0, 241.0, 353.0, 756.0, 884.0, 1002.0]

Next Peak 930.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
| 119 - 930 | = 811 > 75.576
| 241 - 930 | = 689 > 75.576
| 353 - 930 | = 577 > 75.576
| 756 - 930 | = 174 > 75.576
| 884 - 930 | = 46 < 75.576 ⇒ Condition Not Satisfied
}
checkLaneWithDifference_0point75
{
0.75 * 62.98 = 47.235
| 119 - 930 | = 811 > 47.235
| 241 - 930 | = 689 > 47.235
| 353 - 930 | = 577 > 47.235
| 756 - 930 | = 174 > 47.235
| 884 - 930 | = 46 < 47.235 ⇒ Condition Satisfied ⇒ **Does not qualify for a Lane**
}
Distance between peak and adjacent lane is < 0.75*median_width : 930.0 Is Not a Lane
Lanes [119.0, 241.0, 353.0, 756.0, 884.0, 1002.0]

Next Peak 1068.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
| 119 - 1068 | = 949 > 75.576
| 241 - 1068 | = 827 > 75.576
| 353 - 1068 | = 715 > 75.576

| 756 - 1068 | = 312 > 75.576
| 884 - 1068 | = 184 > 75.576
| 1002 - 1068 | = 66 < 75.576 ⇒ Condition Not Satisfied
}

checkLaneWithDifference_0point75
{
0.75 * 62.98 = 47.235
| 119 - 1068 | = 949 > 47.235
| 241 - 1068 | = 827 > 47.235
| 353 - 1068 | = 715 > 47.235
| 756 - 1068 | = 312 > 47.235
| 884 - 1068 | = 184 > 47.235
| 1002 - 1068 | = 66 > 47.235 ⇒ Condition does not satisfy
}
checkPeakLessThan_0point5
{
Left Adjacent Lane 1002.0
Difference between Next_Peak and Left Adjacent Lane 66.0
Height of Next_Peak 0.06337123023141739
Height Of Left Adjacent Lane Height 0.22228983365517754
Height Of Next_Peak is Less than 0.5 * left_adj_lane_height
Height Of peak 1068.0 is less than 0.5 of the car count in the adjacent lower count lane and this lane is between 0.7D and 1.2D : **It's Not a Lane**
}
Lanes [119.0, 241.0, 353.0, 756.0, 884.0, 1002.0]

Next Peak 953.0
checkLaneWithDifference_1point2
{
1.2 * 62.98 = 75.576
| 119 - 953 | = 834 > 75.576
| 241 - 953 | = 712 > 75.576
| 353 - 953 | = 600 > 75.576
| 756 - 953 | = 197 > 75.576
| 884 - 953 | = 69 < 75.576 ⇒ Condition Not Satisfied
}
checkLaneWithDifference_0point75
{
0.75 * 62.98 = 47.235
| 119 - 953 | = 834 > 47.235
| 241 - 953 | = 712 > 47.235
| 353 - 953 | = 600 > 47.235
| 756 - 953 | = 197 > 47.235
| 884 - 953 | = 69 > 47.235
| 1002 - 953 | = 49 > 47.235 ⇒ Condition Not Satisfied
}
checkPeakLessThan_0point5
{
Left Adjacent Lane 884.0

Difference between Next_Peak and Left Adjacent Lane 69.0
Right Adjacent Lane 1002.0
Difference between Next_Peak and Right Adjacent Lane 49.0
Diff1 and Diff2 are within the Range >=0.7 and <= 1.2
Height of Next_Peak 0.054864608105506966
Height Of Left Adjacent Lane Height 0.27213118825713595
Height Of Right Adjacent Lane Height 0.22228983365517754
Height Of Next_Peak is Less than 0.5 * left_adj_lane_height or right_adj_lane_height
Height Of peak 953.0 is less than 0.5 of the car count in the adjacent lower count lane and this lane is between 0.7D and 1.2D : **It's Not a Lane**

}
Lanes [119.0, 241.0, 353.0, 756.0, 884.0, 1002.0]

So, the Final Lane_Centers are [119.0, 241.0, 353.0, 756.0, 884.0, 1002.0]

Once we have the lane_centers we assign then the direction as follows:
From the trusted_cross_baseline_list we have the trusted_intersection_list consisting of intersection points of trusted vehicle objects with the baseline along with associated trusted direction in the trusted_direction_list.

These lane centers are nothing but the x-axis coordinates on the baseline so for each of the lane centers the vehicle object passing exactly through that intersection point or closet intersection point is considered, and its associated direction is assigned to the lane center.

Direction issue Print
Lane Considered 119.0
Result of Closet 4
Direction point as per closet 1
##########
Lane Considered 241.0
Result of Closet 38
Direction point as per closet 1
##########
Lane Considered 353.0
Result of Closet 45
Direction point as per closet 1
##########
Lane Considered 756.0
Result of Closet 0
Direction point as per closet -1
##########
Lane Considered 884.0
Result of Closet 5
Direction point as per closet -1
##########
Lane Considered 1002.0
Result of Closet 17
Direction point as per closet -1

##

Therefore, final Lane Directions [1, 1, 1, -1, -1, -1]

Lane Centers: [119.0, 241.0, 353.0, 756.0, 884.0, 1002.0]
Lane Directions: [1, 1, 1, -1, -1, -1]

These lane centers are plotted along with their direction in Fig C.

+1 ⇒ Vehicles moving downwards in the frame i.e., approaching the camera
-1 ⇒ Vehicles moving upwards in the frame i.e., moving away from the camera

Reference:

[1] Temporal Mapping of Surveillance Video for Indexing and Summarization by Saeid Bagheri, Dr. Jiang Zheng, Shivank Sinha (Indiana University Purdue University Indianapolis).