ENPM673 Project2
Perception of Autonomous Robot
Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars

By (Group 2)
B. Sai Praveen(DirID: spraveen)
Preyash Parikh(DirID: pparikh)
Ajinkya Parwekar(DirID: ajinkyap)

March 11, 2020

INTRODUCTION

The objective of this project is to do a simple Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars. An algorithm is designed to detect the lanes on the road and to estimate the road curvature to detect the car turns.

1 PROBLEM 1

In this problem, we were given a video, which was recorded at night time, and we were supposed to enhance the quality of video so that it becomes more suitable for being used in applications like lane detection. A screenshot from original video is shown in figure 1.

Figure 1: Screenshot from Original Video

Initially, we decided to use histogram equalization for enhancing the video quality. But the results were not as expected (figure 2). So we decided to try and adjust adjust the brightness and contrast of the video by altering the alpha and beta values of the video. Even after adjusting the brightness and contrast, the results were not quite satisfactory (figure 3). So we reached the decision of adjusting the gamma value, which yielded us a satisfactory result.

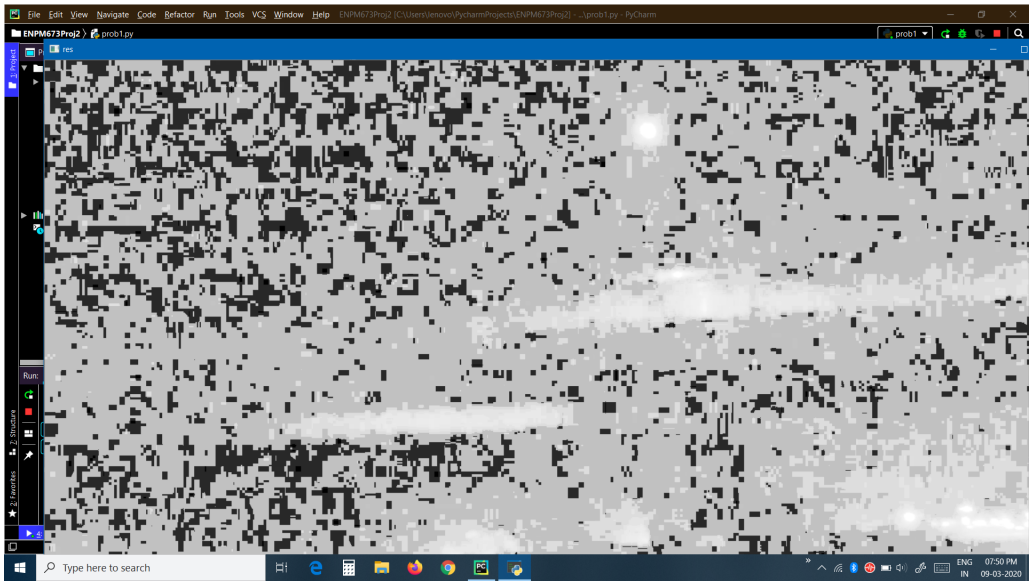

Figure 2: Histogram Equilization

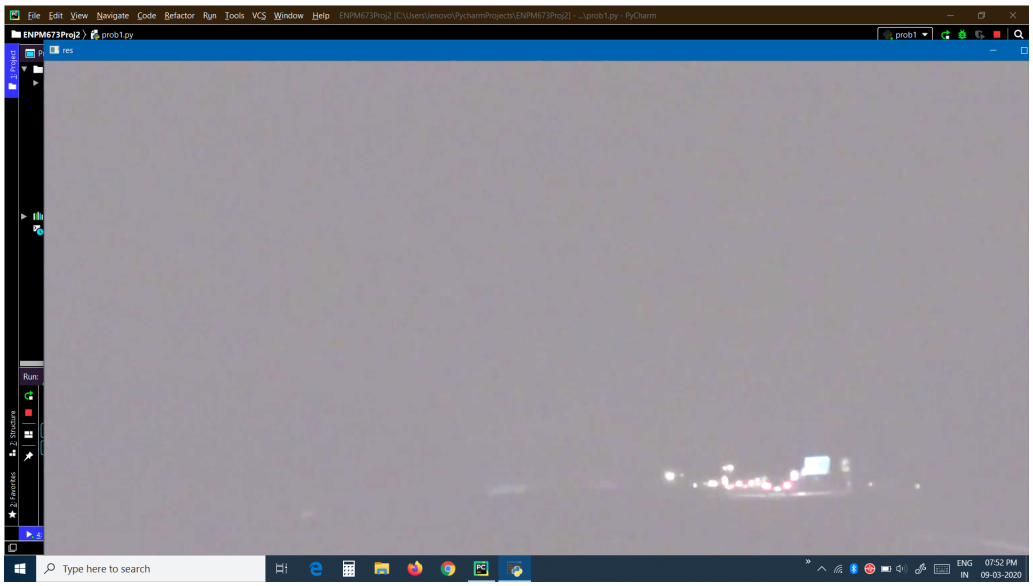

Figure 3: Brightness and Contrast

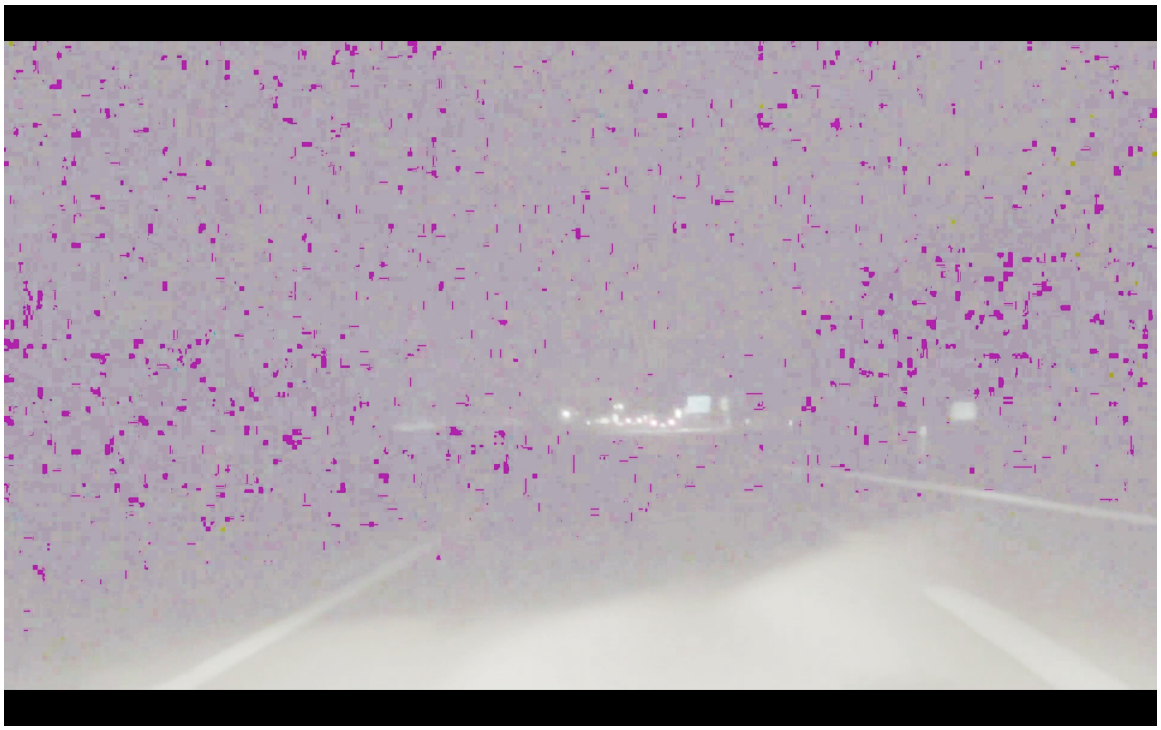

Figure 4: Gama value of 0.1

Figure 5: Gama value of 0.8

Figure 6: Final Output for Gamma value 0.4

To change the gamma value of the video, we split the video into multiple frames. Then we changed the gamma value for each frame and compiled all the frames back to create the video. Then the main challenge was to find the optimum gamma value so as to get the best possible result. For this we used trial and error method, trying some values over a finite range. The results for various gamma values (0.1, 0.8 and 0.4) are shown below (figure 4, figure 5, figure 6 respectively). Gamma value of 0.4 yielded the best result so we finalised it.

2 PROBLEM 2: LANE DETECTION

In this problem we aim to do a simple lane detection where we are provided with 2 videos of a self-driving car. Here, we need to detect the lanes on the road and estimate the road curvature to detect car turns. For this we used Hough Transform technique which is a voting based algorithm.

Representation of lines in Hough Transform
Lines can be represented by two parameters a and b in equation

$$y = ax + b$$

. But this equation is unable to represent vertical lines. Therefore, Hough Transform uses polar co-ordinates. This equation is given by,

$$d = x\cos\theta + v\sin\theta$$

Here, the parameter 'ρ' is the perpendicular distance and 'θ' is the angle made by the line. The following is an image of the polar co-ordinates, (figure 7).

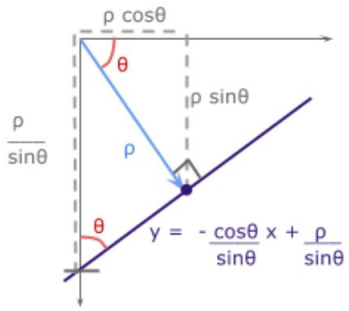

Figure 7: Image of polar co-ordinates

The Hough space for lines therefore has these two dimensions ρ and θ, and a line is represented by a single point, corresponding to a unique set of parameters. Fourier Transform is similar to Hough transform. This introduced a new coordinate system for some pixel, represented by the coefficient of the polynomial that it belongs to, however it uses Fourier series expansion instead.

How we have implemented Hough Transform?

We perform Hough Transformation after our lines are defined. These lines which indicates where our lanes are located are very noisy and flickery and to visualize them as a single line, we perform Hough Transformations. First step is to detect all lines and we have defined them as [x1, y1] as start and [x2, y2] as end. After localising them, we can calculate their slopes to determine whether they are right or left ones. Slopes with negative values are left ones and with positive values are right ones.
Then we need to derive single left and right line. Finally, we can draw our lines using in-built function "cv2.HoughLinesP()".

Mapping Points to Hough Transform
The idea is, that a point is mapped to all lines, that can pass through that point. This yields a sine-like line in the Hough space. (see figure 8).

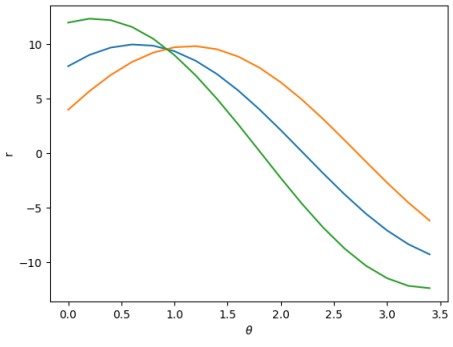

Figure 8: Intersection point represent line

2.1 IMPLEMENTATION: PART 1

For the implementation part, first we defined the region of interest of our video frame which is bottom half of the image (see figure 9).

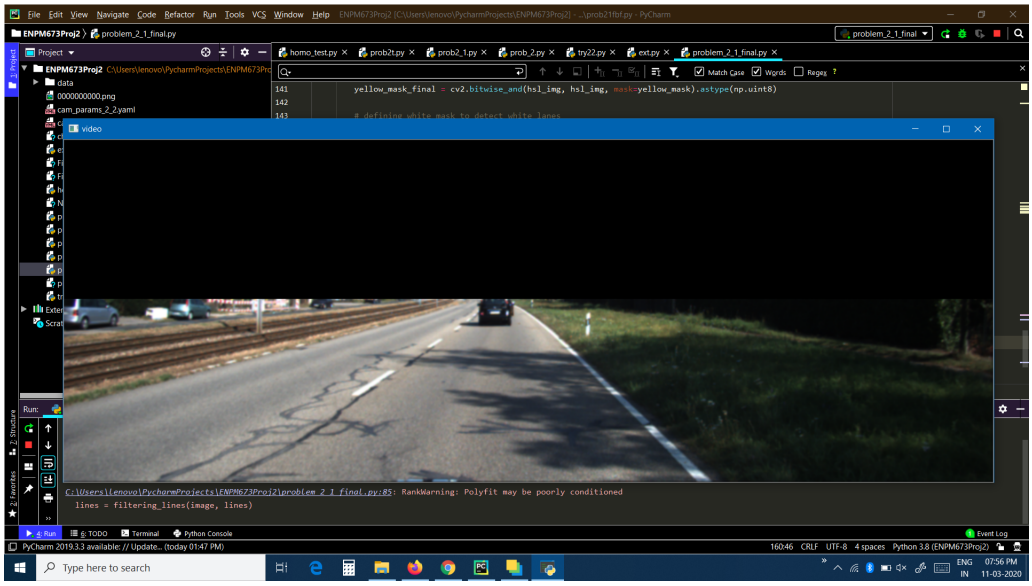

Figure 9: roi

Then we used color masks to detect the white color lane boundaries. We had to use the color mask because, by simply applying canny edge detector on the raw frame did not yield expected results. The result is displayed in figure 10.

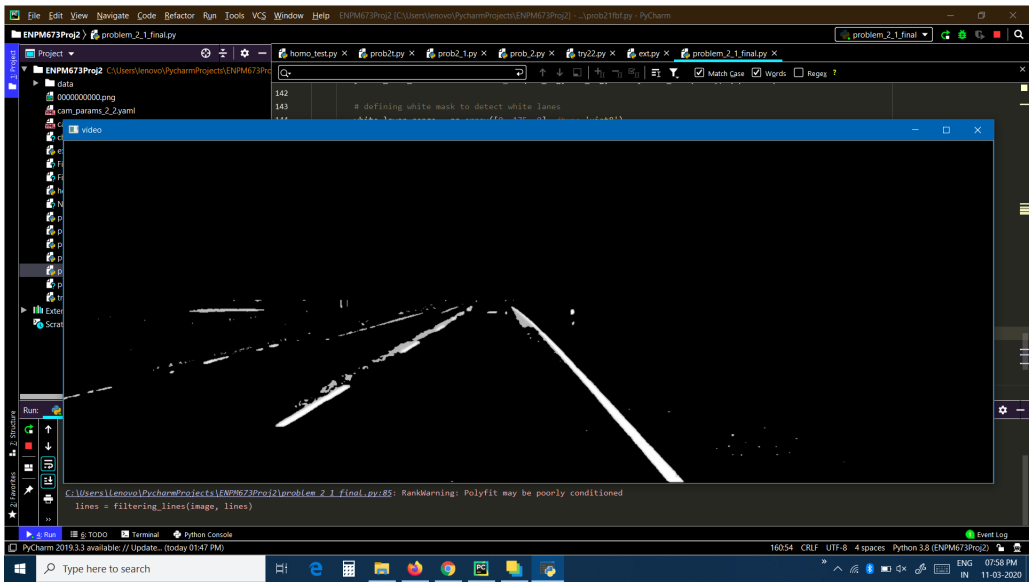

Figure 10: Lanes detected using masks

Then we applied canny edge detector on the detected lanes (figure 11).

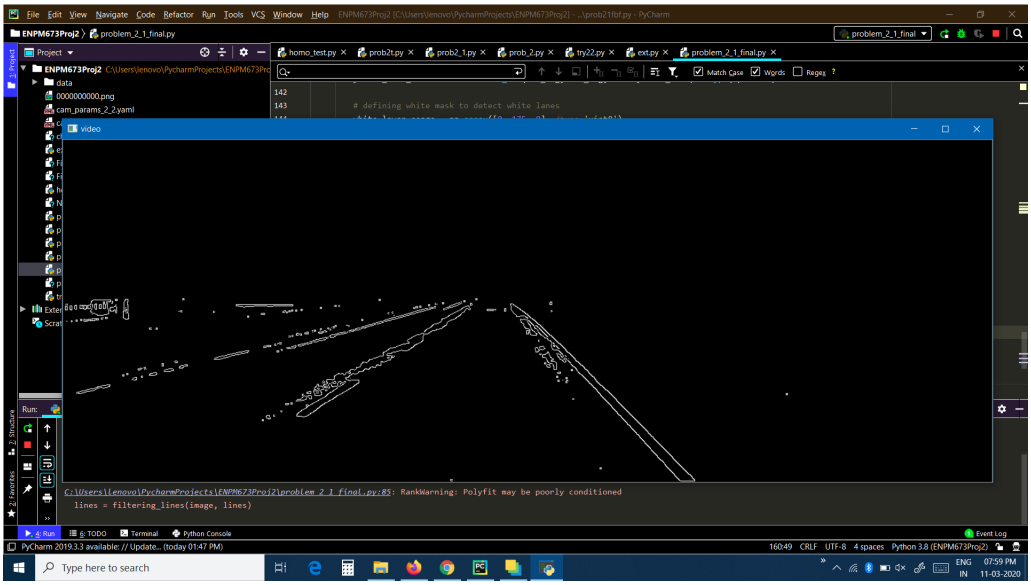

Figure 11: Applied canny on lanes

Then we used the Hough transform function to draw lines on the detected edges and also drew the mesh (yellow coloured) between them (figure 12).

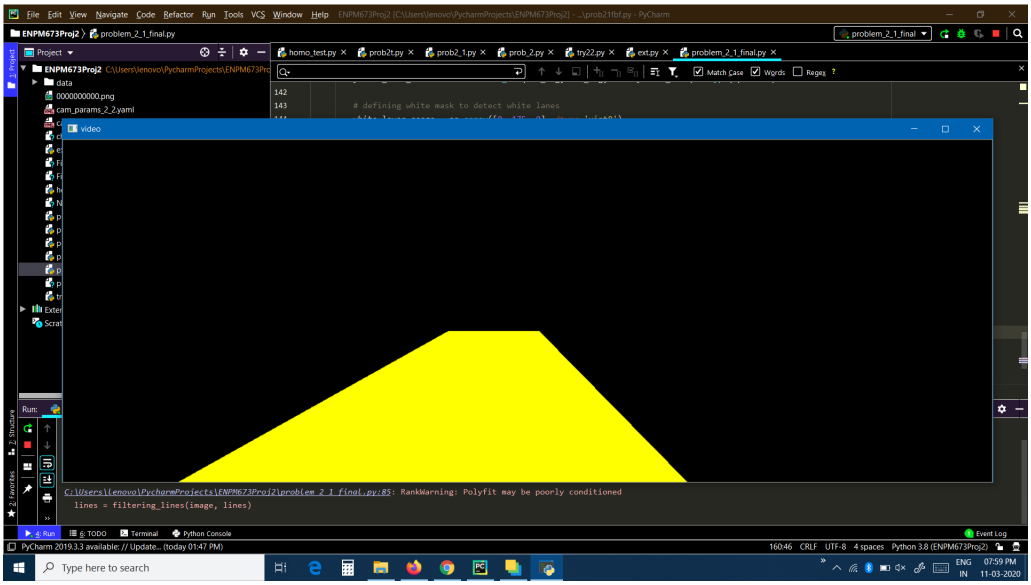

Figure 12: Drawn mesh on detected lane

Then finally we imposed the mesh onto the original frame (figure 13).

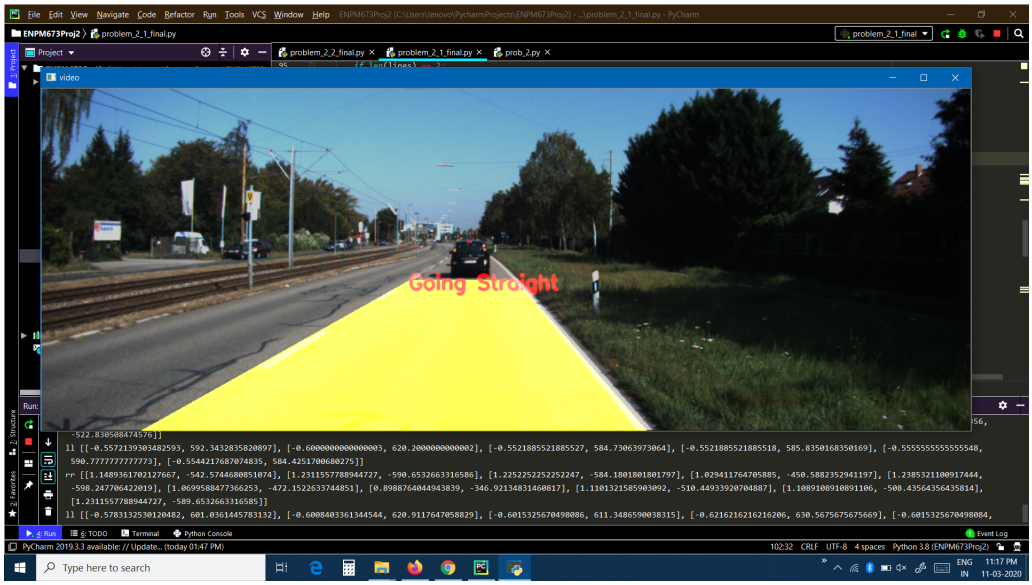

Figure 13: Combined the result with original frame, with turn prediction

2.2 IMPLEMENTATION: PART 2

For the part 2, we followed a similar way as we used for the part 1. First we defined the region of interest of our video frame which is bottom half of the image (see figure 14).

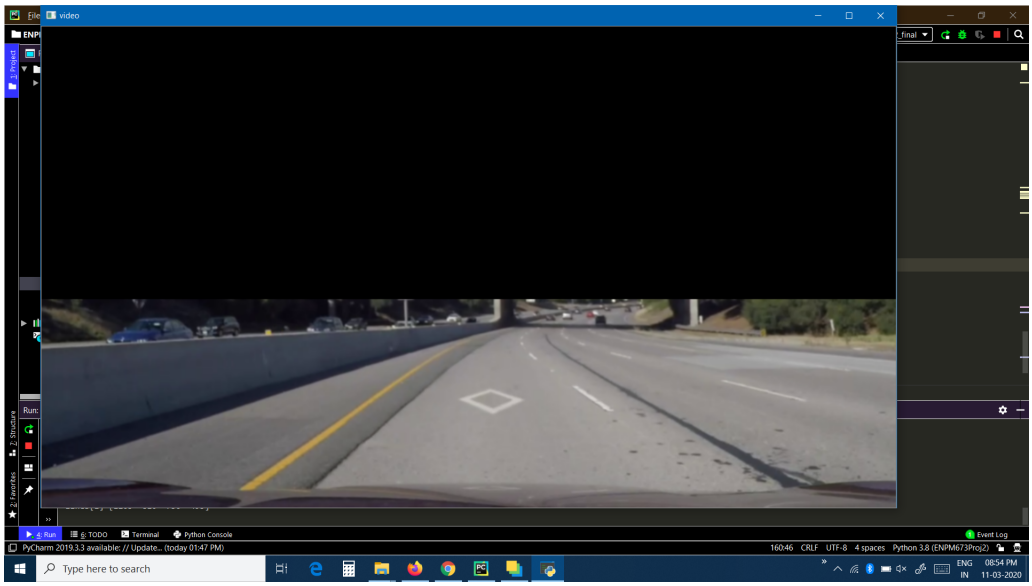

Figure 14: roi

UNDISTORTING

To remove distortion, we have used give correction matrices. To implement so, we have in-built undistort function.

SEPARATING THE TWO LINES

We always want to detect lanes which are yellow and white in such a way that they are distinguishable. In order to do so, we performed HSL conversion on image and created two masks. One for yellow lines and and one for white lines. After experimenting, we ended at a particular value which gave us the perfect result. We have used inbuilt function bitwise to get the image as shown.(see figure15).

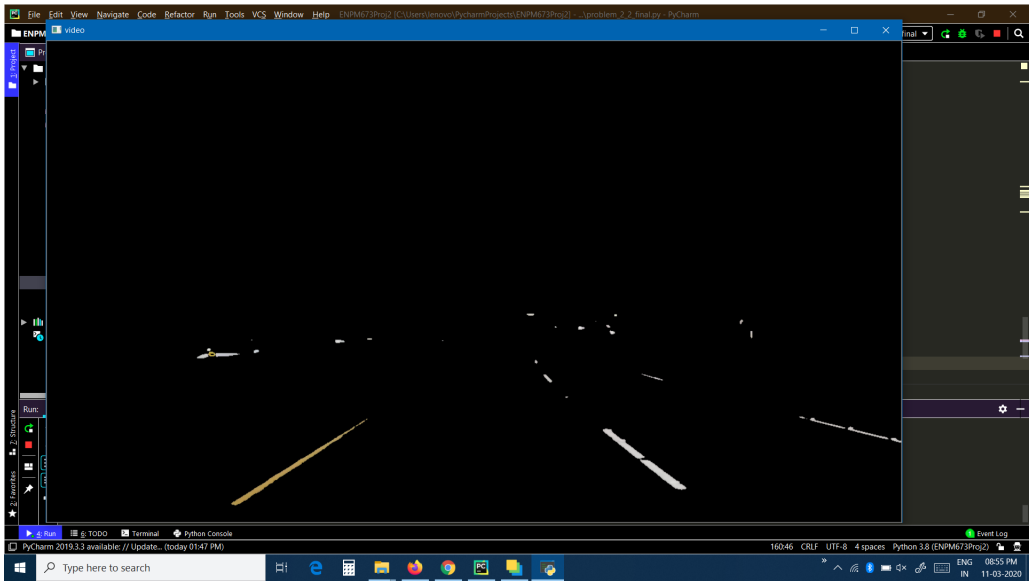

Figure 15: Lanes detected using masks

Then we applied canny edge detectors on the lanes. (see figure16)

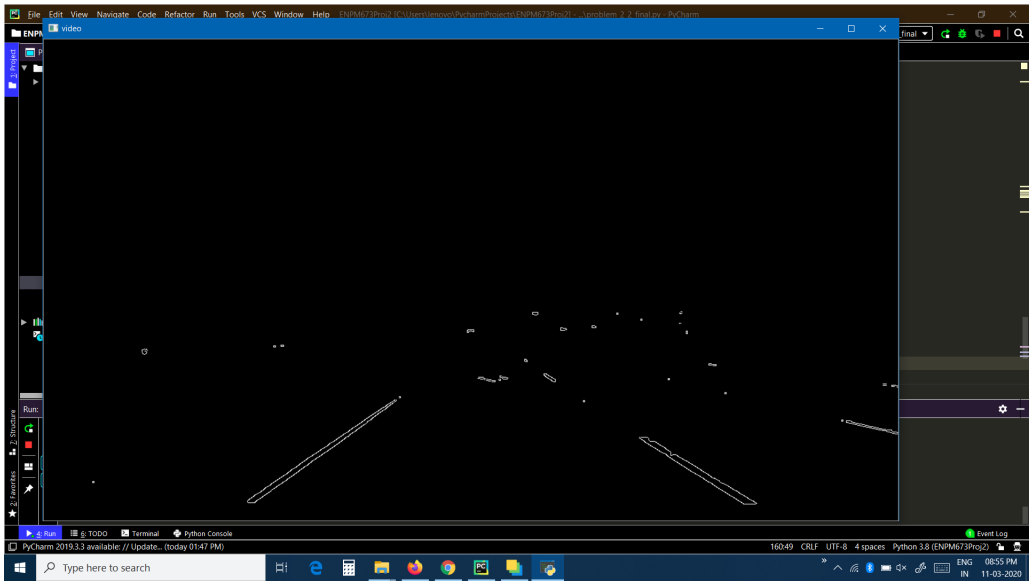

Figure 16: Applied canny on lanes

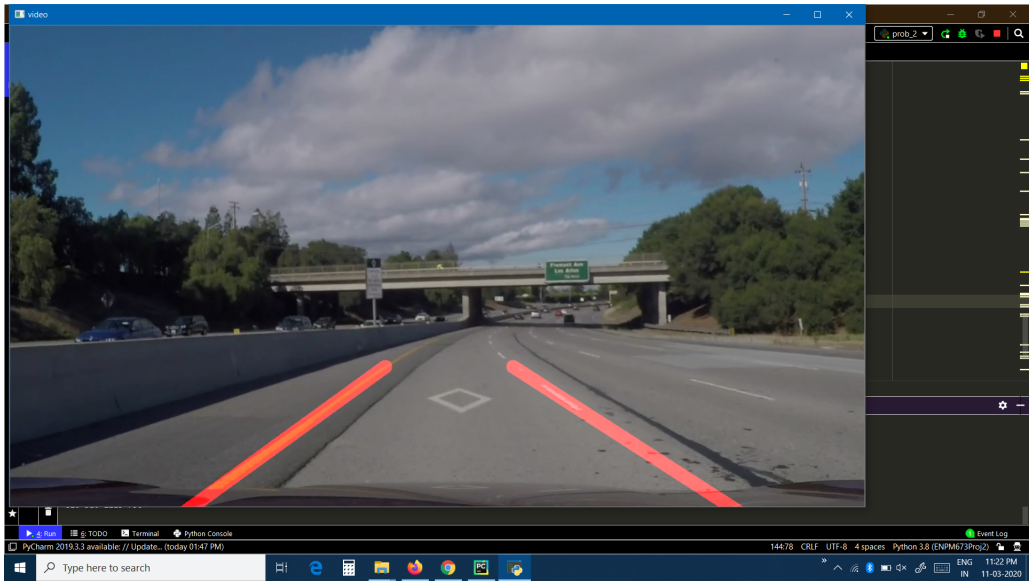

Figure 17: Hough lines, as explained above

Then we used the Hough Transform on the detected edges and drew a mesh between them. (see figure 18).

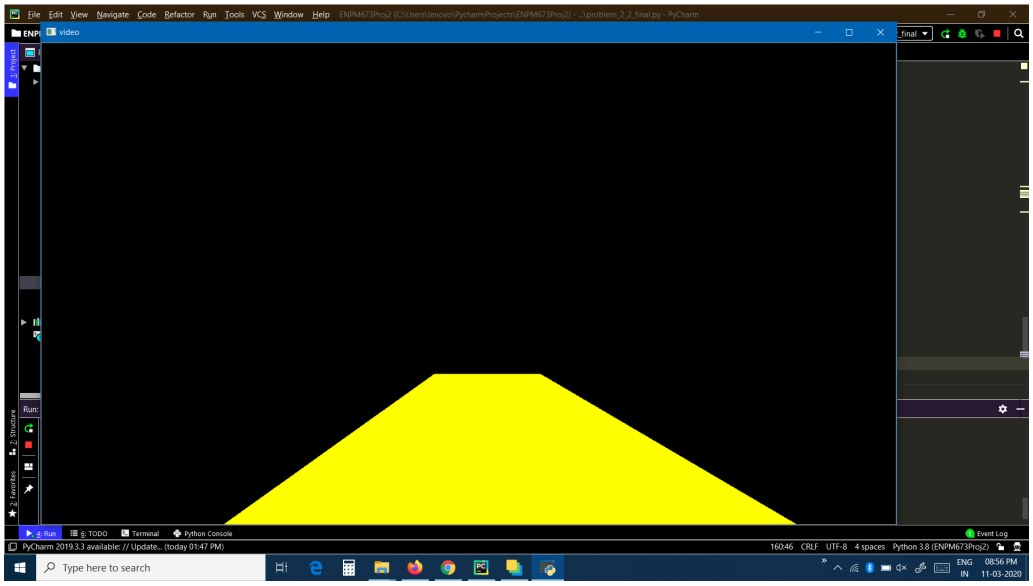

Figure 18: Drawn mesh on detected lane

Then finally we applied the mesh onto the original frame. (see figure 19).

Figure 19: Combined the result with original frame, with turn prediction

TURN PREDICTION

The final step is done with the help of value of slope. So if slope is greater then certain value, then its should turn right and vice versa. If the slope is in certain range then predict "straight".

This program will run for most of the environments. If the road is so curved, then the program might not run. But it will run for such curved roads as provided in the given videos.

The following is the google drive link for the final output videos of the project.
https://drive.google.com/drive/folders/1subzjEo5C5ovcrHrNWKeOjO6URZqlfVQ?usp=sharing

3 CHALLENGES FACED

A) VIDEO PROBLEMS

When the same code for dataset 1 was run for challenge video, initially we did not get hough lines continuously on white lines as these lines were not continuous.

To overcome this challenge, we changed last three parameters of in-built Hough function. In addition to this, lower value of white mask is also changed and minimum distance required for the points to be defined as lines are also increased.

B) TURN PREDICTION

Because the logic we have used in this project, the slope values of lines are changing very rapidly and due to this it was difficult to stabilise the output of turn prediction.

C) COLOR SELECTION RANGE

To get proper white and yellow lines using mask we had to try different range of values. This task took our good amount of time.

4 REFERENCES

1) https://medium.com/@tempflip/lane-detection-with-numpy-2-hough-transform-f4c017f4da39
2) https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html
3) https://www.youtube.com/watch?v=VyLihutdsPk&t=1392s