ENPM 673

Perception of Robotics

Project 2

By

Mothish Raj Venkatesan Kumararaj

Question 1:

Problem 1: Histogram equalization [30]

Dataset for Problem 1

Here, we aim to improve the quality of the image sequence provided above. This is a video recording of a highway during night. Most of the Computer Vision pipelines for lane detection or other self-driving tasks require good lighting conditions and color information for detecting good features. A lot of pre-processing is required in such scenarios where lighting conditions are poor.

Now, using the techniques taught in class your aim is to enhance the contrast and improve the visual appearance of the video sequence. You **cannot** use any in-built functions for the same. You have to use **histogram equalization** based methods, both **histogram equalization** and **adaptive histogram equalization** and compare the results.

Note:

1. We understand that adaptive histogram equalization will be slow, that is why only 25 frames are provided in the dataset.

Process:

Histogram:

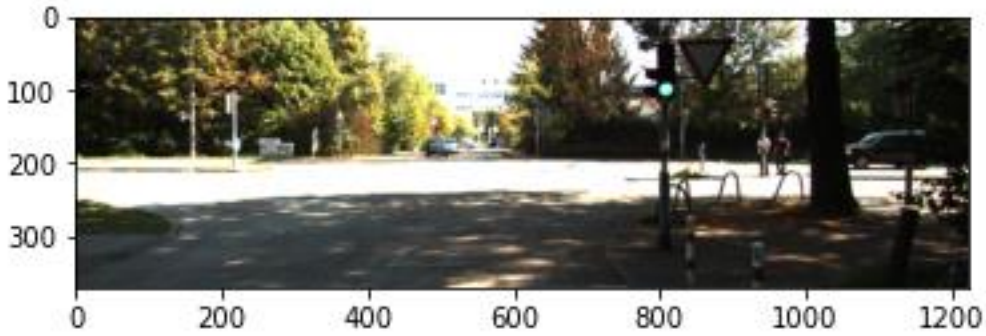
- Get all the image from the adaptive_hist_data folder.
- Read the given image using OpenCV.
- Flatten the image in 1D array.
- Calculate the histogram graph for each color gradient from 0 to 255.
- Get the cumulative sum for each intensity in histogram graph and normalize it.
- Replace the position in the flattened array of the image with the corresponding cumulative sum.
- Reshape the image

Adaptive Histogram:

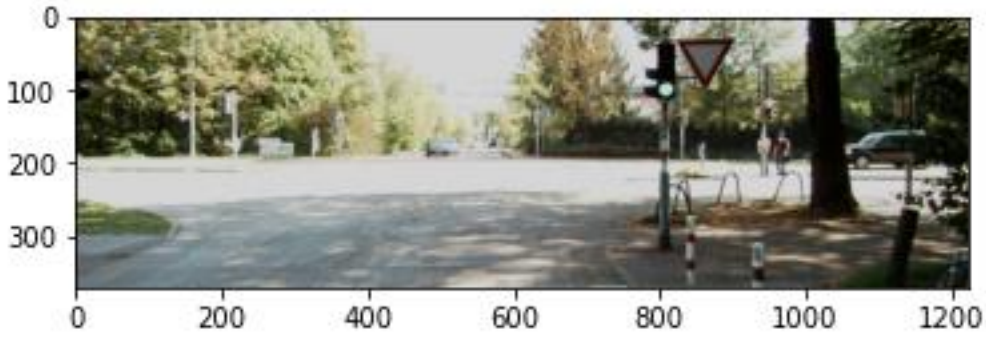
- Read the given image using OpenCV.
- Flatten the image in 1D array.
- Separate the image into small grid.
- Apply the histogram for each grid separately
- Replace the position in the flattened array of the image with the corresponding cumulative sum for each grid.
- Reshape the image

Results:

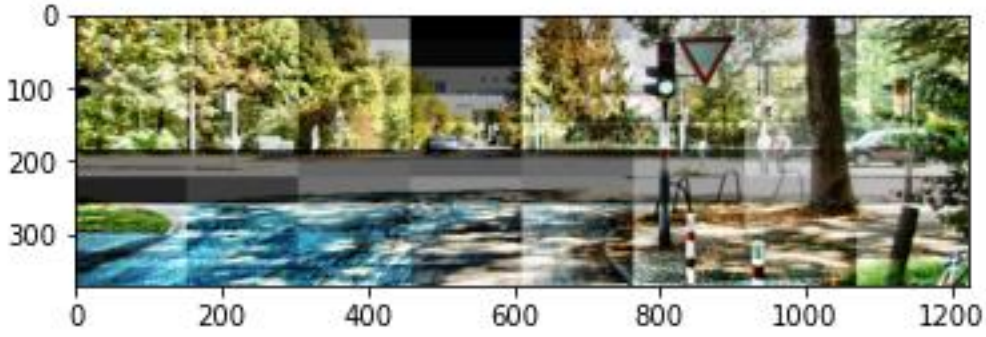
Input Image:



Histogram Image:



Adaptive Histogram Image:



Question 2:

Problem 2: Straight Lane Detection [35]

Dataset for Problem 2

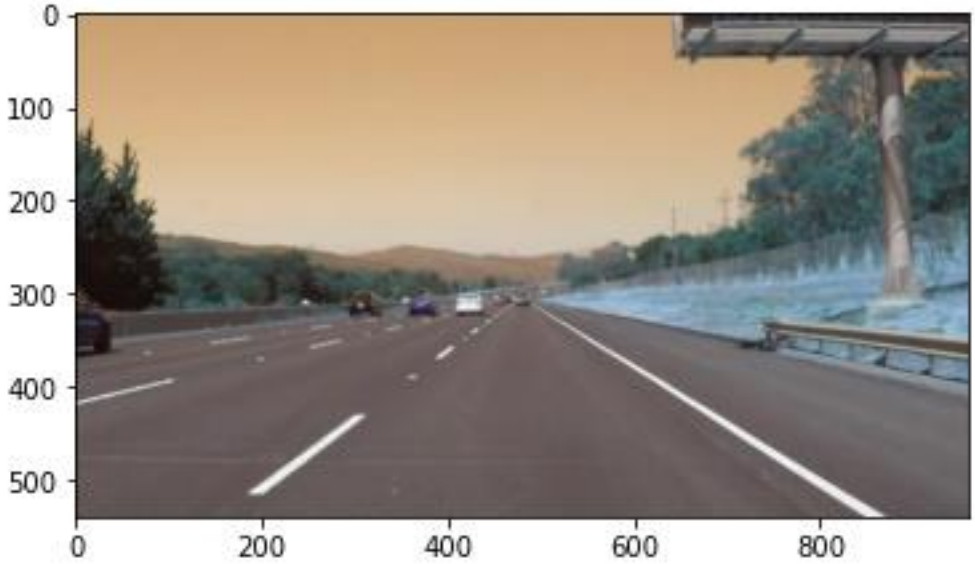
In this problem we aim to do simple Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars. You are provided with a video sequence, taken from a car. Your task is to design an algorithm to detect lanes on the road, and classify them as dashed and solid lines. For classification of the line type, you have to use different colors. Use **green for solid** and **red for dashed.**

Process:

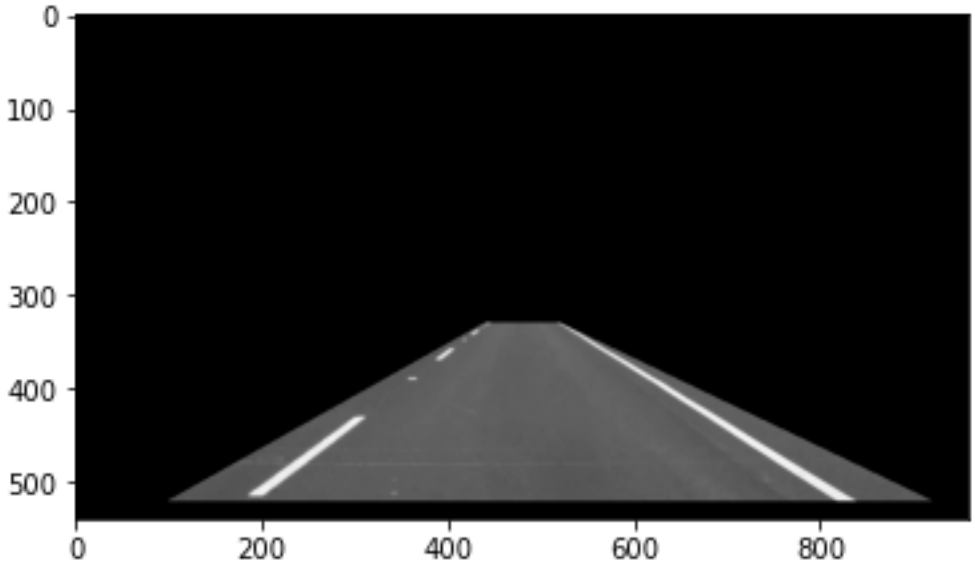
- Read the video and get separate frame.
- Convert the image to gray scale and mask the image to specific coordinate to get only the road.
- Convert the masked image to binary format.
- From the binary converted image, do Hough line transform to get the line coordinates.
- The lines received is ordered from longest to short so get the slope of the longest line.
- Check the signs of the slope of the longest line with other lines.
- If the sign is same, then they belong to solid line else it belongs to dashed line.
- Fill green for solid line and red for dashed line.

Results:

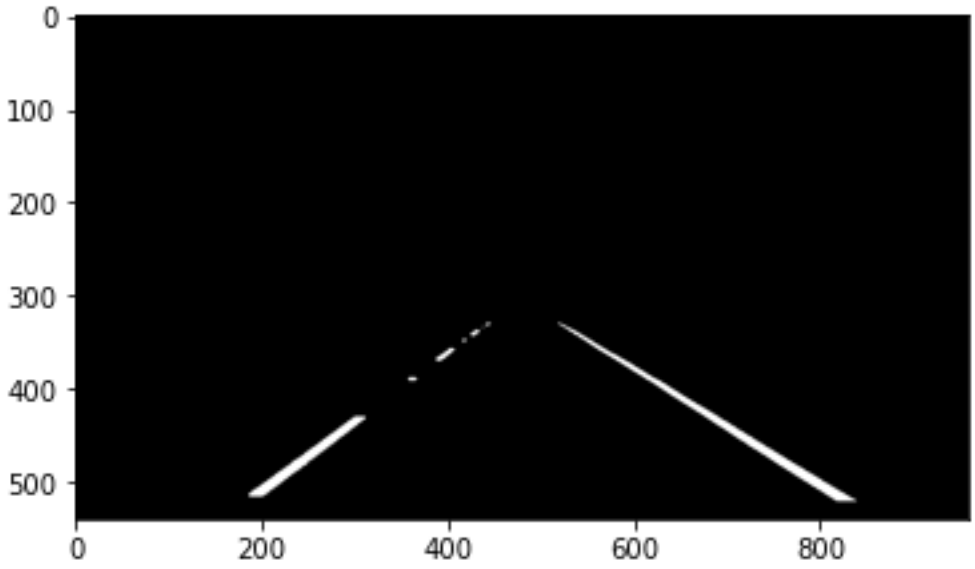
Input Image:



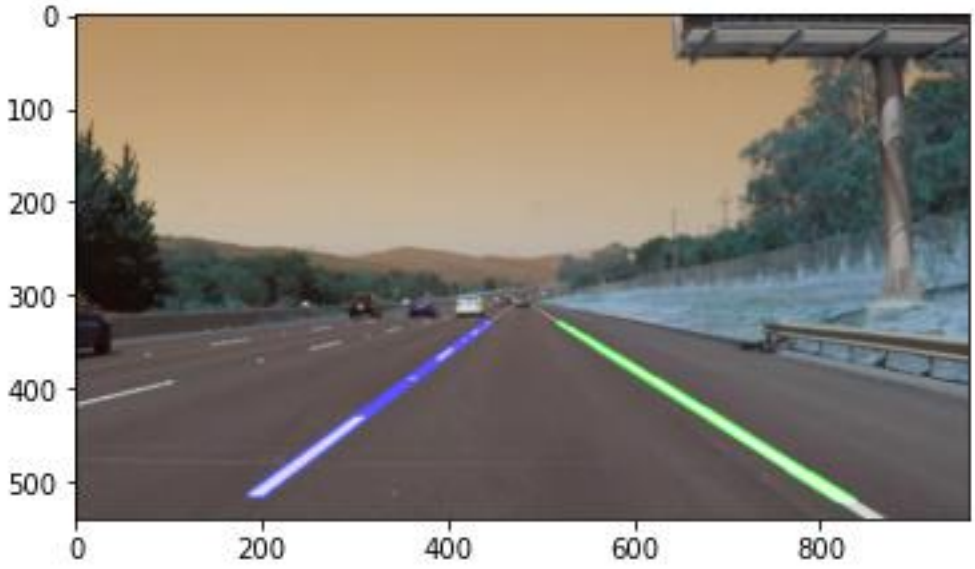
Masked Image:



Binary Masked Image:



Output Image



Link for output video:

https://drive.google.com/file/d/1OHV8_M0kzU-1JKFEya9jfW6A0WmcJLFG/view?usp=sharing

Question 3:

Problem 3: Predict Turn [35]

Dataset for Problem 3 :

In this problem, we aim to detect the curved lanes and predict the turn depending on the curvature: either left or right turn. The dataset provided has a yellow line and a while line. Your task is to design an algorithm to detect these lanes on the road, and predict the turn. Please note that for the output of the lane detection, you have to superimpose the detected lane as shown in fig.4. Also compute the radius is curvature for the lane using the obtained polynomial equation. Refer to this.

When your lane detection system loses track(cannot find lane lines), you must manage to use the past history of the lanes from the previous image frames to extrapolate the lines for the current frame.

(a) Polynomial fitting (b) Backprojection onto the original image

Figure 4: Polynomial fitting and an example of the final output

Process:

- Read the given image using OpenCV.
- For each frame do binary and canny edge detection and combine the two outputs for better results.
- Specify the coordinated points to perspective warp the threshold image to get only the lanes.
- Get histogram for ones in each pixel of the first half and second half of the image in x direction separately.
- The maximum intensity of the left and right half gives us the lanes location along x
- Add margin of 100 to get the start and end of the left and right lanes.
- Form a non-zero matrix for x and y direction of the image
- Separate the image into 9 grids for the y direction and obtain the non-zero points on the left and the right.
- Concatenate the left and right land index points and put the index in the non-zero matrix in x and y direction to get the coordinate position.

- Use Polyfit to get the constants of curve equation and calculate the variable.
- Convert the pixels to meters and multiply the conversion to the variables and constants to get left and right lanes fit.
- Get the warped image with marking of the lanes for the output.
- The radius of curvature can be calculated using the formula.

Radius of Curvature, $R = \frac{(1 + (\frac{dy}{dx})^2)^{3/2}}{|\frac{d^2y}{dx^2}|}$

Radius of Curvature Formula

- In polar coordinates $r=r(\Theta)$, the radius of curvature formula is given as: $\rho = \frac{1}{K} \frac{[r^2+(\frac{dr}{d\theta})^2]^{3/2}}{|r^2+2(\frac{dr}{d\theta})^2 - r\frac{d^2r}{d\theta^2}|}$
- $R = 1/K$, where R is the radius of curvature and K is the curvature.
- $R = \frac{(1 + (\frac{dy}{dx})^2)^{3/2}}{|\frac{d^2y}{dx^2}|}$

where K is the curvature of the curve, K = dT/ds, (Tangent vector function)

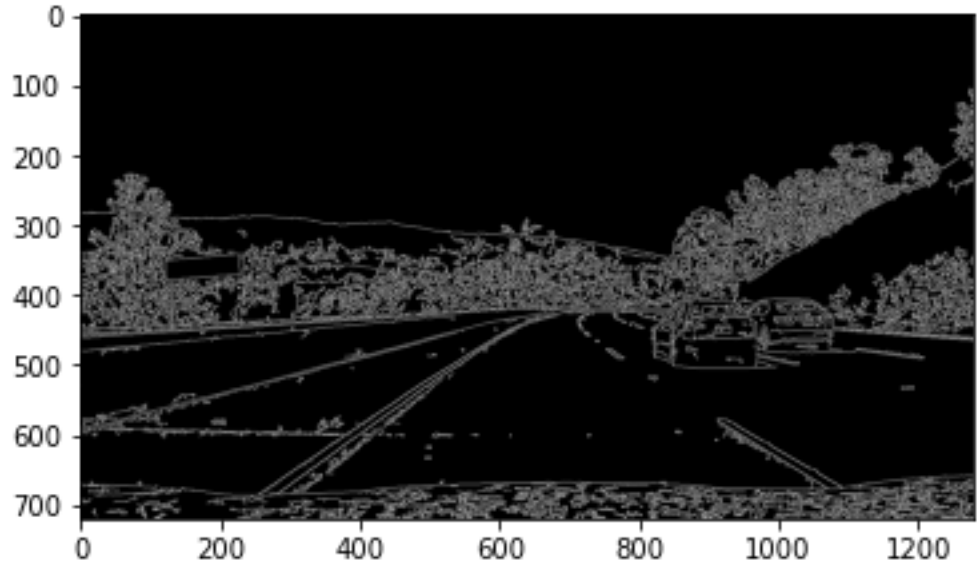
R the radius of curvature

- Use fill poly to draw the marking on the lane and warp it according to the original image and use added weightage
- Add the radius of curvature to the image and get the final image
- Store the final image for each frame in a video and display it in the console.

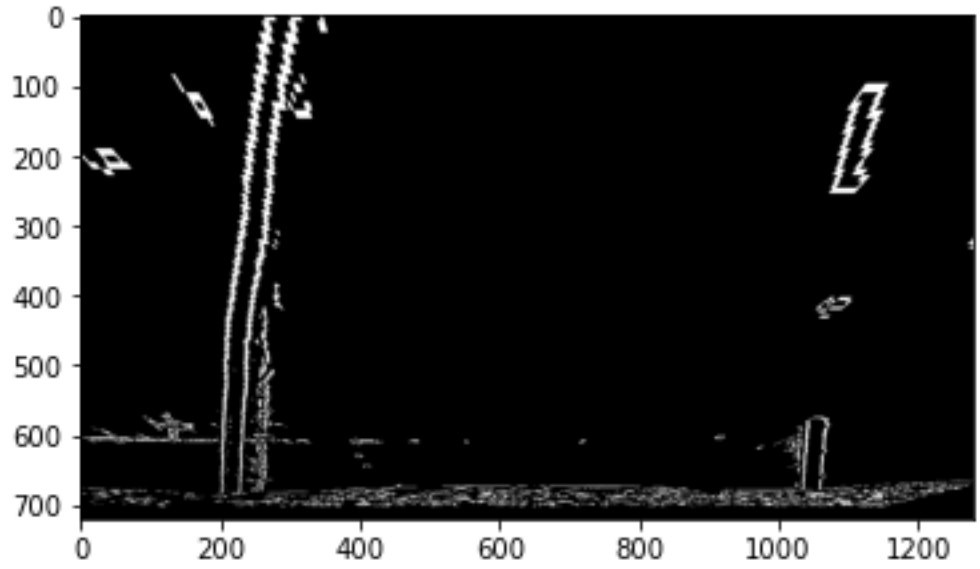
Results:

Input Image:

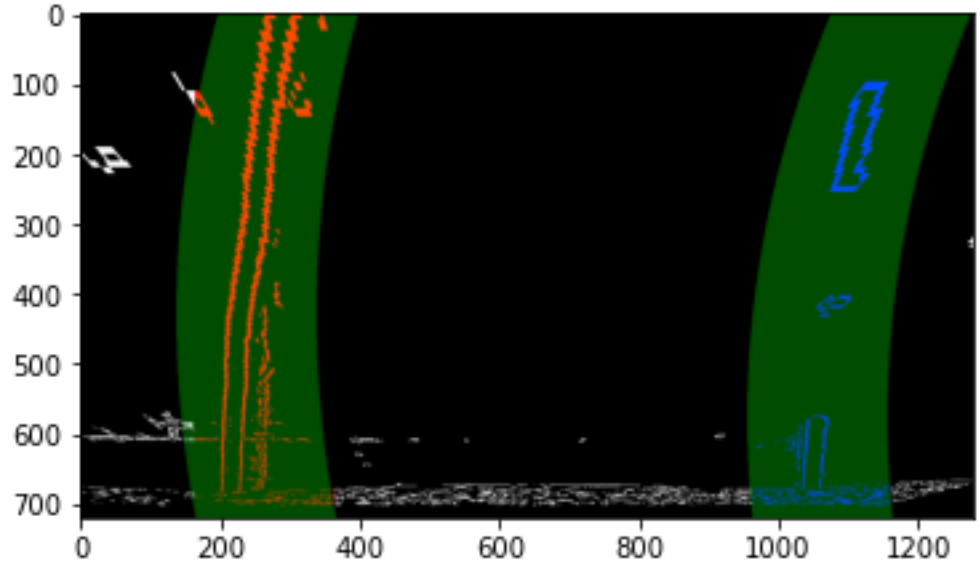
Threshold Combined Image:



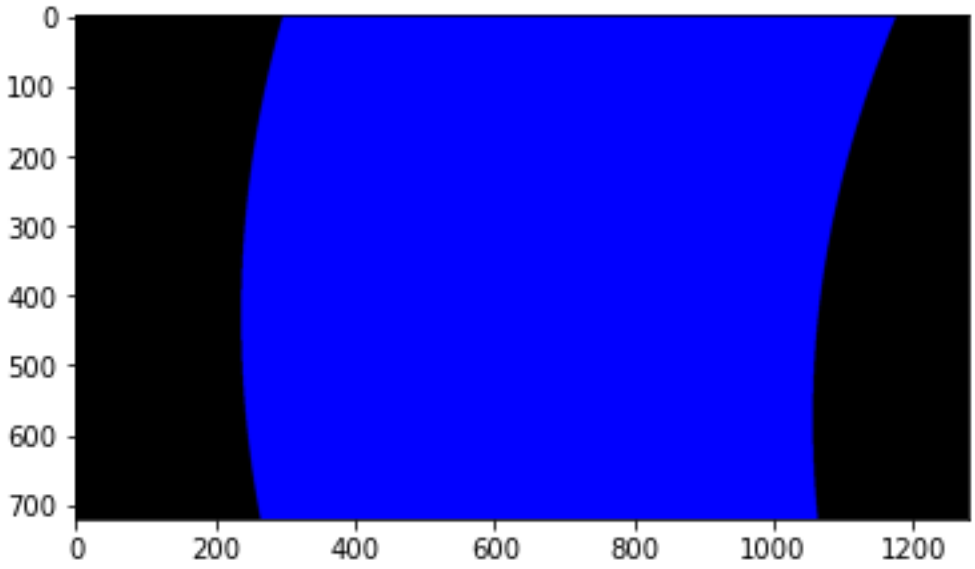
Perspective Warped Image:



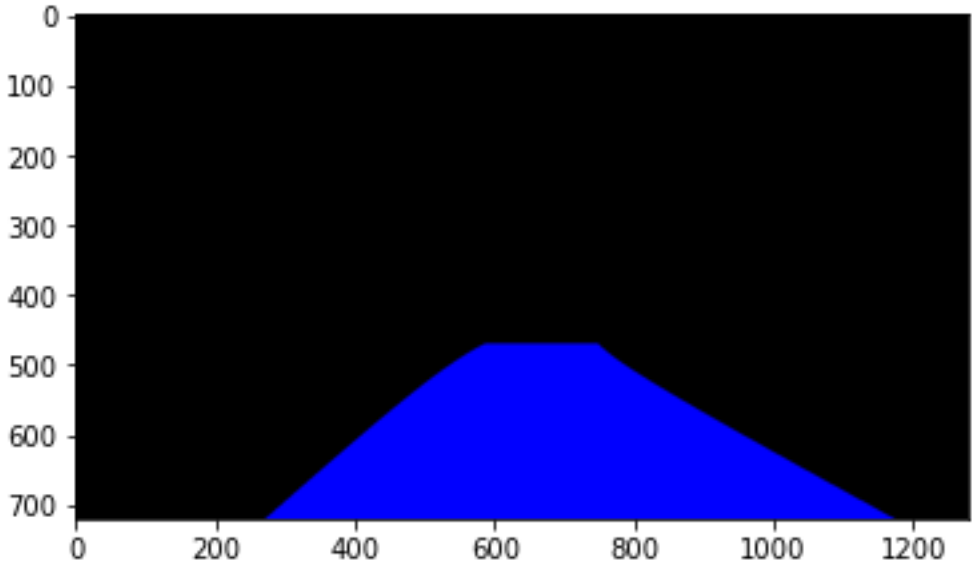
Warped Image with lane marking:



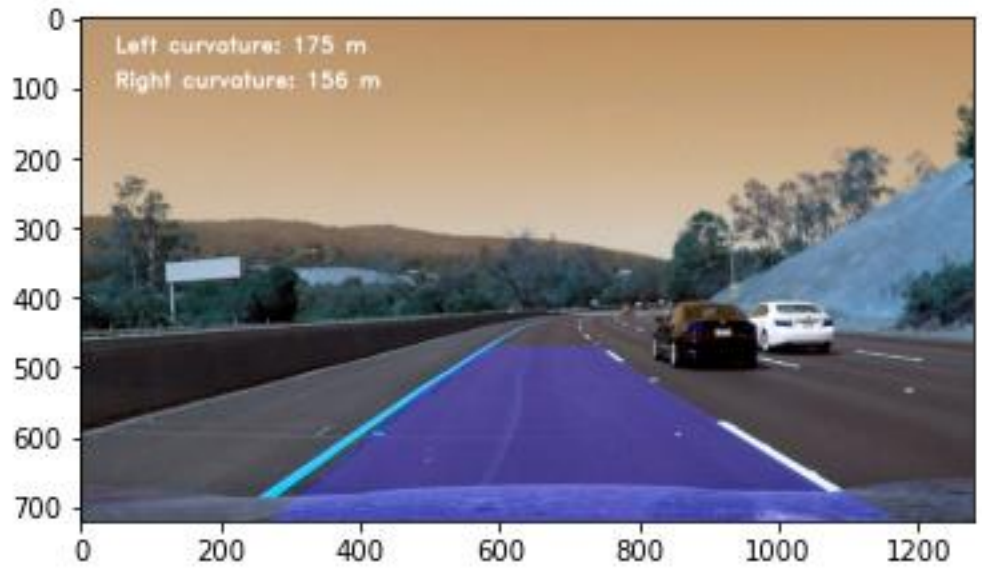
Warped image with fill poly:



Warped image to input size with fill poly:



Final Image



Link for output video:

https://drive.google.com/file/d/19ir8WMhl9TR8-uBay-YFlS10FwGCHrZG/view?usp=sharing

Problems Faced:

Question 1:

In question 1 when looping through the grid, I was reusing the code used in AR tag and since it was a square the loop did not have any issue. But for the given problem, it was a rectangular image so the divisor of the x and y length was used instead.

Question 2:

Initially, the left and right lines were separated by using positive and negative slope which was inefficient. Then the longest line was detected according to the sign when longest line and any line in the image is multiplied, the left and right lane were separated considering that the longest line lies in the solid lane.

Question 3:

For detecting the lanes, the logic used in question 2 was tried initially which did not render the required output. So, histogram of the threshold image was taken in the left half and right half of the image along x axis and the highest intensity of 1 on either half gave the position of left and right lanes and margin was added to it.

To get a better threshold image, I combined canny and threshold method which gave an efficient output.