ENPM673: Perception for Autonomous Robots
Project 2
Vivek Sood – 117504279

Output File Link : Here

Problem 1

The given problem asks us to improve the quality of the given input video. Improving the quality of a frame sequence can mean many things but in our case the frame sequence is a road during night video from a car dashcam and it is very clear that the features in the input video frame as seen in Fig.1 are not very detectable. Hence improving the quality in terms of the given input means that we need to make key features like lanes more detectable when using conventional detection pipelines.

Figure 1. Original video frame.

To achieve this we can improve the contrast of the image using histogram equalization. Histogram in our application is the number of pixels for each intensity. Mathematically a histogram in defined as

$$h(i) = \sum_{x}\sum_{y}[I(x,y) = i]$$

Where h is the histogram, I is the intensity at the position (x,y)

The histogram of the frame in Fig.1 can be seen in Fig.2. We can observe that the histogram has most of its pixels near zero intensity which implies that the image is underexposed. Histogram equalization basically redistributes the histogram to use the entire dynamic range.

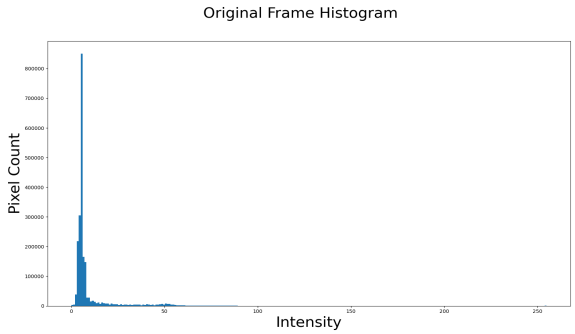

Figure 2. Histogram of blue channel of original video frame

To achieve this using cumulative distribution function (CDF) which calculates the fraction of pixels with an intensity equal to or less than a specific value. Mathematically CDF is defined as

$$C(i) = \frac{1}{N} \times \sum_{i,y}^{N} j <= i(h(j))$$

Where C is the CDF, N is the number of bins or intensities, h is the histogram.
I split the original image into Blue, Green and Red channels and then perform histogram equalization on all three individually and we can see a better contrast when compared to the original frame as seen in Fig.4. It can be observed that the lanes are more visible in the right side when compared to the left side.

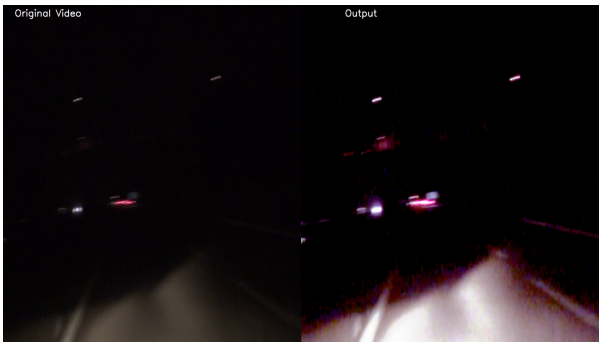

Figure 4. Final Output

Problem 2

The given problem asks us to detect lanes and the radius of curvature of the lanes and using that we need to then predict whether the car is turning or not. The section below explains my pipeline.

1. First the image is undistorted using the given inputs by using openCV's inbuilt function *cv2.undistort()*.

2. To make the lanes stand out from the background I improve the contrast using contrast limited adaptive histogram equalization (*CLAHE*). I apply this by first converting the image to LAB color space and apply *CLAHE* to the L channel of the image using the *cv2.createCLAHE* function of openCV.

3. To make the edges more prominent I sharpen the image using a special filter known as unsharpening filter kernel for which can be seen below

$$M = -\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

4. Then I do color filtering to isolate the lanes. For yellow color I use HSV space to create a mask and for white I use the standard BGR color space. These two masks are combined using the *OR* operator. The individual masks for yellow and white can be seen in Fig.5 and Fig,6. The combined mask can be seen in Fig.7.

Figure 5. Yellow Mask on 2nd data set

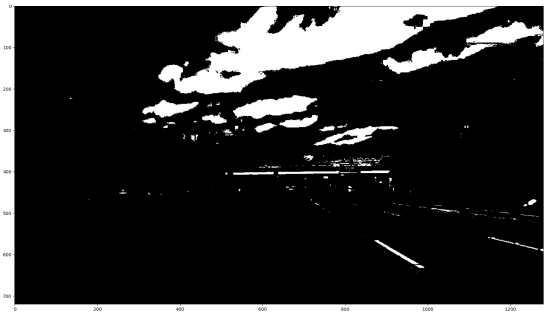

Figure 6. White mask on 2nd dataset

Figure 7. Combined Mask

5. Now I unwarp the image using *cv2.warpPerspective()* which uses homography to change the view from perspective to bird's eye which is essential to detect lanes and compute the radius of curvature. The points for homography are chosen in a way to remove unnecessary elements like the sky, etc from the view. The unwarped binary image is shown below in Fig.8.

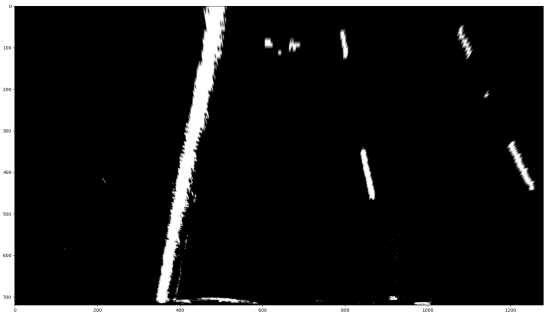

Figure 8. Unwarped binary image

6. I then apply a median filter with a 5x5 filter to remove salt and pepper noise. After generating we start the lane detection and polynomial fitting. First I divide the entire image into 16 horizontal slices as shown in Fig.9

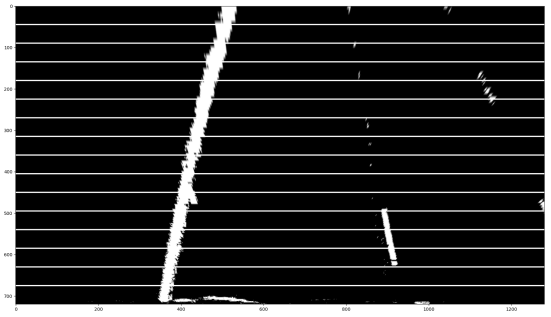

Figure 8. Sliced Image

7. Now in order to detect the lanes I compute the histogram of first the entire image and then the histogram of the first or in other words the bottom most slice.

8. Now I detect the 2 peaks in the histogram of the lowest frame. If maxima's are detected we start the search by creating a window with the otherwise I use the global maxima for the same procedure. Now I first split the slice into two parts which each contain the left and right lanes.

9. Now in each part of the slice I create a window centered around the detected maxima's with the width decided with a threshold and the height is the height of the slice.

10. Now the program iterates through each slice and detects the bottom edge center coordinates for the next slice by taking a weighted average of the points found inside the previously found window and using the x-coordinate from that. I use a weighted average algorithm to give more weight to points in the center.
To visualize the algorithm better I am using a colored version of the warped image in Fig.9 instead of the actual binary image that is being used by the program.

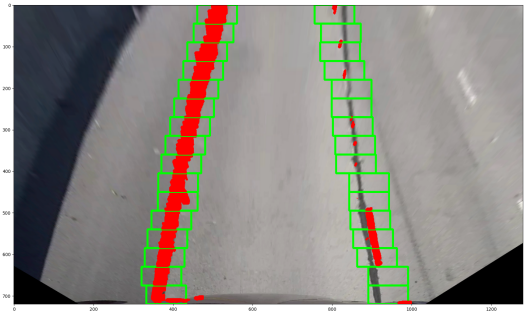

Figure 9. Slice Based algorithm visualized

11. The red points are the points of interest which are then used to fit a curve using *np.polyfit()*

12. If we implement the above mentioned algorithm directly we will get a result but lanes will not be proper for a lot of frames as *np.polyfit()* doesn't give the correct result every time as the points detected can have error due to nearby lanes or cars.
To combat this problem I devised a variance based algorithm. I calculated a few parameters.

a. Variance of the above calculated polynomial
b. Variance in distances of the two computed lanes.
c. Mean difference of distances between the bottom points for both lanes
d. Mean difference of distances between the top points for both lanes

Variance of both lanes is calculated to check how far the polynomial is from a straight line. The variance of distances between the two lanes is calculated to see how close or far are the new lanes from the global average distance. This helps eliminate lanes that are too close or far. The mean differences helps us to identify the skewness of the lanes.

13. If any of the above parameters move beyond the defined threshold I use a fallback mechanism wherein I use the last detected lanes.

If both lanes are satisfy the threshold then the overlay becomes green as seen in Fig.10

Figure 10. Both Lanes are good

In case any one of the lanes don't fit the above mentioned parameters the lane is corrected and the overlay becomes yellow as can be seen in Fig.11

Figure 11. Left Lane Corrected

If both lanes need correction the overlay becomes red as seen in Fig.12

Figure 12. Both lanes corrected

Hough Transform

As I have not used hough transform in my implementation I am explaining it here.

Hough Transform is a method used to detect lines in an image and works based on a voting scheme. A line can be mathematically represented as

$$y = mx + c$$

where m is the slope of the line and c is the y-intercept. The above line equation is true in the cartesian coordinate system. The Cartesian coordinate system is not the ideal system for hough transformation as the slope can become infinite when the line is vertical. Hence to counter this problem we use the polar coordinate system.

A line in polar coordinate system can be expressed as

$$x = d\,Cos\theta$$
$$y = d\,Sin\theta$$

$$d = xCos\theta + ySin\theta$$

where d is the perpendicular distance of the line from origin and θ is the angle with x-axis. Now A line in image space becomes a point in hough space (d,θ).

Basic line detection algorithm using Hough transformation.

1. *for each edge point (x,y) in the image*
2. *for* θ *varying from 0 to 180◦*
3. $d = xSin\theta + yCos\theta$
4. H(d,θ) += 1
5. max(H) is a line
6. Equation of the detected line : $d = xSin\theta + yCos\theta$

Algorithm Viability

The viability of my algorithm can be analyzed by analyzing the pros and cons of the algorithm.

Pros:

1. The algorithm can handle a lot of cases and scenarios as seen in the dataset2 video.

2. The variance detection and fallback algorithm can be essentially independent of the scenario.

Cons:

1. The color thresholding is highly dependent on the lighting conditions.

2. The window detection is susceptible to deviation if we have less points of interest.

According to the pros and cons I would conclude the algorithm in its current state is not viable for all use cases but further work can improve its viability.