ENPM673 - Perception for Autonomous Robots

Project - 2

Ninad Harishchandrakar - 118150819

ninadh@umd.edu

1 Histogram Equalization

1.1 Histogram Equalization

For each frame converted to Grayscale, a Frequency matrix of size 256 was created where the element at each index of the matrix was the number of pixels with the value of that index.

The Cumulative Sum of this matrix was found using np.cumsum() and this was divided with the Number of pixels in the frame to get the CDF matrix. Each element in this CDF was multiplied by 255 and these values were filled in the corresponding pixels to get the Histogram Equalization Image as shown in Figure 2.

Figure 1: Original Frame

Figure 2: Histogram Equalization

The Histograms of the Original Image is shown in Figure 3 and that of the Histogram Equalized Image is shown in Figure 4.

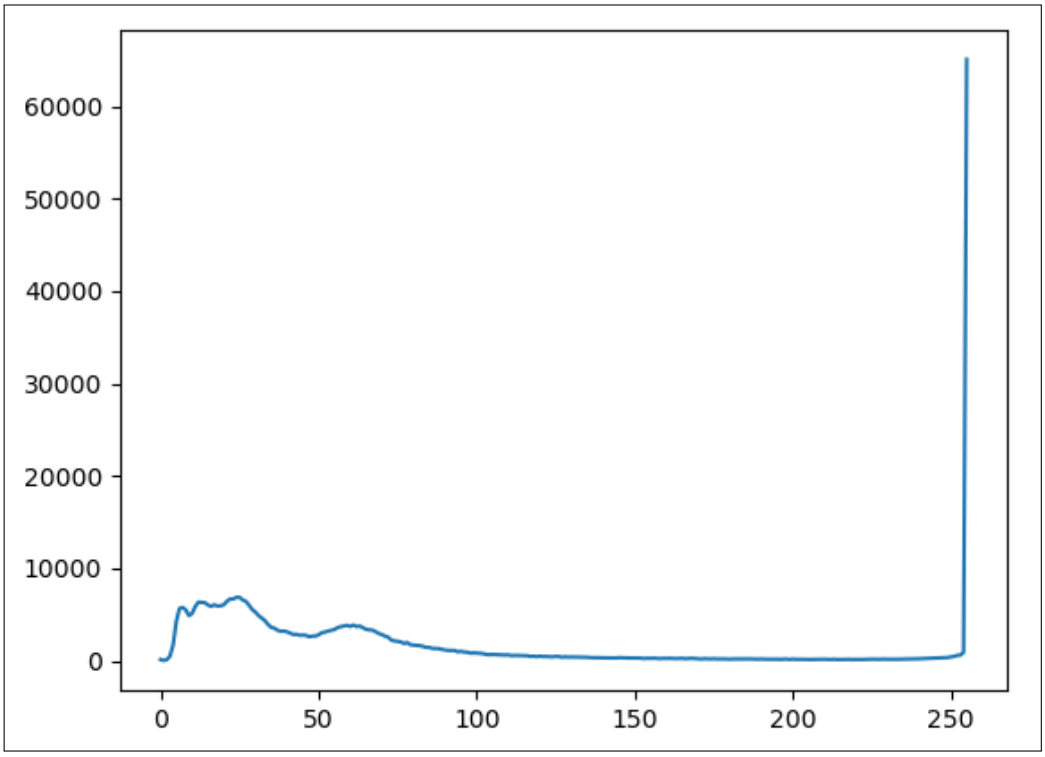

Figure 3: Histogram of Original Image

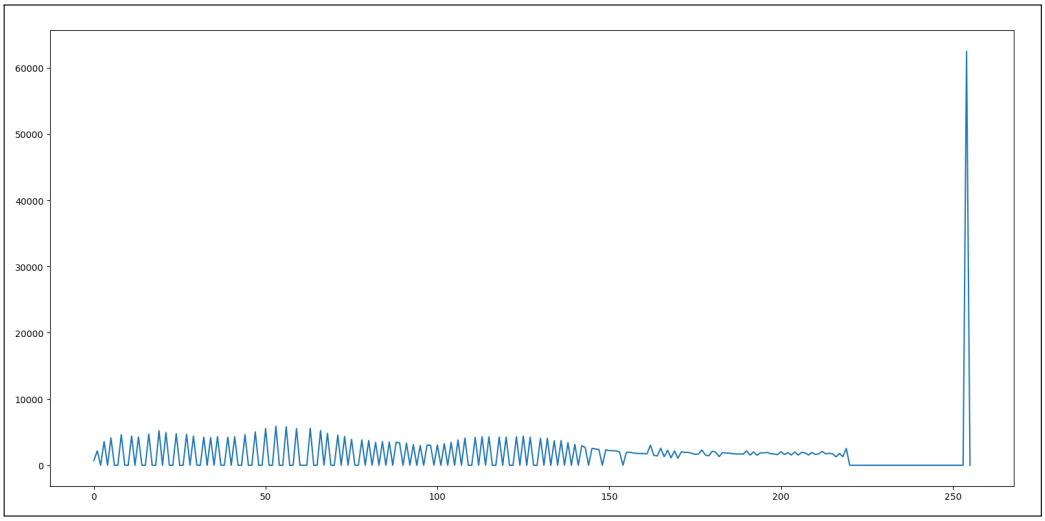

Figure 4: Histogram of Histogram Equalized Image

1.2 Contrast Limited Adaptive Histogram Equalization

For this, the frame was divided into 8x8 parts and the following process was done on each part:

Similar to Section 1.1, a Frequency Matrix was created for each pixel intensity. A limit of 40 pixel was set and all the pixels who had a intensity frequency greater than 40 were set to 40. The total number of excess pixels for all intensities was saved and equally divided between all the intensities.

With this new frequency matrix, normal Histogram Equalization was done as in Figure 1.1.

After this process was carried out on all 64 parts, they were combined again to form the entire image.

The output of Contrast Limited Adaptive Histogram Equalization is shown in Figure 5.

Figure 5: Contrast Limited Adaptive Histogram Equalization

The Histogram of Figure 5 image is shown in Figure 6.

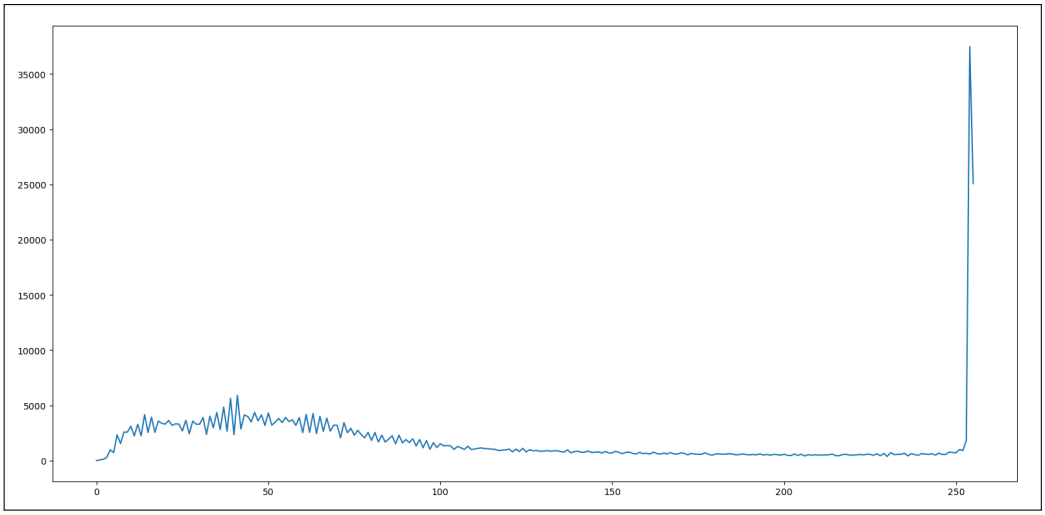

Figure 6: Histogram of Contrast Limited Adaptive Histogram Equalized Image

The video output for Problem1 of the project can be found here.

1.3 Comparison

Normal Histogram Equalization considers the global contrast of the image, and hence contrast is enhanced for the entire image. This results in un-

necessary contrast enhancement in various spots, which results in a loss of information for the image. On the other hand, since Adaptive Histogram Equalization works on smaller sections of the image, it does do unnecessary contrast enhancement and most of the information is retained. Contrast limiting further prevents unnecessary noise enhancement and a far better result is obtained.

2 Straight Lane Detection

For each frame, the trapezoidal section enclosing the two lanes was considered by applying a Bitwise AND operation with a mask and thresholding was done on this image as shown in in Figure 7.

Figure 7: Trapezoidal section isolated

Next, Lanes were detected using Hough Lines with the function cv2.HoughLinesP().

It was found that many of these lines overlapped over each other. To filter these out, the angles made by all lines with all other lines was checked and the ones which were less than 12 degrees were removed.

The lengths of all the remaining lines were found and the ones with length greater than 200 were deemed to be corresponding to the continuous lane and colored green. Whereas ones whose lengths were less than 200 were colored red as shown in Figure 8.

Figure 8: Colored Lanes

The video output for Problem2 of the project can be found here.

3 Predict Turn

The trapezoidal portion having the two lanes as in Problem2 was warped to a rectangular image using the cv2.findHomography() and cv2.warpPerspective() functions. This was then converted to grayscale and adaptive thresholding was done on it using cv2.createCLAHE(). Median blurring was done on this and it was then thresholded with cv2.threshold() as shown in Figure 9.

Figure 9: Binary Warped Image

A sliding window approach was used to detect lanes. For the left and right lanes, the column with the highest number of white pixels was selected using histogram and the column was divided into 10 parts.

Based on the center of the white pixels in one division, an ellipse was drawn with that center for all the divisions. Using the function numpy.polyfit(), 2 curves for left and right lanes were fit based on the centers of these ellipses. This was plotted onto the warped image as shown in Figure 10 using cv2.polylines().

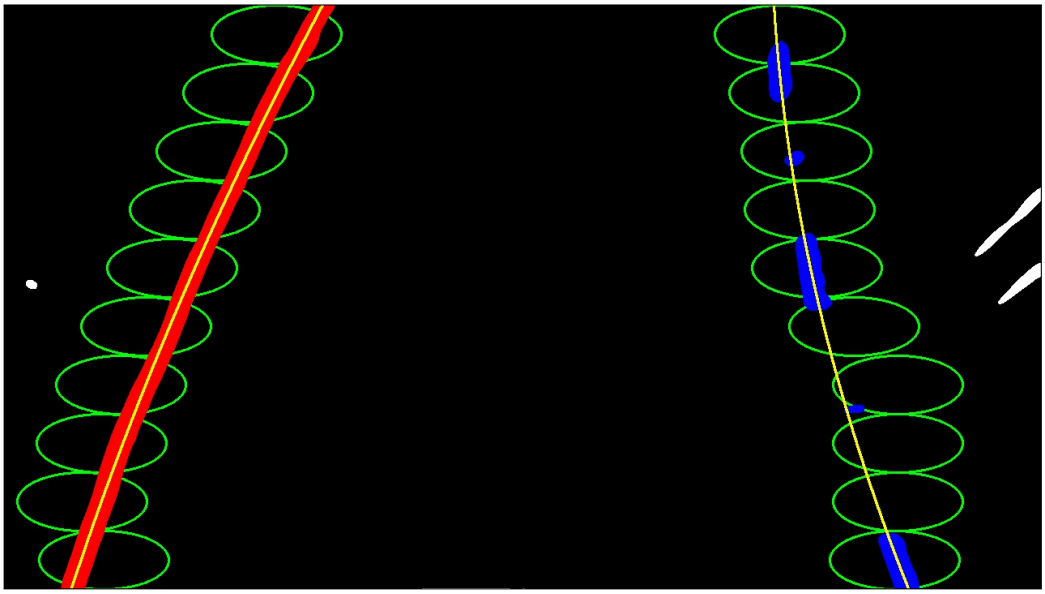

Figure 10: Lane Plots Highlighted

Next, the corresponding coordinates on the original image of the curves on the warped image were plotted by using Inverse Homography without the use of any inbuilt function. A weighted trapezium on the lane was also displayed to depict the lane as shown in Figure 11.

Figure 11: Highlighted Lanes on the Original Frame

The distances were converted from pixels to meters and the radii of curvatures of the left and right lanes was found. Based on the average curvature, decisions were made whether the vehicle needs to go left, straight or right.

Also, when the number of white pixels in the binary image exceeded a certain limit, it was considered that there was no lane found and the lane information from the previous frame was used.

All of this information was combined and displayed as shown in Figures 12 and 13. As seen in Figure 13, when there was no lane found, the ellipses were not found and displayed.

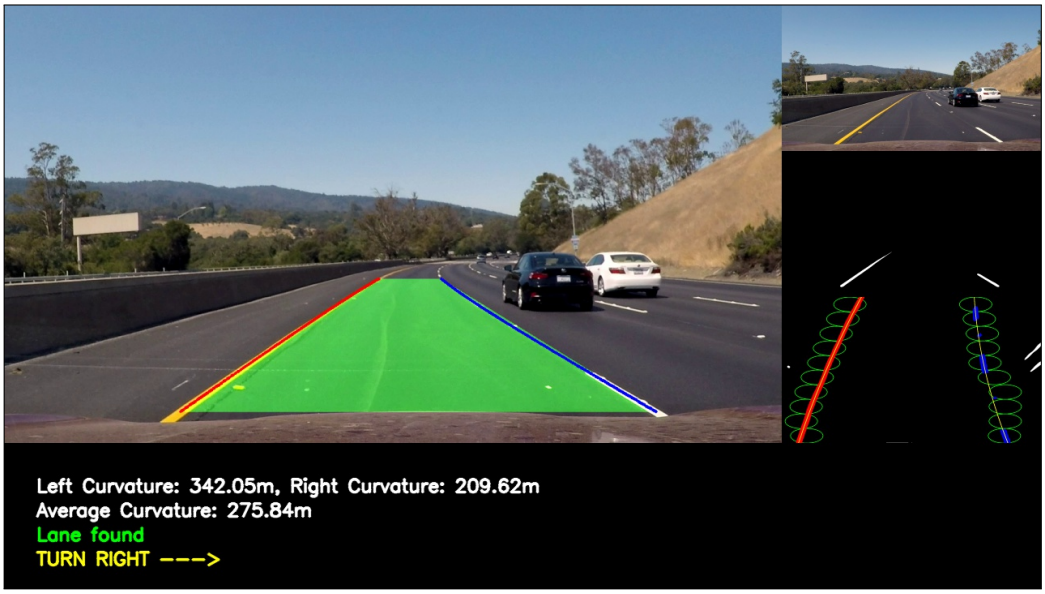

Figure 12: Output when Lane is Found

Figure 13: Output when Lane is not Found

The video output for Problem3 of the project can be found here.

4 Further Information

4.1 Homography

Homography is the transformation of points from one coordinate frame to another. With this, the perspective of an image can be changed from one view to another. This can be used to transform image coordinates to planar surface coordinates using Equation 1 as shown in Figure 14.

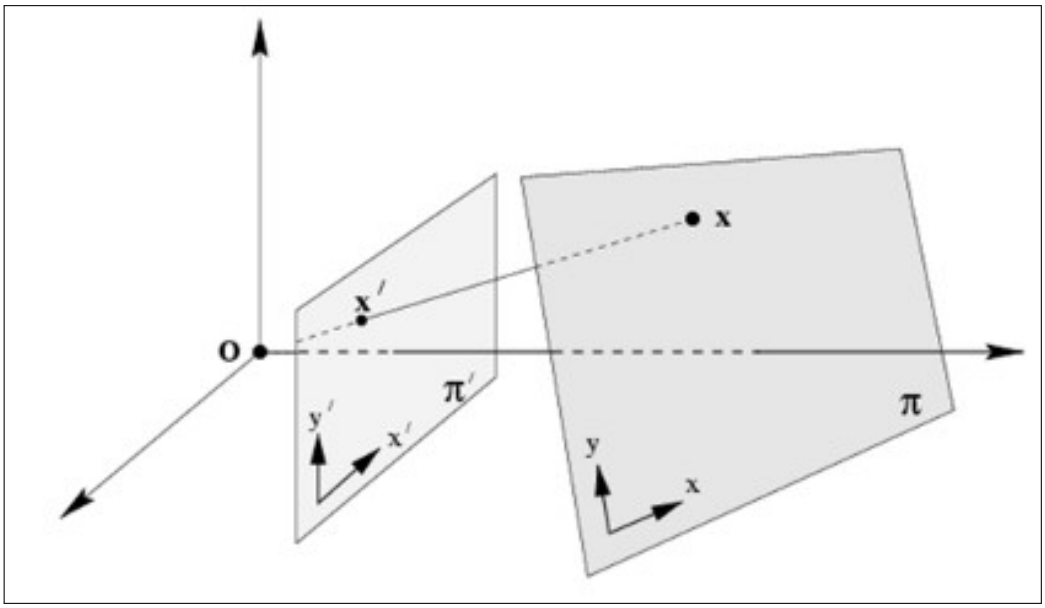

Figure 14: Homography in Images (Source: OpenCV Documentation)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

Homography has been used in Problem 3 to visualize lanes from the top view in order to find the radius of curvature.

4.2 Hough Lines

Hough lines are used to identify straight lines in a binary image.

For a given point (x_0, y_0) , the locus of all the lines that pass through that point is given by:

$$r = x_0 cos\theta + y_0 sin\theta$$

This locus is represented by the pair (r, θ).

This is represented by a sinusoidal wave on the r vs θ graph.

This is done for all points in the image and if there is a point where multiple sinusoidal waves intersect like in Figure 15, that means the points corresponding to those waves lie on the same line.

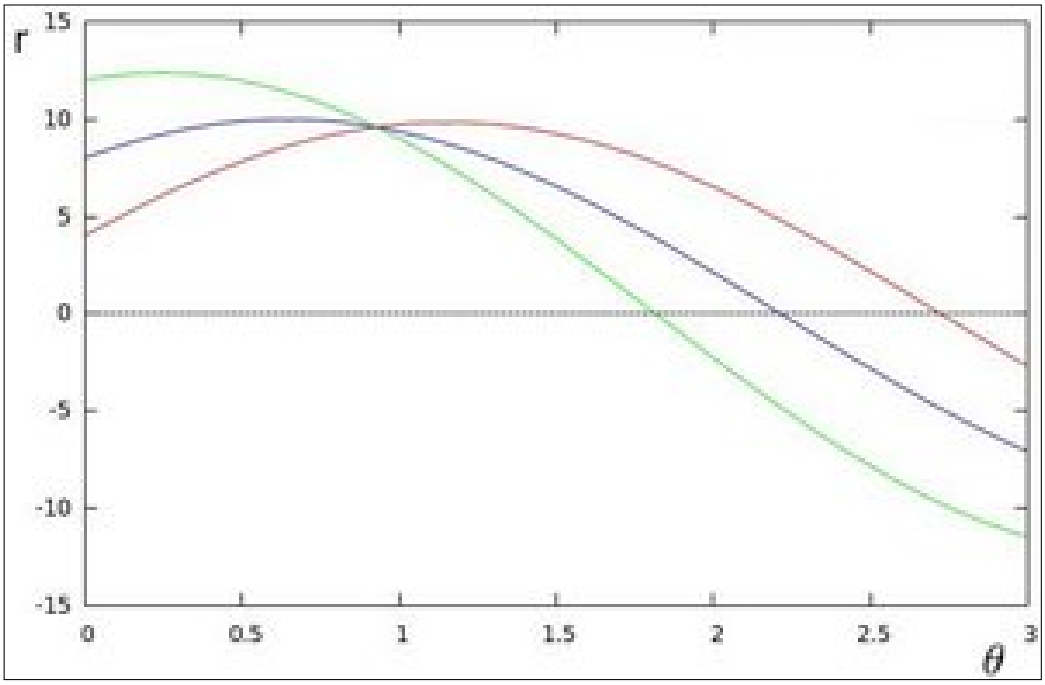

Figure 15: Hough Lines (Source: OpenCV Documentation)

4.3 Pipeline

The pipeline used for all the problems can be generalized in most of the situations unless the quality of the image is extremely bad. For instance, in problem 3, information of previous lanes were used in cases where lanes were not found. This worked well since for this input, lanes could be found starting with the first frame. This will not work in cases where lanes are not found in the first frame itself.