# ENPM673 – Perception for Autonomous Robots

Project 2

Pratik Acharya

117513615

April 4, 2022

# Problem 1: Histogram equalization

## Part A: Histogram Equalization

### Pipeline

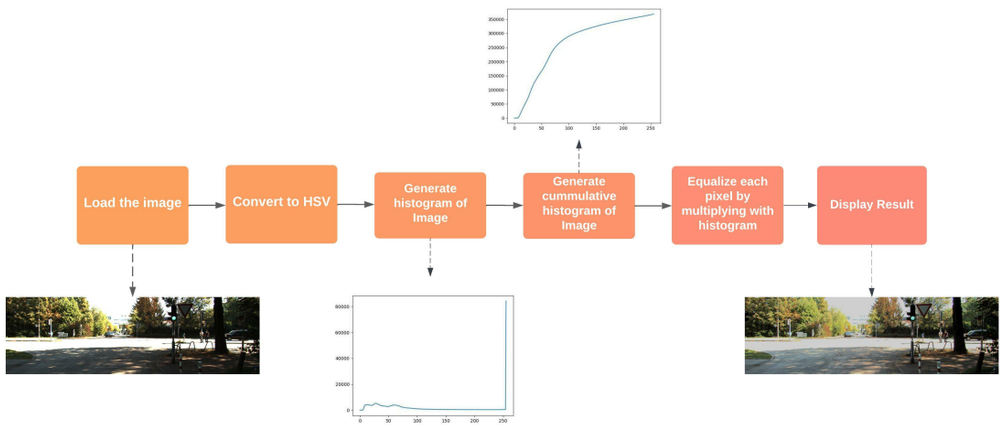


Figure 1: Pipeline

**Steps:**

- HSV Conversion: As histogram equalization cannot be perfome on an RGB image, the image is converted to HSV format. The V parameter from HSV corresponds to brightness and as contrast is dependent on brightness, only this channel is modified.
- Generate Histogram: Based on the value of V for each pixel, it's histogram is created which measures the total pixels for each value between 0 to 255.
- Generating Cumulative Histogram: Using the histogram, a cumulative histogram is created which is the sum of all the previous values.
- Equalization: The cumulative histogram is then used, where for each pixel, it's respective bin value is multiplied with 255, which distributes the histogram equally.
- Result: The result is displayed on the screen.

Figure 2: Original Image

Figure 3: Histogram Equalized Image

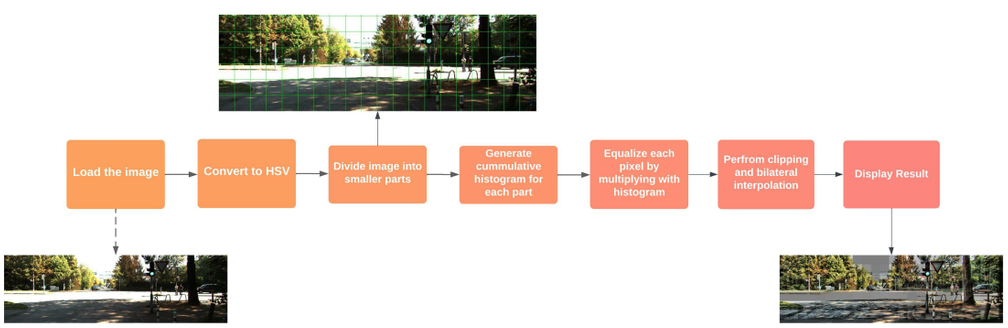


Figure 4: Pipeline

## Part B: Adaptive Histogram Equalization

**Pipeline**
**Steps:**

- HSV Conversion: As histogram equalization cannot be perform on an RGB image, the image is converted to HSV format. The V parameter from HSV corresponds to brightness and as contrast is dependent on brightness, only this channel is modified.
- Divide Image: The image is divided into smaller grids, after which histogram equalization will be performed on individual grids.
- Generate Histogram: Based on the value of V for each pixel, it's histogram is created which measures the total pixels for each value between 0 to 255.
- Generating Cumulative Histogram: Using the histogram, a cumulative histogram is created which is the sum of all the previous values.
- Equalization: The cumulative histogram is then used, where for each pixel, it's respective bin value is multiplied with 255, which distributes the histogram equally.
- Clipping: To ensure that noisy pixels don't dominate the histogram, the histogram is clipped at a threshold and the pixels are equally distributed at the bottom of the histogram.
- Bilateral Filter: As the image after adaptive histogram is patchy, bilateral filter is used for smoothing the image.
- Result: The result is displayed on the screen.

Figure 5: Original Image

Figure 6: Adaptive Histogram Equalized Image

## 0.1 Comparison

Even after multiple tries with different parameters and changing code for performing adaptive histogram, the results obtained were undesirable. Due to this comparison between adaptive histogram and normal histogram equalization cannot be made. Logically, as the image has some highly lit areas, and some dark areas, adaptive histogram would be a better choice for such images.

# Problem 2: Straight Lane Detection

### Pipeline

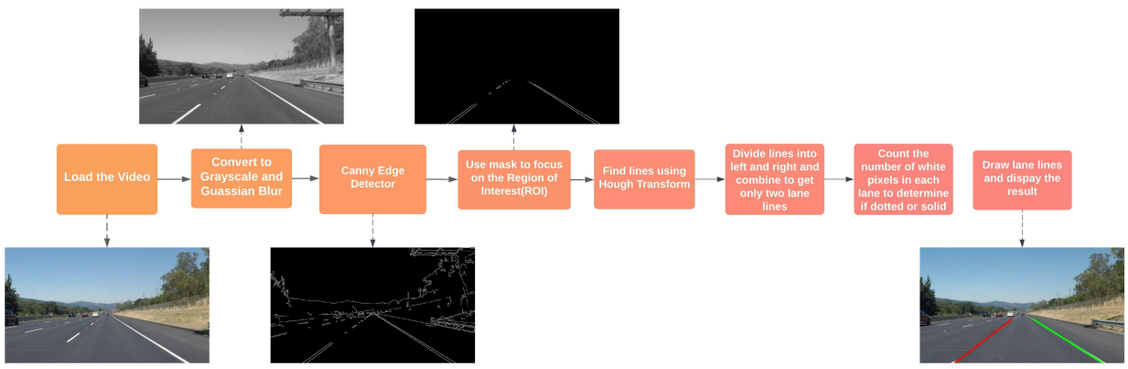


Figure 7: Pipeline

### Steps:

Figure 8: Original Frame

- Color Conversion and Blur: Images were converted into grayscale to be able to detect the edges. Gaussian blur is applied to smoothen the image and reduce noise.

Figure 9: Gray and Blurred image

- Canny Edge Detector: Canny edge detector is applied to the image with threshold of (50,150).
- ROI Mask: A traingular mask is created to focus only on the Region of Interest(ROI) which eliminated the background. As the camera is always fixed with respect to the road, the ROI mask will always include the lanes that are to be detected.

Figure 10: Canny Edge Detector

Figure 11: ROI Mask

- Hough Lines: To find the lane lines in the frame, hough transform is used, which will find all the lines from the detected lane points. The explanation for Hough Transform is given at the end of the document. Hough transform will provide multiple lines. These lines need to be combined into left and right lane lines. Based on slopes, the lines are divided into two groups, for the left lane and right lane. Then, individually in these groups, the slopes and intercepts are averaged to find only one line which will be the optimal solution for the lanes.

- Classifying Lanes: To classify the detected lane lines, first a rectangular mask is created for both the lines, which will enclose the lane lines. This mask is applied for the canny edge detector output. Now, the white pixels are counted for both the left and right lanes, and the number of white pixels are compared for both, and the one with higher number of white pixels is classified as the solid line and the other is classified as dotted line.

- Drawing Lines: Depending on the output of the classified lane lines, the solid line is colored green, and dotted line is colored red, and the lines are drawn on the original frame.

## Generalization

Based on this pipeline, as there is no contrast adjustment and the video quality may be affected by shadows and other artifacts, the lane detection maybe be affected. If lane lines are not detected for some frames, the lines detected from the older frames are used, which tackles this problem to some extent.

For lane classification, as the lines will only be classified based on comparison between

Figure 12: Lane Detection Result

the two detected lanes, hence, this will not work for lanes which are both white or dotted. This can be avoided by classifying the lines based on a threshold of number of white pixels, but comparison was providing better result for the given dataset, hence it was utilized.

# Problem 3: Predict Turn

**Pipeline:**

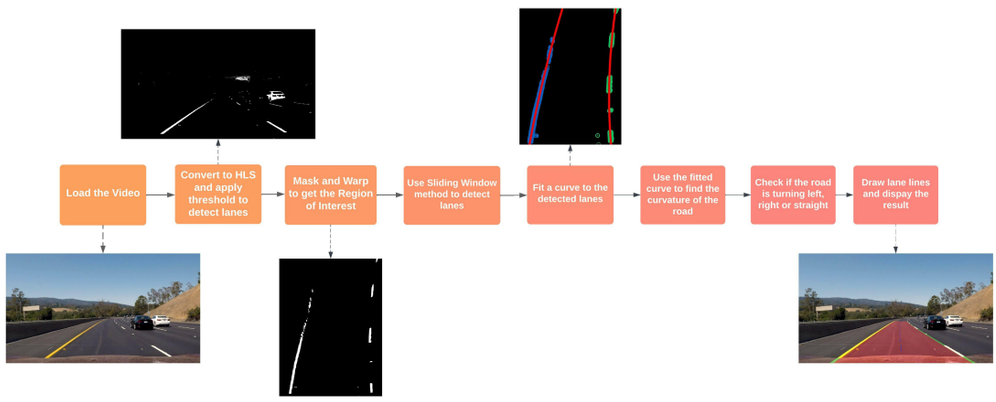


Figure 13: Pipeline

**Steps**

Figure 14: Original Image

- Convert to HLS: Due to the uneven lighting conditions in the dataset provided, lane detection is difficult in BGR colorspace, hence the colorspace is converted to HLS to isolate the white and yellow lane lines.

Figure 15: Thresholding in HLS colorspace

- Warping: As the end goal of the algorithm is to find the curvature of the road ahead, the road image needs to be warped, as per the mask, in the shape of a trapezoid to

only highlight the road and the lanes in front of the car, and this region is then warped to change the shape from trapezoid to a rectangle.

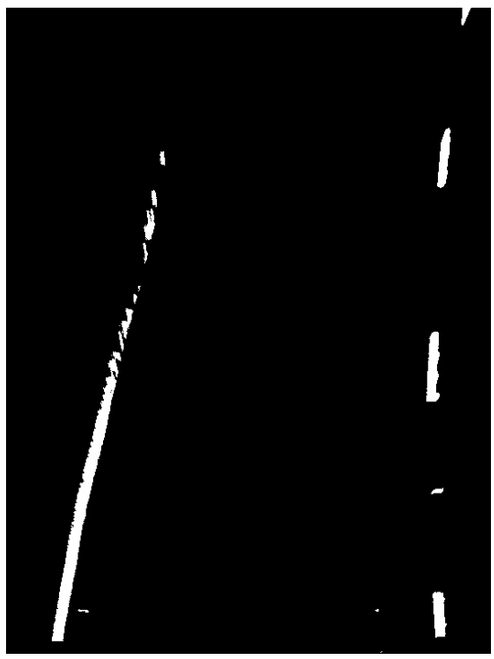

Figure 16: Warping the ROI

- Sliding Window: To detect the lane lines, first the image is divided into left and right halves. Histogram is calculated which includes the number of white pixels for each column. The column with the highest white pixels is detected. A rectangular window is created, which will slide over the image from bottom to top. For the first position of the sliding window, the column with the highest white pixels is chosen as the mid point. After detecting all the white pixels, the next window is placed above the previous one, with the center decided by the mean 'y' position of the previously detected pixels. Hence, the sliding window moves left or right as it moves up the image. Position of each detected white pixel is added to a list, which will be used to fit a curved line. This is done for both the halves of the image, and the positions are stored as left and right lane points.

- Curve Fitting: The detected white pixels for left and right lanes are used to fit a curve using the 'polyfit' method of numpy.

- Finding Curvature: The coefficients of the fitted curves are used to calculate the curvature. This curvature is individually for left and right lanes. The average curvature is then calculated from the left and right curvature.

- Check for turn: To check for the turn, the mid point of both curves is found at the halfway of the image vertically. This is compared with the mid point of the car, which is the mid point of the ROI. 'Y' coordinated of both these points are compared to find if the road ahead turns left, right or is straight.

- Drawing Lanes: After detection of turn, the fitted polynomial is projected onto the original image to show the detected lanes.

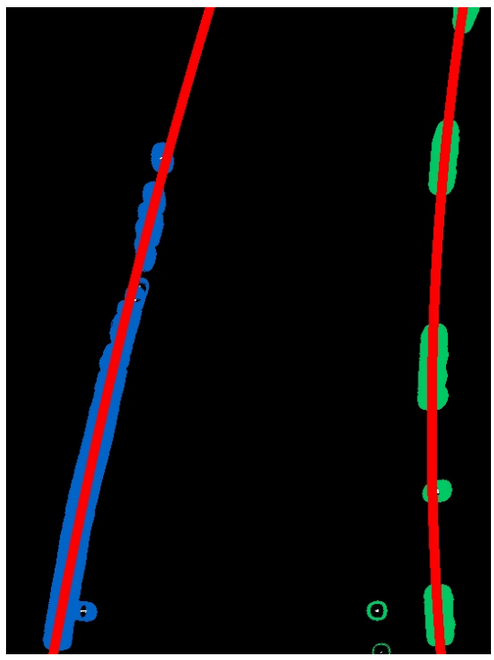

Figure 17: Fitted curve


Figure 18: Final Result

## Generalization

As HLS space and sliding window is used to detect the lanes, this pipeline is highly generalized to various lighting conditions.

## Homography

Homography is used to transform the view of an image from one perspective to another. In the third problem. as curve fitting was to be done on the lanes, the image required would be of a top view of the road. As the top view is not available, it can be created by using homography to transform the merging lane lines into a top view where the lines are almost parallel. This enabled the calculation of curvature of the road.

## HoughLines

Hough tranform is a method that is used to convert the coordinate system from cartesian to polar coordinates. A line in cartesian space, can be represented by two parameters, that same line can also be represented by two parameters in the polar coordinates. Now these coordinates are unique to each line, but if a point is present in cartesian coordinate, it has infinite kines passing through it, and each line can then be represented by two polar coordinates. These polar coordinate, when plotted in a graph with the coordinates as the axis, they form a sinusoidal way. This can be done for multiple points on a cartesian plane, with each point on the sinusoidal wave representing a line. The intersections of these multiple sinusoidal waves will be the point which represents the line that passes through all the points that are represented by the intersecting sinusoids.

Finding these intersections will give the line that passes through all the points. This property is used to find the lines passing through multiple points in Computer Vision.

For the second problem, Hough Lines is used to find the lines that pass through all the detected lane points.

## Generalization

Explanation for generalization is included in the respective problem section.

## Results

The results of the problems can be found in the google drive : ENPM673 Project 2 Videos

## Code

The code is included in the zip file submitted on ELMS and it can also be found at the following GitHub repository : Lane Detection and Curvature Calculation

# References

1. http://amroamroamro.github.io/mexopencv/opencv/clahe_demo_gui.html
2. https://www.hackster.io/kemfic/simple-lane-detection-c3db2f
3. https://www.hackster.io/kemfic/curved-lane-detection-34f771