# ENPM 673 - Perception for Autonomous Robots

## Image Processing, Lane detection and Turn Prediction for Self Driving vehicles

*Submitted towards completion of Project-2*

Abhishek Banerjee
Arjun Gupta
Vishnuu AD

March 2020

## 1 Introduction

The purpose of this project is to perform a lane detection task for an autonomous vehicle along with improving the image quality for a scenario under low light conditions.
In the first part we use techniques to enhance a night-time video to enable the perception subsystem to detect relevant features in the environment around it.
In the second part, we implement a lane-cum-curvature detection pipeline, which is a precursor for 'Lane Departure Warning System'. We determine the lane from the camera image, in essence, it's boundary and then determine it's heading direction based on it's curvature. In this report we detail the concepts, pipeline and implementation for our project.

## 2 Image Enhancement

The data/video provided for image enhancement is a video recorded during night. Since the data is recorded at night most dominated pixel values lie in the lower range which can be evidently seen in the histogram shown in figure 1. Accumulation of pixel intensities around a certain value results in bad contrast in the image. Thus, in order to enhance the contrast we use histogram equalization to evenly spread the intensities and thus enhance global contrast. It is a fairly straightforward method which enhances and uniformly spread the intensities resulting in better contrast.

For histogram equalization we first calculate the cumulative distribution of each pixel intensity and then normalize it by dividing the histogram values by maximum possible value (total number of pixels) and finally multiply by 255 to de-normalize the intensity values. One of the disadvantage of this method is that it is indiscriminate i.e. it can increase the contrast of background noise and reduces the usable signal. In order to overcome this, we threshold the image by setting the threshold values for the noise. All the values below the threshold value are set to 0 and above it are kept as it is. Figure 2 shows the histogram after equalization. We have plotted the histogram from a specific intensity value for better visualization and understanding of the histogram.

The results of the image enhancements can be found here

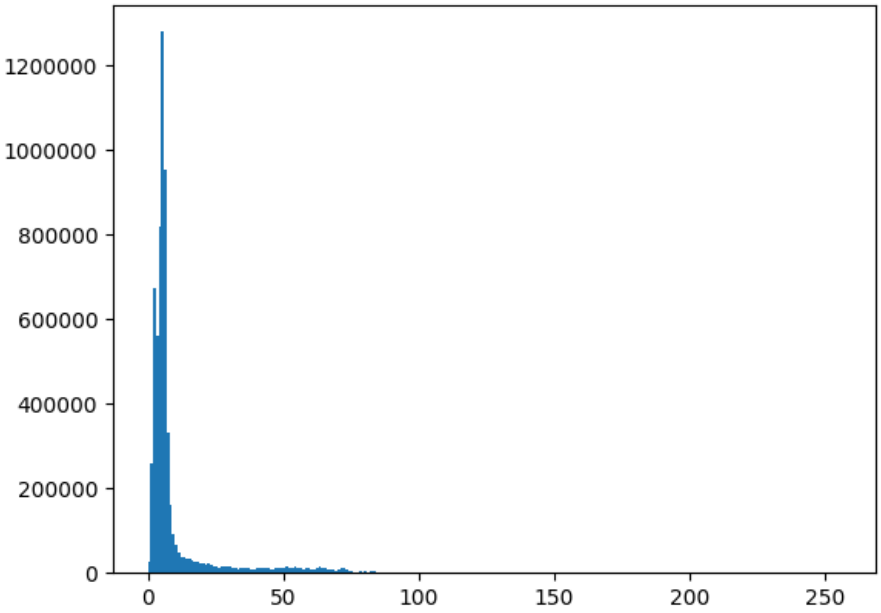


Figure 1: Histogram of image before equalization

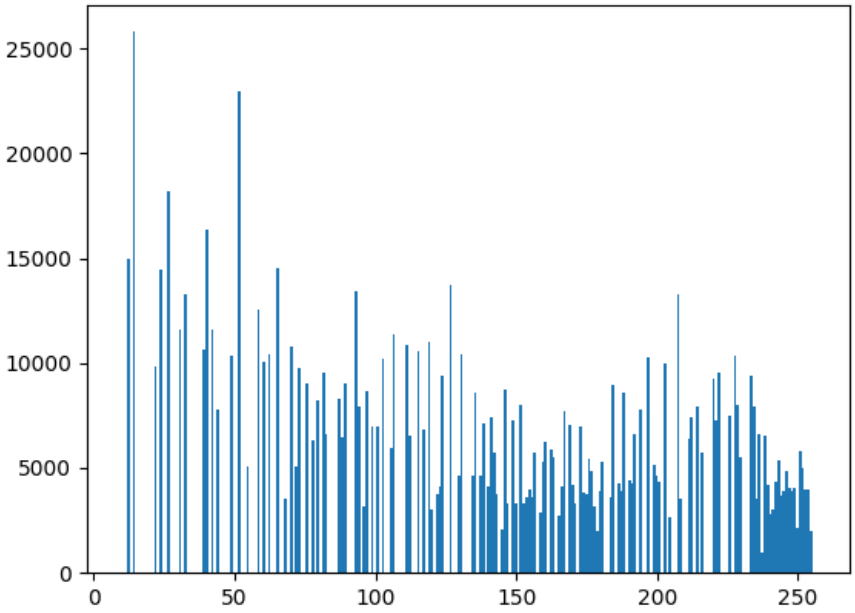


Figure 2: Histogram of image after equalization

# 3 Lane Detections

The problem statement has 2 data sets where road lanes need to be detected. The dataset found here has two different kind of challenges.

The **1st Data set** has a white continuous lane boundary on the right side of the image and discontinuous lane boundary on left side of the image. A sample image is shown below.**INSERT IMAGE**. Looking at the image, we decided to implement a simple image thresholding followed by Hough Lines operation on the bird's eye view of the image. This seems to work for most cases except in the shaded area. For the shaded area, we additionally used other features of the lanes such as high gradient inherently present at the lanes edges. We use a different threshold along with Canny edge detector, then increase the gamma followed by Hough line transform to detect the lanes. We merge the outputs of the two pipelines and perform edge detection.

The **2nd Data set** contains a video with a continuous lane boundary on the left side of the image(yellow coloured) and discontinuous lane boundary on the right side of the image. The challenge in this video is that the discontinuous lane boundary are covered with dark areas as well. **INSERT IMAGE** We also see through the video that a third line(occurring due to road reconstruction) appears starting from the left continuous boundary making it harder to detect the true lanes. Our pipeline uses a color threshold found from the plotting a novel function that highlights the yellow region(explained in detail below). We then apply canny edge detection on the thresholded image and then apply the Hough line transform to find the edges and points corresponding to the lane. In this process we also pick up lines from the false positive: the arbitrary lane appearing at the center. Given the fact that lanes exist only at a certain part of the image (fixed by limiting the points being used to see the top view of the road.), we filter away the lines which don't appear in this specific region.

Several common concepts are involved in solving the lane detection problem and an overview is given below here.

## 3.1 Homography

Homography can be defined as the relation between the two images of the same planar surface. The relation is derived from the projection matrix itself by assuming the Z=0 and thus eliminating the third column and resulting into 3×3 matrix. Since the scale of the projection is arbitrary we divide the whole matrix by the last element. The resultant matrix has only 8 unknowns we need 8 equation to solve or it. Each point gives two equations and thus we need only 4 matching points between two images. The final equations look like:

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

The above equation need to be solved for all $m_i's$ and it can be done by converting the above equation to the form AX=0 and use SVD of A to solve for X. For this project we use homography to obtain the rectified image of the lanes. We do so by manually selecting the 4 points on two parallel lanes and then selecting the corners of the rectified frame as the destination points. We further use the inverse of this homography to project back the detected lanes on the original image.

### 3.2 Hough Lines

Hough lines is a image processing technique that utilizes a voting procedure to identify a particular instance in an object (line for Hough lines). The method uses a parameter space to locate the local maximas and thus estimate the lines. Algorithm for hough lines is:

**Algorithm 1** Hough Lines

```
Initialize H[d, θ]=0
for all (x,y) in I[x,y]: do
    for all θ in (0, 180): do
        d = xcos θ + ysin θ
        H[d, θ] += 1
    end for
end for
Find the values of (d, θ) where H[d, θ] is above a threshold value.
return the line
```

Due to the nature of the dataset we do not feed raw images for hough lines in order to avoid any false positives. We pre process the image (which we discuss in detail in the later section) and then use hough lines on the processed image for better estimation. In Algorithm 1 the image I(x,y) is the processed image.

### 3.3 Refining Lane Detection

The idea behind the lane estimation, is to leverage the points which were extracted in the previous steps and generate a boundary for the detected lane.

The points that are detected, are split into two lists corresponding to the left and right lane boundaries. This is done by using the image center as the reference point, and designating points left of it as being on the left and right otherwise.

To determine a polynomial that fits the points in a single lane, we use the least square errors metric. The method minimizes the sum of square errors for a set of points fit onto a polynomial of some degree $D$. In python this is done using the the $polyfit$ function available in the *numpy* library. The workspace is discretized into a number of points, which in our case is the number of pixels in the vertical direction. The new polynomial is drawn over this entire range. This step is carried out for both the lanes.

The next step is to lay a mesh over the detected lane. For this step we use the $np.polyfill$ function in python. The function forms a polygon with the given vertices. In our case, we feed the terminal points along with the center point of the determined polynomial curve, in a cyclic manner. This polygon is filled in *Green*, and overlayed onto the image using the $weightedAdd()$ function in OpenCV. This function creates a weighted sum of the warped image and the polygon that was defined earlier.

### 3.4 Turn Prediction

For the purpose of turn prediction, we evaluate the curvature of the robot. We determine the the curvature of the right lane (which in this case is the one with white lane markers), by using the

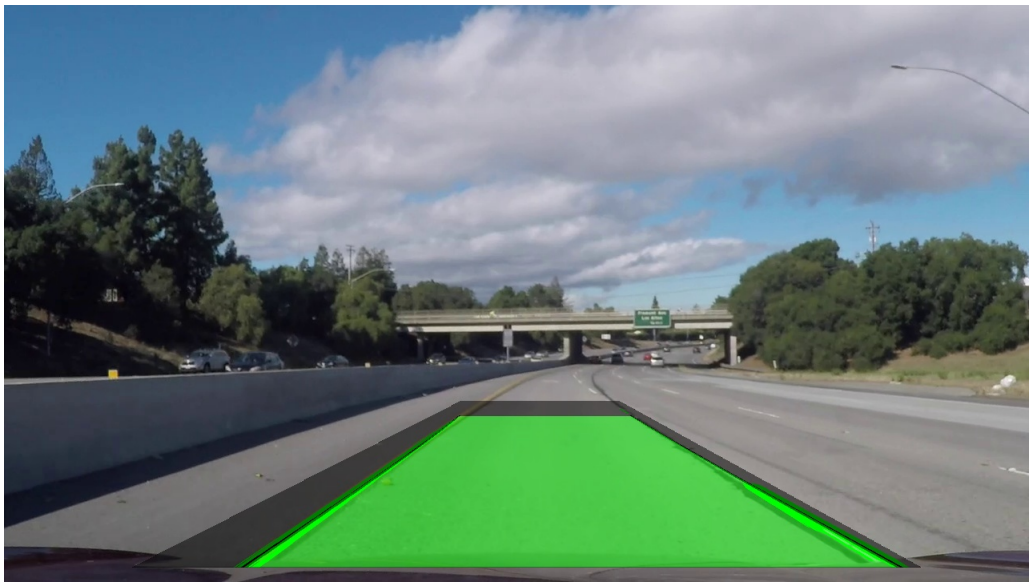

Figure 3: Lane Boundary Prediction and Mesh Generation : Challenge Video

Figure 4: Lane Boundary Prediction and Mesh Generation : Dataset 1

formula:

$$R = \frac{(1 + y'(x)^2)^{\frac{3}{2}}}{y''(x)}$$

For curvatures which are more than zero, this corresponds to right turns, while those which are negative correspond to left turns. In reality, we never get a perfect zero curvature for straight lines, hence we set thresholds for the absolute value of the curvature to be below 3000, in which case, it corresponds to a straight line. For more than 3000, this refers to a right turn and a left otherwise. We display the current status of the vehicle by overlaying the turn command on the screen.

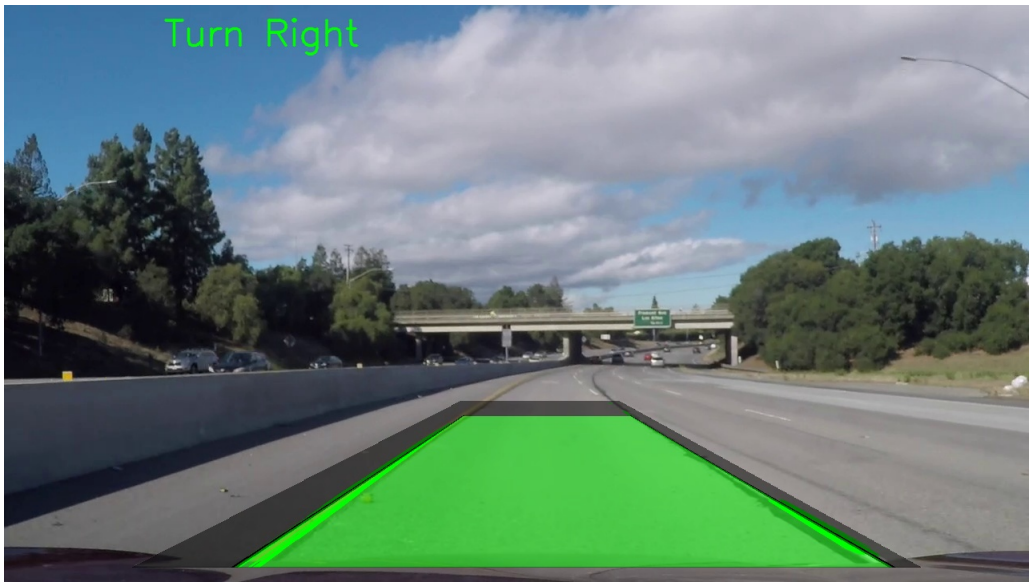


Figure 5: Turn Prediction : Challenge Video

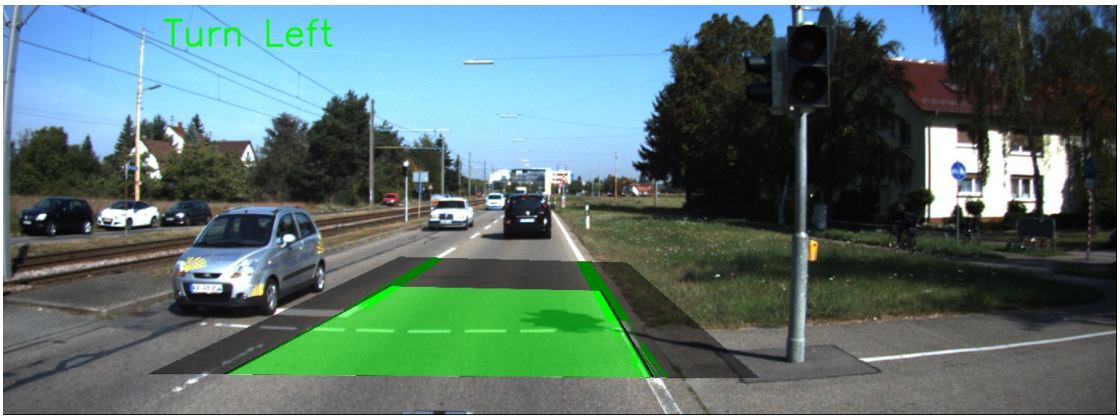


Figure 6: Turn Prediction : Dataset 1

# 4 Discussion

**Data Set 1**: The following is the image pipeline we've followed.

- **Homography & Warp Perspective**: We manually select 4 points on the input image : ([526, 322], [767, 328],[354, 428],[866, 434]) and transform it to points ([200, 62], [1292, 62], [200, 462], [1292, 462]). This gives us a rectified/ birds eye view image of the road as shown below in figure 7:

- **Image thresholding**: We then threshold the image to extract only the lane candidates. The threshold is done between ([220, 255]). Result of thresholding for an image in non shadow region is shown below in figure 8.

- **Gaussian Blur**: We apply gaussian blur to make sure the abnormal jagged edges are not detected as edges. The result is shown in figure 9 and 10.

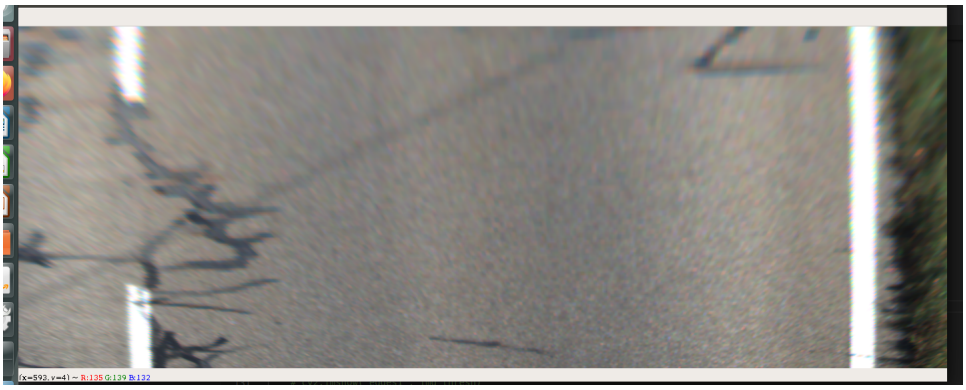

Figure 7: Rectified image

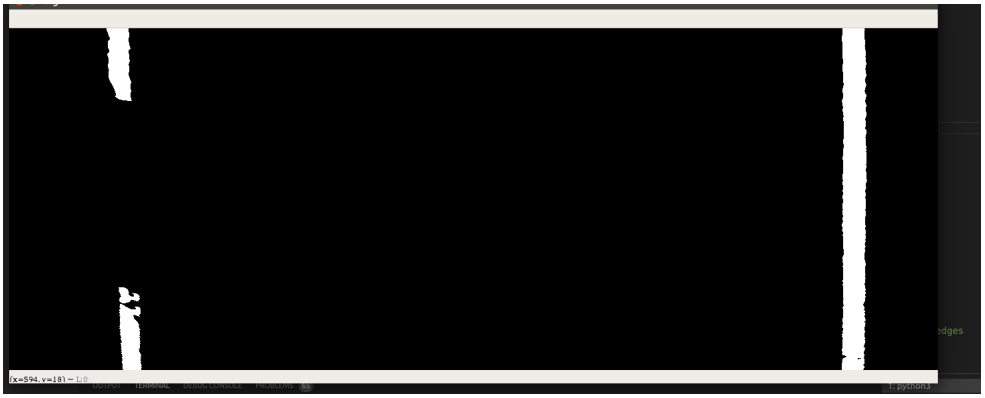

Figure 8: Image in no shadow region thresholded


Figure 9: Detection result after using gaussian blur

- **Detect lane in dark or highly bright regions**: This step is present to identify if our current image is in the shaded region. This is done by simply checking if the median of image intensity is below a certain value. If the image s median value is below a threshold, we perform additional separate operations mentioned below.
  - **Image thresholding**: We perform image thresholding with a different range of threshold as the image is now dimmer than before. Output of image before and after image


Figure 10: Detection result without using gaussian blur

thresholding is shown below in figure 11 and 12.


Figure 11: Image before thresholding

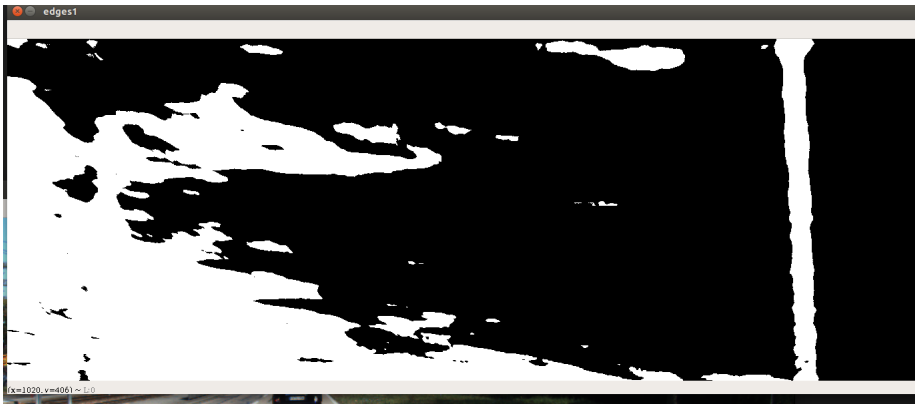


Figure 12: Image after thresholding

– **Canny edge detection**: We perform canny edge detection on the above thresholded image to detect the lanes. Refer figure 13 for the sample output.
– **Gaussian Blur**: We perform gaussian blurring on this image to make sure edges point-


Figure 13: Canny Edge Detection output

ing in random directions are not the only ones that would show up as line in the hough line transform.

– **Combing edges**: We then combine the edges of both the pipelines to have features of both the right lane and the left lane. The left lane is show as a patch from the earlier operation and the edges shown on the line are from the additional special operations. The output is shown below in figure 14.


Figure 14: Image after combining edges

– **Adjust Gamma**: We adjust the gamma in the image for better hough line transforms. The output is shown below in figure 15.

- **Hough Line Transform** The images with edges and required features is then passed into the cv2.HoughLinesP() which returns sets of points belonging to lines in the image. The drawn points can be seen in the figure 16. Taking all such points we then use the polyfit function to get the best quadratic that fits all these points.

- **Slope filtering** We only consider lines that have slope of greater than value 2. This rules out any arbitrary lines generated in the frame. Two images are shown below. One showing lane detected at a normal scene and the second showing lane detected at the shady area.

  The results of the above tests can be found here


Figure 15: Image after gamma enhancement

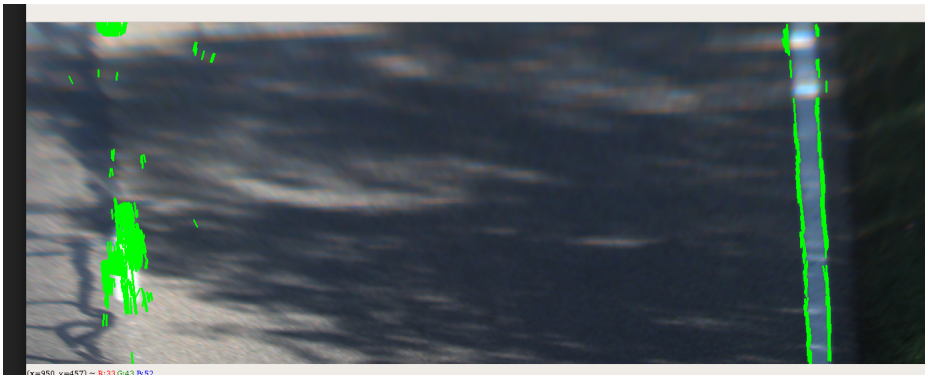

Figure 16: Hough Lines Output of the image in the shaded region

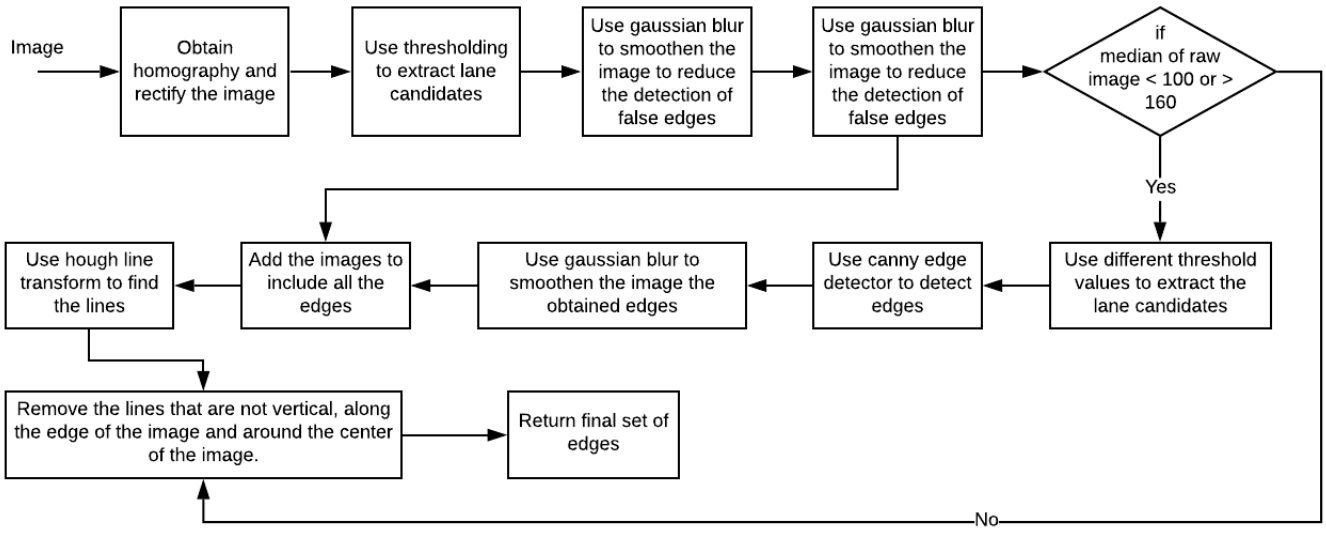


Figure 17: Pipeline for dataset 1

**Data Set 2:** The following is the image pipeline that's being followed.

- **Homography** We manually select 4 points on the input image :([407, 615], [886, 615],[312, 677],[962, 677] ) and transform it to points ([150, 630], [1080, 630], [150, 701], [1080, 701]). This gives us a rectified/ birds eye view image of the road.
- **Identify colour threshold**: To identify the exact color range of yellow we plot the following 2 graphs. The first graph has the 3d plot of intensity value of blue channel over the entire image. The second graph has the 3d plot of the function $y[i,j] = \frac{R[i,j] - G[i,j]}{(R[i,j] - G[i,j]) + 1}$ Where $R[i,j]$ and $G[i,j]$ are the red and green channel intensities at $(i,j)$. The plots are shown below. As expected , there is a sharp dip in the blue channel plot at the location of the lane. At the same locations, we see that the 2nd plot peaks for the custom graph generated. Such a peak indicated the color yellow. As yellow has almost equal values of R and G channel while having very low B channel values.

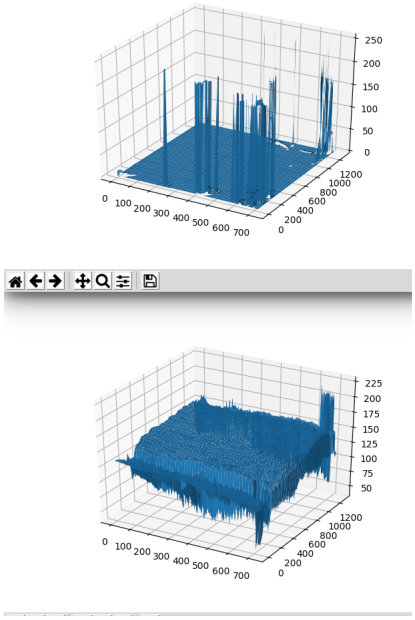


Figure 18: 3D histogram that are used for colour thresholding. Top image is the histogram for the values obtained from the formula discussed above and the bottom histogram is for the blue channel.

- **Color Thresholding**: We get the color thresholding region from the above method where we identify the B,G,R channel values for the lane. We find that the lower bound on the color was found to be (120, 180, 200) and the upper bound on color was found to be (200, 250, 250).
- **Canny edge detector**: We apply a canny edge detector on the colour thresholded image to give us the edges of the lanes. The output of such an image is seen below.
- **Gaussian Blur**: We then apply a gaussian blur on the edges to make sure the random direction edges don't contribute towards arbitrarily formed lines.

- **Entering/Exiting dark regions**: We noticed that while entering(too dim) and exiting(too bright) the dark regions (such as those below the bridge), the color thresholding required seems to change.
  - **Sharpen image**: When entering below the bridge, we find the need to sharpen the image to enhance detection of the edges using cv2.addWeighted() function.
  - **Color thresholding**: We perform a different thresholding for both the cases. The optimum color thresholding for the frames in the dark region was found to be lower bound: (140, 150, 135) and upper bound: (200, 250, 220). The optimum color thresholding for the frames while exiting the dark region was found to be lower bound : (120, 180, 200) and upper bound : (200, 250, 250).
  - **Canny edge detector**: We perform the Canny edge detection on the above image to extract the edges in the two regions.
  - **Gaussian blur**: A gaussian blur is applied so that random direction edges don't contribute to hough lines.
- **Hough Line transform**: The images with edges and required features is then passed into the cv2.HoughLinesP() which returns sets of points belonging to lines in the image. Taking all such points we then use the polyfit function to get the best quadratic that fits all these points.
- **Slopes Filtering**: We only consider lines that have slope of greater than value 2. This rules out any arbitrary lines generated in the frame. Two images are shown below. One showing lane detected at a normal scene and the second showing lane detected at the shady area.

  The results of the above tests can be found here

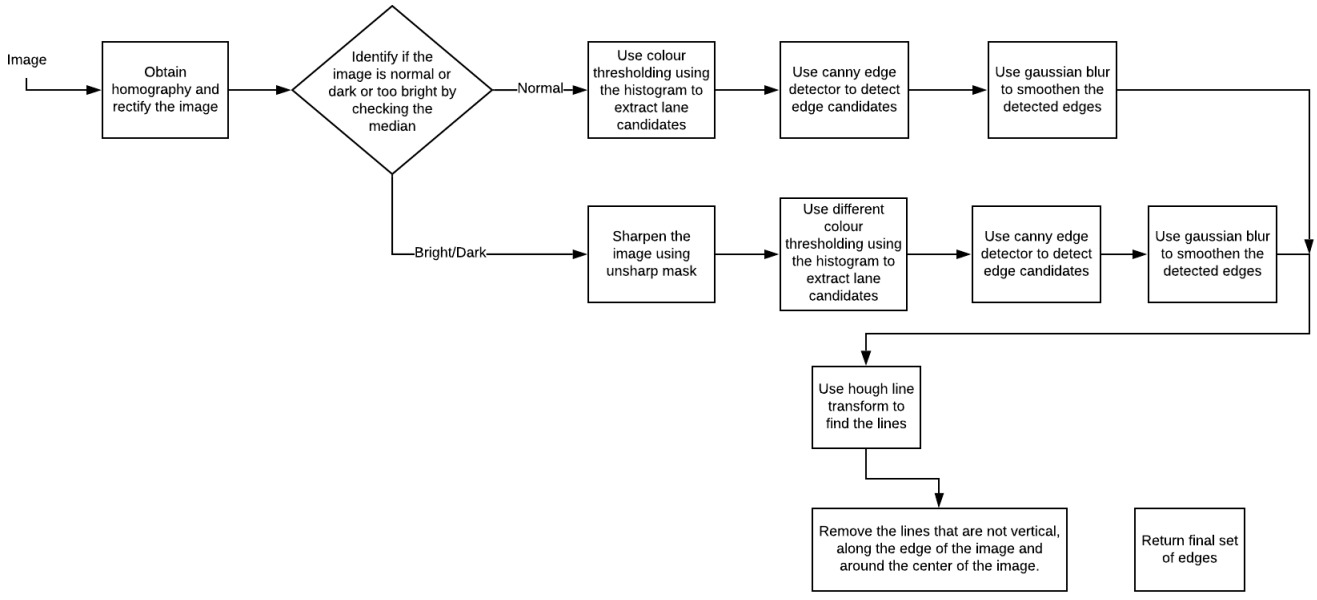


Figure 19: Pipeline for dataset 2