Lane Detection : ENPM673

Rahul Karanam
Robotics Graduate Student
University of Maryland, College Park
College Park, MD
rkaranam@umd.edu

I. Histogram and Adaptive Histogram Equalization

In this Section , we will be looking at optimizing our image using two methods such as histogram equalization and adaptive histogram equalization.

Histogram Equalization

Please refer figure-1 to show you the output from these two methods.

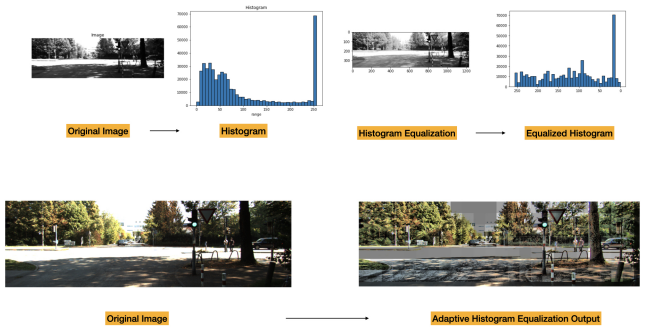

Fig. 1: Pipeline for Histogram and Adaptive Equalization

In general, Histogram equalization is a process where we find the histogram of an image and then we equalize all the intensities of the image in order to make the histogram have a proper or equalized pattern.

It basically kind of normalises the image intensities and averages them.

These are the steps followed to equalize an image using histogram equalization.

Steps to achieve the above process:

- First, we load the image and convert it to different color space i.e HSV(Hue , Saturation Value).
- We will be only working on the Value as it contains the intensity values(brightness).
- First we find the histogram of our image(V-channel) and then find the cumulative histogram for the same image.
- Then we normalise our cumulative sum with a data type of uint8.
- We will be updating our image with the cumulative histogram values.
- We then Merge our new v-channel with our previous h s channel.
- Convert our HSV image to BGR which is the image which has been optimised using Histogram Equalization.

It works with dark images but it is not an optimal one because there are some areas in the image which are over samplified which might change the overall texture of the image.In order to overcome this issue we use adaptive histogram equalization which equalizes the intensities of the image based upon the windows selected and it equalizes these windows.

Please refer to figure-2 for the histogram equalization outputs and their comparison.

Fig. 2: Histogram Equalization

Adaptive Histogram Equalization

As we have seen from the previous section , the image is over samplified i.e white shades become more brighter, which is not a efficient way and we need a technique which is adaptive and works in each enclosed region.

This method works in similar way where we divide the image into multiple windows and then tend to equalize these blocks using our histogram equalization technique. Overall , it will optimize the image and works well with all the images. I have faced issues while performing the adaptive histogram equalization technique as the image was not processing properly and you can see the output in the figure - 4. I have tried to smooth the image using bilateral filter to make sure the blocks are smoothed out in the image.

Steps taken to achieve the above process:

- First, we load the image and convert it to HSV color space.
- We will only be working on the V channel as it contains the brightness values.
- We will be choosing a window size which we needed to divide our image into several windows.
- After dividing the image into windows, we then apply the histogram equalization to each block.
- After applying the histogram equalization , we can see some blocks are not properly optimized and we can see the blocks.
- In order to remove the dark edges around the blocks , I have used bilateral filter to smooth the image.

- After the above process, we merge back our new v-channel with our h s channel, convert it back to bgr to show our output.
- Please refer to the fig-3 for the showing the output from the adaptive histogram equalization

Fig. 3: Adaptive Histogram Equalization

I have compared my output with the CLAHE method in opencv which is a contrast limited adaptive histogram equalization technique.

Here you can refer to the figure-4 for the comparison.

Fig. 4: CLAHE vs Adatpitve Histogram Equalization

II. Straight Line Lane Detection

Pipeline for the above problem:

- Pre-processing
- Apply Hough Transformation
- Perspective Transformation
- Fitting a polynomial
- Drawing lines on lanes

Here, you can find the pipeline followed to detect the straight lane and differentiate between solid and dashed lines.

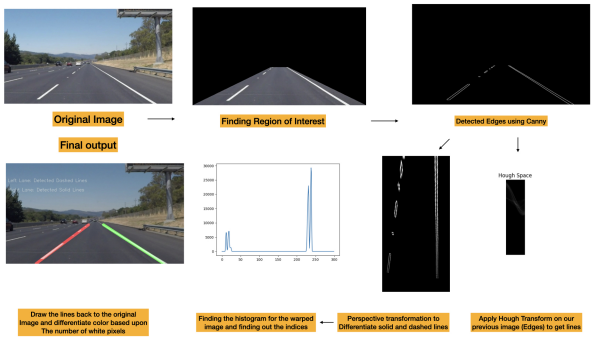

Fig. 5: Pipeline for Straight Line Lane Detection

Pre-processing

As you can see in the figure above, we need to detect the lane which have dashed lines on one side and solid lines on the other side.

First step is to find the region of interest where we want to detect. I have selected trapezoid shape for my ROI(Region of Interest) as it gives me better edges.

After determining the ROI , we then convert our image to grayscale and apply gaussian blur to smoothen the image.

Later we apply Canny edge algorithm to find out the edges from the smoothed image.

I initially thought of applying mask and then finding the edges but the other way worked around great. Now I apply my ROI mask to my image using fillPoly and bitwise and operation to apply the mask to my original image.

Please see figure 6 and figure 7 for your reference regarding the mask and detected edges.

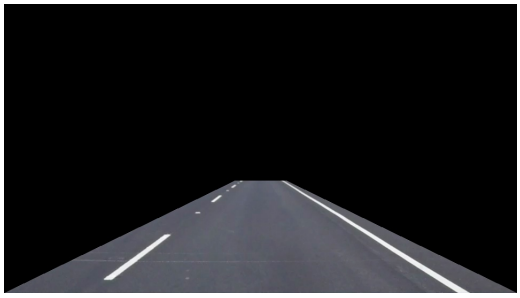

Fig. 6: Edge Detection

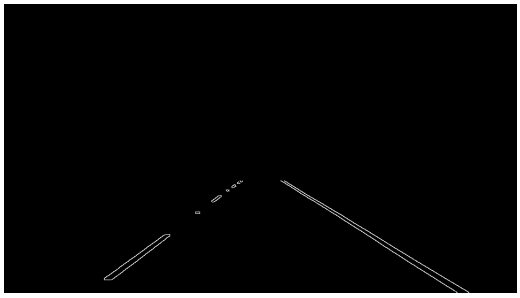

Fig. 7: Image Mask (Region Of Interest

Detecting Lines using Hough Transform

The next step after detecting the edges is to apply hough transform to our output from the above pre-processing. I have used Hough Probabilistic transform rather than the standards as the standard one has detected more lines in the same area which has caused issues when finding accurate left and right points.

Now we use HoughP method apply the transform to our edge image in order to get the lines. The way hough transform works is each edge pixel in the image space will become a line or a curve in the hough space and which is easier to work.A straight line is converted to polar domain and the number of intersections of these points in the hough space represents a line in the image space.

$\rho = x * cos(\theta) + y * sin(\theta)$

The straight line is denoted by two parameters rho and theta in the hough domain.

I have created hough line from scratch just to show the hough space whenever we detect a line.Please refer to this figure for hough space.

Perspective Transformation

Now before drawing lines and finding the left and right coordinates, we first apply perspective transformation to our

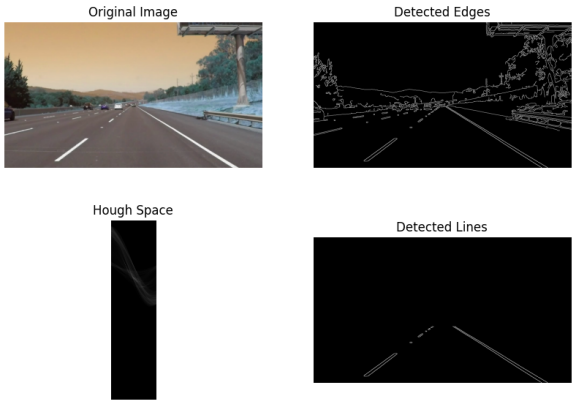

Fig. 8: Hough Space output

detected edge image in order to differentiate between dashed and solid lines.

I have selected source and target points by visually from the plot and warped the image using warpPerspective to get the warped image which can be seen in figure.

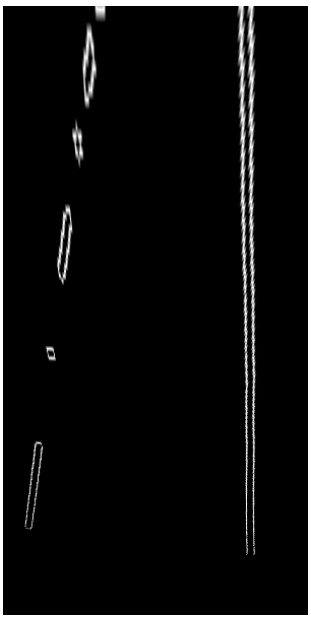

Fig. 9: Perspective Transformation of the detected lane

After getting the warped image, I tend to calculate the histogram of the image and then differentiate the left and right halves.After dividing them into two halves i have found out the highest pixel from both the halves, the more the number i.e it indicates that the line is a solid line.

But the main issue came when i tried to flip the image, my algorithm was working for some frames and the histogram output for the flipped image was not consistent , because of that I had to go for another method.Please see the histogram for the original image which differentiates solid and dashed lines.

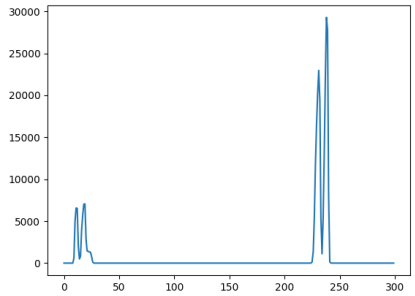

Fig. 10: Histogram output for the warped image

I have divided my image into two halves using the midpoint, and then used the opencv function cv2.findNonZero to find the number of non-pixels or white pixels. I have used this counter to differentiate the color for the lines and text overlaying the original image.

Please find below the Final output depicting both the flipped and original image.

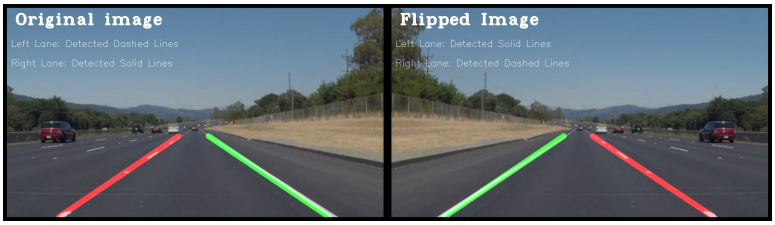

Fig. 11: Final output for both directions

Fitting a Polynomial

Now we tend to work on the left and right lane points in order to fit a polynomial to it and draw lines to showcase the lane detection.

We now iterate through our lines formed using the hough transform and fit a first order polynomial to these points.We then use these polynomial coefficients i.e slope to determine the left and right coordinates.

If the slope is greater than zero and image center position than we append to the right lane coordinates array else we append to the left lane coordinates array , similarly we append the slopes to the respective arrays.

With the appended arrays, we calculate the new x and y coordinates and draw lines with the image bottom. We have found out the color using the number of pixels in the left and right halves of the image.

Here we draw the lines with different colors such as green for solid lines and red for dashed lines.

Please find the final output from the above pipeline.

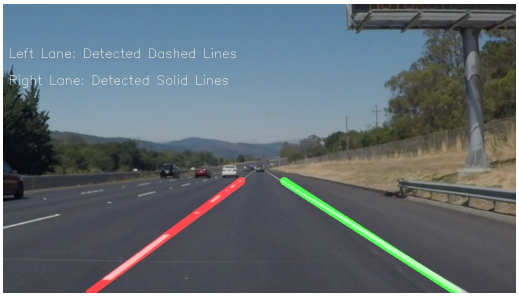

Fig. 12: Final Output

III. Predict the direction and radius of Curvature of the vehicle

In this section, we will be working on a dataset video (Udacity Lane Detection Video) to predict the turn and radius of curvature for the given frames.

Pipeline for the above problem

Please find the steps below in order to achieve the demonstrated pipeline which can be seen at figure-8.

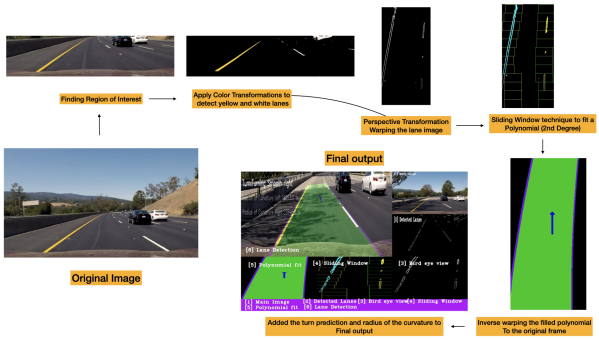

Fig. 13: Pipeline for Problem 3

- First ,we find the ROI from our video frame.
- Undistort the image using the camera calibration matrix and distrotion coefficients.
- Apply color transform(HLS color space), apply seperate masks to yellow and white,smooth the image using bilateral filter in order to create a binary image.
- Apply Canny edge algorithm to find the edges.
- Now we find homography between the source and target coordinates and apply perspective transform to get the bird eye view of the image found above.
- Here we use sliding window which is dividing our image into left and right halves and searching from bottom to top for detecting lane pixels.
- We identify the non-zero pixels in each windows and append these points to either left or right lane points which are used for fitting a polynomial.
- If we don't find any pixels in some frame, we update the current position with the mean of our previous positions.
- We then fit a second-order polynomial using polyfit to our detected left and right lane points.
- We use these polynomial coefficients to find out the radius of curvature for both the left and right lanes.
- For predicting the direction of the vehicle , I have used the histogram technique , we find the histogram for the binary image after warping.
- We tend to find two peaks for the yellow and white lane.From these peaks , we find out the image left , right and center position.We use the lane center coordinates and their difference to predict the turn.
- Now in order to project back our detected lane to our original image, we apply inverse warping using the inverse homography matrix found between src and target points.
- We display our radius of curvature and direction of the vechicle onto our original image.

Fig. 14: Original Image

Pre-processing

The first step in the pre-processing is to find the region of interest from our image.We can get the ROI(region of interest) using various methods, here I choose to crop out the image for a certain region which covers most of my lane instead of using shapes such as trapeziod or a triangle as it gave me good results during the warping.Please refer to the figure 9 for the ROI.

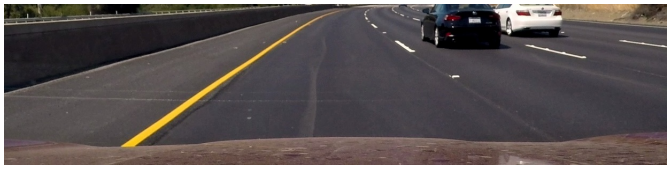

Fig. 15: Region of Interest

As we can see in our video, the frames are distorted radially, in order to remove these distortion we use findCheckboard and CalibrateCamera function methods to find the correct K matrix for our camera and to undistort the image.I have used sample checkerboard images to find out the K matrix and distortion coefficients.

We apply this to our image frame to remove the distortion.

The next step is apply color transformations,smoothing the image to get finer edges.

Fig. 16: Applying yellow and white mask to detect lanes

- First convert our image to HLS color space,apply yellow mask to get the yellow lane and white mask to get the white lane from the image.
- Then we merge these two images and convert it back to gray scale.
- Apply Bilateral filter to smooth the image.
- Now we detect edges in the image using Canny Edge detection.

In the below figure , you can see the binary images and the detected edges.

Fig. 17: Detected Edges using Canny

Perspective Transformation

Given the above binary image found using the above pre-processing steps , we then perform perspective transformation to get a bird's eye view to interpret the lanes better.

I have choosen the source points as per the frame and worked with the different coordinates and found out the best one. My target coordinates are the world coordinates where i want to transform my original image to a bird eye view.(600,300) is my window size.These points were visually determined rather than using a function.

Here is the example image , after applying the above transformation.

Fitting a Polynomial for the detected Lanes

Now after getting the warped image from the previous pipeline we then tend to fit a second order polynomial to the above left and right lanes.We use sliding window technique which is used to divide image into windows and detects the lane pixels in each windows(we extract indices) and later we fit a polynomial to these points.

- First, we Calculate the histogram for our warped image.
- As you can see from the figure , we can see two peaks which denote our lanes.
- Now we divide our image into windows i.e 12 our case and window height will be image height divided by the number of windows.
- We then find out the non-black pixels or non-zero pixels in the image using nonzero function which returns
- We repeat this process for every windows(i.e 12) and separate the left and right lane points using the margin and the current left and right points identified using the histogram image.

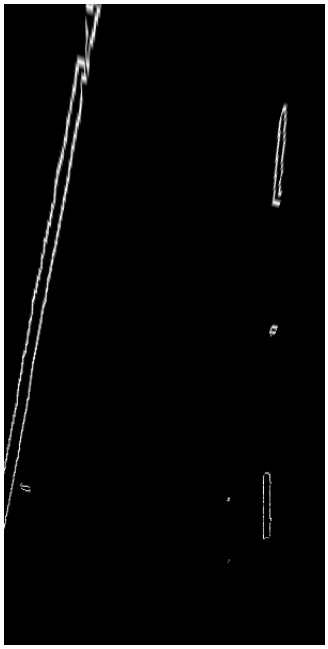

Fig. 18: Perspective Transformation of the detected edge image

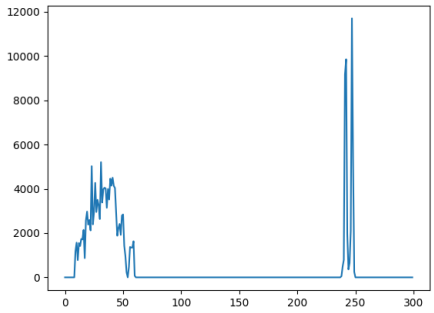

Fig. 19: Histogram of the warped image

- We append these locations to either left or right array depending upon the mean of the pixel values found in each frame this is used when you miss some data or there are too many pixels.
- After finding out these two arrays or points(left and right lane points) , we then apply a second order polynomial to these points.
- Prediction of the direction of vehicle is done using the image center,left and right coordinates.
- It mainly calculates the lane center and image center if it is less than 0 , it implies a left turn and if it less than 8 then go straight else turn right.

Please find below the sliding window technique and the lanes which are fitted using the above polynomial.

Radius of Curvature and Projecting the Lane to the image

Now We find the radius of curvature of the image.

We first need to convert our pixel unit to meters.

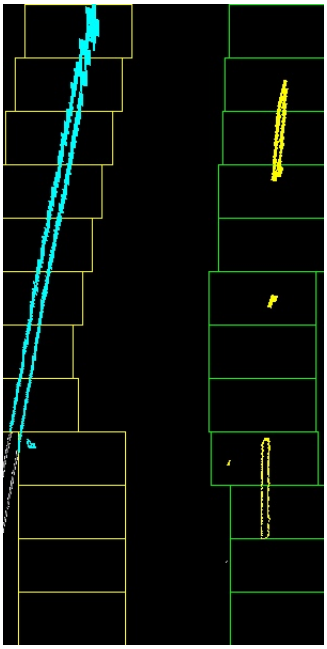

Fig. 20: Sliding window technique

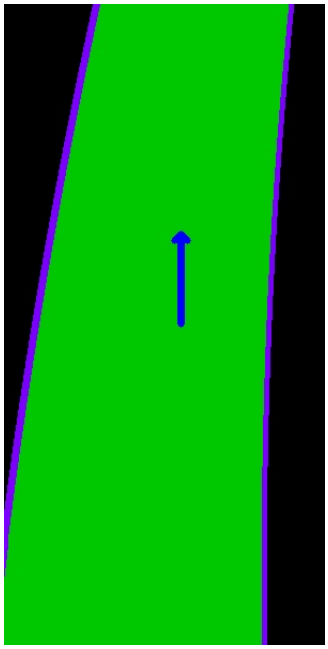

Fig. 21: Fitting a polynomial and filling the polynomial

For example, our lane length is 32 m and out image height is 720 to get the y-meters-per-pixel we divide our length by image height.

Radius of Curvature formula is given by :

ROC= $((1 + (2 * fit[0] * y0 * ymeter_{perpixel} + fit[1])^2)^{1.5})/np.absolute(2 * fit[0])$

We find the left and right curvatures using the above formulae.

After finding out all the required parameters we now project back all the details to the original image with proper textual descriptions.

- We first create a blank image same as our image.
- Append the left and right lane points into proper format.
- Using fillPoly we tend to fill the poynomial generated using the pixel locations which return an image with filled color refer to figure.
- We then apply inverse warping to get back from the bird eye view to our normal coordinates.(Inverse homography matrix generated using target and source points).
- After warping, we need to project back into our original image.
- We use opencv Addweighted function which will do this work for us.
- We add the radius of curvature and turn prediction turn onto the image.

Here you can final output generated after this pipeline.

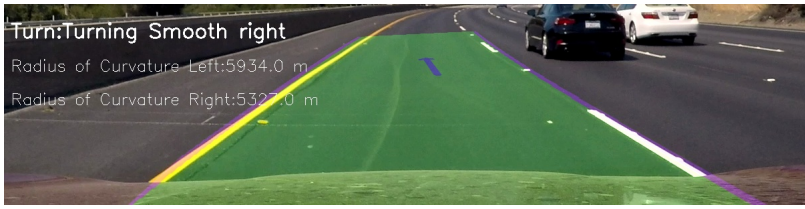

Fig. 22: Final Output with turn prediction and radius of curvature

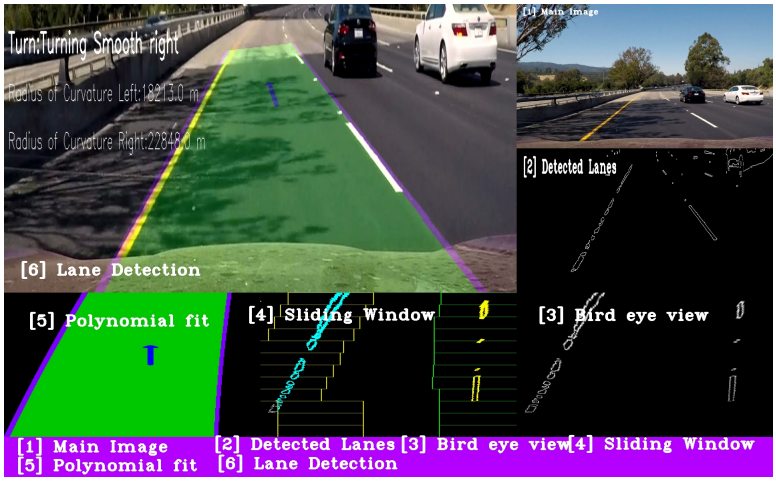

Fig. 23: Final Output with turn prediction and radius of curvature

A. Generalisation to other videos

I have implemented the above algorithm on different videos which have a similar lane. The outputs were not convincing as i have choosen the ROI certain to my frame which is not working accurately in the new video frames. I have attached

the output video in the above google drive for your reference.
I have tried to generalize other functions overall.

Output Video Link

Please find the above project in the below github repository

- **Github: https://github.com/karanamrahul/Lane-Detection-using-OpenCV**

- **Output Videos : https://drive.google.com/drive/folders/17uvekw6EzbJL63fmnQd51FWaF0KMO2NO?usp=sharing**

B. References

- https://www.cuemath.com/radius-of-curvature-formula/
- https://github.com/charleswongzx/Advanced-Lane-Lines
- https://kushalbkusram.medium.com/advanced-lane-detection-fd39572cfe91
- ENPM673 Homography and Hough Transform Lectures