ENPM673-Project2

Perception for Autonomous Robots

April 5, 2022

Student:
Pranav Limbekar (118393711)

Semester:
Spring 2022

Course code:
ENPM673

Contents

1 Problem-1

Task: To enhance the image quality

Approach: The purpose of this project was to improve the visual appeal of the provided video. The image sequencer depicts a view on a road. The input video sequence is dark, thus there aren't many details to be gleaned. To improve the quality of the image there is a well known method called Histogram equalization. In this method basically we find histogram of the image of the desired channel(R,G,B,Gray scale) and perform equalization on the acquired histogram. What equalization does is enhances the image's quality, it spreads out the most frequent pixel intensity values or stretches out the intensity range of the image.

Normal Histogram equalization enhances the image but does not account for the contrast of the image to full extent. To enhance the contrast we can use Adaptive Histogram Equalization(AHE). AHE is performed by dividing the image into a grid and performing the histogram equalization on each window in that grid separately. This method enhances the image image quality and adjusts the image contrast as well.

Although the AHE method can give better results, it depends on the existing conditions of the image. For instance in the image data provided for this project if we just perform the AHE on the images the output has the borders of the grid lines on it. According to my understanding the AHE process does not account for the the borders of the grid which might have high intensity pixels and according to histogram equalization they are normalized considering just the intensities in that window which when performed on all the windows in an image it might show darker grid edge lines. This can be avoided using the CLAHE which is Contrast Limited Adaptive histogram Equalization which basically enhances the quality of the image by adjusting the gamma value of the image.

Pipeline used for this problem is as follows:

- Get the input frames from dataset.
- For each frame get the histogram.
- Equalize the histogram uising normal equalization and Adaptive equalization.
- Generate videos of both outputs.

The cumulative sum is always between 0 and 1, as seen in the following figure which is cumulative sum for first frame in dataset,

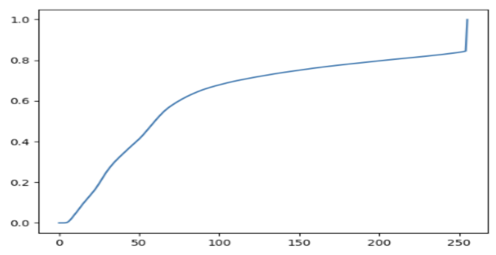

Figure 1: CDF

In my opinion the Adaptive Histogram equalization is more effective in low contrast images. For the dataset provided, normal histogram equalization give better output than the Adaptive histogram equalization. Sample output f the first frame of the dataset can be seen in the following figure

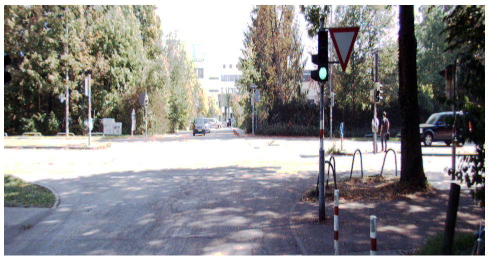

Figure 2: Final output of first frame

2 Problem-2

Task: To detect lanes lines on a input video.

The task here was to detect the lane lines in the input video and plot lines on the lanes based on the type of the line. Pipeline followed here is as follows:

- Get the input vide.
- For each frame, convert it to grayscale and detect edges using canny edge detection.
- Mask the image with a predefined polygon and get the region of interest.
- Get lines using HoughLine function.
- Saperate the lines and get the type of line i.e. if dashed or solid.
- Get the final image with lines drawn.

The first step here was to take the input frame convert it from BGR to GrayScale and process it to get the masked image which is done in the processImage function. In this function a mask in a shape of predefined polygon in applied to get the region of interest which is done by changing the pixel values of the image outside the area of polygon to 255 which is black if the passed imaged is an 8 bit image. By applying this mask we get just the region which has lane lines in it.

Before applying the mask we perfrom the canny edge detection whose algorithm gives us the edges in the image. This is done before applying the mask because if it is done after the mask is applied the algorithm also accounts for the edges of the polygon which we do not need. After processing the image and getting the masked output we then use HoughLines function in opencv to dtect the lines in the image. The HoughLines method is widely used method for line detection in Computer vision which detects the lines present in the hough space. Lines can be represented uniquely by y = a (x + b). This form is, however, not able to represent vertical lines. Therefore, the Hough transform uses the equation in the form ($r = xcos\theta + ysin\theta$). The Hough space for lines has therefore these two dimensions; and r, and a line is represented by a single point, corresponding to a unique set of parameters (θ, ρ). The mapping of the line to point can be seen in the following figure,

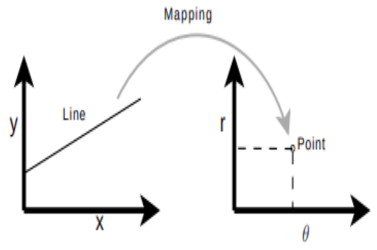

Figure 3: Mapping from line to point

An edge point produces a cosine curve in the Hough Space. From this, if we were to map all the edge points from an edge image onto the Hough Space, it will generate a lot of cosine curves. If two edge points lay on the same line, their corresponding cosine curves will intersect each other on a specific (ρ, θ) pair. Thus, the Hough Transform algorithm detects lines by finding the (ρ, θ) pairs that have a number of intersections larger than a certain threshold. The following fig shows the demo of hough lines detection,

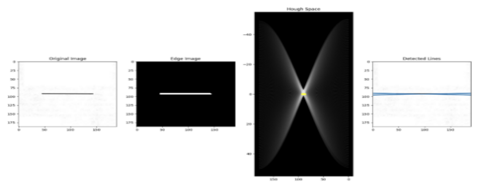

Figure 4: Hough line detection

Getting back to our task, after performing the hough line detection we pas the image in a function named getLineImage which returns the lane lines draun on the frame. In this function, after we get the lines from hough ine we sort the left and right lines and store them in a list which contains the points to draw the lines but before drawing them we need to know the type of line i.e. if it is dashed or solid. To know the type of the line we pass the warped image in a function called getLineType. This functions basically splits the image in half and finds the contours in each half. If the contours found in one of the halfs in more than the other, it means that half contains the dashed lines as the dashed lines will have more contours than a solid line, Based on the output of this function a line is drawn on the image with blue line representing the dashed and other as solid. The output of the can be seen in the following fig,

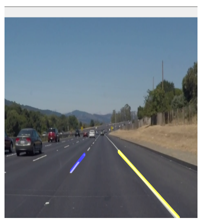

Figure 5: Final output

3 Problem-3

Task: Lane detection and polynomial fitting using radius of curvature

The task here is to fit the polynomial using polynomial equations curve fitting algorithms. The pipeline of this problem is as follows:

- Get the input video.
- For each frame apply the color mas i.e. perform color segmentation.
- Get the polynomial fit points and the polygon drawn on a dummy frame same as the input frame using the fit points.
- Get the radius of curvature in Km and display it on the output.
- Finally predict the turn using the image centre and fit points.

The process of this problem begins with color segmentation of the image. This is done by passing a frame in the function named ApplyColorMask which is converted from BGR to HSV format. The Masking function basically applies the color mask by putting a threshold on the image which is upper and lower limit of the image pixel values that are allowed to pass. The out put of this image will give yellow and white lane lines in this case.

After segmenting the image based on colors of lane, we get the birds eye view by of the lanes by passing it into the said function. Here to get the birds eye view we have used the getperspective-transform function to get the matrix which can be used to map the image from the source point to the destination points. In general perspective transform can be expressed as follows,

$$\underbrace{\begin{bmatrix} t_i x' \\ t_i y' \\ t_i \end{bmatrix}}_{\text{Scaling Factor}} = \underbrace{\begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}}_{\text{Transformation Matrix (M)}} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Figure 6: Perspective Transform

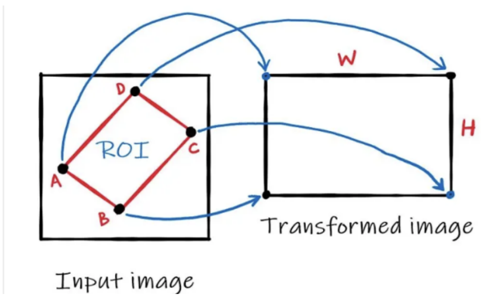

Figure 7: Perspective Transform

After getting the perspective transform which is the warped image or the birds eye view, we then perform the polynomial fitting of the lane line points. To achieve this task we divide each lane into 10 windows and get the fit points by sliding these windows over the lane lines. The visualization of this task can be seen in the following figure,

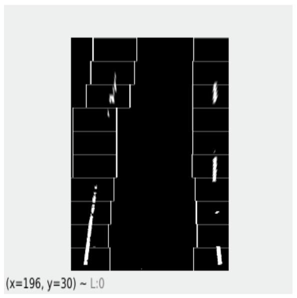

Figure 8: Windows on lane lines

The window is traversed through the whole image height and the indices of the white pixels are appended into a separate list using the nonzero function. The windows are shifted to left or right side depending on the mean of the pixel value indices found in the previous window.This will help us to take into account the steep turns that vehicle might encounter and better trace the lane line points.

After getting the pixel indices for both the lanes in separate lists, we need to get the polynomial function that fits the pixel locations. The polynomial can be found using the PolyFit() function available in numpy-python that returns the coefficient values of the equation. The degree of the polynomial chosen is 2 so as to take into consideration the curves during turning. Then after getting the curves of the lane, we fill that detected region with a particular color. This is done using the fillPoly() function.

The radius of curvature can be calculated using the following formula and displayed on the video(in Km) and the out put fig,

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

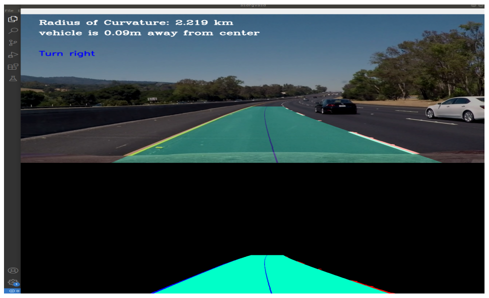
Radius of Curvature: 2.219 km
vehicle is 0.09m away from center
Turn right

Figure 9: Final output

4 Challanges faced

- For the first problem the challenge was to avoid the grid lines showing up on the image.
- For second problem the task was to separate the dashed lines and the solid line, i.e get the optimal output even if the video is flipped horizontally. This was resolved by using the line getlinetype function.
- For third problem the challenge was to get the curved lane line points. At first the fitting was working for comparatively straight lane lines but not for curved. This was resolved by performing the polynomial fitting on the obtained straight or base x and y coordinates.