Project 2

Gokul Hari
Directory ID: hgokul@umd.edu

Fig. 1. a) Raw Image; b) $mask_{color}$ c) Bitwise-OR($mask_{color}$, $mask_{grad}$)

Fig. 2. Lane rectification - bird's eye view

I. PROBLEM 2

Question

Do a simple Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars. You are provided with two video sequences, taken from a self-driving car. Your task will be to design an algorithm to detect lanes onthe road, as well as estimate theroad curvature to predict car turns.

Answer

I performed the following preprocessing and outlier rejection steps in order to detect the lanes consistently throughout the video frame.

Color and Gradient Thresholding: To identify the lane lines, I converted the RGB (Red-Blue-Green) image frame to HLS (Hue-Lightness-Saturation) color scheme and performed color thresholding to obtain a mask. This is denoted as $mask_{color}$ In addition to color thresholding, I also perform gradient-based thresholding. I computed a thresholded sobel derivative map along x axis (G_x) and a gradient direction map ($\tan^{-1}(\frac{G_y}{G_y})$), thresholded between $45°$ and $65°$. I combined these two maps with a bitwise-OR operation and applied the opening morphological operation with a 1×2 white pixel structural element. This output of gradient thresholding is denoted as $mask_{grad}$

I fused both the $mask_{grad}$ and $mask_{color}$ with a bitwise-OR operation to obtain the resultant seen in figure 1c. Eventhough, it accurately identifies the lane pixels than simple color thresholding, I noticed that gradient thresholding includes a lot of outlier elements from the road due to illumination discontinuities/ shadows. This in turn affects the lane tracking and hence, in the final pipeline, I only used $mask_{color}$ in my final pipeline

Lane Rectification: Lane rectification is required to obtain a bird's eye view of the road and compute the curvature of the road in order to make turn-predictions. To perform this rectification, 4 corner points ,denoted as C_{lane}, belonging to a trapezoidal region of interest (ROI), where the lane line is present has to be picked.This can be done manually or using a Hough Transform-based vanishing point estimation approach, which will be discussed below. The homography between the corners C_{lane} and the corners C_{rect} of a destination image frame was calculated. This homography can be used to warp the lane ROI to obtain a bird's eye view of the road. This is denoted as B_{map} This is shown in figure 2c

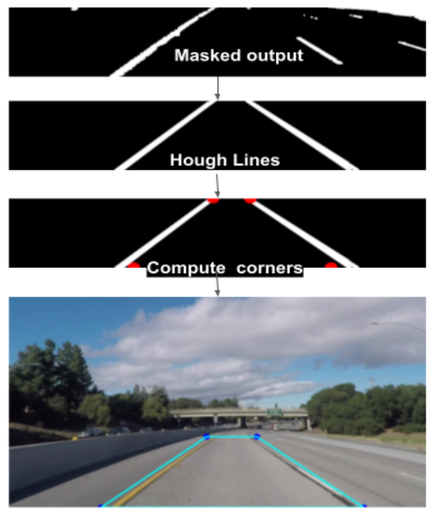

Fig. 3. Hough transform based lane corner detection

Choosing C_{lane}: It is crucial for the detected lane lines in the bird's eye view B_{map} to resemble parallel to each other. This, highly relies on the choice of the corners C_{lane}. To automatically compute the corners C_{lane}, I cropped and removed out the upper half of the image frame which contained the sky and other outliers. Next, obtained the bitwise-OR resultant of $mask_{grad}$ and $mask_{color}$, which is seen in 3 (block - masked output). I computed the hough lines of this binary mask to obtain a set of lines that only belong to the lane lines. These hough lines are segregated to right and left set of lane lines by computing their slopes. The mean of the set of lines is computed to obtain a single right and left lane each. This is shown 3 (block - hough lines). The start and end points of the left and right lane are noted, and this information is used to compute the C_{lane} corner points. The block diagram of this procedure is shown in 3. However, this procedure to obtain lane points is not consistent throughout all the frames, as houghlines fail in some scenarios and hence, to maintain accuracy and simplicity of the pipeline, I chose to manually choose the C_lane and pass them as hyper parameters.

***Lane Search using sliding windows**:* The rectified output B_{map} contains the lane lines to be tracked. However, it is highly likely to contain a lot of outlier non-lane pixels in B_{map} that can affect the lane tracking. Hence, we perform a sliding window search of the B_{map} to find the lane lines. The block diagram of this procedure is shown in 4

For a B_{map} of height *h* and width *w*, we can fix the total number of windows N_{win}. The height of each window is given as h_{win} = *h*/N_{win} and the width of each window is given as w_{win}. In this search from $0 - N_{win}$, the y-coordinate location of the windows span from *0-h*, in steps of h_{win} while the x-coordinate location of the windows is calculated from the preceding window.

To begin the search, we need to compute the window region where the lane lines begin. This can be done by plotting the white pixel intensity distribution of B_{map}. Notice in figure 4 that there are two peaks in this distribution that correspond to the left and right lanes. The x-coordinate of these two peaks (x_{l1}, x_{r1}) point out the starting location of the lane. I obtained the left and right lane windows W_{l1} and W_{r1}, both of size $h_{win} \times w_{win}$ using x_l, x_r as the center along x-axis in left and right lane. I computed the centroid of the white pixels within W_{l1} and W_{r1}. The x-coordinate of this centroid, becomes the x_{l2}, x_{r2}) for the next window W_{l2} and W_{r2}, and the search continues till $W_{lN_{win}}$ and $W_{rN_{win}}$ The non-zero pixel coordinates from all the left lane windows ranging from W_{l1} to $W_{lN_{win}}$ is recorded as LP_{left} and the non-zero pixels of right lane windows is recorded as LP_{right}.

***Lane Line fitting**:* The non zero pixels LP_{left} and LP_{right} for the left and right lanes, computed using the sliding window search method, can be used to fit a polynomial line ranging from (0 - h) for the left and right lanes. These predicted left and right 2nd order polynomials (L_{line}, R_{line}) can be used to compute the lane curvature and predict turns. The L_{line}, R_{line} lines are plotted in a black mask, and the cyan coloured lane area is printed, as shown in figure 5a. By inverse warping this image and superimposing it back in the input image frame, I printed the detected lane area as shown in 5a

***Predicting Turns**:* To predict the direction of the turn, I computed the average of x and y points in (L_{line}, R_{line}), and fit a first order polynomial to it (green line between lanes in figure 6). Then I computed the slope (m) of this first order polynomial line. Based on the slope m, I classified whether the lane is curving to the right or left as following

- If m $>$ 0.1, the direction is left
- If 0.1 $>$ m $>$ 0.01, the direction is slightly left
- If 0.01 $>$ m$>$ -0.01, the direction is straight
- If -0.01$>$ m $>$ -0.1, the direction is slightly right
- If -0.1$>$ m , the direction is right

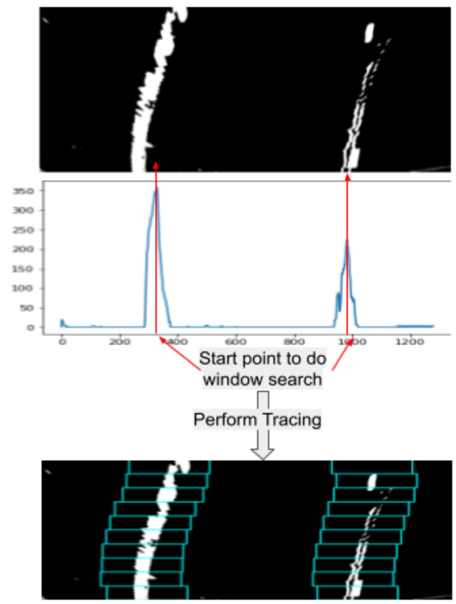

Fig. 4. Window searching

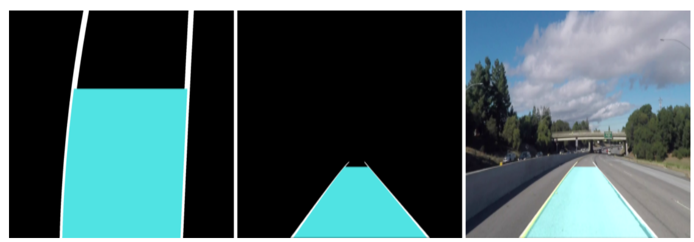

Fig. 5. a) Print lane lines and area, b)Applying inverse homography; c)Superimposing the lane area

***Computing Lane curvature**:* I also computed the radius of curvature of the left an right lane lines L_{line}, R_{line}. I computed the radius in real world units (meters) by converting from pixel space to meters by defining the appropriate pixel height to lane length and pixel width to lane width ratios. I assumed the real world parameters, that is, the width between two lanes as 3.7 (as per US highway standards) and the length of the lanes to be 32 meters. I made these assumptions based on this reference [1], Knowing the width *w* and height *h* of lane ,in pixels, from the B_{map}, I computed the *width per pixel* and *height per pixel* which I used to compute the polynomial coefficients (a,b) of the lane curves in terms of meters (real world units). Using these coefficients (a,b), I computed the radius of curvature using the following equation

$$Radius\ of\ curvature = \frac{[1 + (\frac{df}{dY})^2]^{\frac{3}{2}}}{\left|\frac{d^2 f}{dY^2}\right|} \tag{1}$$

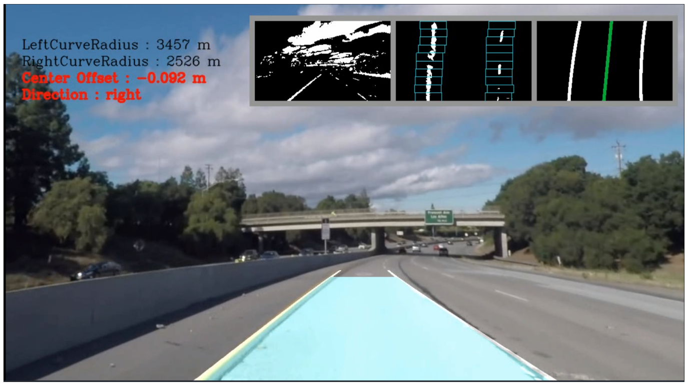

Fig. 6. final output frame

where $\frac{df}{dY}$ = aY+b, where Y is the length of lane in meters, given by Y = *height per pixel* * lane height h in B_{map}

Next, I computed the car's offset from the lane center. For this, I obtain the center between the two lanes from the first order polynomial (green line between lanes in figure 6) $LC_{current}$. The center point between the two lanes from the car was visually observed to be at 600 at the start of both the videos, and this is denoted as LC_{start}. The offset value from the lane center (in pixels) is calculated as,

$$C_{offset} = LC_{start} - LC_{current} \tag{2}$$

By multiplying this with the previously computed *width per pixel*, we can obtain the car's offset from lane center in terms of meters.

All these values are printed in the final output frame shown in 6

Motion filtering and Outlier rejection: As the video proceeds, there are several cases where the lane output is not favourable to compute curvatures/ predict turns.

Case 1: There are possible cases when the color thresholding function **identifies outlier** components of the road as lane pixels. This results in wrong x_{l1} ,x_{r1} start point in the sliding window search, and thus sliding window loses direction, resulting in bad lane detection. Such an instance is shown in figure 7. In order to notify this, I keep a record of moving average of the start points x_{l1}, x_{r1} as the video proceeds, denoted as x_{l1avg} ,x_{r1avg}. The absolute difference d_l between the x_{l1} and x_{l1avg}, and d_r between the x_{r1} and x_{r1avg} is computed for every frame. Any sudden changes/flickers in start points x_{l1} ,x_{r1} result in high values of d_l and d_r which can be used to detect a "bad lane". This function can be tuned to accomodate classifying whether a lane is being changed purposefully by the driver or if it is due to the noise in $mask_{color}$, but since I did not have a dataset where lane lines are changed by the driver, I could not test this.

Case 2: In some cases the color thresholding function **doesnt identify** sufficient number of non zero pixels to detect a lane. Such an example is shown in figure 8. Insufficent number of non-zero pixels signify a 'bad lane'.

Fig. 7. Failure case : Sudden change in lighting

Fig. 8. Failure case : No lane found - poor lighting

I perform moving average filtering of the lane lines, and so I compute and record the mean of the left and right lane lines of the past n_{mva} window frames as $L_{lineAvg}$, $R_{lineAvg}$. When a 'bad lane' is detected, $L_{lineAvg}$ or $R_{lineAvg}$ replaces the bad lane, thus rejecting failure cases.

Results: The video outputs are provided in here

II. Problem 1

Question

Here, we aim to improve the quality of the video recording of a highway during night. The aim is to enhance the contrast and improve the visual appearance of the video sequence. The suggested pipeline for improving lighting conditions is the Histogram equalization method.

Answer

To enhance the lighting conditions of the image frames in the video, I first attempted to perform histogram equalization. To perform histogram equalization, I computed the histogram of every channel of the image with 256 bins, with each bin corresponding in pixel intensities from 0-255. The histogram is shown in figure 9a The normalized cumulative distribution of this calculated histogram was computed for every channel, which is shown in figure 9b. Cumulative distribution function is given as,

$$C(i) = \frac{\sum_{j<=i} h(j)}{N} \tag{3}$$

where h is the histogram, and h(i) indicates the number of pixels with intensity of i, as N is the total number of pixels. The normalized cumulative distribution (scaled within 0-255) is used to reconstruct the image with new pixel intensities. This is histogram equalization. The equalized histogram of the output image is shown in 9c.

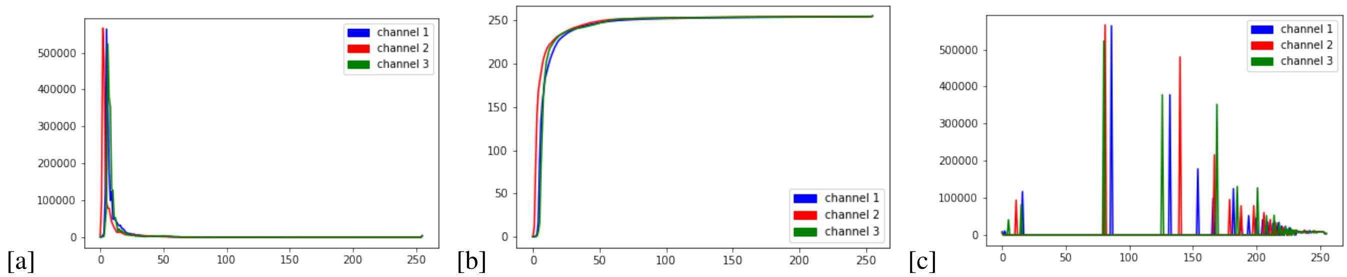

Fig. 9. a) Histogram of image; b) Cumulative Distribution; c) Equalized Histogram

Fig. 10. a) Raw Image frame; b) Output of Histogram Equalization; c) Output of gamma correction

However, this method did not yield satisfactory results since the output image is too bright. This is a well known shortcoming of the normal histogram equalization procedure and hence the variants like contrast limiting adaptive histogram equalization were suggested. However, I was unable to try out other methods due to time constraints. Instead, I applied gamma correction with a gamma value of 2.2, to obtain a better result. The gamma correction of input image pixels to the output is given by the relationship,

$$I_{out} = I_{in}^{\gamma} \tag{4}$$

The comparison in output of both the methods (histogram equalization and gamma correction) is shown in figure 10. The output videos are available here

REFERENCES

[1] https://towardsdatascience.com/teaching-cars-to-see-advanced-lane-detection-using-computer-vision-87a01de0424f