Project 2
ENPM673

Sakshi Kakde
M. Eng. Robotics
University of Maryland
College Park, MD, 20742
Email: sakshi@umd.edu

I. Problem 1

This section of the report discusses in brief the methods to improve the quality of the given video. I used the following approaches:

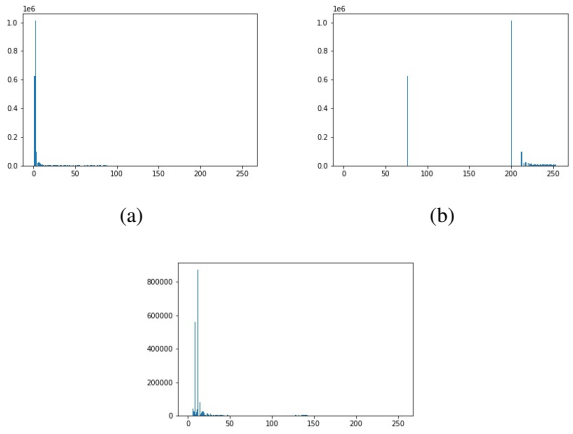
- Histogram Equalization
- Gamma Correction
- Contrast Limited Adaptive Histogram Equalization(CLAHE)

I will discuss each approach in brief.

A. Histogram Equalization

*1) **Histogram**:* A histogram represents the number of pixels for each intensity value considered[5]. Refer fig. 1(a).

*2) **Equalization**:* In this method, we spread out the most frequent intensity values and thus increase the intensity range of the image. This method usually increases the global contrast of images when its user data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast[5]. Refer fig. 1(b). As we are changing the intensity distribution, applying histogram equalization to R, G, B channel separately will result in color change of the resultant image, which is not desired. So, usually, we convert the RGB image to HSL image and apply equalization to only luminance channel. This way we preserve the color of the image.



(a) (b)

Fig. 1. Histogram of L channel without equalization(a), with equalization(b) and with CLAHE

B. Gamma Correction

Steps involved in Gamma correction are:

1) Convert the image from range [0, 255] to [0, 1] range. let's call it I.
2) For a selected value of Gamma(G), compute the output image O as $O = I^{1/G}$
3) Scale back the output image to the range [0, 255].

For a Gamma value less than 1, the value of $\frac{1}{G}$ will be greater than 1. Since the image values after conversion are less than or equal to 1, the output image values will be even lower. Hence, when we re-scale the image, the image gets darker for a Gamma value less than 1. Similarly, when Gamma is greater than 1, the image appears lighter. I have used a Gamma value of 2.

C. Contrast Limited Adaptive Histogram Equalization(CLAHE)

CLAHE is a variant of Adaptive histogram equalization(AHE) which takes care of over-amplification of the contrast. CLAHE operates on small regions in the image, called tiles, rather than the entire image. The neighboring tiles are then combined using bilinear interpolation to remove the artificial boundaries [7]. It also puts a limit on the contrast values [6] Refer fig. 2 to see results using all the three methods.

The following part of the report discusses, in brief, the algorithms to detect lanes for a given set of images and videos. I tried two approaches: One using Hough transform and the other using the curve fitting technique. The following section will explain both approaches.

II. Problem 2 - Approach 1

A. Image Prepossessing

The following operations were performed before lane detection:

1) Image undistortion using *cv2.undistort.*
2) Image blurring using a Gaussian kernel of size 5×5.
3) Thresholding the gray image to get a binary image.
4) Obtained ROI, which is almost 55% of the image.

Refer fig. 3 for output of this section.

(a)

(b)

(c)

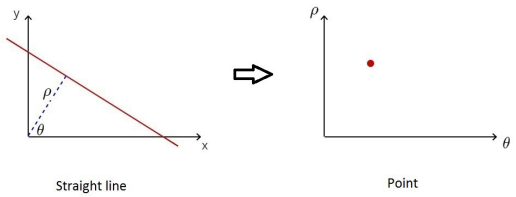
Fig. 2. Results after histogram equalization(a), Gamma correction(b) and CLAHE(c)

Fig. 3. Output after thresholding and getting ROI

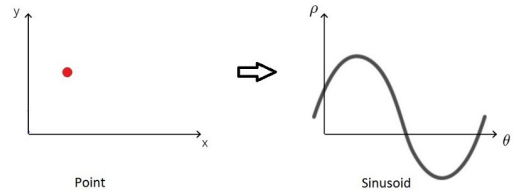
B. Hough Line Transform

1) ***Equation of line****:* A line can be represented as $y = mx + c$, where y and c are slope and y-intercept respectively. This form is called slope and intercept form. We can represent it in polar coordinates as well as $\rho = xcos() + ysin(\theta)$, where ρ is the shortest distance from the origin to the line (approaching the line perpendicularly) and θ is the angle between x-axis and the distance line[1]. The reason for choosing the polar coordinate system is that we can handle cases with vertical lines.

2) ***Image Space to Hough Space****:* A line in image space becomes a point in Hough Space, which is represented by (ρ, θ). From any given point, there can be a multiple line passing through it. Thus, there will be multiple sets of (ρ, θ) for a given point, which resembles a sinusoidal curve. Refer fig. 4.



(a)



(b)

Fig. 4. Image space to Hough space[1]

3) ***Detecting line****:* For a given set of points in image space, there will be set of corresponding values of (ρ, θ). For all the points lying on a line, the values (ρ, θ) will be similar. In other words, the sinusoidal curves for the points lying on the line will interest at a point, giving us the appropriate line parameters. Refer fig. 5

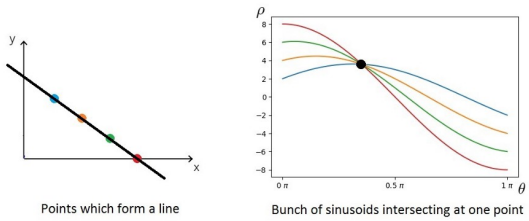


Fig. 5. Detecting hough line[1]

4) ***Hough line parameters****:* During implementation, we need to count the number of votes for each set of parameters.

So, we discretize the space using ρ resolution and θ resolution. Then we choose the values with votes greater than a threshold set by the user.

*5) **Optimzed approach: Probabilistic Hough Transform**:* In the Hough transform, it takes a lot of computation. Probabilistic Hough Transform is an optimization of Hough Transform. It does not consider all the points, instead takes only a random subset of points and that is sufficient for line detection. This implementation has additional two parameters:
minLineLength - Minimum length of a line.
maxLineGap - Maximum allowed gap between line segments to treat them as single line[2]. Refer fig. 6

Fig. 6. Detected Hough line

C. Filtering lines

As seen in figure 6, the detection is quite noisy. It can be seen that there are a lot of horizontal lines, which we are not interested in. I selected the line whose magnitude of the slope is within 20^o and 70^o. Next, I divided the set into two parts: left lane lines and right lane lines, depending on the sign of the line angle. To filter it further, I chose left lines with angle values within -35^o and -25^o and right lines with values within 45^o and 65^o. Still, there can be multiple detections. To choose the best fit, I checked their y coordinate values. From both left and right sets, I pick up a line with a maximum y value. Refer fig. 7. In case there is no new detection, I used the old detected lines.

Fig. 7. Filtered Hough line

D. Extending the lines and overlaying

Since the left line is not continuous, I set the low value for minimum line length. I extended the short detected lines by using simple line equation. Then using the four points(two from each line), I used *cv2.fillPoly* function to get overlayed lane. Refer fig.8

Fig. 8. Detected Lane using Hough line

E. Analysis

The following are the drawbacks of the above method:

1) This method will give incorrect results on turns.
2) Color thresholding is very sensitive to the environment. This approach worked here because the right side is grass(green) which created a good contrast for right lane markings. If there was pavement or footpath with a little color difference, this approach would fail.
3) If there is a traffic jam, then the left side will be packed with vehicles, which can result in wrong detections. Similarly for areas under construction.

III. Problem 2 - Approach 2

A. Image Prepossessing

The following operations were performed before the lane detection:

1) Image undistortion using *cv2.undistort.*
2) Obtained ROI, which is almost 62% of the image.
3) Thresholding the image that was converted to HSL color scheme to extract yellow and white lines. The output is a binary image.
4) Morphological operations to reduce noise. The final output from these steps is as shown in fig. 9

Fig. 9. Thresholded yellow and while lanes

B. Image warping

To detect the curved lanes, we warped the image to get a bird's eye view. The following steps were followed:

1) Four points were selected on the image in fig. **??**.

2) We need another set of four points to get the homography matrix. Since it is a bird's eye view, the points will correspond to the four corners of a rectangle of a chosen dimension.
3) The homography matrix H was computed using the OpenCV function *getPerspectiveTransform*.
4) The computed H was used to warp the image using the *warpPerspective* function.

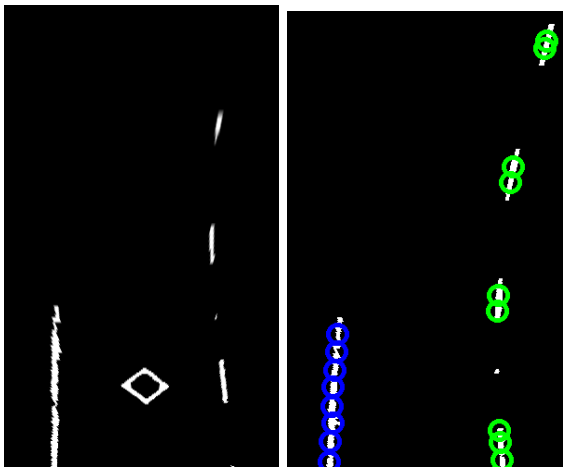
Refer fig. 10(a)

C. Detecting points for curve fitting

To obtain a curved lane, we need to fit a polynomial for a set of points. I used the following steps:

1) I divided the whole warped image into horizontal stripes of height 30 pixels and then divided it into two parts: set of left stripes and set off right stripes
2) For each stripe, I counted the number of white pixels. If the sum is greater than a set threshold, I will consider that strip, else I will discard it.
3) For the chosen stripes, I calculated the mean of locations for all the white pixels. This will give me the point coordinates that I will use for curve fitting.
4) At times, the points get noisy due to noise in the binary image. I compared the location of point from n^{th} strip with the $n-1^{th}$ strip. If the difference is large, I discarded the point.

Refer fig. 10(b).

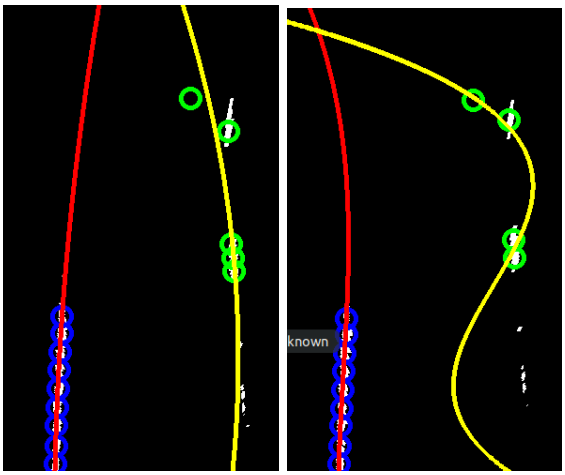


(a) (b)

Fig. 10. (a)Warped binary image, (b) Detected points for curve fitting

D. Curve fitting

We need to choose a n^{th} order polynomial to fit the set of points from the previous section. As the order polynomial increases, the curve tries to fit as many points as possible, thus giving us an over-fitted solution. Refer fig. 11 for a comparison between second and third-order polynomial. Hence, to avoid overfitting, I chose a $n = 2$.
As seen in fig. 11, the curve can sometimes get a bit unstable, mainly due to noise in the detected points. To make it more stable, I used a moving average filter with a window size of 10.



(a) (b)

Fig. 11. (a)2^{nd} order polynomial, (b)3^{rd} order polynomial

E. Finding Curvature

The second order polynomial is written as:
$y = a_0x^2 + a_1x + a_2$ The first and second derivative can be written as:
$\frac{\delta y}{\delta x} = 2a_0x + a_1$
$\frac{\delta^2 y}{\delta x^2} = 2 * a_0$
The curvature is given by: $R = \frac{(1+\frac{\delta y}{\delta x}^2)^{3/2}}{\frac{\delta^2 y}{\delta x^2}}$ [3]

F. Predicting Turn

Depending on the value of curvature, we can detect the lane turn. The curvature will be high when the lane is straight. It will be positive when there is a right turn and negative when there is a left turn. There can be noises in the turn prediction, which can be avoided by checking the past values of the turn. For example, if the past five predictions are saying 'right turn', and a new prediction says 'left turn', the chances are the new prediction is wrong.

G. Overlay Image

To overlay the detected lane onto the original image, I used the inverse of the homography computed from section III-B. Once I get the re-projected points, I used the *fillPoly* function to color the detected lane. The final results are shown in fig. 12

H. Analysis

This method has the following benefits as compared with the approach

1) It works for curved lanes
2) Since we are selecting the four points for warping the image, this approach is not very sensitive to the road

Fig. 12. Final results for lane detection using approach 2

surroundings. I think this will work in cases of traffic jams and road constructions.

The drawbacks are:

1) Since we are using color threshold, this is still sensitive to changes in the environment.
2) As we are manually choosing four points to warp the image, there can be cases where the lane is not present within those selected four points.

Lane detection using Deep Neural Networks as discussed in [4] can be better solution for real time applications.

REFERENCES

[1] https://medium.com/@tomasz.kacmajor/houghlines-transformexplained-645feda072ab
[2] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html
[3] https://tutorial.math.lamar.edu/classes/calciii/curvature.aspx
[4] https://github.com/qinnzou/Robust-LaneDetection
[5] https://towardsdatascience.com/histogram-equalization-5d1013626e64
[6] https://towardsdatascience.com/clahe-and-thresholding-in-python-3bf690303e40
[7] https://www.geeksforgeeks.org/clahehistogrameqalization-opencv/#:~:text=CLAHE%20is%20a%20variant%20of,to%20remove%20the%20artificial%20boundaries.