Effectively detecting Lane Lines on the roads which can be used for self-driving cars for following a lane

Varun Satyadev Shetty

***Abstract*—I will be detecting lane lines on the road from images and videos taken from a front end of a car camera under ideal conditions using different computer vision techniques such as Canny edge detection and Hough transform and apply it to different datasets and present a condition where It will fail and show different more robust computer vision techniques that can be used to detect lane lines in a more realistic environment having varying lighting conditions and distortion, and at the end I will also be calculating the radius of curvature of the road lane lines and the position of the vehicle on the road in real time**

***Index Terms*—Canny edge detection, Hough Transform.**

I. Introduction

In this section, I will talk about the problem statement and the motivation behind using the two different computer vision techniques that I am using to solve the problem.

A. Problem statement and motivation

For a Self-driving car to follow a lane on a road the first thing is to efficiently detect the lane lines on the road. I will be using different computer vision techniques such Canny edge detection and Hough transform to detect the lane lines and show that how these techniques work properly only in ideal cases and cannot be used to detect lane lines in real environment and then I will be implementing more advanced computer vision techniques such as Camera Calibration, Distortion correction, Filtering image and generating binary image using Sobel algorithm, color thresholding, Perspective transform, region masking and polynomial fitting techniques that can be used to detect lane lines in a real environment consisting of undistorted frames of image and varying lightning conditions. I will also be calculating the radius of curvature of the lane lines in the pixel space and later transform it in the real space and also calculate the vehicle position in the real space.

II. Methodology

In this section I will explain the Two techniques that I have used to detect the lane lines on the road.

Fig. 2. gray scale image

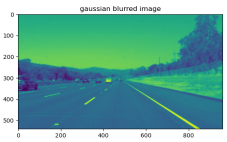

Fig. 3. Gaussian blurred image

A. *Basic computer vision technique*

Fig. 1. Original image

1) ***Converting the Image into Grayscale image and doing Gaussian Blurring****:* I have used the function cv2.cvtColor() And passed the colored image and the command cv2.COLOR_RGB2GRAY to the function to convert the colored image into grayscale image.

Gaussian blurred scaled image I have used gaussian blurring for obtaining a smooth image, gaussian blurring is done by using cv2.GaussianBlur() function and I have passed to it the grayscale image and a **kernel of size = 3**

2) ***Canny edge detection****:* I have used the cv2.Canny()function to which I pass the blur_image and the low_threshold = 200 and high_threshold = 255. Canny edge detection is a very common method to detect edges in an image. For kernel of size **3**. after performing Canny edge detection I will do a regional masking to concentrate only on the region of image

which is most likely to have the lane lines.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (1)$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (2)$$

$$G = \sqrt{G_x^2 + G_y^2} \quad (3)$$

$$\theta = \arctan \frac{G_y}{G_x} \quad (4)$$

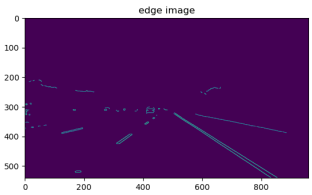

Fig. 4. Edge image

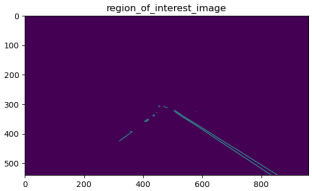

Fig. 5. Region of interest masked image

*3) **Performing Hough Transform**:* Hough transform is used to detect lines on the image.The Hough Transform is used for conversion from image space to Hough space. A line in a image space is a point in the Hough space and a point in the image space is a line in the Hough space. I have used the Probabilistic Hough Transform function cv2.HoughLinesP() as it is more computationally efficient.

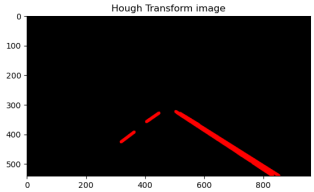

Fig. 6. Hough transformed image

*4) **Weighted addition**:* I am using the cv2.addweighted function() weight_1 = 0.8, weight_2 = 1.0, additionalterm = 0.0 so as to show the lane lines on the image

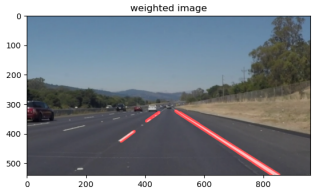

Fig. 7. final output of the basic computer vision techniques

B. Advanced computer vision technique

Fig. 8. Test image

*1) **Camera Calibration and Distortion correction**:* I will be using multiple chessboard images to calibrate my camera, which can be later used to undistorted any image of the same size. By using multiple images of a same chessboard on a flat surface from the camera, we an compare the shape of the images that we are getting to the actual image that we would be requiring, from this we can create a function that maps from the undistorted points to the distorted points and use this function to undistort any image of the same size. The reason for using chessboard images for camera calibration is due to the chessboards high contrast regular

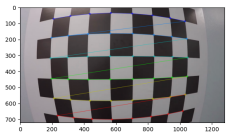

Fig. 11. Chessboard image with corners marked

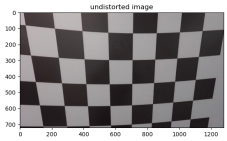

Fig. 12. Undistorted chessboard image

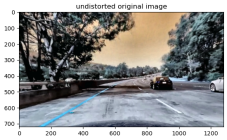

Fig. 13. Undistorted test image

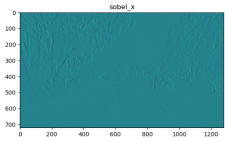

Fig. 14. sobel in the x direction

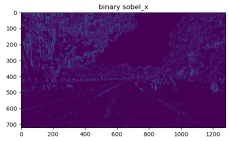

Fig. 15. binary sobel in the x direction

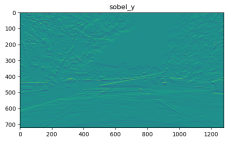

Fig. 16. sobel in the y direction

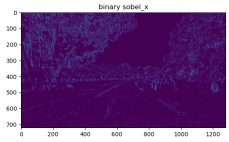

Fig. 17. binary sobel in the y direction

pattern they are easy to measure the distortion. I am going to generate the object points and the image points using the cv2.findChessboardCorners() function. These object points and image points can later be used to undistorted any image of the same size taken from the same camera.

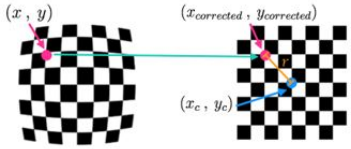

Fig. 9. Translation mapping demonstration

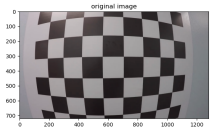

Fig. 10. ditorted chessboard image

*2) **Gradient and color thresholding**:* I have used a combination of color thresholding and gradient thresholding to get rid of the unwanted pixels. Gradient and color thresholding is applied to the undistorted image. **2.1) Gradient thresholding:** For gradient thresholding I am using the Sobel operator. Applying the Sobel operator to an image is a way of taking the derivative of the image in the xx or yy direction. For a kernel of size 3.I first applied sobel operator in the x direction and then took a sobel operator in the y direction then did a combination of the sobel_x and sobel_y and computed the gradient function. After this applied a number of thresholds to get the best estimate of the lane lines on the road.

$$sobel\_combined = \sqrt{sobel\_x^2 + sobel\_y^2} \quad (5)$$

$$\theta = \arctan \frac{sobel\_y}{sobel_x} \quad (6)$$

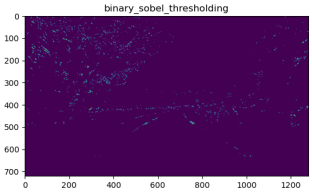

Fig. 18. Gradient thresholded image

Constraints applied:

$$temp_1 = new_abs_sobel_x > 40 \quad (7)$$

$$temp_2 = new\_abs\_sobel\_combined > 70 \quad (8)$$

$$temp_3 = (gradient > 0.7) \ \& \ (gradient < 1.4) \quad (9)$$

$$binary\_sobel\_thresholding[temp\_1 \ \& \ 2\_3] = 1 \quad (10)$$

2.2) Color Thresholding: I have done color thresholding by converting the image from the RGB channels to the HLS(Hue Lightness Saturation) channels. Saturation value gives us the measure of how colorful or dull the

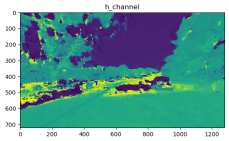

Fig. 19. h_channel

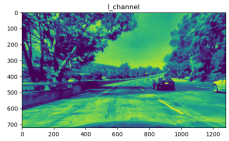

Fig. 20. l_channel

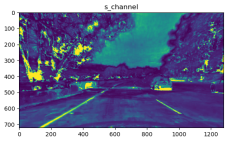

Fig. 21. s_channel

image is, the Hue value gives us representation based on the combination of red green and blue and the Lightness value tells us how close to the white value is the image.

Constraints applied:

$$temp\_4 = (s\_channel > 120) \ \& \ (l\_channel > 40) \tag{11}$$

$$temp\_5 = l\_channel > 205 \tag{12}$$

$$binary\_ls\_channel\_thresholding[temp\_4|temp\_5] = 1 \tag{13}$$

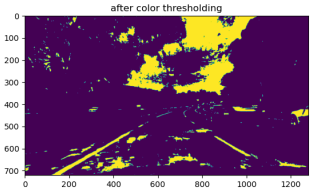

Fig. 22. color thresholded image

Combining the gradient and color thresholding

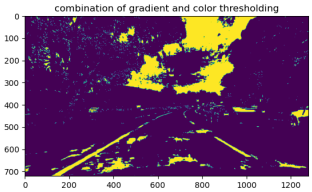

Fig. 23. combined thresholded image

*3) **Perspective Transform**:* I am using perspective Transform to change the view of the undistorted image and get a bird's eye view of the image, as it would be easier and more efficient to find lane lines on such a view. I am using the cv2.getPerspectiveTransform() function to get the wrapped matrix and then passing it to the cv2.warpPerspective() function to get the view that we desire.

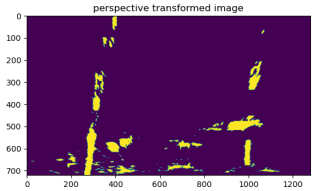

Fig. 24. Perspective transformed image

*4) **Polynomial fitting using Sliding window Algorithm**:* I will be doing polynomial fitting by using a second degree polynomial function

$$x = A * (y)^2 + (B * y) + C$$

. To find the coefficients A,B,C I will be using the built in function polyfit().

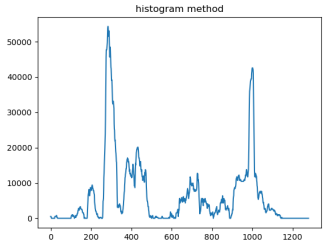

Fig. 25. output of the histogram method to find the x_base points for sliding window method

Sliding Window Algorithm: After finding the starting position of the lane lines, the next step is to determine the hyperparameters of the window. Now I will be looping through each window to find the nonzero lane pixels within the window and if the number of nonzero lane line pixels in a window in a particular step is more then the minimum number of pixels specified then I will recenter the window based on the mean position of this pixels

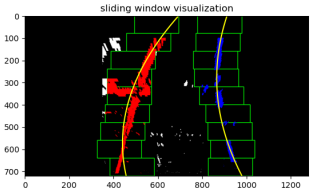

Fig. 26. output of sliding window method

Doing a smart search: With sliding window algorithm I have built an algorithm that uses sliding windows to track the lane lines out into the distance. However, using the full algorithm from before and starting fresh on every frame may seem inefficient, as the lane lines don't necessarily move a lot from frame to frame. In the next frame of video I don't need to do a blind search again, but instead I can just search in a margin around the previous lane line position.In fig 28 green and red shaded area shows where we searched for the lines this time. So, once you know where the lines are in one frame of video,I can do a highly targeted search for them in the next frame.

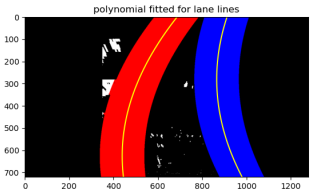

Fig. 27. Visualization of Smart search

*5) **Unwrapping the image and filling the pixels between the lane lines**:* This is very much similar to the perspective transformed step the only difference is that in this we are swapping the source points and the desired points.

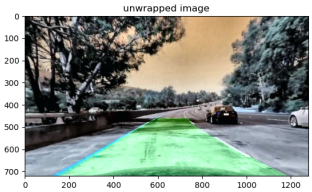

Fig. 28. Unwrapped image with lane pixels filled

*6) **Finding the radius of curvature and the vehicle position**:* In the previous step I had fitted a second order polynomial equation,

$$f(y) = Ay^2 + By + C$$

I am fitting f(y) rather than f(x), because the lane lines in the wrapped image are near vertical and may have the same x values for more than one y value.

$$f(y) = \frac{dx}{dy} = 2Ay + B \tag{14}$$

$$f''(y) = \frac{d^2x}{dy^2} = 2A \tag{15}$$

Formulae for radius of curvature is:

$$R_{curvature} = \frac{(1 + (2Ay + B)^2)^{\frac{3}{2}}}{|2A|} \tag{16}$$

The above formulae gives us the radius of curvature in the pixel space, so we need to convert it into real space : assuming: the lane width is 3.7 meters.The dashed lane line is 30 meters (According to the US regulations) Therefore we need to calculate the radius of curvature using y_meter_perpixels = 30/720 (Since my image has a height of approximately 720pixels) x_meter_perpixels = 3.7/700 (Since my image has a width of approximately 700pixels) By knowing the x positions of the the two lane lines and the curvature of the lane lines I have calculated the vehicle offset position.

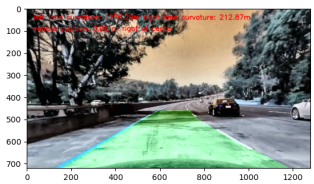

Fig. 29. Unwrapped image with lane pixels filled and with radius of curvature and vehicle position displayed

III. Results and Discussion

Results obtained using simple Computer vision techniques:

Input video test_case_1:link.

Result obtained:link.

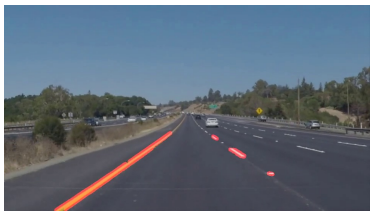

Fig. 30. one of the frame of the result video(frame 17)

Input video test_case_2:link.
Result obtained:link.

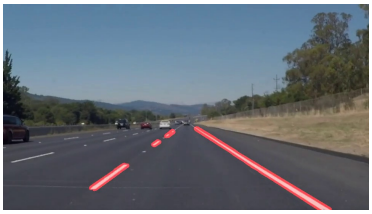

Fig. 31. one of the frame of the result video(frame 10)

Observation:
The above cases do not contain any distortion, lighting variation and shadows and it can be observed that in such a situation the simple computer vision techniques do not work properly.
Input video test_case_3:link.
Result obtained:link.

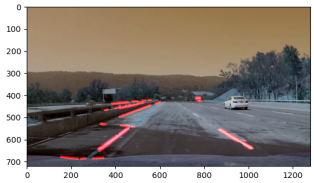

Fig. 32. one of the frame of the result video(frame 555)

Observation:
The above cases contains distortion, lighting variation and shadows and it can be seen that Simple computer vision techniques used do not work properly and the output obtained is not desirable.
Results obtained using Advanced computer vision techniques:
Input video test_case_4:link.
Result obtained:link.

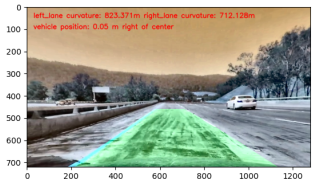

Fig. 33. one of the frame of the result video(frame 555)

Observation:
It can be observed that even though the video stream contains a camera distortion, lighting variations and shadows the advanced computer vision technique that I have used works properly.
observations for the radius of curvature and the vehicle position:
It can be observed that the Radius of curvatures for each frame is reasonable according to the US government specifications for highway curvature reference and also the vehicle offset position seems reasonable.

IV. Conclusion

The objective in the problem statement has been satisfied. I have successfully detected the lane lines in a video stream containing ideal case(i.e no distortion, no light variation and no shadows) and shown that this technique for lane detection fails for a case containing distortion, light variation and shadows and have successfully used more advanced computer vision techniques to detect lane lines under more real world situations i.e situations containing distortion, light variation shadows etc. I have also successfully calculated the radius of curvature of the two lane lines and also the vehicle offset position.The code for this Project is documented and open-sourced on my GitHub repository. I highly recommend that the reader have a look at the code (link provided in the Appendix) and runs the code. This seems to be the only way to reproduce and verify the results shown in this report, **I also highly recommend to watch my output videos (link provided in results and discussion)**

A. Credits

UDACITY: SELF DRIVING CAR NANODEGREE