University Of Maryland

College Park

Project Report

673-Perception for Autonomous Robots

Project 2

Aditya Goswami
116951968

Mahmoud Dahmani
116777896

Sukoon Sarin
116955522

March 12, 2020

Contents

1 Introduction **3**

2 Problem 1 **4**

3 Problem 2 **6**

4 Using Histogram on Lane for Sliding Windows **8**

5 Generalisation **10**

List of Figures

1 On left enhanced brightness, on right original video 5
2 Finding HSV Values for yellow and white color masking 8
3 Sliding Windows 9
4 Warped Binary Image 10
5 Lane Detection Challenge Video 10
6 Lane Detection Challenge Video with info 11
7 Lane Detection Project Video with info 11

1 Introduction

This project emphasises on computer vision- more towards vision for autonomous cars: Lane detection. This project entails use of image processing tools to process a video and detect the lanes on the road.

- The project has two problems answered.
- First, we aim to improve the quality of the video sequence provided to us in data set for Problem 1. This is a video recording of a highway during night. Most of the Computer Vision pipelines for lane detection or other self-driving tasks require good lighting conditions and color information for detecting good features. A lot of pre-processing is required in such scenarios where lighting conditions are poor.
- This problem was approached with the thought of increasing the brightness of each frame in the video.
- The second problem is somewhat the major hurdle as well as one crucial problem in learning perception for autonomous robots/cars.
- In this project we aim to do simple Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars. We were provided with two video sequences, taken from a self driving car. Our task was to design an algorithm to detect lanes on the road, as well as estimate the road curvature to predict car turns.
- This problem is where we faced the most of our challenges.
- Further below is the detailed report of our steps for each problem.

2 Problem 1

Problem Statement :
Here, we aim to improve the quality of the video sequence provided above. This is a video recording of a highway during night. Most of the Computer Vision pipelines for lane detection or other self-driving tasks require good lighting conditions and color information for detecting good features. A lot of pre-processing is required in such scenarios where lighting conditions are poor.
Now, using the techniques taught in class your aim is to enhance the contrast and improve the visual appearance of the video sequence.
You can use any in-built functions for the same.

- The approach to solve this problem was obvious - increase the brightness of the video by using some techniques that we studied in the class.
- The first technique that we tried applying histogram equalisation method which enhanced the contrast of the dark video also introduced a lot of noise.
- While going further we found that the noise is way too high when using histogram equalisation or normalisation.
- While looking for a denoising method we came across a method called Gamma Correction.
- when twice the number of photons hit the sensor of a digital camera, it receives twice the signal (a linear relationship). However, that's not how our human eyes work. Instead, we perceive double the amount of light as only a fraction brighter (a non-linear relationship)! Furthermore, our eyes are also much more sensitive to changes in dark tones than brighter tones (another non-linear relationship).

  In order to account for this we can apply gamma correction, a translation between the sensitivity of our eyes and sensors of a camera.
- Gamma correction is also known as the Power Law Transform. First, our image pixel intensities must be scaled from the range [0, 255] to [0, 1.0]. From there, we obtain our output gamma corrected image by applying the following equation:

$$O = I^{(1/G)}$$

- Where I is our input image and G is our gamma value. The output image O is then scaled back to the range [0, 255].
- Gamma values < 1 will shift the image towards the darker end of the spectrum while gamma values > 1 will make the image appear lighter. A gamma value of G=1 will have no affect on the input image.

Figure 1: On left enhanced brightness, on right original video

- Now that we understood what gamma correction is and how it works, we used OpenCV, numpy and Python to implement it.
- Following is the images of the output. The one on the left is enhanced video (image) and the one on the right is original video (image).

.

3 Problem 2

Problem Statement :
In this project we aim to do simple Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars. You are provided with two video sequences (both are required for this assignment), taken from a self driving car. Your task will be to design an algorithm to detect lanes on the road, as well as estimate the road curvature to predict car turns.

1. For this problem, a pipeline was suggested. In our simplified pipeline we will process every frame independently from each other. This pipeline will rely on the fact that lanes are parallel to each other and almost always 'vertical', if you look at the image straight from the top. But the actual camera does not look at the road from the top, but it always looks at the road from the same angle. So if we could compute the Homography and distort the image so that the lanes look parallel to each other, we will get the top view.

2. a. Pick one image when the car moves straight and manually extract the coordinates of four points on the lanes(two for each lane). It should not matter which points you select.
   b. Using these points compute the homography(you can use built-in cv2 functions for this), so that the lanes look parallel

3. Despite computed only on a single image, this homography transformation will work for most images, since the tilt of the camera is constant.

4. Using the video we have manually selected 4 points src, which is source point and dst has been chosen randomly to form a rectangle which includes the video without the sky. Now to find the homography, we know that:

5. Theory: You can compute the homography matrix H with your eight points with a matrix system such that the four correspondance points $(p1, p'1), (p2, p'2), (p3, p'3), (p4, p'4)$ are written as 2×9 matrices such as:

$$p_i = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x'_i & y_i x'_i & x'_i \\ 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y'_i & y_i y'_i & y'_i \end{bmatrix}$$

6. It is then possible to stack them into a matrix P to compute:

$$PH = 0$$

Such as:

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y_3' & y_3y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_4' & y_4y_4' & y_4' \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

7. While adding an extra constraint $|H| = 1$ to avoid the obvious solution of H being all zeros. It is easy to use $SVDP = USV^\top$ and select the last singular vector of V as the solution to H.

   The elements of homography, H, is the last column of the V matrix in the above formula.
   Once you have your homography matrix H, you can compute the projected coordinates of any point $p(x, y)$ such as:

$$\begin{bmatrix} x'/\lambda \\ y'/\lambda \\ \lambda \end{bmatrix} = \begin{bmatrix} h_11 & h_12 & h_13 \\ h_21 & h_22 & h_23 \\ h_31 & h_32 & h_33 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

8. In the code, the P matrix is created and its SVD singular value decomposition is done using the inbuilt function `(np.linalg.svd`. This gives us the homography matrix.

9. After finding homography from source coordinates to warped coordinates, we perform warping by multiplying the source coordinates to the obtained homography matrix. The image so far is not GRAY scaled and thus before apply CV2.threshold to convert all the lane marking to white, irrespective of colour, we will convert the image to gray scale. This converts the image to binary image. Image is undistorted. Perspective transformation using homography and getPerspectiveTransform function in openCV.

10. We use the camera calliberation matrix K and distortion coefficients for undistorting the frames.

11. All this is done to convert the image into binary form.

12. A mask is created for Yellow lane detection and another mask is created for the detection of white lane.

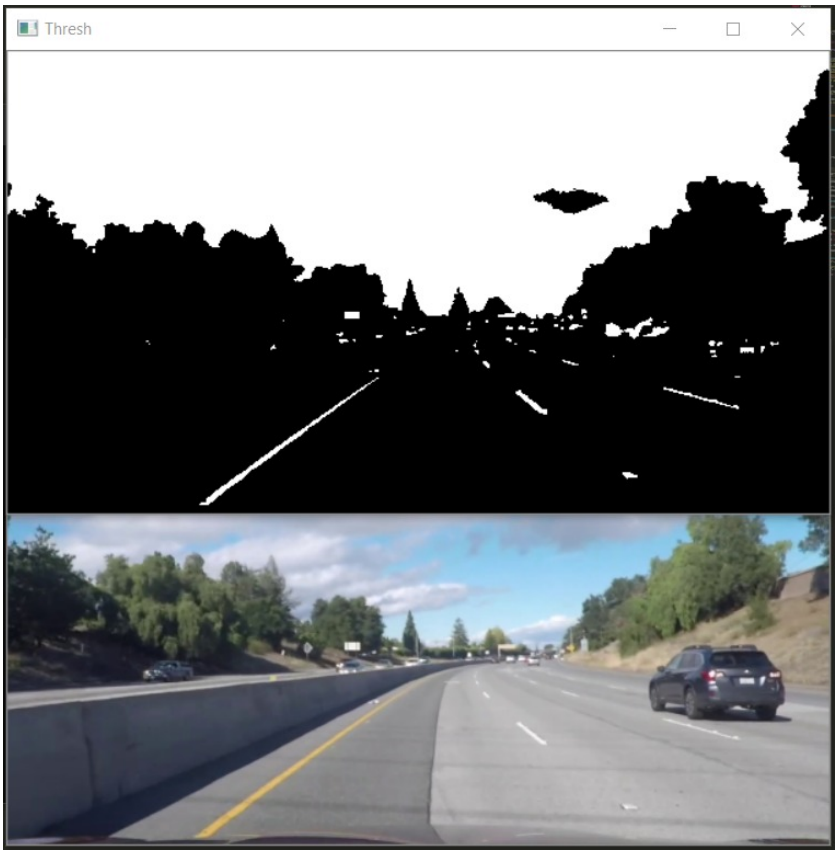

Figure 2: Finding HSV Values for yellow and white color masking

13. HLS colorspace is used to mask out the yellow lane and HSV colorspace is used for masking out the white lane.

14. Here HLS is a scheme of colors where HLS stands for Hue of Color, Lightness of color and Saturation of Color.

15. Similarly HSV is a scheme of colors where HSV stands for Hue of Color, Saturation of color and Value of Color.

16. Both masks are EXORed to form one mask to detect both the yellow and white lanes together from our warped image.

4 Using Histogram on Lane for Sliding Windows

1. We now have a threshold warped image. Now to detect lanes we have used peaks in the histogram generated by the lane pixels.

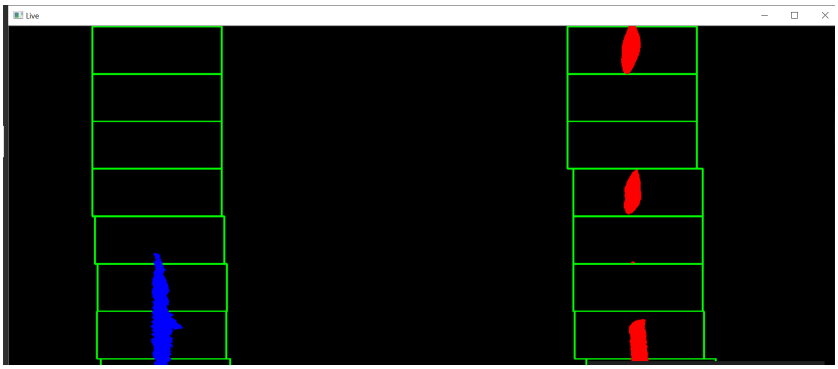

Figure 3: Sliding Windows

2. After per-processing and applying threshold, and perspective transform to a road image, we have a binary image where the lane lines are standing out.

3. We now decide which pixels are part of the lines and which belong to the left line and which belong to the right line.

4. In my threshold binary image, pixels are either 0 or 1, so the two most prominent peaks in the histogram were good indicators of the x-position of the lane lines and the non-zero pixels are picked from along Y-axis.

5. From here we have used a sliding window, to find the lines.

6. The Sliding Window is a rectangular region of fixed width and height that slides across an image.

7. For each of these windows, we have applied an image classifier to determine if the window has a nonzero pixel or not.

8. Since we have a binary image if the pixel is nonzero that means its white and we append it to a queue for either of the half (left or the right halves).

9. The step Size for us mini pixel indicates how many pixels we are going to skip in both the (x, y) direction.

10. We then concatenate the above queues for each window to finally form the left and the right lanes.

11. There were many challenges that we came across and many terms so confusing we had to study other sources as well like Ross Kippenbrock [1] and OpenCV[2].

Figure 4: Warped Binary Image

Figure 5: Lane Detection Challenge Video

5 Generalisation

- The pipeline we have set up is generated using individual channels of the image frame. These threshold values are not specific to color and luminosity. The only variable will be the camera calibration matrix and the distortion coefficients, rest everything can be kept same for detecting lanes in a video sequence.

Figure 6: Lane Detection Challenge Video with info

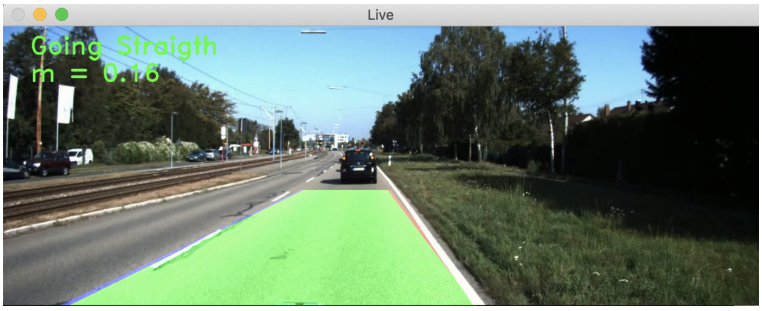

Figure 7: Lane Detection Project Video with info

References

[1] Ross Krippenbrock. *Finding Lanes for self Driving Cars.*

- Finding Lanes for self Driving Cars.

.

[2] ProgrammingKnowledge. *OpenCV Python Tutorial - Find Lanes for Self-Driving Cars (Computer Vision Basics Tutorial).*

- OpenCV Python Tutorial - Find Lanes for Self-Driving Cars.

.