ENPM 673 : Perception for Autonomous Robots - Project 2

Akanksha Patel - 116951029

Aruna Baijal - 116130110

Sri Manika Makam - 116697415

March 12, 2020

Abstract

In this project we aim to do simple Lane Detection to mimic Lane Departure Warning Systems used in Self Driving Cars. We do the lane detection using two methods- Hough Lines and Histogram generation of lane pixels. The outputs obtained in the latter method are good. We also implement the turn prediction in this project.

1 Problem 1

To recognize any lanes or features in our video, we require good lighting conditions and color information to detect required features. Since images/videos don't always satisfy this condition, we apply algorithms and computer vision techniques to improve our video feed quality.

In the data set provided to us for Problem 1, the highway recording is during the night with minimal lighting conditions. Any lane detection algorithm would fail in such a case. One good way to improve the lighting conditions in our video and enhance features is to perform a histogram equalization. In the video feed, most of the intensity values lie within 0-50 value. We want the frequency to be evenly distributed throughout the range. Hence we apply histogram equalization.

On directly applying $cv2.equalizeHist()$, we notice there is a lot of noise in the result. That is because the 'darkness' isn't 'equally dark' everywhere. After applying histogram normalization the resultant values are varying hence the large noise even though the expected values are the same.

To combat this we applied a threshold to our image, set all almost black pixel values to black using $cv2.threshold(image, 10, 255, cv2.THRESH\_TOZERO)$. This function sets all values below the threshold, 10 in our case, to 0 and retains the pixel values above the threshold. Now that the dark background is uniform, on applying histogram equalization we get the desired result, as shown below.

Figure 1: Original frame

Figure 2: Frame after histogram equalization

2 Problem 2

2.1 Step 1: Image Pre-processing

We have applied the following operations on the images before processing the images to detect the lane candidates:

1. **Undistort the image:** We are given the camera calibration matrix and the distortion coefficients for both the datasets. Using these and OpenCV function *undistort*, we remove the distortion in the image.
2. **Remove Noise:** To remove the noise from the images, we use a Gaussian filter (kernel size = (5, 5) and $\sigma_x = \sigma_y = 5$). After this we apply histogram equalization on the image to counter the effects of intensity change.

2.2 Step 2: Segment Lanes

In this section we talk about our approach to segment the lanes. We want to detect lanes which are white and yellow in color. We used color segmentation and segment out the white and yellow segments from the image.

In RGB color scheme, a white pixel is represented by $\{255, 255, 255\}$. Therefore, to extract the white segments, we selected the pixels that have high RGB values. Extracting yellow pixels isn't as straight forward. To extract the yellow pixels, we convert the image to HLS color scheme and threshold the image as per the color range of yellow. The thresholding values used are stated in the table below:

The resultant binary images for Data Set 1 and Data Set 2 are shown in Figure 3 and Figure 4.

	Lower Threshold	Upper Threshold
White (BRG)	(200,200,200)	(255,255,255)
Yellow (HLS)	(10,120,120)	(70,200.250)

Table 1: Thresholding values used for color segmentation

Figure 3: Segmented lanes from Data Set 1

Figure 4: Segmented lanes from Data Set 2

2.3 Step 3: Detect Lanes

In this section we fit a polynomial to the segmented lanes. The lanes are parallel to each other but because of the properties of projective geometry they appear to meet at horizon. Our pipeline relies on the fact that the lanes are parallel. To view them as parallel lines, we compute the required homography and distort the image accordingly. The four points to compute homography are chosen manually. Figure 5 the images from Data Set 1 and Data Set 2 after applying homography transformation.

Figure 5: Parallel lanes on Data Set 1 and Data Set 2

The next step is select the lane candidates, i.e. the pixels in the image above that are a part of the lanes. We follow the following steps to select the lane candidates:

1. Get an initial estimate of x-positions of our right and left lines. For this, calculate the histogram of this binary image along the x-axis and pick the maximum values in the left and right half of the plane. These become our initial x-positions for the window centers.
2. Divide the picture in 20 horizontal stripes. For each strip, select a window of 70x40 centered about the current estimate of the location.
3. Pick all the white pixels that lie in the window and add them to the list of lane candidates.
4. Update the x-positions of the window centers as the mean of the all while pixels in the window.

Steps 3 and 4 are followed for each horizontal stripes (20 times). This gives us a list of the left lane and right lane candidates. Finally, we use $np.polyfit$ function to fit a quadratic polynomial and to generate a closed polygon on the lanes. The resultant image is shown in Figure 6.

As the final step, we warp the image back to its original candidate and use $cv2.addWeighted$ to blend the two images together. The final overlayed image is shown in Figure 7.

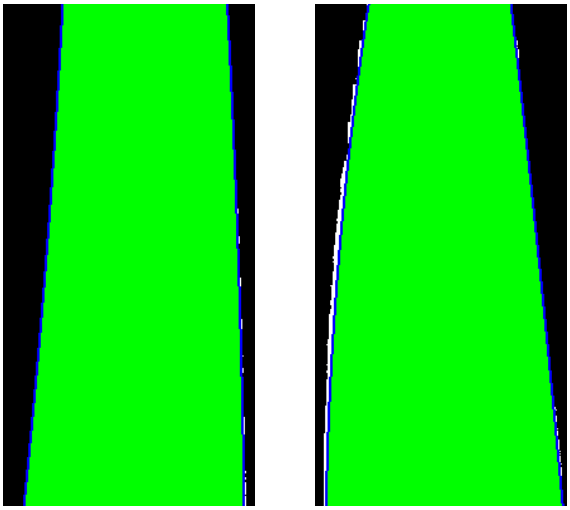

Figure 6: Detecting lanes and generating polygon on Data Set 1 and Data Set 2

Figure 7: Polynomial Overlay on Data Set 1

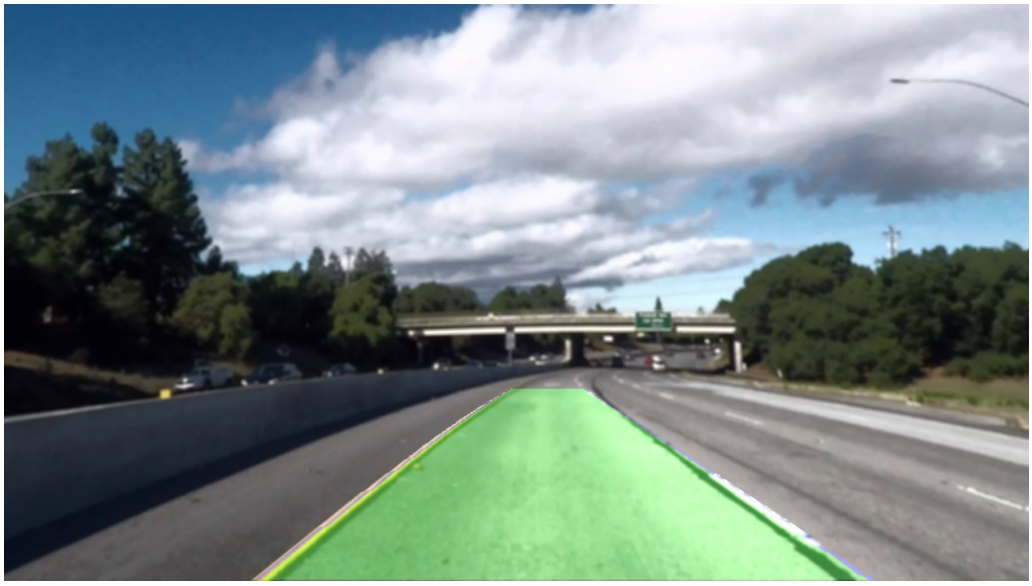

Figure 8: Polynomial Overlay on Data Set 2

3 Turn Prediction

To predict the turns, we take the difference of lane centre and vehicle centre. We calculate the lane center by evaluating the left and right lane polynomials at the maximum Y and find the middle point. We take the vehicle centre as the mid point at minimum Y. Ideally, if the difference between the lane centre and vehicle centre is greater than zero, it's predicted as right turn; if the difference is lesser than zero, it's predicted as left turn and if zero, it's going straight. But in practical situations, we cannot always get the difference as zero when it's going straight. So, we have chosen a range [-10,10], wherein if the difference lies, it's predicted as straight. If greater than 10, it's right turn and if lesser than -10, it's predicted as left turn.

The turn prediction on data set 1 and data set is shown in figures 9 and 10 respectively.

Figure 9: Turn Prediction for Data Set 1

Figure 10: Turn Prediction for Data Set 2

4 Problems faced

1. For Problem 1, directly applying histogram equalization resulted in a lot of noise. There is minute difference in the background intensity though they all appear black. After equalization, these differences get aggravated hence resulting in noise as shown in figure 13.

2. The developed pipeline fails in case of sudden illumination changes. This is visible in the data set 2 when the car passes from under the bridge, the lane estimation is lost.

3. Our first attempt to segment the lanes was using the Sobel filter. We tried different combinations of sobel_x, sobel_y, it's magnitude and gradient but the resultant image contained a lot of noise. Ultimately, we had to shift to color segmentation instead of Sobel.

4. Detection of yellow lanes also posed a problem. Detection of yellow lines using RGB color space isn't a straight forward task. We had to use HLS color scheme in order to effectively detect yellow lanes.

5. To detect the lanes, we first generated Hough Lines from the detected lane marking edges. After providing appropriate parameters and choosing the best detected Hough Line we could draw lines where the lane edges were detected. Using these points, we created a polynomial and then overlayed this filled polynomial on the original image as shown in figures 11 and 12. But the lane estimation we are getting from this method doesn't fit the lane exactly as these are straight lines. So, we implemented the method of using homography and histogram generation of lane pixels for lane detection.

Figure 11: Polynomial Overlay on Data Set 1 using Hough Lines

Figure 12: Polynomial Overlay on Data Set 2 using Hough Lines

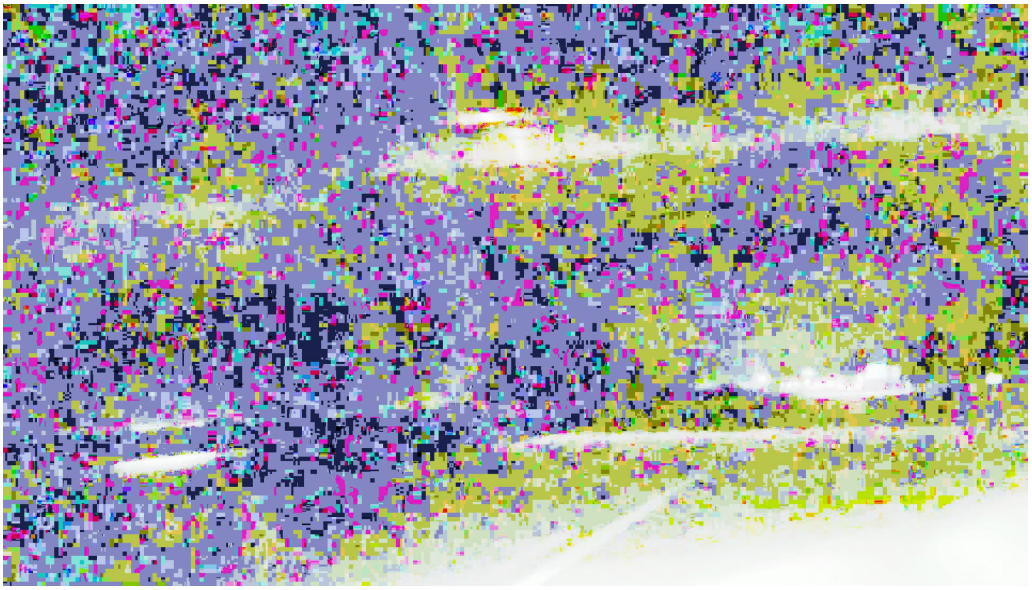

Figure 13: Noise without thresholding in Problem 1

5 Outputs

The video outputs can be seen in the link below:

https://drive.google.com/drive/folders/1Oe3rB3p5MhFcHhUC2igstVb5UC1eVaTI?usp=sharing