ENPM673 - Perception for Autonomous Robots
Project 2

Cheng Chen, Haixiang Fang, Timothy Werder

Due March 10, 2020

Introduction

Detecting lanes lines is one of the up-most importance for cars which use the lane departure warning system and corrects the position of the vehicle with respect to the detected lane. Nowadays, all vehicles come equipped with such a system and hence need a robust lane detection algorithm whatever the lighting conditions on the road might be.
The following project describes a pipeline for lane detection for vehicles. It uses blue and red to highlight the lane edges in videos while predicting the direction in which the car is turning, using OpenCV in Python.

Problem 1 - Improving Imagery

To start we began with a dark and grainy image. We were asked to improved the image. To do this we called the alpha and gamma of the photo, increased the contrast to make dark and light lines more distinguishable. Finally, we used a median filter to remove as much of the salt and pepper noise as possible. As an added touch, to highlight some additional details, we increased the green hue to counteract some of the darker hues. See below for the imagery improvement.

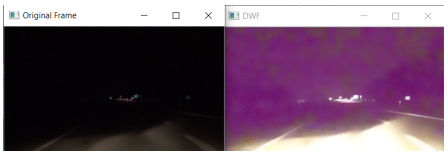

Figure 1: Left: Original Image Right: Improved Image.

Problem 2 - Lane Following

Importing the Imagery

The first aspect to take into account was the difference in parts one and two of the second question for lane following. While some cameras will export videos as video files, some will export frames as a set of still images. For problem two part 2. We simply read in the video. For part one though we had to read through a file and import each individual image as a frame. This was done by looping through the folder and assigning them as an OpenCV supported file.

Correcting and Denoising the Image

After reading the frame from the video, we used the undistort function in opencv to correct the image. The undistort function requires the camera intrinsic matrix and distortion coefficients which can be imported from the given camera parameters. Before performing any mathematical analysis on the image, we also used the GaussianBlur function to smooth out the high frequency noise in the image.

Extracting the Region of Interest

Since we cannot assume that the car does not change lanes, we only cropped the top half of the image by using fillPoly function to build a black mask to eliminate unnecessary environment information such as sky and bridge.

Using Homography to stabilize the Image

The homograhy matrix between the camera and ground plane is assumed to be a constant matrix, because the camera is rigid attached to the vehicle and the vehicle height barely change while driving. In order to obtain the homography for a specific driving scenario, four source points (dash line corners and its parallel points on solid line)are manually chosen from a source image (a typical frame captured from video), shown in Figure 2. Then four destination points are arranged as a 108 pixels by 400 pixels rectangle, according to the US lane specification.

Figure 2: ID extraction on a single tags from data1.

The homography is then compute as following:

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 * x_{p1} & y_1 * x_{p1} & x_{p1} \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 * x_{p1} & y_1 * x_{p1} & x_{p1} \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2 * x_{p2} & y_2 * x_{p2} & x_{p2} \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2 * x_{p2} & y_2 * x_{p2} & x_{p2} \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3 * x_{p3} & y_3 * x_{p3} & x_{p3} \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3 * x_{p3} & y_3 * x_{p3} & x_{p3} \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4 * x_{p4} & y_4 * x_{p4} & x_{p4} \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4 * x_{p4} & y_4 * x_{p4} & x_{p4} \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix} = 0 \tag{1}$$

Elements of h is the right singular vector corresponding to the smallest eigen value, and computed by using SVD (singular value decomposition), where $A = UDV^T$, $A{=}UDV^T$, the smallest eigen vector of A are the element of homography.

$$h = \frac{[v_{19}, ..., v_{99}]^T}{v_{99}} \tag{2}$$

H found now is reshaped into a 3x3 square matrix and normalised, so that the last column last row element of this matrix is one. The resultant unwrapped frame is shown in Figure 3.

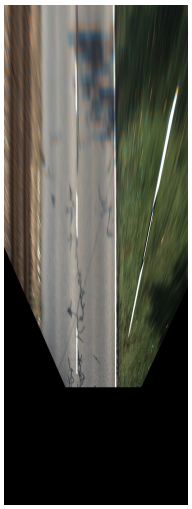

Figure 3: Unwrapped frame

Identifying the Lines using Contours, Canny and Color Masking

Before and after the homography created the flat-field, we use the HSV values of yellow and the the gray scale frames for the road to find all yellow and white lines. We then mask the colors and Canny over the lines. Then using contours we identify all points in a binary image to find the lines.

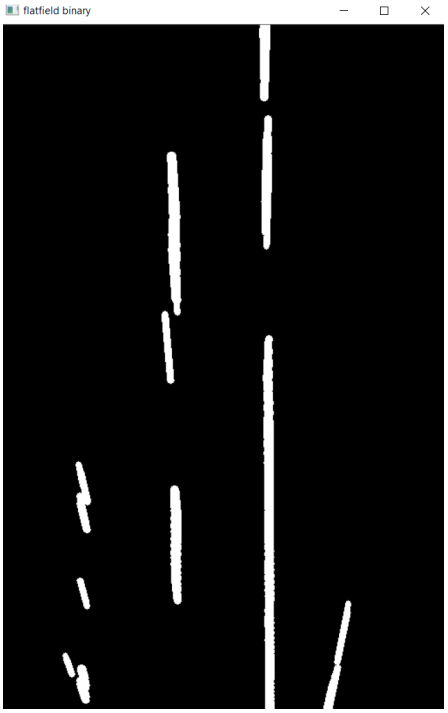

Figure 4: Binary Flat field Image

Using the Histogram to Identify the Lane Lines

After threshold the unwrapped road image, the binary image have two parallel lane lines are standing out. After building a histogram of pixels along x axis, as shown as top one of Figure 5, the x coordinates of one of lines has the highest value. Then, since we know the distance between two line is approximately 108 pixels in unwrapped image, set all histogram value to be zeros give the coordinate of another line, as shown in bottom of Figure 5.

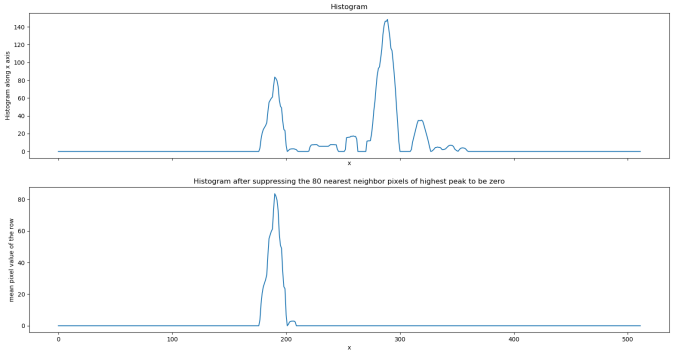

Figure 5: Histogram for lane end detection

Reconstructing the Lanes using Polyfit

After identifying the lines from the histogram, we take those X value points and append them with corresponding y value points. Then using Polyfit we create a polynomial than connects either the left or right line. We then run all of the values of the y axis. We then use the homography to translate these line values into the original frame and impose them onto the image.

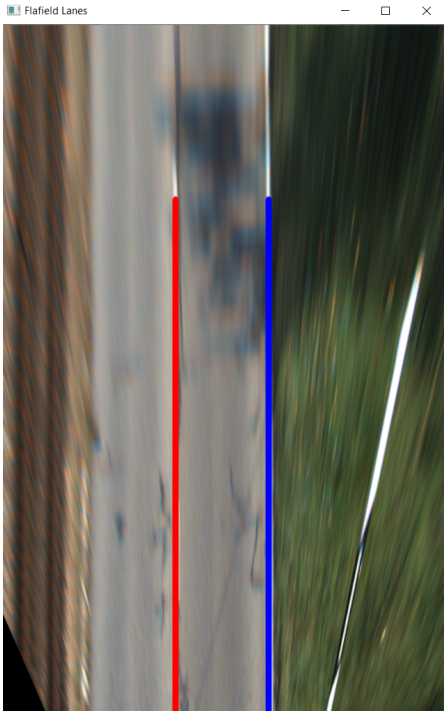

Figure 6: Imposed Lanes on Flatfield Image

Figure 7: Imposed Lanes on Original Frame

Determining Turning

Using the top and bottom of the each of the lane lines, we then Identify the midpoint at the top and bottom. Using Polyfit once again to find the line that characterized the middle line between the lanes. We then checked and summed the gradient to determine whether the car was turning left or right.

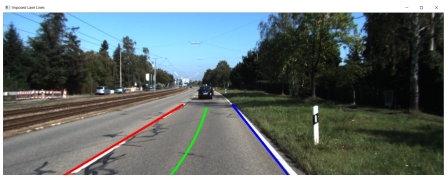

Figure 8: Turning / Mid-line imposed in Final Image

Problem 3 - Lane Following Challenge Video

The second part of the second problem involved two aspects that were different from the first lane follower video. As with the first video we developed our lane finder to find both yellow and white lines. We were unable to correct for the lines disappearing for a couple of frames due to the darkness of the overpass. However, we were mostly able to use the same procedure and functions we used in the the first part with some minor tuning.

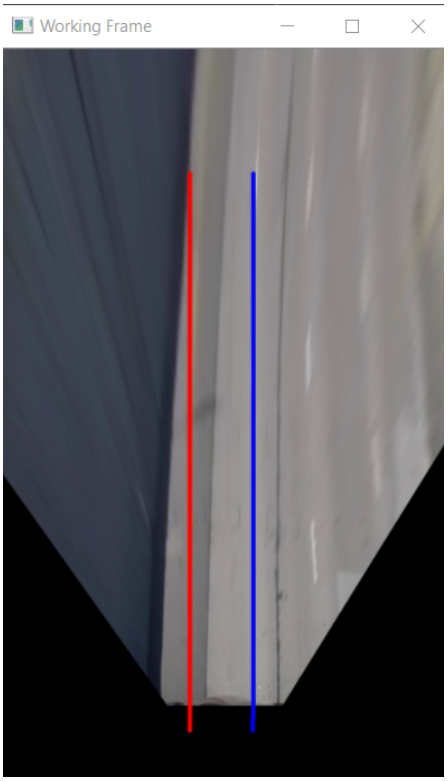

Figure 9: Imposed Lanes on Flatfield Image Challenge Video

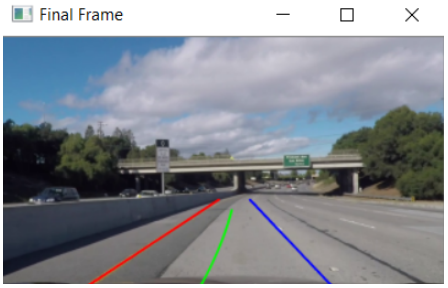

Figure 10: Turning / Mid-line imposed in Final Image Challenge Video

Conclusion and Lessons Learned

This project really emphasised our strengths that were our weakness from the last project. For this project as a team we setup a Github and setup update-commit-push -pull protocols through Pycharm to quickly and efficiently modify our project. This also helped with enabling team members to work independently.

Some issues we encountered mainly surfaced with with trying to get the functions to work for both the the first and second problems. Eventually we were able to make them all work, though some parameter tuning was necessary based on the position of the of the camera in the vehicle.