Advanced Lane Line Finding

Samuel Navarro

May 30, 2019

Contents

1 Camera Calibration

The code for this step is implemented in the third and fourth cell of the jupyter notebook in the functions `show_image` and `calibrate_camera`.

The output of `calibrate_camera` gives us the objpoints and the imgpoints arrays. I then use this arrays to obtain the distortion coefficients and the camera matrix to obtain the undistorted image.

An example is in the Figure 1 and 2 1

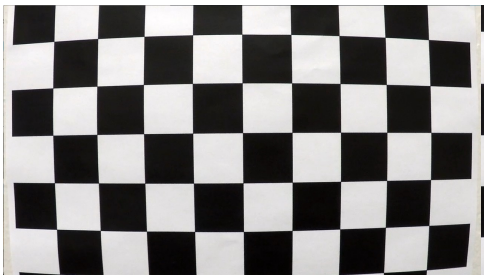

Figure 1: Original

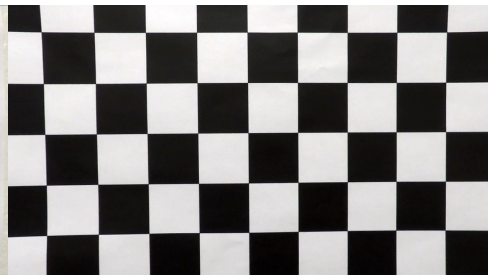

Figure 2: Undistorted

2 Pipeline Images

2.1 Undistorted test

We can see the distortion correction being applied in one of the test images:

Figure 3: Undistorted test

2.2 Perspective Transform

The perspective transform is in the function `perspective_transform` in the same notebook.

The parameters I used was hardcoded. This is obviously something that can be improved. The most straightforward way I can think of based on what we've learned is applying Hough Transform in the image and apply the perspective transform on that.

The result is in the Figure 4:

Based on the reviews and the suggested changes, I made changes in the `perspective_transform` function. The result is on Figure 5:

2.3 Color transforms and gradients

Then, with the warped image at hand, I used a combination of color and gradient thresholds. The code is in the function `color_pipeline`.

I figure out that when the road is clear, the s_channel is better to find the lanes but when the road is dark because of some shadow caused by the threes, the l_channel was better.

Because of that, I used a combination of l and s channel images with and without x gradient.

The result can be found in the Figure 6

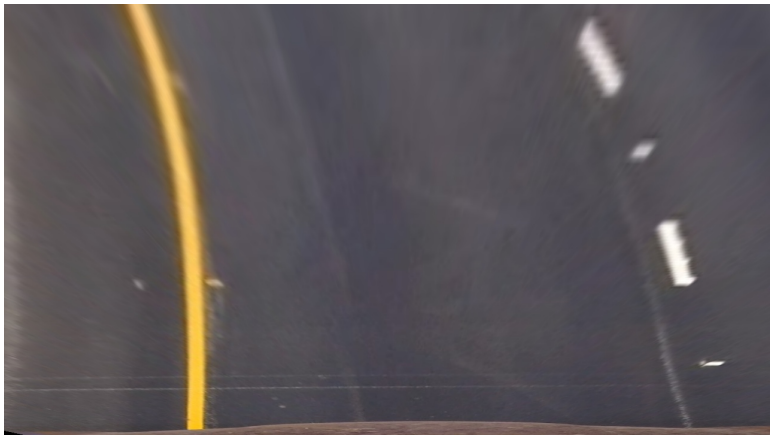

Figure 4: Warped

Figure 5: Warped Straight Lines

2.4 Fitted Lane Lines

The functions I used to fit the lines was `search_around_poly`. The result can be found in the Figure 7.

For the video I used the `search_around_poly` but for illustration I used the `old_search_around_poly` function to illustrate the windows.

2.5 Curvature

The curvature is being measured in the function `measure_curvature_real`. The useful pixels of the images were taken from the warped image so I endup with:

- `ym_per_pix` = 30 / 720

Figure 6: Combined Binary

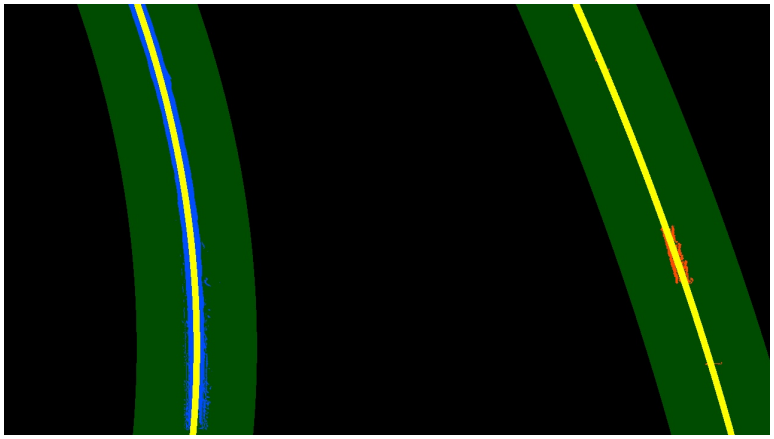

Figure 7: Fitted Lines

- `xm_per_pix` = 3.7 / 700

The location of the vehicle was taken from the fitted lines. I computed the distance from the left line to the center of warped image. Also, I calculated the distance from the middle of the warped image to the right line. If the difference between this to values is greater than zero, the vehicle is on the left and if the difference is less than zero, the vehicle is on the right.

The final output is in the Figure 8

Here's a to the output video

3 Discussion

- One of the problems I encounter was the fact that you have a difference in the frames when you are trying to diagnose the problems. It seems

Figure 8: output

obvious when you know you are dealing with the undistorted frames but at the beginning I was trying to diagnose the problems with the original frame but applying the functions into the undistorted images.

- As in the previous project, I would like to implement this in C++ and check what are the differences.
- Another thing I could do to improve the pipeline is to compute the Exponential Moving Average of the fitted lines to give more weight to the recent frames and to update the lines based on the EWMA of the lines.
- Also, I wanted to provide a video with 4 different frames with the different process being showed in the video but I didn't have enough time.