Advanced Lane Finding P4: Andriy Kumanovskyy Write-up

Camera Calibration

The camera calibration procedure was rather straightforward – I largely taking influence from the class lessons. The function “findChessBoardCorners” was used to find the coordinates for each of the corners in the image in order to be able to be transformed into an un-warped version. The function “calibrateCamera” was then used to calculate the camera matrix, rotation as well as serve as a pre-requisite for the next step – undistorting.

Undistort Images

After calibrating, the camera matrix was applied to undistort an image via the “unditort” fuction. The image used was the same one from the class project (it was found online), since this image was very clearly distorted:

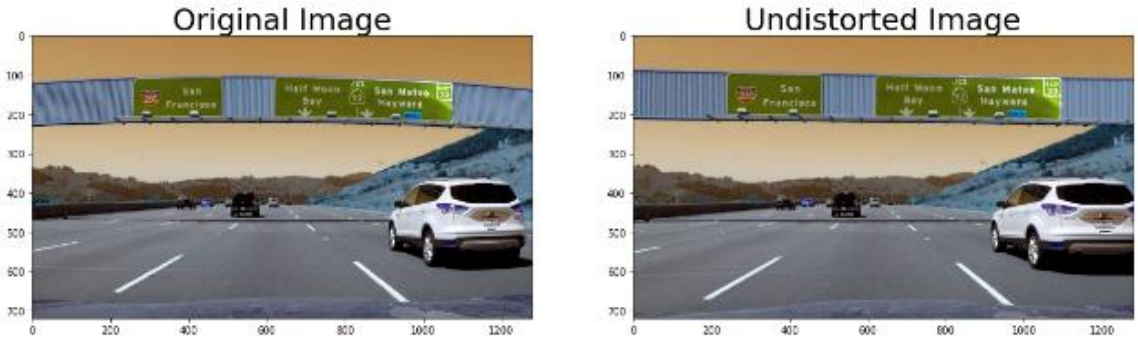

Figure 1: Original Image & Undistorted Image (for camera calibration)

The before and after for this image can also be found in the Jupyter project file or HTML file very close to the beginning of the project, as the third visible set of images titled “Original Image” & “Undistorted Image”.

Perspective Transform

Next, the “getPerspectiveTransform”, & “warpPerspective” functions were applied to generate an overhead birds’ eye view of the image. The SRC & DST coordinate arrays were fed in order to generate such an image. The Images are titled “Undistorted Image” and “Unwarped Image” in the Jupyter notebook just after the camera calibration. It should be noted that for the rest of the project, the “Unwarped Image” will be referred to as “Original Image”, since it will be the original image that various different sobel transforms will be taken place such as magnitude, gradient, HLS colour space, ect.

The Undistorted and Unwarped images are presented on the next page:

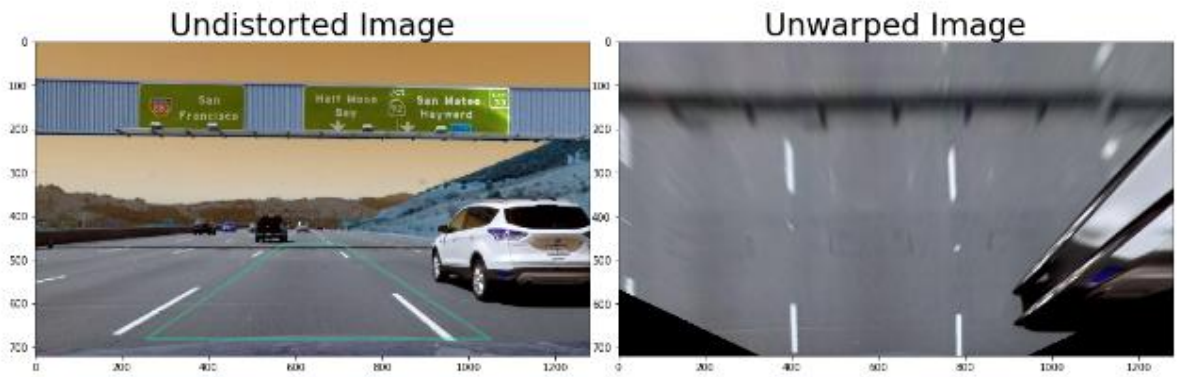

Figure 1: Undistorted Image & Unwarped Image (Starting point for binary image processing)

Binary Image

The next stage of this project would be to create a robust method for detecting lane lines and rejecting anything else that is not relevant. There are various different ways this can be achieved, ranging from HLS colour space, to more advanced gradient, directional and magnitude approaches utilizing sobel (expand)

The image used as a starting point for all of these outputs is the unwarped image presented in figure 2.

I utilized the **ipywidgets** library for interactive sliding of various parameters such as threshold to minimize the amount of time doing empirical works (as fun as it may be for me, this project implementation was a bit long so I wanted to save time there).

The final approach was to utilize several the following:

1. HLS (L) threshold

2. LAB (B) threshold

3. if this combination was not able to succeed, the next steps would be to include more.

However, as you will be able to see, this approach was able to complete both the project video and the challenge video (the harder challenge video was still a bit too much).

Under each of the outputs of various binary processing (after figuring out the ideal setting) I printed out the best settings that I found in order to be able to set them up again (every time I open it seemed to not remember my settings, look into this if time permits).

Lane Lines

As inspired by the class lesson, the function “sliding_windows_polyfit” was used to identify the left and right lane lines. This function works utilizing a histogram of the bottom half of the image as a starting point. Furthermore, the function iterated through the windows to identify the rest of the lane line through the “sliding_windows_fit_polynomial” function. The process as well as outputs can be found in the jupyter notebook and/or html output file.

Lane Curvature

As inspired by the class lesson, the “fit_cr” function was used to calculate the curvature:

left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])

This function applied a mathematical transform taught in the lesson to calculate the curvature via the x & y coordinates. Additionally, the average of the radius is calculated along with how far the vehicle has drifted left or right from the centre of the lane.

Problems/Issues/Discussion

This project was very difficult to implement in terms of engineering, since so many lines of code needed to be written and troubleshooting errors took weeks. The first approach was to utilize a polynomial fit not based on the lesson, however since it was giving too many errors, I decided to take a more direct approach of following the lesson. The process became easier to troubleshoot as well as tune thresholds once I utilized the **ipywidgets** “interact” library. This process helped me refine the many different lighting conditions which were especially tricky to perfect in terms of analysis and trial and error. Some improvements can be made to utilize more colour spaces to be able to do the “harder challenge” video, as well as smooth the algorithm on different levels. Perhaps this can reduce some of the flickering I experienced.