Advanced Lane Finding Project

By: Sraven Saradhi

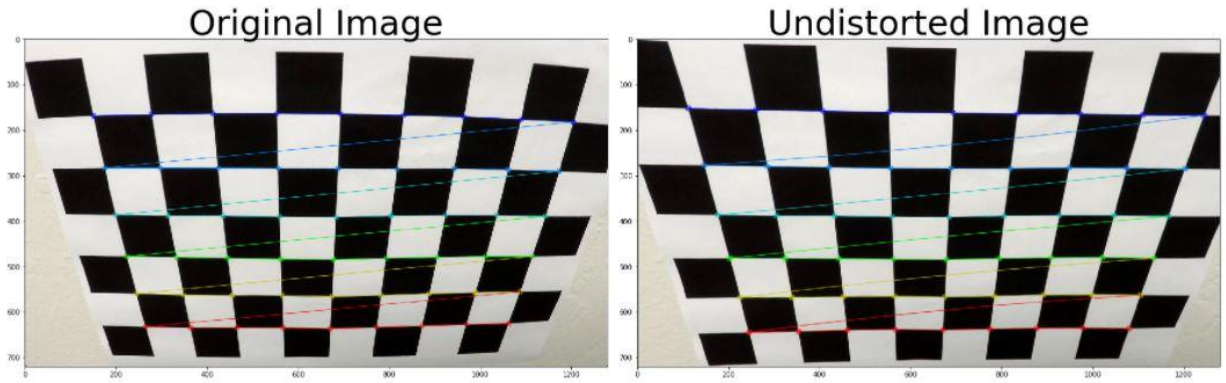
The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

1. Camera Calibration:

For this part of the project, the calibration was performed in the first cell of the python notebook. I start by noting the number of inside corners and outside corners of the chessboard. From these images, I know that the number of x corners are 9 the number of y corners are 6. I then move onto converting the image into grayscale, this is then passed into the find corners and draw corners functions.

I early on created 2 arrays for storing the object points and image points. The object points would be from the mgrid function, and the image points would be from the corners return value. This would then be used to calibrate the camera, once the camera has been calibrated the image would be undistorted. For comparison, an example has been shown below.



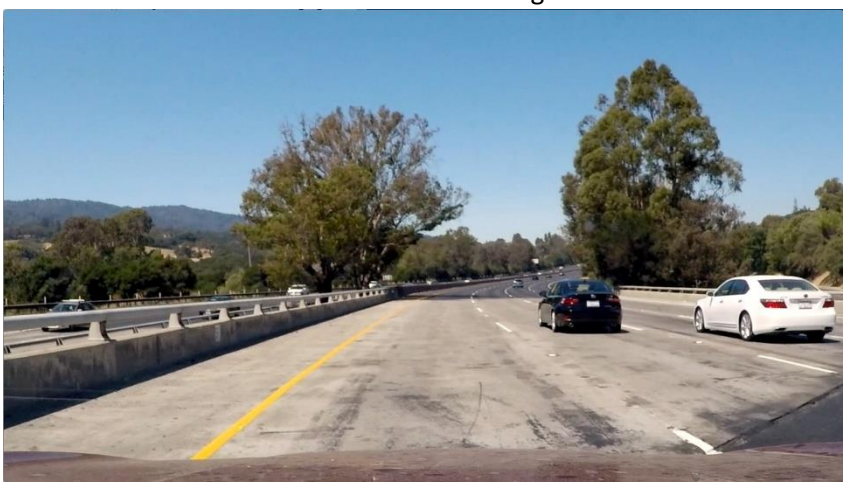
After the camera calibration has been completed the mtx and dist coefficient values will be used to undistort the images from the lanes.

2. Pipeline for images:

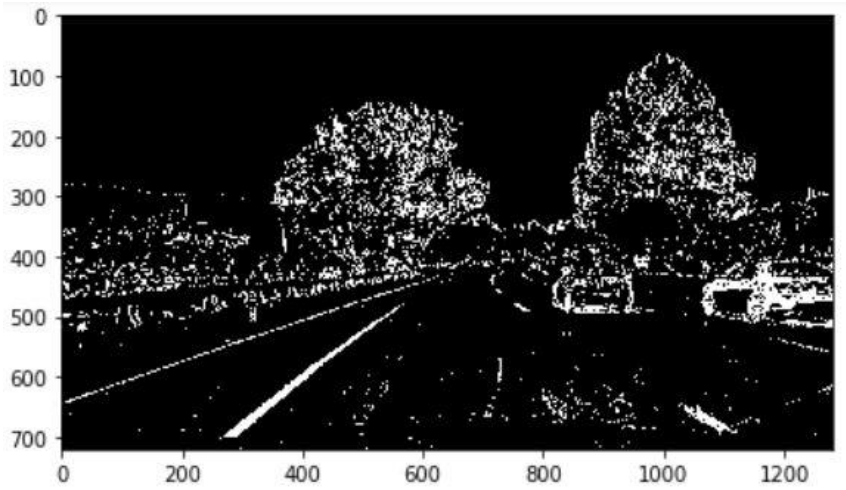
Step 1: image conversion into combined Sobel and s channel:

In this step the images for each image in the test images is sent through a for loop into the pipeline, the for loop starts by converting the images into grayscale and s-channel. The combination of grayscale with the s channel in the HLS conversion gave the best result for the lane lines. Below is an example of the original image and the conversion after into the combined version.

Before distortion correction and thresholding:



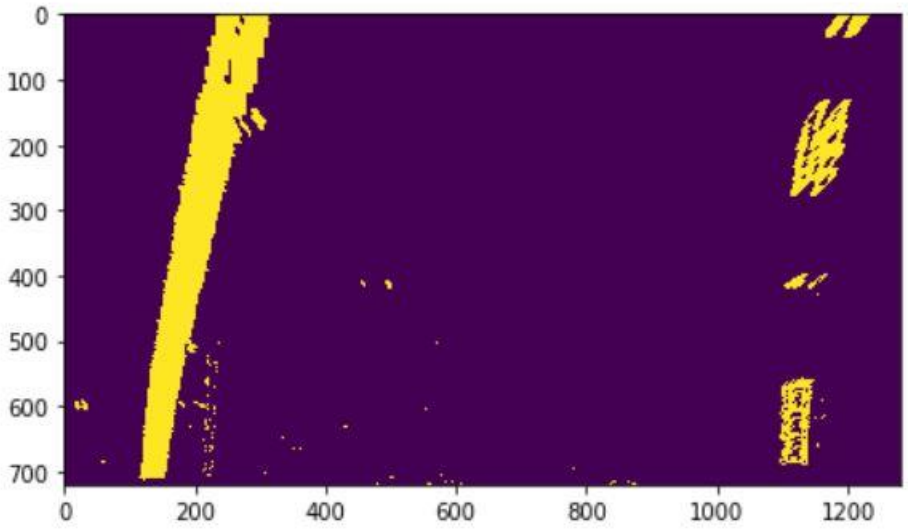
After distortion correction and thresholding:



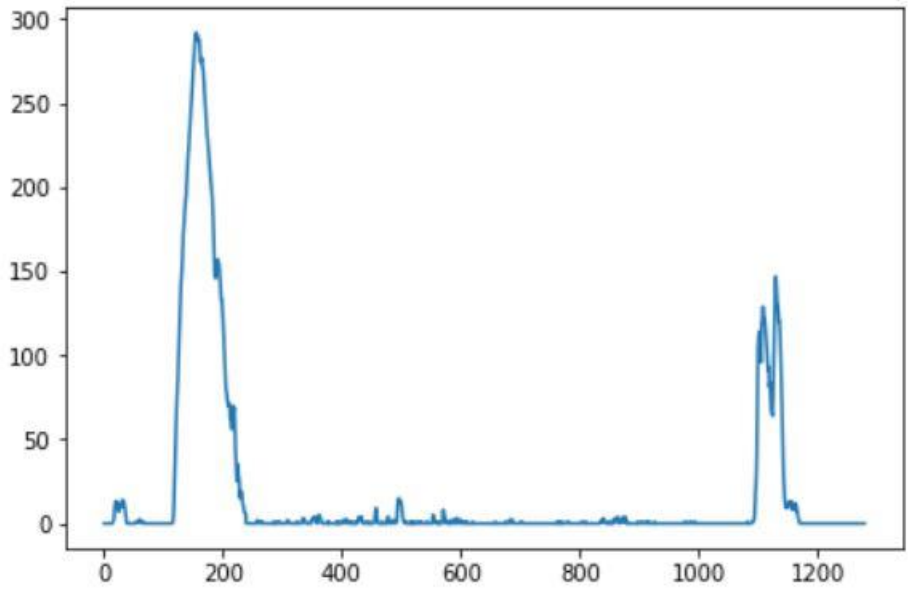
Step 2: Perspective transform and Histogram:

After the images have been corrected for, it was time to identify the curves within the lane lines. To better apply a 2nd order function, we need a 2d bird's eye view of the image. After identifying the original image points and the destination points, I am interested in. The perspective transform was performed with the resulting warped image saved as warped. An example image is shown below:

Warped image:



I now have areas of high pixel density and brightness, with areas of low pixel density and brightness. I can then get a histogram of the pixels and the peaks within the histogram plot correspond to the highlighted lanes; the histogram plot can be seen below:

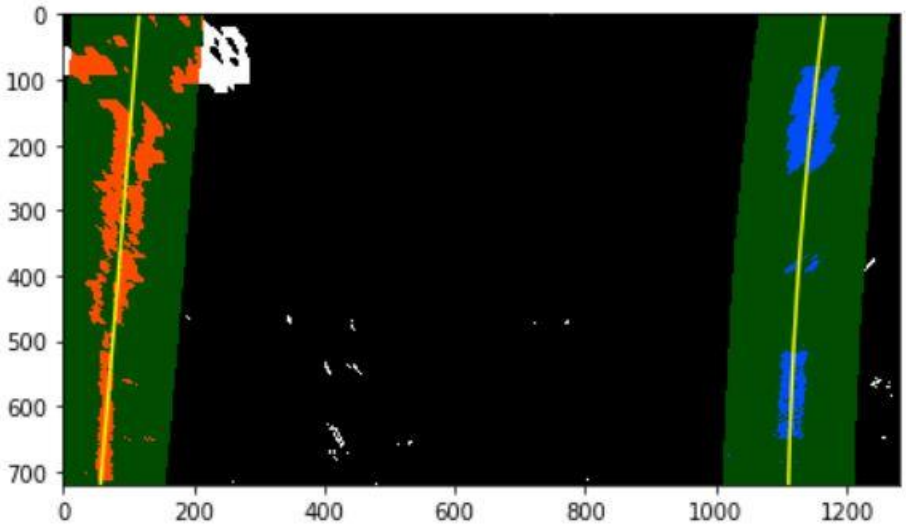


The left peak corresponds to the left lane and the right peak corresponds to the right lane.

Step 3: sliding window and 2nd order :

To set the 2nd order polynomial that would dynamically change to the changing lane, I need to set some parameter like the number of windows, margin size and minimum number of pixels within the box. For each window, the box dimensions need to be specified. The position of the pixel will be adjusted based on the new mean if the pixels are greater than the minimum pixels.

After the window size and dimensions are set, the polyfit will be applied and the 2nd order polynomial will be applied. The lines along with the highlight is then applied and placed onto the image giving the following look:



Step 4: Radius of curvature and overlaying onto the actual image:

The left and right radius of curvature in meters for test_images/test1.jpg respectively 713.883 & 1992.148 . This curvature was calculated based on the derivatives of the 2nd order equation. The simplified version of this formula is shown below:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

The radius of curvature is initially calculated in pixels but is then converted to meters based on pixel to meter ratio. The position of the vehicle is based on the offset and is given as 15m.

When overlaying the image onto the road, the inverse of the perspective transform is used, and the result is shown below:

Step 5: Final Video:

The final part of the project involved creating a processs_image function containing the entire pipeline except camera calibration as that is separate to the road images. The final video is shown in the final cell of the notebook

3. Discussion:

I took the step-by-step approach mentioned above to complete the project. I would take the advantage of camera calibration, color combination, gradients, perspective transform and finally 2nd order polynomial fitting. The result is then overlayed onto the actual image and videos. The color combination and 2nd order polynomial fitting along with perspective transform worked as it would account for factors such as variations in light and shadow. This would make the algorithm more realistic to real work scenarios, the perspective transform improved the 2nd

order polynomial fitting which allowed for the algorithm to better adjust to the curves seen on the road. Overall improved performance to real world scenarios.

The areas in which the algorithm might fail would be in areas where the road colors and amount of light varies simultaneously and to an intense degree. To improve advanced lane detection further, I would add more color channels into the combined gradient and tune the parameters to adjust for any abrupt changes.