Advanced Lane Lines Project

Camera Calibration:

The first step to measuring real values from a camera is the calibration process. With images taken by the camera of a checkerboard from multiple angles and distances, a camera can be calibrated. Given those images 3D points (Object Points) in real space are compared with 2D points (Image Points) in real space. The OpenCV method calibrateCamera will take the Object and Image points and give you the Camera Calibration matrix and the distortion coefficients necessary to distortion correct images with the undistort method.

Here is an example of the original image of a checkerboard, and the undistorted image.

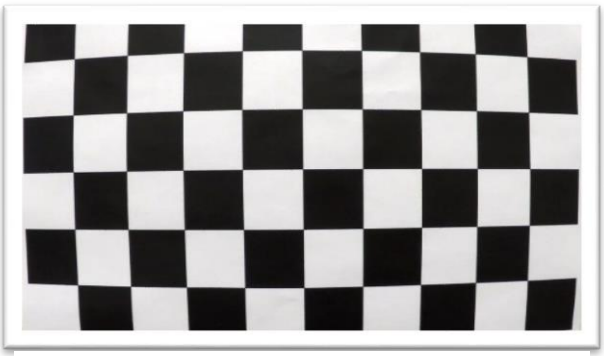

Figure 1: Distorted Image

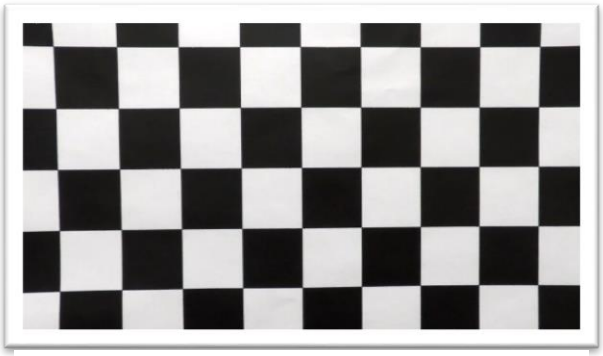

Figure 2: Undistorted Image

Pipeline

At the start my image pipeline, the first step is to distortion correct images.

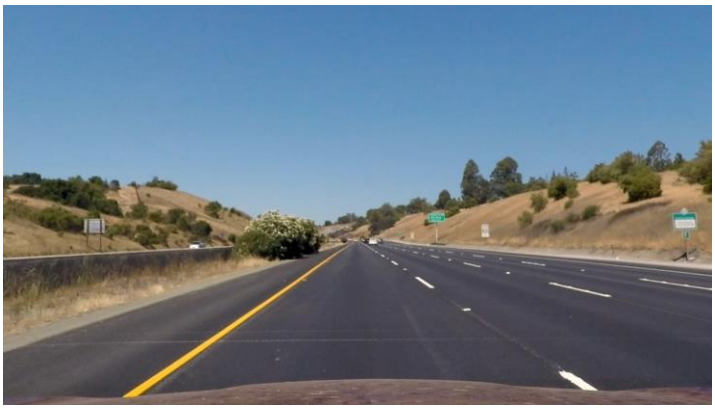

Figure 3: Undistorted Lane Line Image

With the distortion corrected image, we can use color transforms, gradients in the x and y directions, as well as the direction of the gradient with thresholding to get a binary image result of the lane lines.

I tried a combination of thresholding many different things from changing the color space to HLS, and using each channel with thresholding to see what would produce the best most robust lane line image for the images and video provided. What I found to be the best result was thresholding the lightness and the saturation, along with taking the gradient a gray scale image.

The result is as shown below.

Figure 4: Processed Image

Transforming the Image

After a good processed binary image of the undistorted image was found, I performed a perspective transform on the image. In order to perform a perspective transform, you need to choose four source points and four destination points, where the source points are rectangle on the same plane as the road, where the lane lines are parallel with the line joining the points. The Destination points are the four points the source points will be mapped to for the perspective transform. We can get a transformation matrix by using the source points and the destination points that will transform an image to the desired position. We wish to transform the image to a birds-eye view of the lane in order to find the lane lines.

I verified that the Source points I choose worked well by drawing the destination points onto the transformed image and checking if the lines drawn are parallel to the lane lines.

Here is an example.

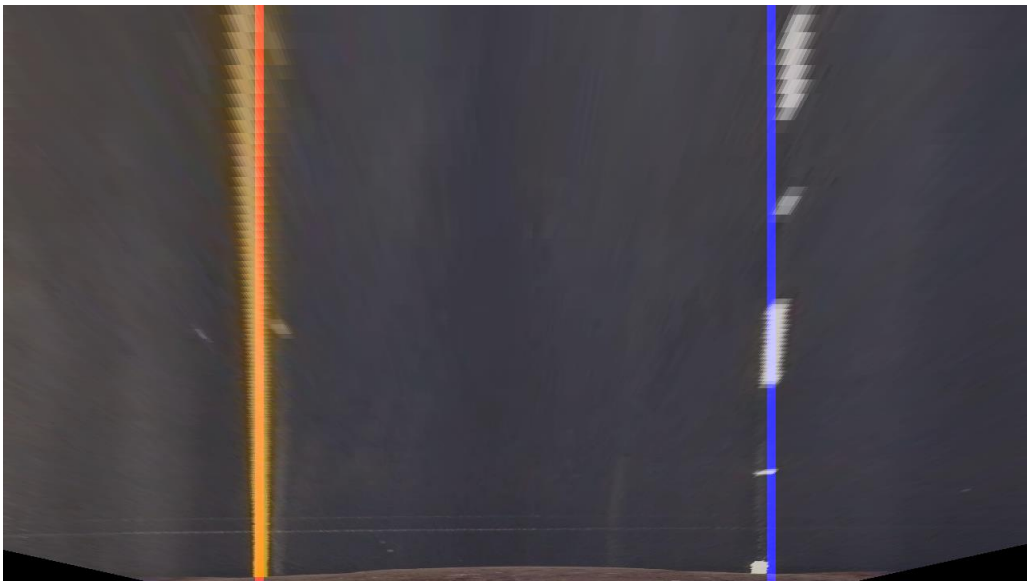

Figure 5: Perspective Transform Check

The transformed processed lane line image is also shown below.

Figure 6: Perspective Transformation of processed Image

Finding the Lane Lines

With the transformed processed image, the lane lines can be found by taking the histogram of the column values for bottom half of this image, the peak value of this histogram will be a great starting point to find the center of the lane. The peak on the left half of the image corresponds to the left lane, while the peak on the right half of the image corresponds to the right lane.

With that center value, the image can be segmented into windows vertically (in my case I used 9 windows), where pixels within a boundary of the current center prediction can be counted. Pixels within that window are considered part of your lane line, where if the number of pixels surpasses a set level, a new lane center can be calculated. Once this is completed we will have a set of lane points, which can be fit to a second order polynomial in order polynomial representing your lane.

Below is an example of the polynomial found using the above image.

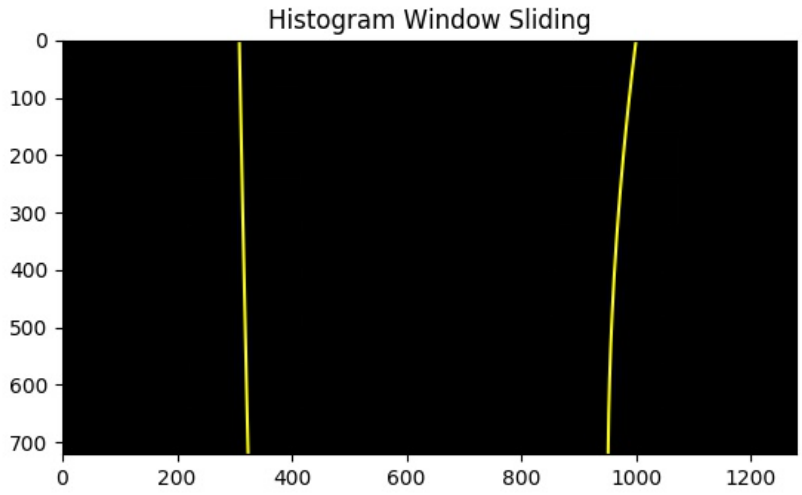

Figure 7: Lanes found in Perspective transformed image

With the points in the lane line, we can also convert the pixel values to meter values and fit them to another polynomial representing the lane in meters. With this lane, we can perform the following calculation using the point at the bottom of the image (where the car is) as an input. A and B are the polynomial coefficients, and y is the values for the bottom of the image.

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

With this result, we can transform the lane line back to the original lane with the inverse matrix and draw in between both lines to get a final result.

Figure 8: Final Image of Lane Lines

Pipeline

Running the process on a video will provide a solution to track the lane lines on every frame, but the output is still jittery. To solve this problem we can keep track of the lane line values found for the past few frames and average them to smooth the output.

The result of the video is name resulting video, and can be found on github, or in the submitted zip folder.

Discussion

Testing out the different processes I can use to try and highlight the lane lines took a lot of time to find a robust enough process that can handle all the issues in the video, from the shadows and the color of the lane changing. I do feel that my process is still limited and with further playing around, a more optimal solution could be found. Thresholding the image with different values and seeing what artifacts are still present in different values was a lengthy process. Trying to identify a way to find a more optimal process would one way to improve this pipeline.