Advanced Lane Finding Project

By: Tyler Zamjahn

APPROACH

In order to develop an image pipeline, I adapted code from the Udacity lesson plans and tested each step in a jupyter notebook. I used the notebook to finetune the pipeline, and produce the images used in this report. After testing, I then created a python file that eliminated some of the unnecessary image displays and condensed the pipeline to produce the video results. I saved the calibration information using the pickle module to transfer it from the jupyter notebook to the python file.

CODE

Camera Calibration

In order to calibrate the camera, the chessboard calibration files were read in using OpenCV and then processed using cv2.findChessboardCorners() and cv2.drawChessboardCorners() to verify the corners were being found. This was done in a jupyter notebook and an example of original chessboard image, the corners being drawn on that image and then finally the undistorted image using cv2.calibrateCamer() and cv2.undistort() are shown below.

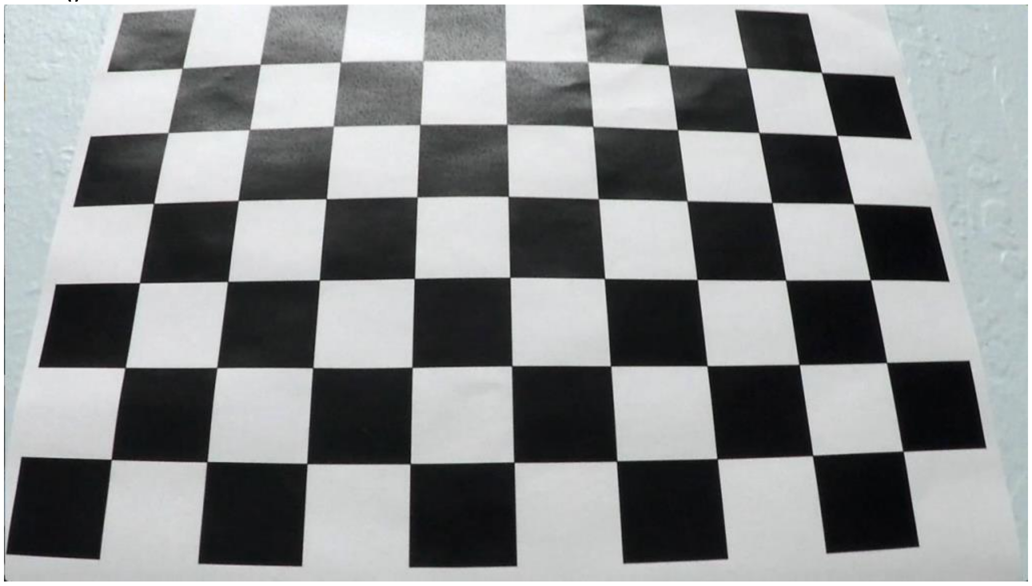

Figure 1: Original Image

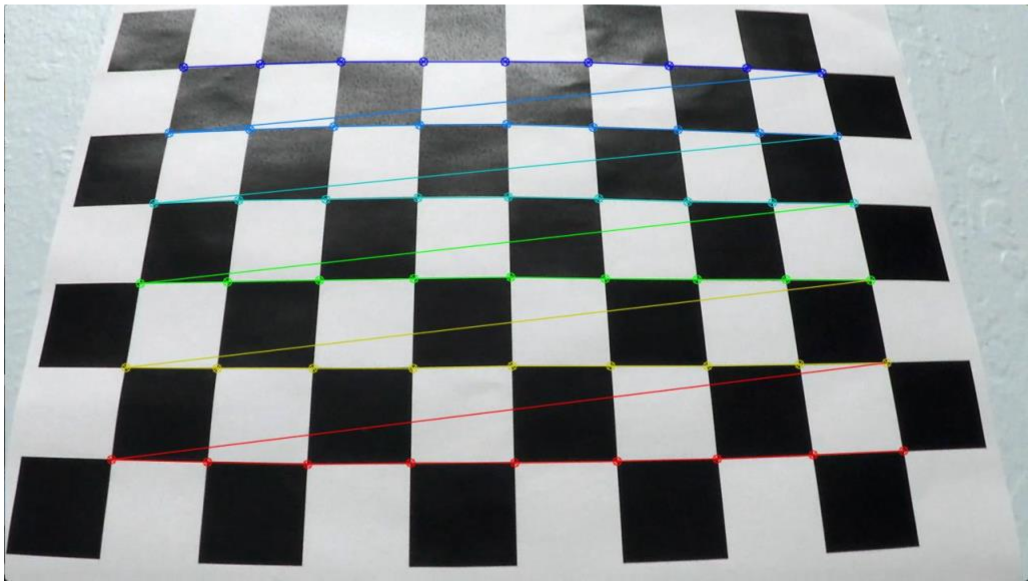
Figure 2: Corners drawn on the Image

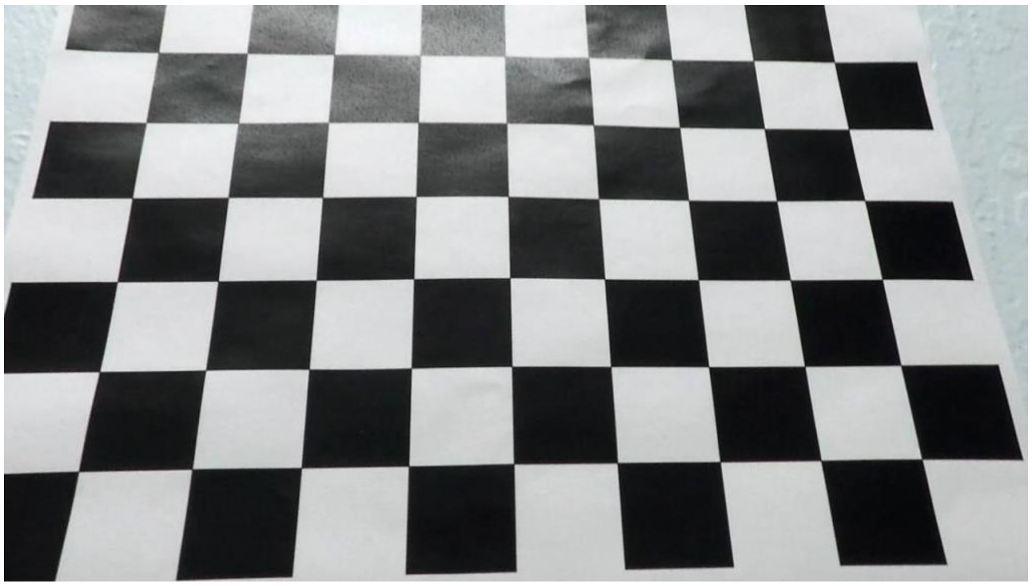
Figure 3: Undistorted Image

Undistorted Images

Once the calibration information was found using the chessboard images, the same calibration matrix and distortion coefficients were then applied to images from the roadway using cv2.undistort(). An example original and undistorted image are shown below.

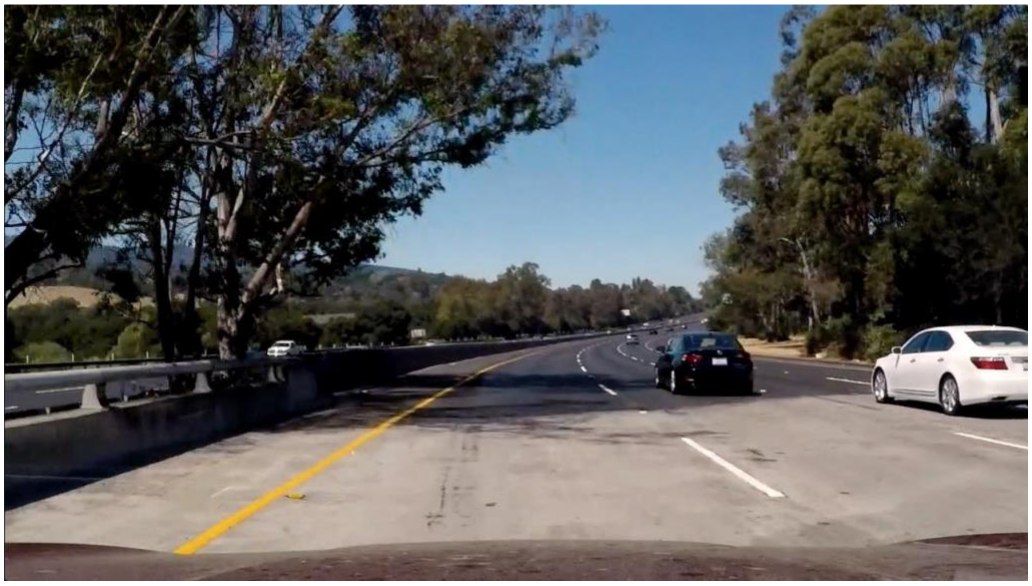
Figure 4: Original Image

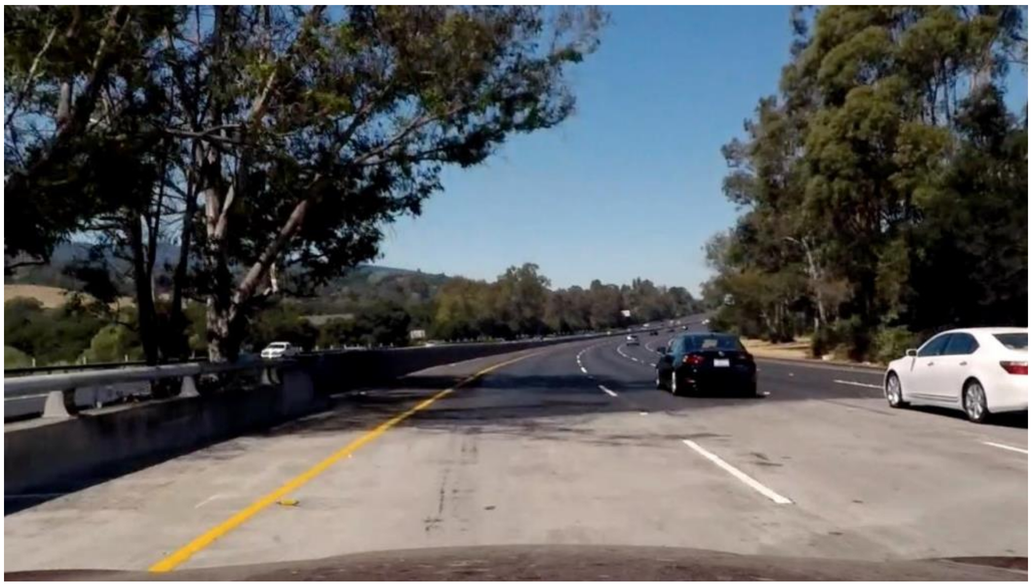
Figure 5: Undistorted Image

Creating a Bird's Eye View

After undistorting the image, four source points (shown below) were then identified on an image with straight parallel lanes as a region of interest to highlight the lane lines. These four points are shown below and were used to warp the image in to a top down view using cv2.getPerspectiveTransform() and cv2.warpPerspective(). The destination points were chosen such that the source points were mapped to straight perpendicular lines to the x axis of the image. The destination points were chosen so that they were equidistant from the outside edges of the image so that the location of the vehicle in the lane could be accurately measured.

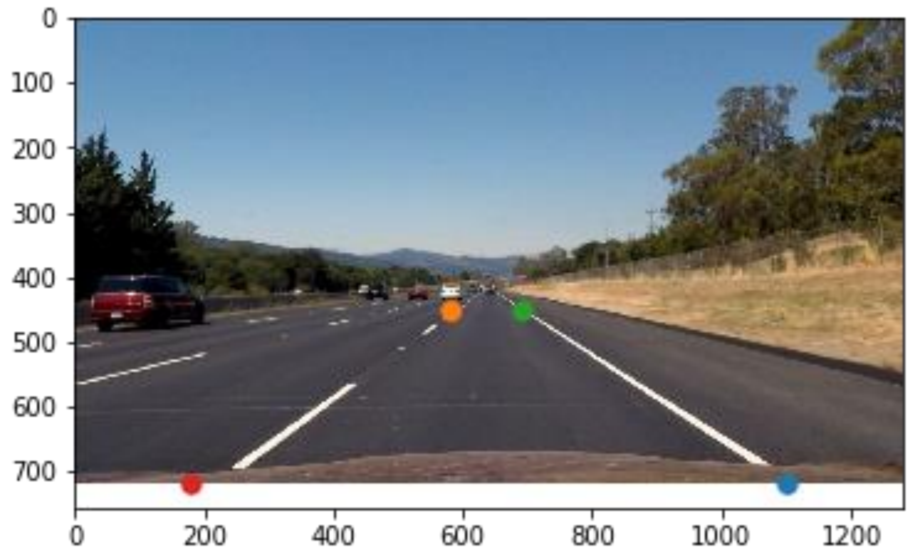

Figure 6: Source Points

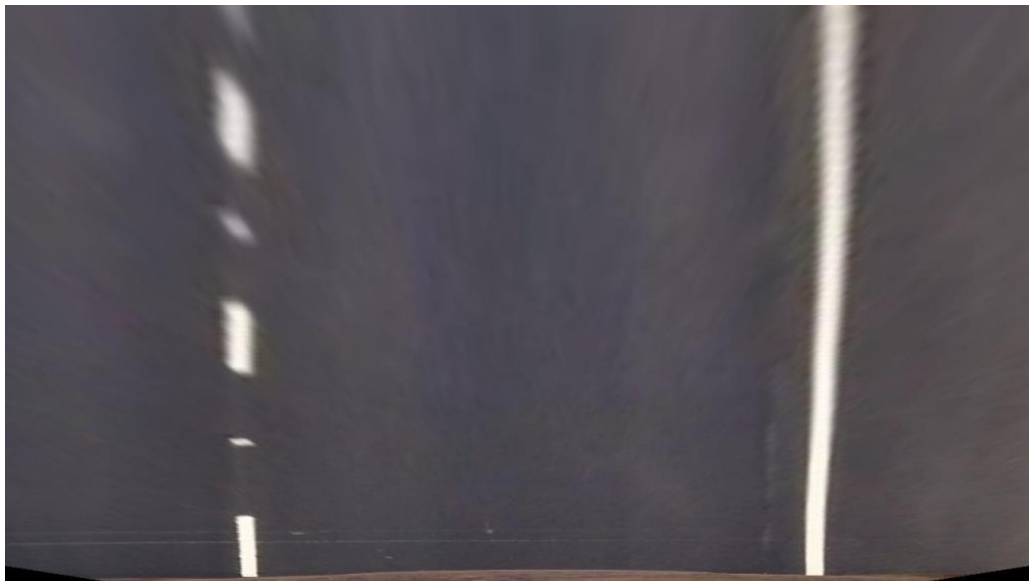

Figure 7: Bird's Eye View after Warping to Destination Points

The image below shows the inverse mapping from the warped, destination points, to the unwarped, source points.

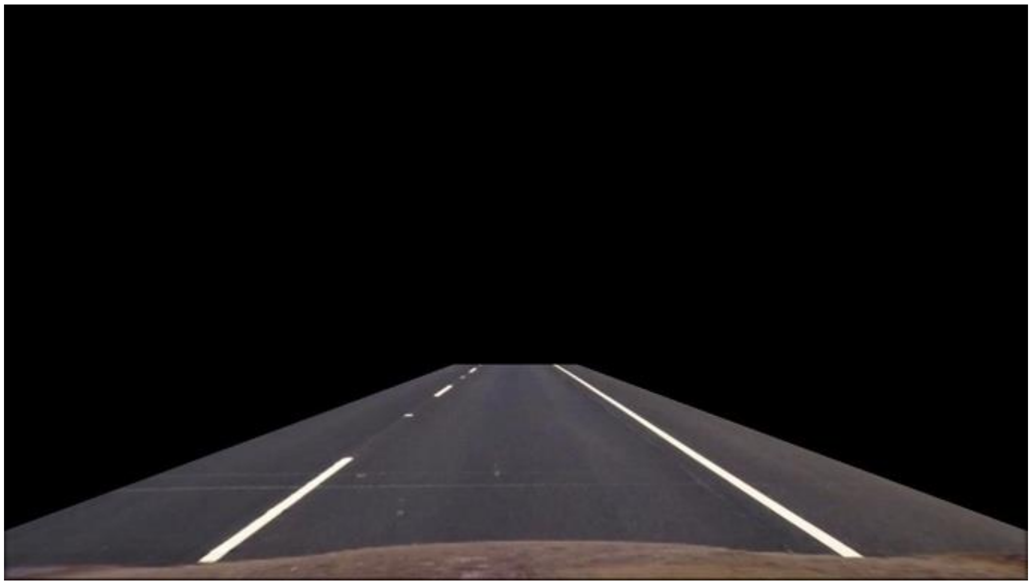
Figure 8: Mapping Back to the Original Image Using Inverse Distortion Matrix

Creating A Binary Image Using Color and Gradient Thresholding

The warped image is then processed using both color and gradient thresholds to produce a binary image that highlights the lane lines in the image using a function called binary_image(). Firs the warped image was converted to the HLS color space and the S channel was isolated. This channel was used to set a color threshold to eliminate non-lane line pixels from the image. Then, a gradient was taken in the x direction using the cv2.Sobel() function to identify vertical shapes in the image. The gradient was taken over the R channel from RGB space and L channel from HLS space and then these two resulting binary images after thresholding were combined with the color channel binary to form a final image binary shown below. The S channel was chosen because it best identified lane lines in images and the R and L channels both maintained the most useful representations of lane lines using only one channel.

Figure 9: An Example Combined Binary Image

Drawing and Mapping a Best Fit Polynomial

After a binary image is produced, a sliding window method using a histogram was then used to identify which pixels were part of the lanes. This was done using finding_line(). After the pixels were identified, a best fit second order polynomial was fit using np.polyfit().

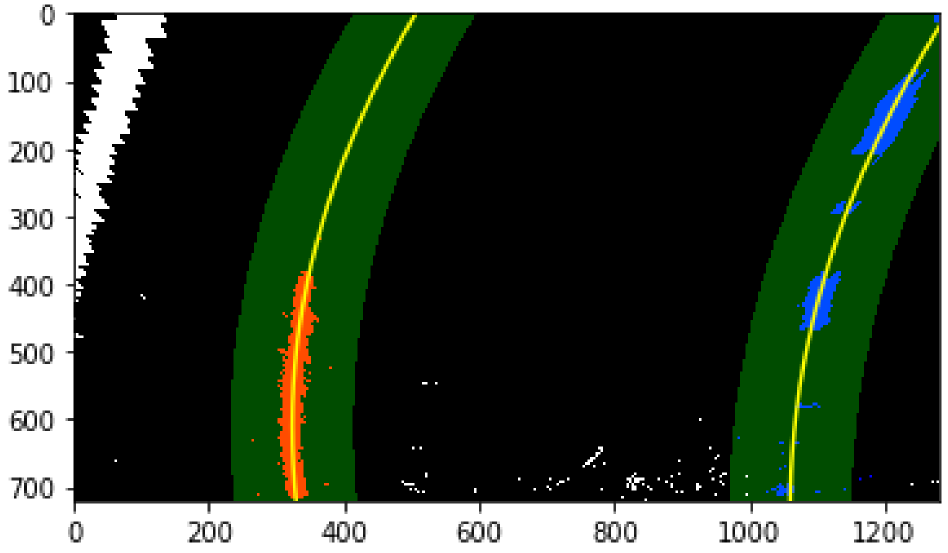

Figure 10: Example of a Best Fit Polynomial (yellow line) Through Both the Left (red pixels) and Right (Blue Pixels) Lanes

Finding Lane Curvature and Distance from Center

The polynomial was calculated twice, once in pixel space, so that it could be plotted, and once in real world estimated dimensions using a pixel to meters conversion of 3.7 m/700 pixels in the x dimension and 30 m / 720 pixels in the y dimension. Using the real world best fit polynomial, the radius of curvature could then be calculated for both the left and right lane. Additionally, by subtracting the right lane position at the bottom of the frame from the left lane and then subtracting the midpoint of the image, the car’s position in the lane can then be estimated.

Results

The resulting polynomial lines were then filled using cv2.fillPoly(). Next, the resulting image was then mapped to the original unwarped perspective using the inverse transformation matrix found earlier. The average of the two lanes’ radius of curvature and the distance from the center of the lane were then displayed on the image to finalize the output. An example image is shown below.

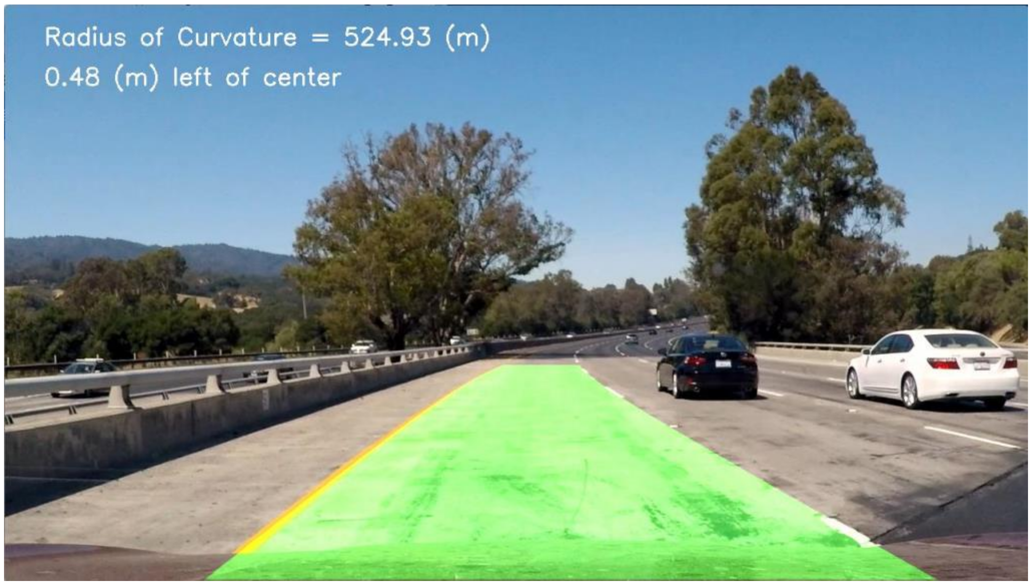

Figure 11: Example Resulting Output from the Image Pipeline

ROOM FOR IMPROVEMENT

The resulting video was accurate and steady for most of the video and lane conditions. However, further efforts to smooth out the lane estimation could be done by using a running weighted average of the last few lane predictions, using previous estimations to find the locations of the new lane lines, and discarding readings of large fluctuations between images. Additionally, dynamic estimations of pixel to real world distances could be used to more accurately estimate lane curvature and car position. While the computer vision approach has its pros in efficiency, it lacks the adaptability of machine learning methods to changes in lighting conditions. A static threshold in color spaces simply does not have the adaptability of an extremely nonlinear neural network.