<u>Advance Lane Finding</u>

By: Brendan Kam

The goal of this project is to create a pipeline to detect lane lines in still images and in a video. In this report, the steps taken to accomplish this and the final results obtained are discussed. The pipeline file to find lane lines in the still images is titled, "***advanced_lane_lines_images.ipynb***". The pipeline file to find lane lines from a video it titled, "***advanced_lane_lines_videos.ipynb***". The pipeline to find lane lines in a video was built based on the pipeline to find the lane lines in still images. Lines of codes for visualization purposes were removed from the former to make the code more concise. The pipeline file "***advanced_lane_lines_images.ipynb***" will be used for explanation and discussion of the pipeline in this report.

The first step of the project was to calibrate the camera. The code to accomplish this can be found second code cell of the pipeline file. Using a set of chessboard images, object points and image points of the chessboards were found. Object points are the (x,y,z) coordinates of the chessboard corners in the real world. Image points are (x,y) coordinates of the chessboard corners in the image plane. To find the corners of the chessboard, the function cv2.findChessboardCorners() was used. Having found the object and image points, the camera calibration matrix and distortion coefficient was found using the function cv2.calibrateCamera(). Using the camera calibration matrix and distortion coefficient, images can be undistorted with cv2.undistort().

Figure 1, shows the original chessboard image on the left and the undistorted chessboard image on the first. Figure 2 is a visualization of the original and undistorted images of the road. As seen in both figures, the undistorted image looks slightly flatter. Image distortion can affect the apparent size and distance of an object in an image, so images must be undistorted to obtain accurate results.

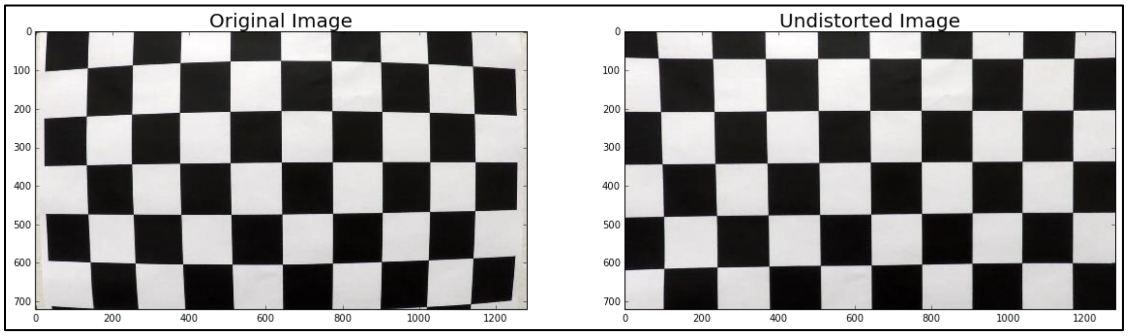


Figure 1: Visualization of undistorted chessboard

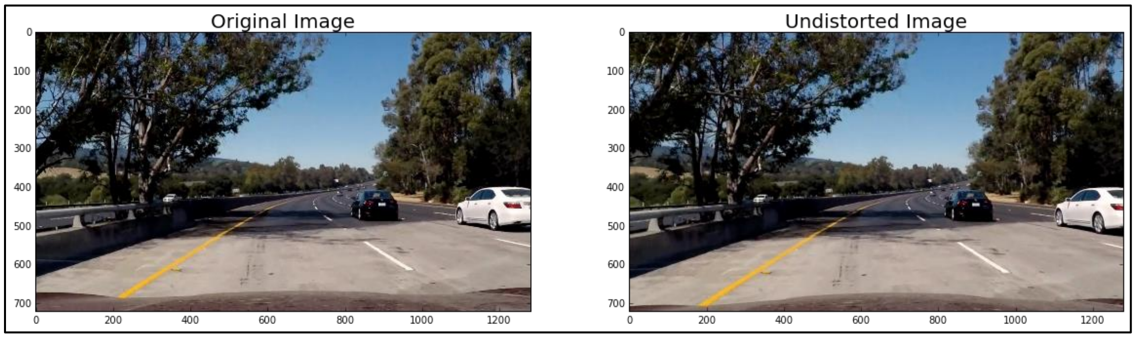
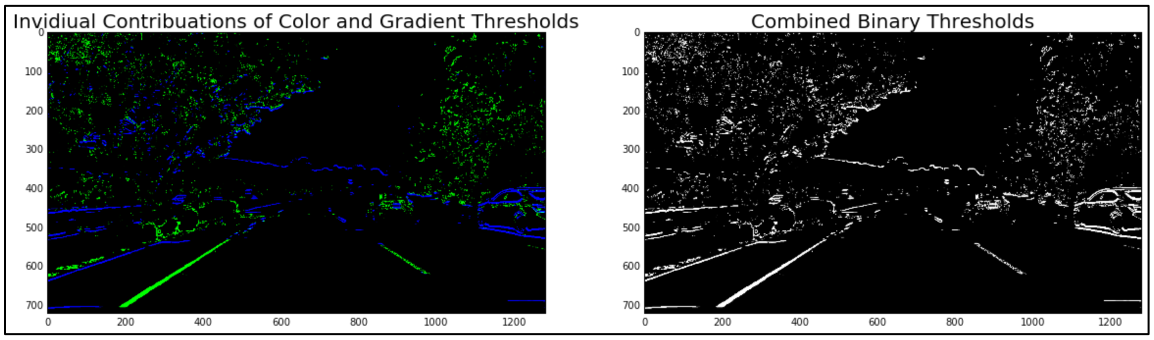


Figure 2: Visualization of undistorted road image

After undistorting the image, the next step was to obtain a binary threshold image. This was done through color and gradient thresholding. To obtain a binary threshold image from color thresholding, a color threshold was used on the saturation, normalized red and normalized green channels of the image. By using these three channels, colored lines can be found even with changing lighting conditions. Gradient thresholding was used to find lines based on the gradient changes in grayscale images. By using gradient thresholds, lane lines of different colors that were not detected by the color thresholds were detected.



Invidiual Contribuations of Color and Gradient Thresholds
Combined Binary Thresholds

Figure 3: Binary Threshold Image

In the left image of figure 3, the lines in green are the contribution of color thresholding while the lines in blue are the contribution of gradient thresholding. The right image in figure 3 shows the overall binary thresholded image obtained from the color and gradient thresholds. The code to obtain these images can be found in the fourth code cell of the pipeline file.

The next step of the pipeline is to warp the binary image to obtain a “birds-eye view” of the road. First, 4 edges of the area of image to be warped were selected. This was manually selected to find the edges of the lane lines. These edges are called in source points. Then, 4 different points, called destination points were chosen. Destination points are places edges where the “birds-eye view” image would be. A function cv2.getPerspectiveTransform() was used to obtain a perspective transform matrix. Then, the cv2.warpPerspective() function was to warp the image, to obtain the “bird’s eye view image”. The code for this are found in the fifth code cell of the pipeline file.

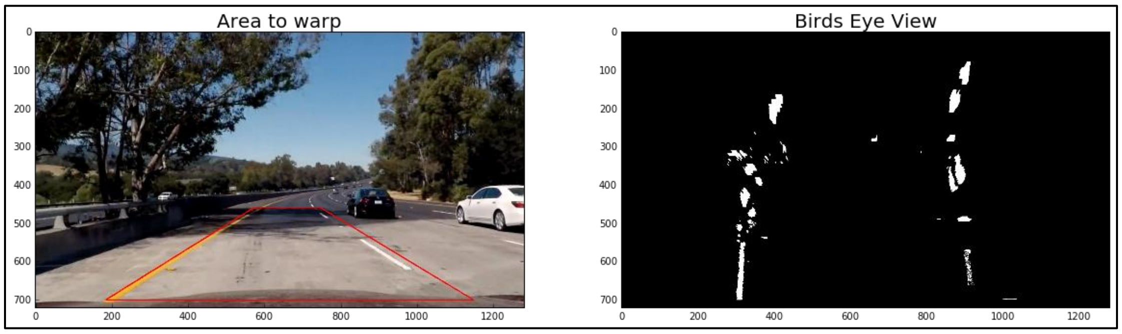


Figure 4: "Birds Eye View of Road"

The edges of the polygon in the left image of figure 4 are the source points. The right image in figure 4 shows the binary threshold bird’s eye view image of the road.

After obtaining the “bird’s-eye view” of the binary thresholded image of the road, the lane line pixels were found. Then, a 2nd degree polynomial fit was done on the line pixels to estimate lane lines on the road. The code for these steps can be found in code cells 6 through 8.

The first step to find the lane line pixels was to get a histogram of the lane line pixels. In general, lane line pixels are where the highest peaks of the histogram are. The two peaks corresponding to the left and right lane pixels were found through this method. Starting from the bottom position of the lane line pixels, a sliding search window was used to find the position of the lane line pixels above it. The search window has a width of 200 pixels and a height of 78 pixels.

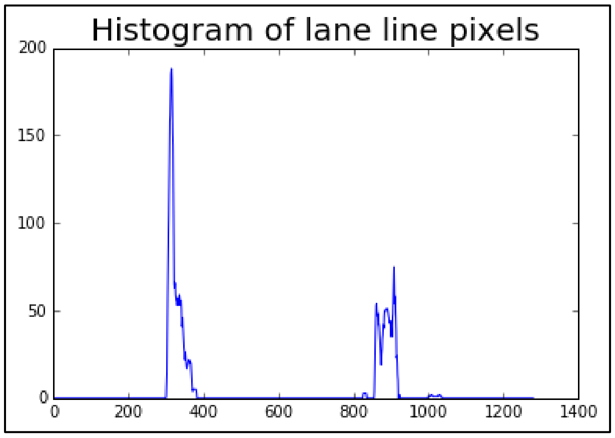


Figure 5: Histogram of lane line pixels

Figure 5 shows the histogram of the lane line pixels. The left peak in the histogram corresponds to the pixels of the lane lines, while the right peak of the histogram corresponds to the pixels of the right lane lines.

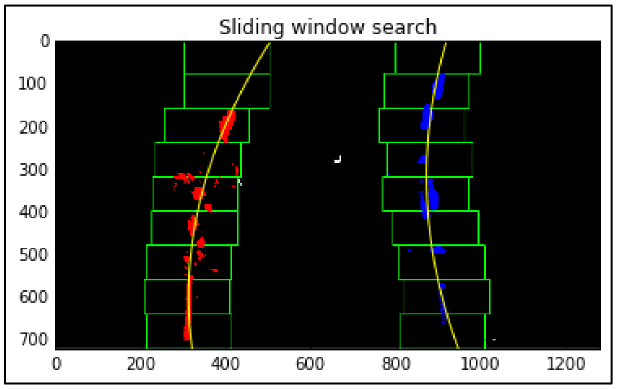


Figure 6: Illustration of sliding with search

The rectangles in figure 6 shows the search window for the lane line pixels. The red pixels are the found from the left lane lines, while the blue pixels are the pixels found for the right lane lines. The two yellow lines are the polynomial fits of the left and right lines.

After finding the lane lines, the next step in the project was to find the car offset from the center of the lane and the radius of curvature of the road. To find the car offset, it was assumed that the camera was located at the center of the car, hence the center of the car would be at the center of the image. By getting the difference in distances between center of the center of the lane, the car's offset was found. To convert the car's offset from pixels to meters, it was assumed that the distance between the left and right lane lines was 3.2m. To obtain the radius of lane curvature, it was assumed that every 700 pixels in the y direction corresponds to 30 meters ahead of the car. The curvature of the lanes is then approximated by assuming its radius is the same the radius of the circle tangent at a point of interest. The codes to finding the car's offset and radius can be found in the ninth cell code.

Finally, the lane lines found from the bird's eye view image was unwarped and drawn back onto the original image.

Figure 7, shows the final image obtained along with the final predicted lane lines.

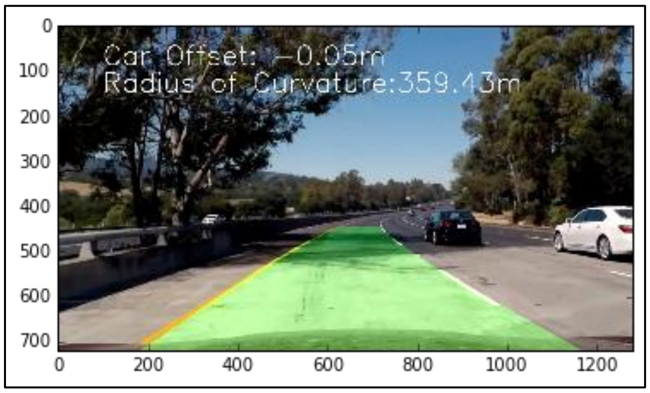


Figure 7: Final Image of detected lane lines

The final video of this project can be found in the zip file included in this submission.

Compared to the pipeline used to find lane lines in project 1, this pipeline could approximate curve lanes more accurately. It was also able to find the car's offset from the center of the lane and the radius of curvature of the lane at a point of interest. However, there are several issues with this current pipeline.

The first issue with this pipeline is that it approximates where lane lines are by looking at the peaks of the histogram. There could be situations however where the peaks of the histogram are not the positions of the actual lane lines. Besides that, changing lighting conditions such as shadows, sun glare and color changes on the road causes this pipeline to lose accuracy. Finally, while running the challenge videos, exceptions are raised if one line was not found, causing the code to fail. This could be a safety hazard if used on the road.

Those 3 issues mentioned are the main reasons why this pipeline was unable to complete the challenge videos provided. To make the pipeline more robust, it is suggested that the biggest issue that should be tackled first is the changes in lighting conditions. There have been various ways proposed to solve this issue. One way to solve the pipelines failure find lane lines in low light conditions is to use a low-light image enhancement via illumination map (Guo, Li, & Ling , 2017) as proposed in a journal article in the IEEE Transactions of Image Processing. If shadows or low lighting conditions were detected, the pipeline would be able to enhance the image to better detect lane lines.

Currently, lane lines are found by combining color thresholding and gradient thresholding. The pipeline might be made more robust if the lane lines were found by doing both thresholding separately and comparing both results to see if they generally agree with one another. Different color channels in varying color spaces should also be explored

Finally, to solve the problem of exceptions being raised, data about previous lane lines should be kept. If a lane line is not detected, previous lane lines would be used in its place until a new lane line is detected. Keeping data of old lane lines would also help to average lines to make it the video smoother and more accurate.

Higher levels of polynomial fits should also be explored for cases where lane lines are very curvy and have sharp turns.

References

Guo, X., Li, Y., & Ling , H. (2017). LIME: Low-Light Image Enhancement via. *IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 26, NO. 2, FEBRUARY 2017*.

Radius of Curvature. (n.d.). Retrieved from Interactive Mathematics: http://www.intmath.com/applications-differentiation/8-radius-curvature.php

Road Design Manual. (n.d.). Retrieved from http://onlinemanuals.txdot.gov/txdotmanuals/rdw/horizontal_alignment.htm#BGBHGEGC