Advanced Lane Finding Project

The goals / steps of this project are the following:

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

Where are the results

My code is in the main.ipynb file.
My image results are in the directory output_images.
My video results are in the directory output_videos.

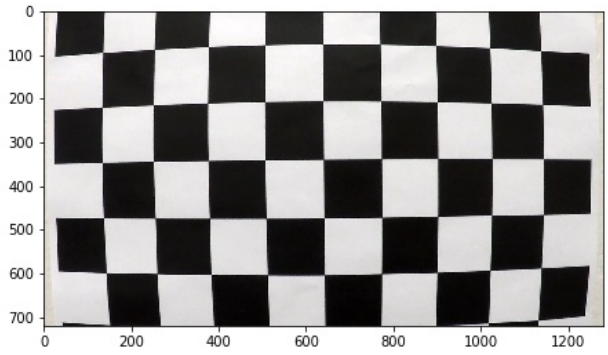
Camera Calibration

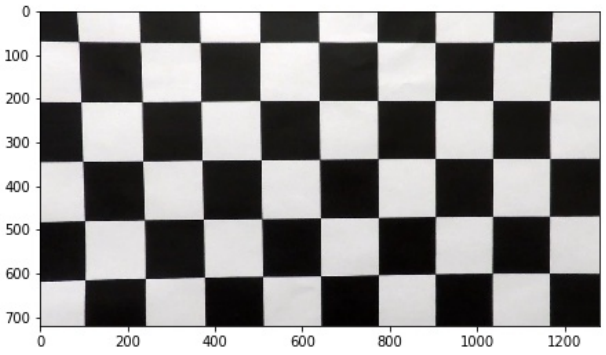
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is the 'calibrateCamera' method of class AdvancedLaneFinder.
I use cv2.findChessboardCorners to find the corners of each chessboard image in the directory camera_cal, and use cv2.calibrateCamera to get the camera matrix and distortion coefficients. cv2.findChessboardCorners will give us sets of corner points in 2d. And these sets of corner points found in different images are all mapped to the same set of corner points in 3d.

After getting the camera matrix and distortion coefficients, we can pass them to cv2.undistort with some image to undistort it.

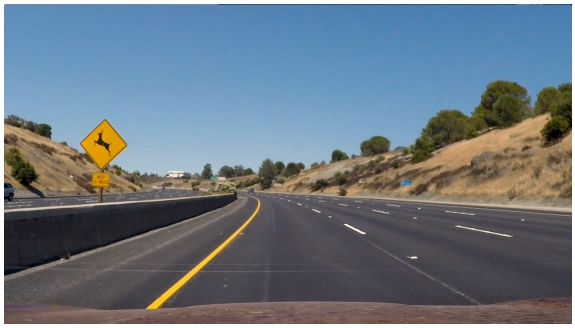
These are the images(calibration1.jpg) before(left) and after(right) undistortion.
There are also results of other images stored in the directory undistorted_chessboard_images.





Pipeline (single image)

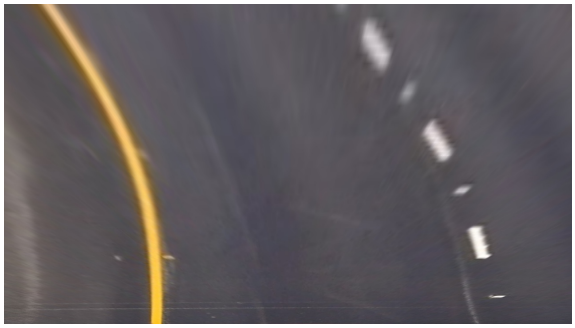
original image



apply undistortion

apply perspective transform

The source and destination points are chosen by my eyes looking at the image.
The source points are [300, 675], [1030, 675], [700, 456], [585, 456]
The destination points are [300, 720], [1030, 720], [1030, 200], [300, 200]
The transformed result is as the following.



process to binary image

The code is in the method process2Binary of class LaneTracer.
First, the image was converted to gray. Then Sobel, Canny, Converting to HLS space, and direction were applied to find lane line pixels. These results are combined together to get one binary image.
The result is as the following.

find lane line pixels

The code is in the method getLanePointsGivenCenter and getLanePointsGivenLine of the class LaneTracer.

getLanePointsGivenCenter use the starting point calculated by histogram of the bottom of the binary image, and set windows to found the pixels in them, which is the same method as that in the quiz.

For video, once we have the fitting line, we use getLanePointsGivenLine instead of getLanePointsGivenCenter. getLanePointsGivenLine set windows centered at the points on the line.

The following shows the result of getLanePointsGivenCenter.

line fitting

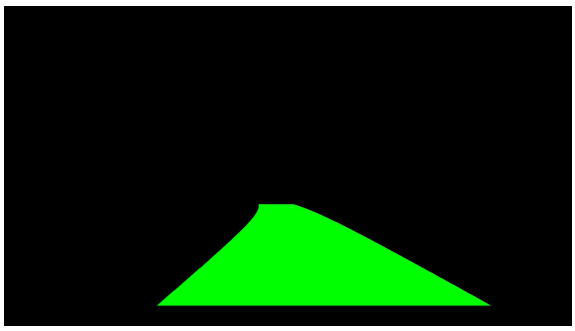
The code is in the method getLeftFitLinePoints and getRightFitLinePoints of class LaneTracer.

For video, once we have the fitting line, every time we get the coefficients of the new fitting line, we weighted-add them to the saved coefficients to get smoothed result. (As in the video)

The following shows the result of the polygon given the points of the two fitting line.

perspective transform back to the original space

The source and destination points passed to cv2.getPerspectiveTransform are exchanged.
The following shows the result.



add to the undistorted image

The result is as the following.

add the radius of curvature and the amount of shift with respect to the image center

The radius of curvature is calculated using the formula mentioned in the course.
The amount of shift with respect to the image center is calculated as the following:
calculate the average of the values of the line evaluated at the bottom of the image, and subtract the half width of the image.

The result is as the following.

Pipeline (video)

The results are put in the directory output_videos.

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

For the project_video.mp4, the performance of my lane finder looks well.
For the challenge_video.mp4, the found line seems to be a little curly in the farther part. It may result from that the farther part of the line are not so clear. We may enhance the contrast of the image to try to tackle this problem.
For the harder_challenge_video.mp4, the performance of my lane finder looks bad. The changing brightness and the curves make it harder to detect the lane lines. We my add some methods to solve the changing brightness problem.