Self-Driving Car Project #4

Advanced Lane Finding

Complete on April 19th, 2018

Haowei Zhang

Part 1 Camera Calibration

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

Response:
The code for this step is contained in the 3rd code cell of the IPython notebook located in $./Advanced_lane_finding.ipynb$. I started by converting 3-channelled RGB image into Grayscale image. Then I detect (9, 6) corners on the chessboard calibration images using OpenCV function $cv2.findChessboardCorners$. Next I would store 2D image points in the camera calibration images and their corresponding 3D object points it they exist. The image points and object points would be useful to correct images for distortion.

The original image and undistorted image are as follows.

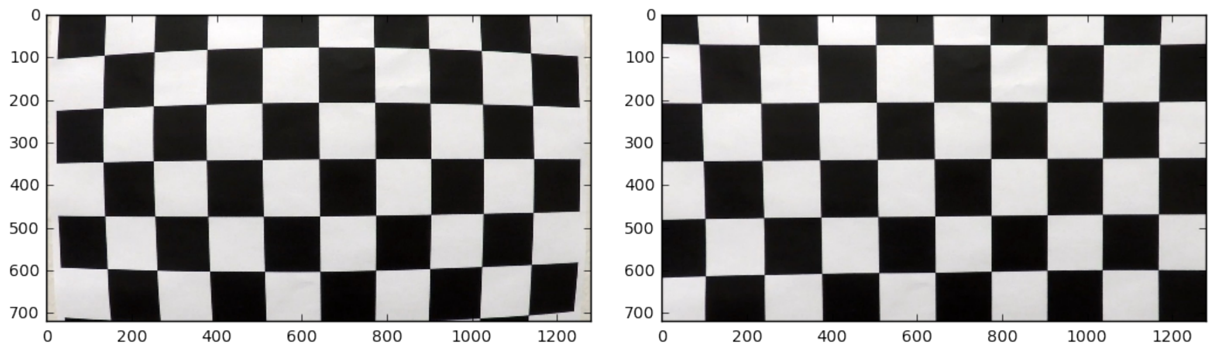

Figure 1: **Chessboard image before and after correction for distortion** (left): distorted chessboard image. (right): undistorted chessboard image after correction for distortion.

Part 2 Pipeline

Task 1:
Provide an example of a distortion-corrected image.

Response:
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

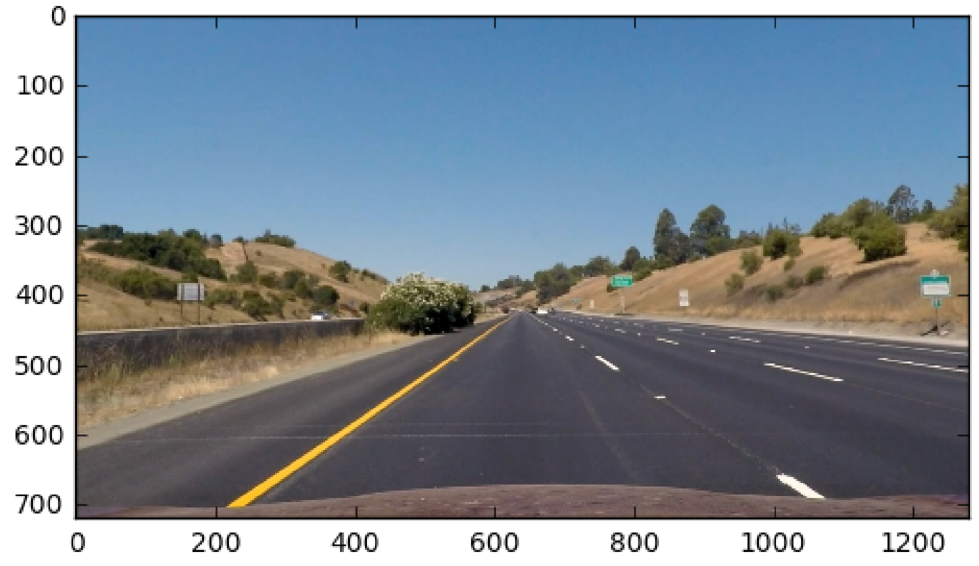

Figure 2: Test image after correction for distortion

Task 2:
Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

Response:
I used a combination of color and gradient thresholds to generate a binary image. The color threshold is on L-channel in HLS image space where I threshold it to an amplitude range of 170 to 255. The gradient threshold is a Sobel x-direction gradient thresholded at a range of 30 to 255. The combination of color and gradient threshold is hoping to keep integrity of the lane while removing as much background redundant information as possible. Here is an example of the step. The code can be found in the IPython notebook at 6th cell labelled as *Color and Gradient threshold*.
Task 3:
Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

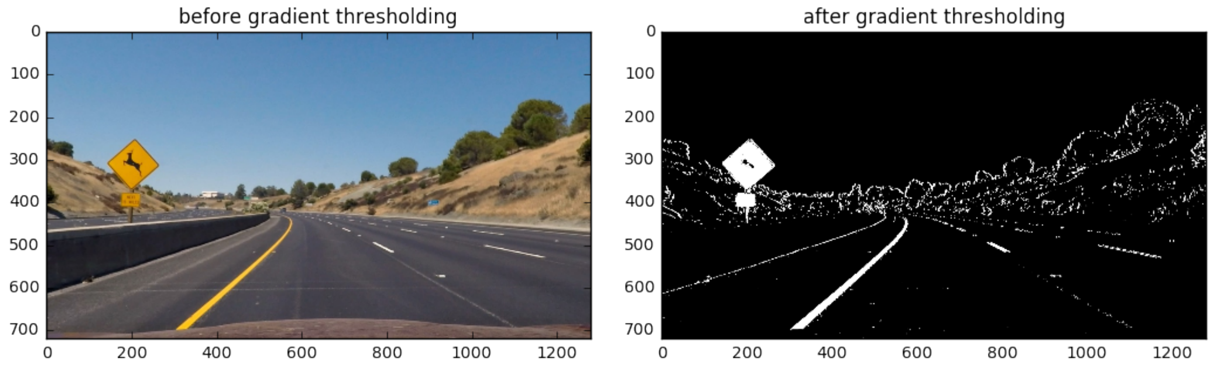

Figure 3: **Test image before and after applying thresholds** (left): undistorted test image. (right): undistorted test image after applying color and gradient threshold.

Response:
To apply the perspective transform, I would need to specify source points and destination points to match. Following are the source and destination points I selected. The code cell can be found in the IPython notebook at the section labelled *Apply perspective transform*.

src	dst
220, 720	200, 720
570, 470	200, 0
720, 470	1080, 0
1100, 720	1080, 720

I verified that my perspective transform was as expected by drawing the *src* and *dst* points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

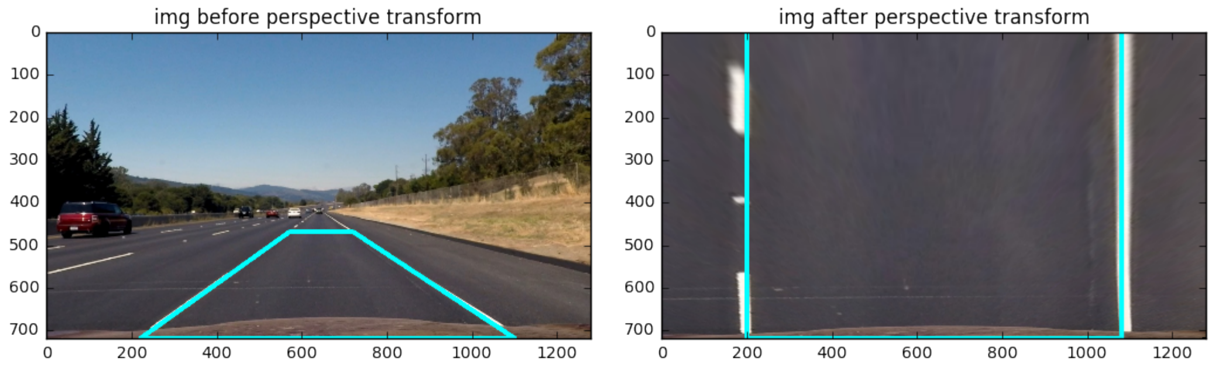
img before perspective transform
img after perspective transform

Figure 4: **Test image before and after applying perspective transform** (left): before perspective transform (right): after perspective transform.

Task 4:

Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Response:
I used the sliding window approach as in the video to fit the second order polynomial to left/right lane after applying perspective transform. The code cell can be found in the IPython notebook at section labelled *Find the lines*. Following is an example of a fitted test image.

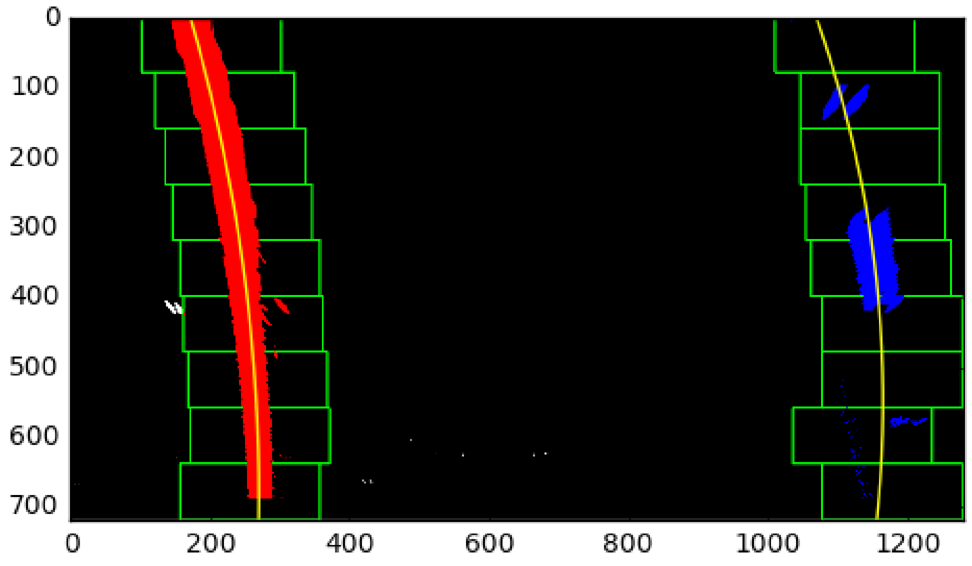

Figure 5: Second-order polynomial fitted lane lines

Task 5:
Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

Response:
The curvature of the lane can be found using the equation as follows.

$$R_{curve} = \frac{(1 + (2 * A * y + B)^2)^{\frac{3}{2}}}{|2A|} \tag{1}$$

The position of the vehicle can be determined by averaging the position of where left and right lane starts and compare with the midpoint of the image. For more detailed implementation, please find the code cell at the 17th cell in IPython Notebook.
Task 6:
Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

Response:
Following is an example test image.

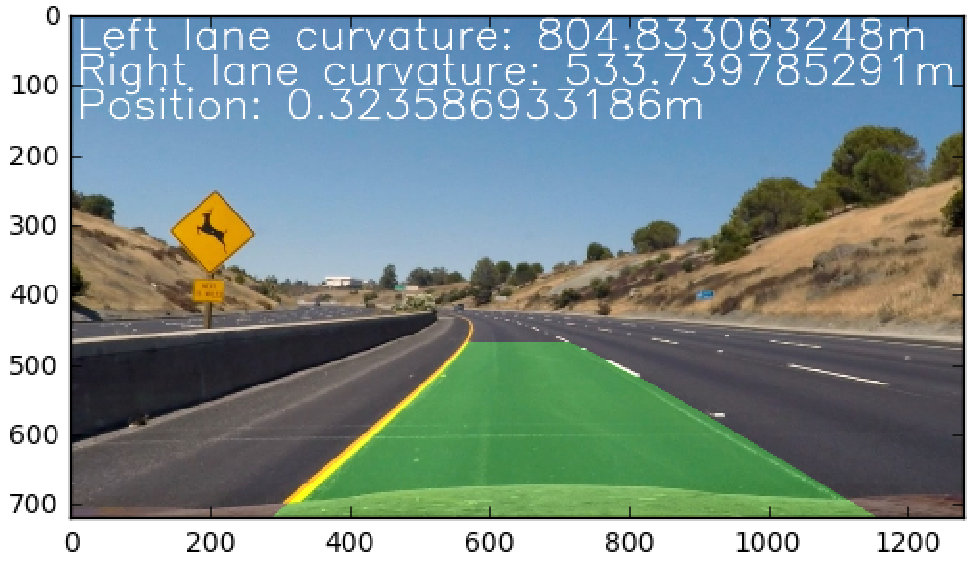
Left lane curvature: 804.833063248m
Right lane curvature: 533.739785291m
Position: 0.323586933186m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Figure 6: Mapping the lane back to the undistorted test image

Part 3 Pipeline video

Task 1:
Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

Response:
Please kindly find the video in the same Github directory.

Part 4 Discussion

Task 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?

Response:
There are multiple tasks I had faced when trying to implement this algorithm.

1) The color and gradient threshold takes effort to combine. Although the L-channel in HLS image space gets a clear image of the lane, the gradient threshold often messes things up by amplifying noisy regions in the image. I was combining magnitude threshold and directional threshold but ended up abandoning those as they introduced too much noise into the thresholded image.

2) The source and destination points from perspective transform take efforts to find. I started off using a set of those points but ended up with mis-aligned image after perspective transform. I had to play around with it until finding an optimal match.