Advanced Lane Finding Project

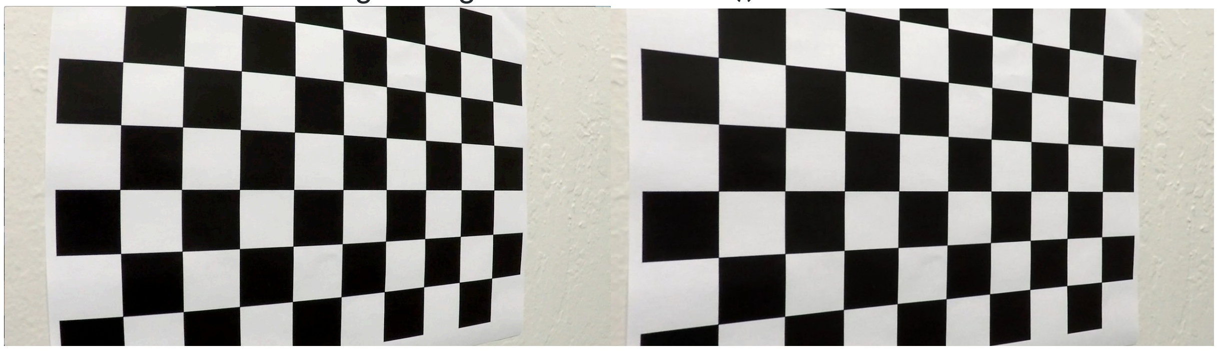
The goals / steps of this project are the following:

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

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 contained in the first code cell of the IPython notebook located in “Advanced Lane Finding CarND.ipynb”(cell 2~3)
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.
I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()`function and obtained this result:



Original Undistorted

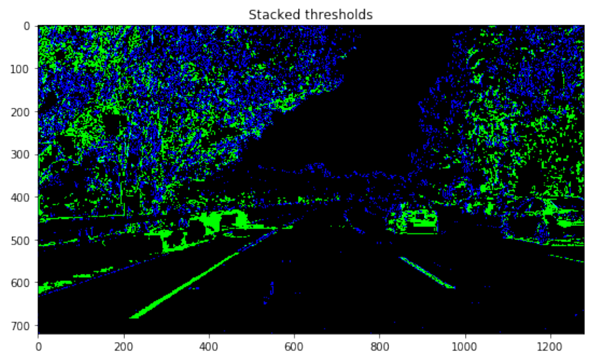
Pipeline (single images)

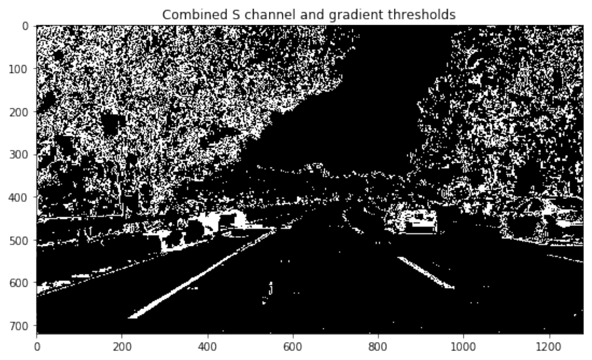
1. Provide an example of a distortion-corrected image.

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

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.

I used a combination of color and gradient thresholds to generate a binary image "Advanced Lane Finding CarND.ipynb"(cell 4~5) I applied the sobel operator on the x axis to detect the lane lines in further distance. And I applied the S channel threshold in HLS color space to detect near distance lane and filter out the shadow. Here's an example of my output for this step. (Blue line is detected by x axis sobel operator and green is detected by color space threshold)



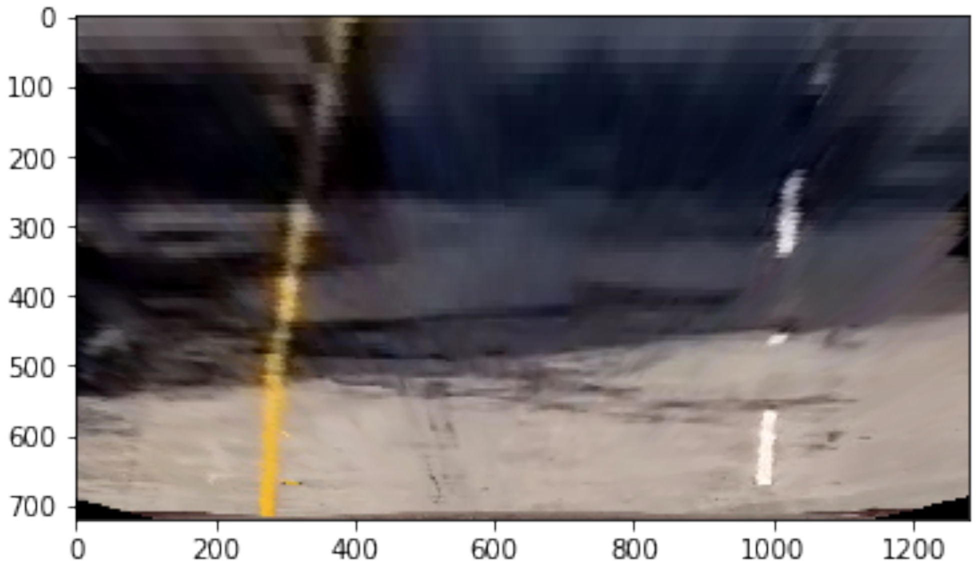


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `warper()`, which appears in “Advanced Lane Finding CarND.ipynb”(cell 6~7). The `warper()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in an interactive image browser. This resulted in the following source and destination points:

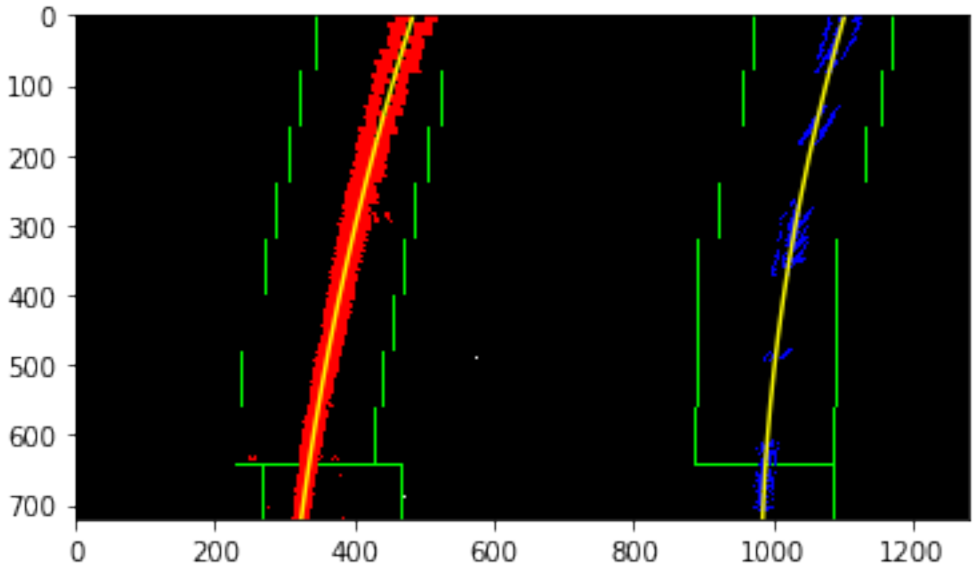
Source	Destination
261,696	320,720
583,459	320,0
703,459	960,0
1062,696	960,720

I verified that my perspective transform was working 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.



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

Then I applied a window search function to find the hot pixels as lane line, which appear in “Advanced Lane Finding CarND.ipynb”(cell 9). An example is following:



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.

I defined a “find_curvature” function to take polynomial fit from previous function and calculate the curvatures from

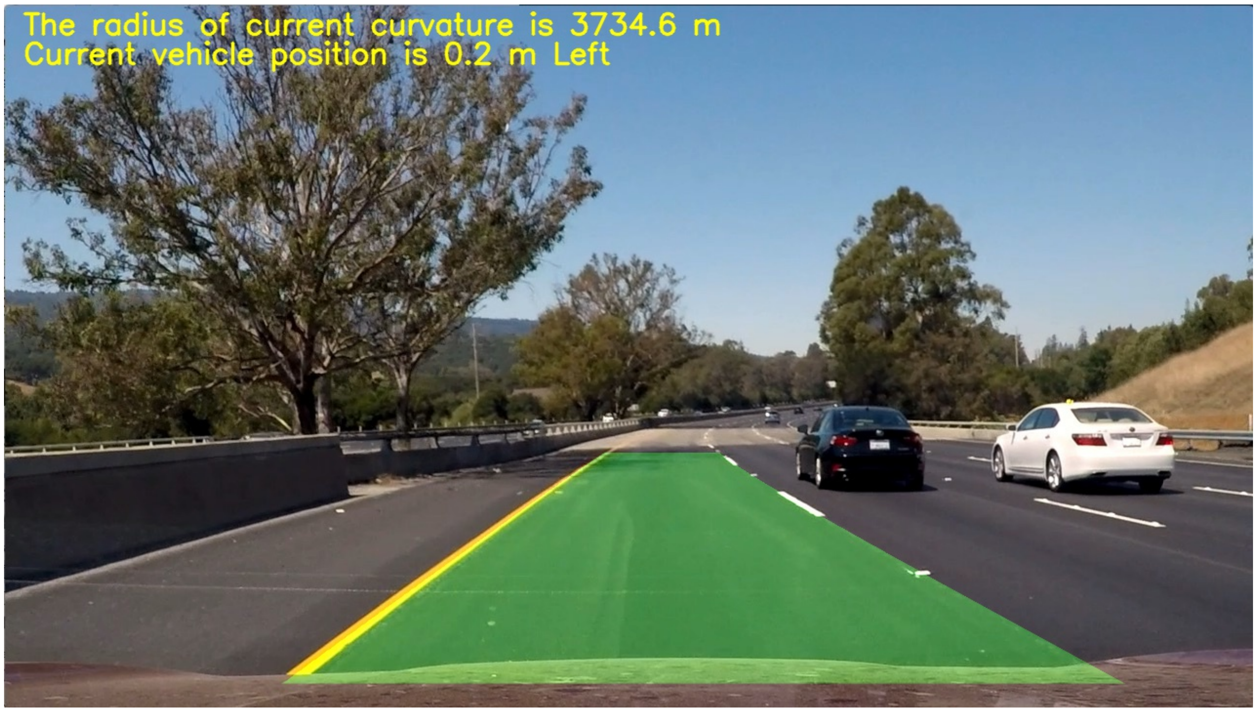
$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$. Then convert the pixel value into meters as: ym_per_pix = 30/720, meters per pixel in y dimension; xm_per_pix = 3.7/700, meters per pixel in x dimension.

6. Sanity check

I added a sanity check step in “detect_lane” function in “Advanced Lane Finding CarND.ipynb”(cell 14). In order to verify the detection is reliable, I compare the left and right curvature’s radius is close or not.

7. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I implemented this step in “detect_lane” function in “Advanced Lane Finding CarND.ipynb”(cell 14). Here is an example of my result on a test image:


The radius of current curvature is 3734.6 m
Current vehicle position is 0.2 m Left

8. 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?

The function is still easy to be tricked by the cracks on the road, a good way to verify this could be calculating the left and right lane distance to check whether it’s close to 3.7m or not.