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.

Rubric Points

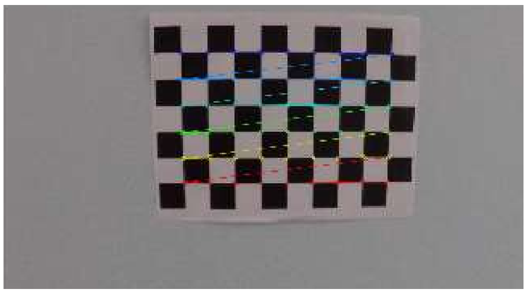
Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Camera Calibration

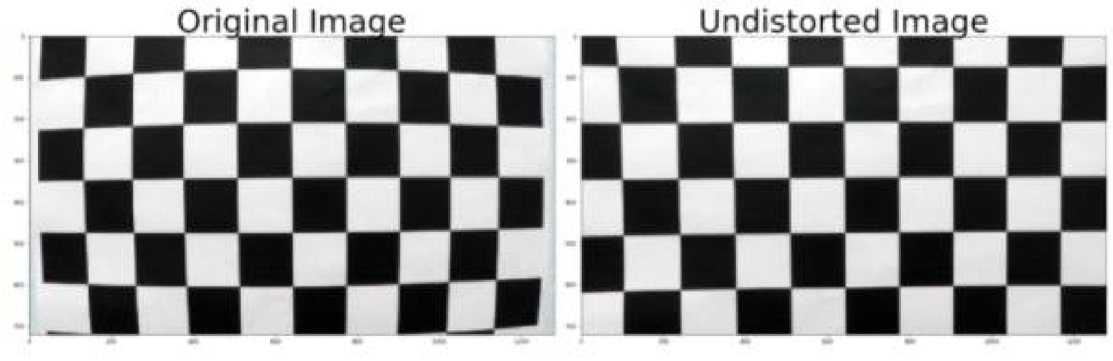
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the first code cell of the IPython notebook located in "./examples/example.ipynb".

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:



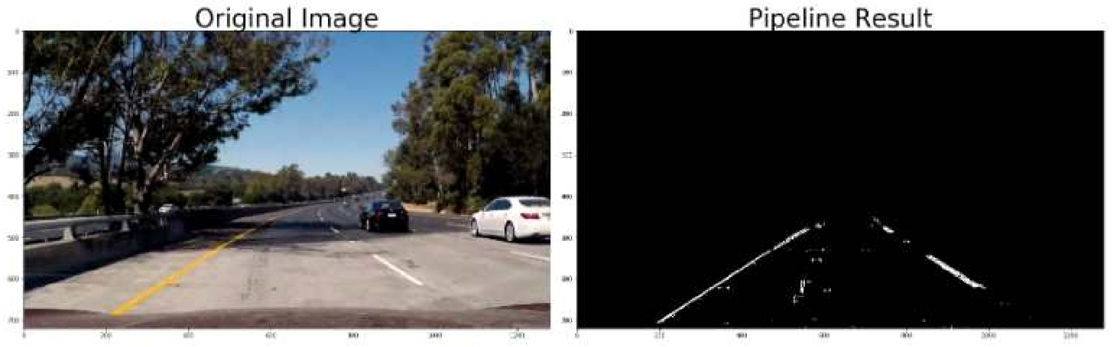
Pipeline (single images)

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

To demonstrate this step, I will describe how 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.

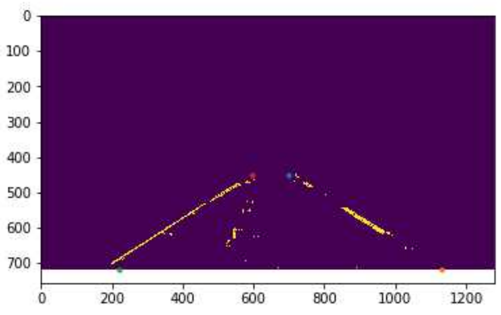
I used a combination of color and gradient thresholds to generate a binary image. Initially, the image is changed to gray scale. And I used the S channel among the HLS colors. Using the Sobel function, we obtain the derivative in x and use it. Then set an appropriate threshold value to highlight the line. After combine the two binary thresholds, here's an example of my output for this step.



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 warp(), which appears in the 6th code cell of the IPython notebook. The warp() function takes as inputs an image (img), as well as source (src) and destination (dst) points. I initially defined Four source coordinates by drawing a dot directly over the image.

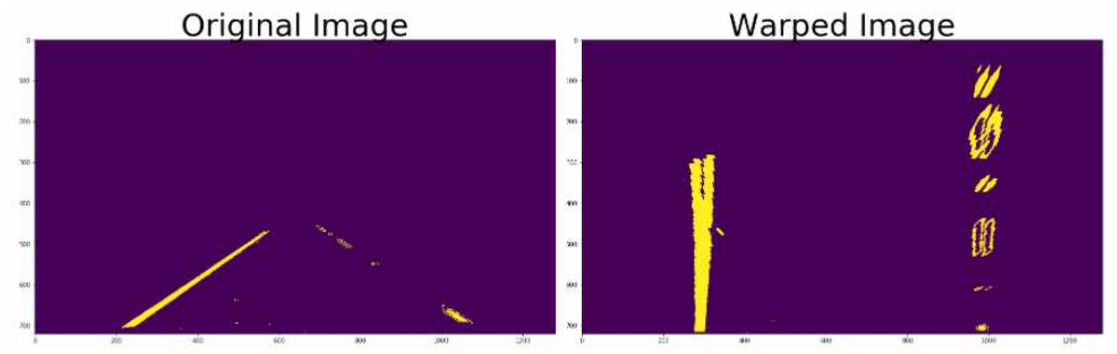
```
img = np.copy(result)
img_size = (img.shape[1], img.shape[0])
plt.imshow(img)
plt.plot(700,450,'.') # top right
plt.plot(1130,720,'.') # bottom right
plt.plot(220,720,'.') # bottom left
plt.plot(595,450,'.') # top left
```



And I confirmed Four source codes and Four desired codes.

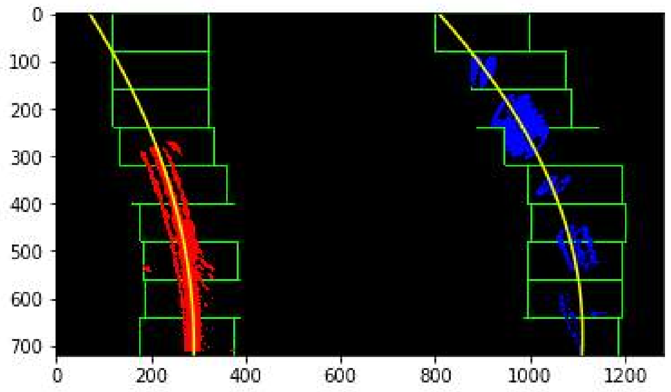
src = np.float32([[700,450], [1130,720], [220,720], [595,450]])
dst = np.float32([[1100,0], [1100,720], [220,720], [220,0]])

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.



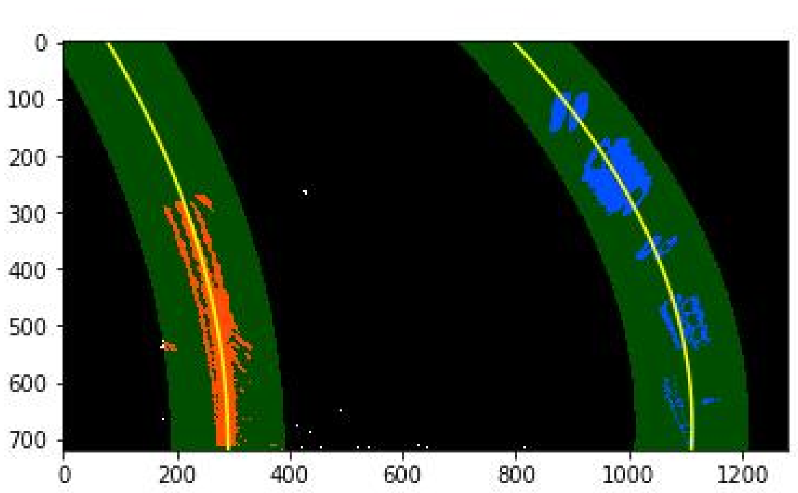
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Take a histogram of the bottom half of the image. And then find the peak of the left and right halves of the histogram. These will be the starting point for the left and right lines. Next, set hyperparameters. I set nwindows = 9, margin = 100, minpix = 50. And Step through the windows one by one. In this step, I draw the windows on the visualization image and identify the nonzero pixels in x and y within the window. Then I draw a curve using the fit_polynomial function.



The function is inefficient because it is on line for the first time and then you've used the following way.
Using the fit_poly function, fit a second order polynomial to each with np.polyfit(). After generate x and y values for plotting, calculate both polynomials using ploty, left_fit and right_fit. An area is created by setting a margin on a curve using the

search_round_poly function.



In the figure above, the quadratic function of the left and right curves is as follows.

left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx, 2)
left_fitx = left_fit[0]*ploty2 + left_fit[1]*ploty + left_fit[2]**
right_fitx = right_fit[0]*ploty2 + right_fit[1]*ploty + right_fit[2]**

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.

The code for my measurement of curvature includes a function called measure_curvature_real(), which appears in the 11th code cell of the IPython notebook. The curvature is determined by the following equation.

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

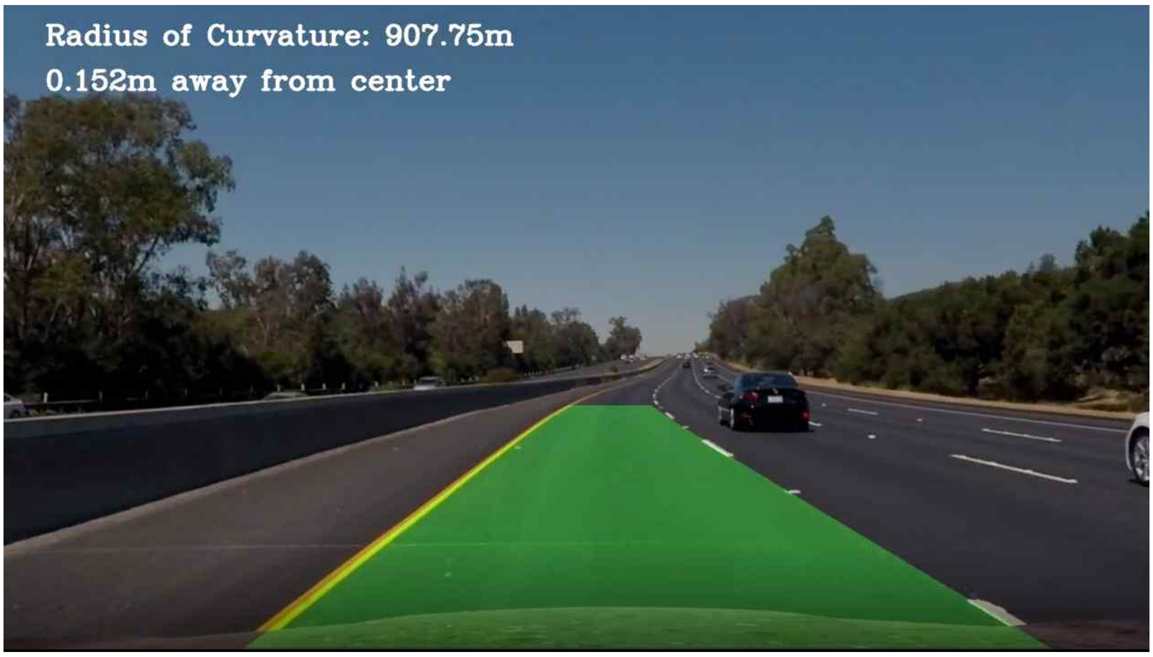
If this is expressed by code, it is as follows.
left_curverad=np.power((1+np.square(2*left_fit[0]*y_eval+left_fit[1])),(3/2))/abs(2*left_fit[0])
right_curverad=np.power((1+np.square(2*right_fit[0]*y_eval+right_fit[1])),(3/2))/abs(2*right_fit[0])
When calculating the above equation, define conversions in x and y from pixels space to meters.
 ym_per_pix = 30/720 # meters per pixel in y dimension
 xm_per_pix = 3.7/700 # meters per pixel in x dimension

```
left_fit = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
right_fit = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)
```

Also, when finding **the position of the vehicle with respect to center.** use car_pos and curvature.

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

The code for my perspective transform includes a function called inverse_warp(). I colored between the left and right curves after inverse_perspective transform. Here is an example of my result on a test image:



Pipeline (video)

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!).

I'll submit it as a compressed file.

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?

When driving on a road with spots, the line was not well detected. I used the appropriate HSL channel and set the optimal threshold value when using the sobel function. In addition, when there is a lot of noise in the image, the noise was minimized by setting the roi.