Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

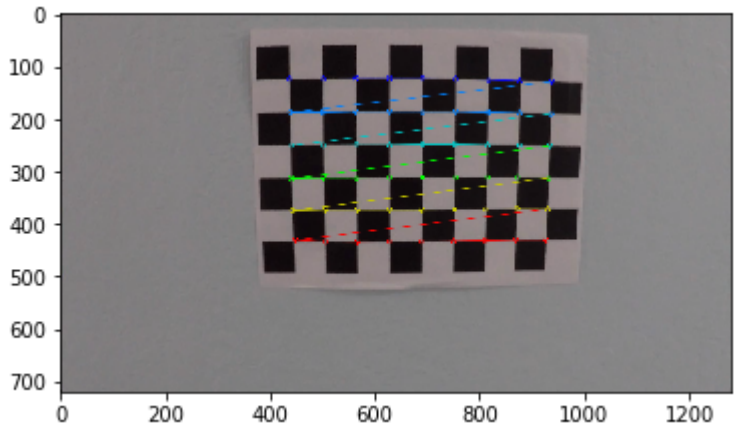
Camera Calibration

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

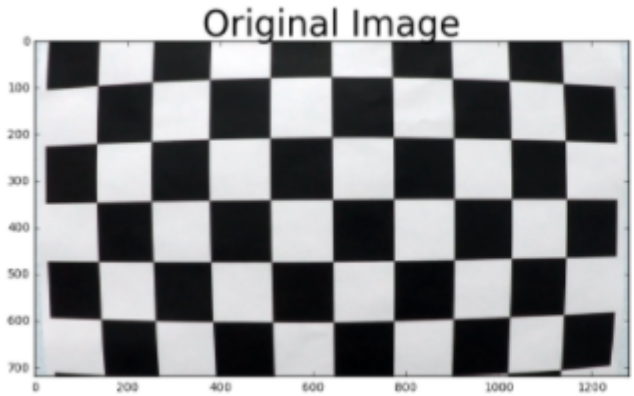
The code for this step is contained in the first code cell of the IPython notebook located in "./examples/example.ipynb" (or in lines # through # of the file called `some_file.py`).

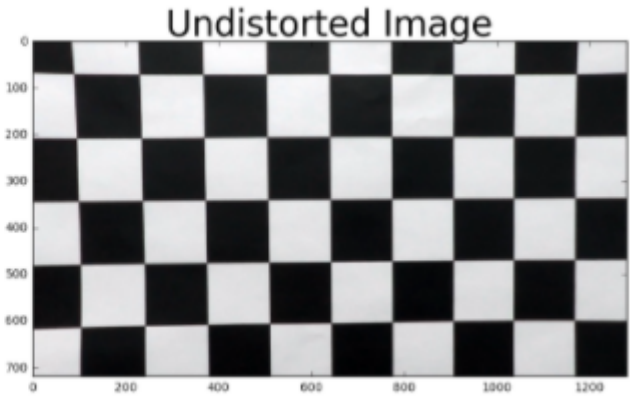
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of 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.

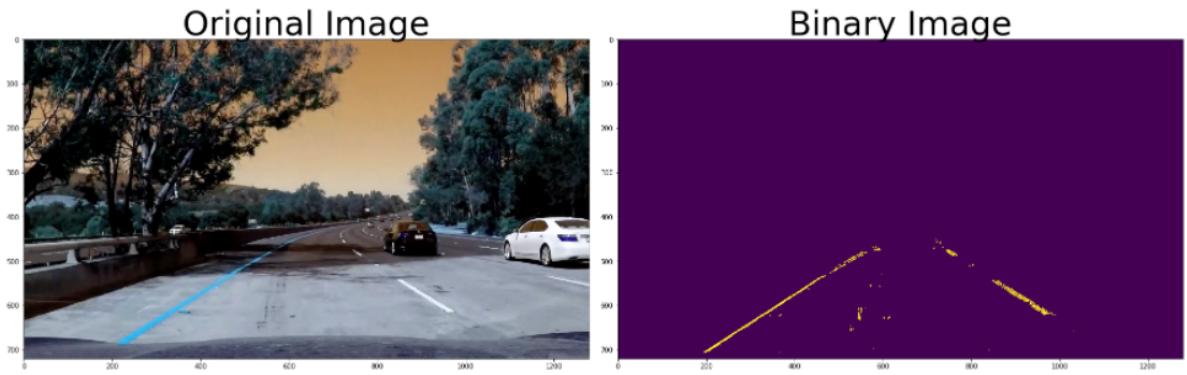
To demonstrate this step, I will describe how I apply the distortion

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 Derivative was extracted with Sobel X to obtain a vertical line segment with its absolute value. S was extracted from the HLS color channel and made robust against light changes. An appropriate threshold was set to enable smooth line detection. And then I combined the two binary thresholds. In addition, using the mask learned earlier, line segments unnecessary for line detection were removed.

Here's an example of my output for this step. (note: this is not actually from one of the test images)



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 lines 1 through 8 in the in the code cell underneath the title Perspective transform. 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 the following manner:

```
src = np.float32([[685 ,451 ],[1100, 720],[ 220 , 720 ],[ 595 , 451]])
dstr = np.float32([[img_size[0]-offset, 0],[img_size[0]-offset, img_size[1]],[offset, img_size[1]],[offset, 0]])
```

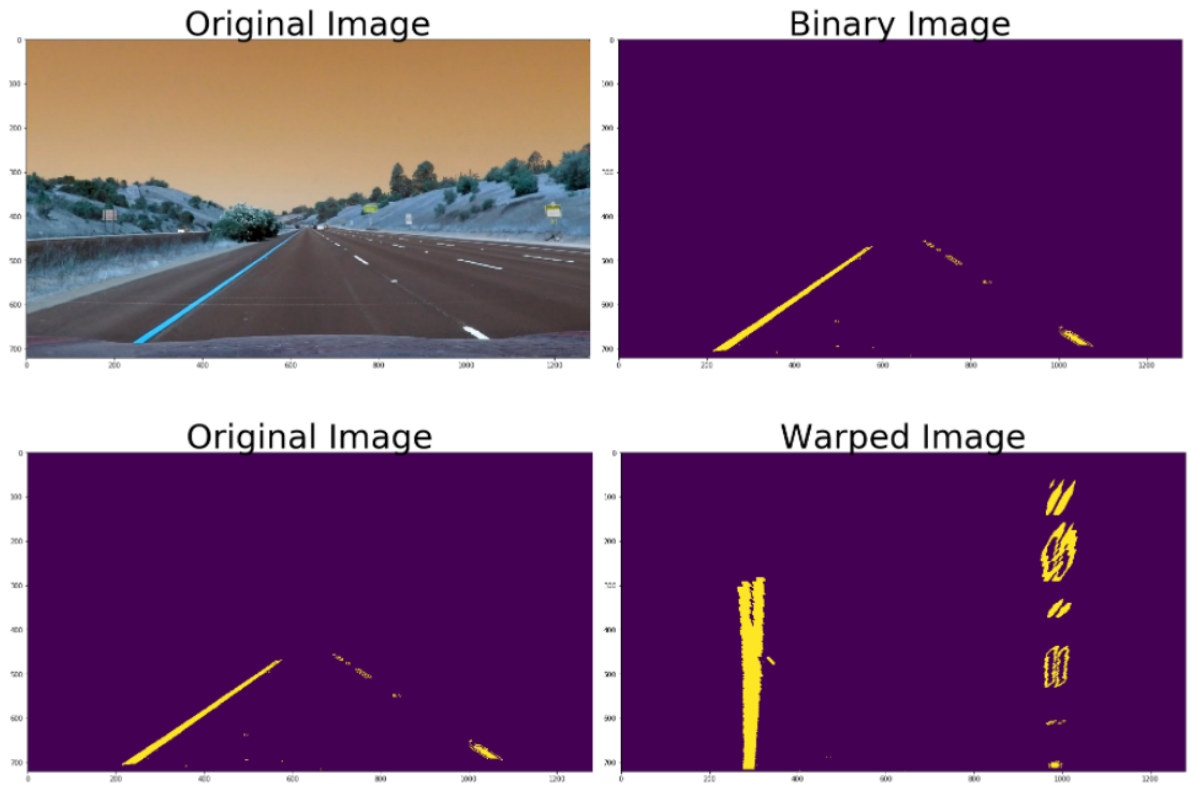
This resulted in the following source and destination points:

Source	Destination
685 ,451	img_size[0]-offset, 0
1100, 720	img_size[0]-offset, img_size[1]

Source	Destination
220 , 720	offset, img_size[1]
595 , 451	offset, 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.

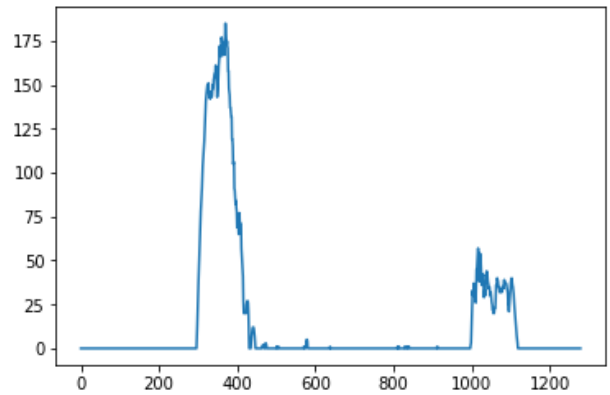
To demonstrate this step,I used the picture named straight_lines1.



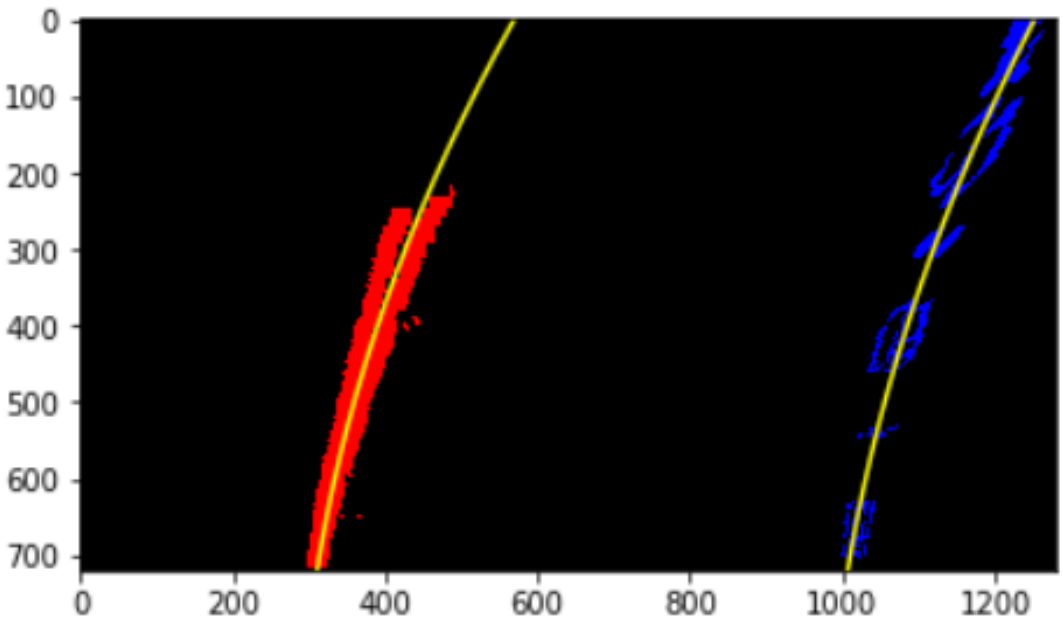
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

In the beginning stage, a histogram at the bottom of the screen was

drawn to roughly determine the location of the lane. Looking at the histogram, it can be seen that the lanes are approximately located at 350 and 1100.At this time, left and right lanes were divided based on the center line.

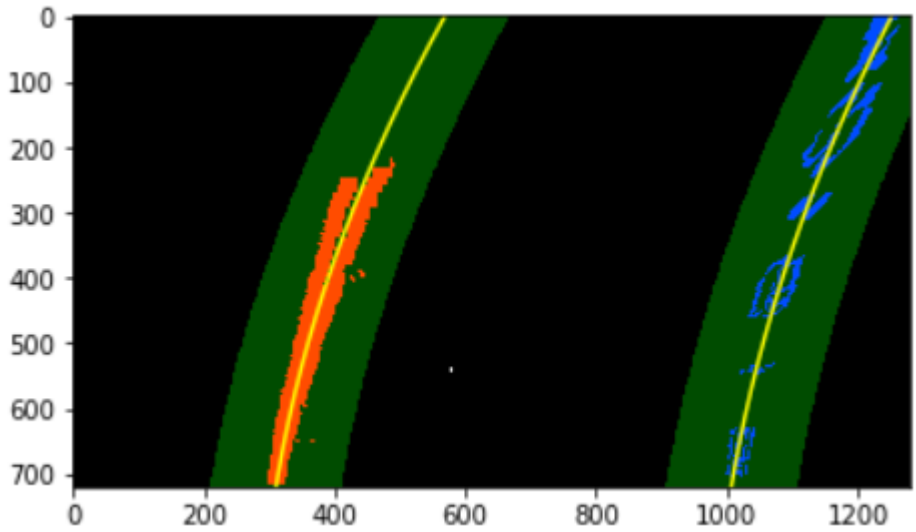


Then, a sliding window moving left and right was created.While circulating through this window, I found an index inside each window. With indexes, nonzerox, and nonzeroy, x and y coordinates of suboptimal points were detected.The points obtained here were painted red on the left and blue on the right.After these points were connected to a function called concatenate, the coefficient of polynomial was obtained using a function called polyfit.Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:



For a more efficient calculation, I used a method of finding suboptimal

points only in the range of +/-margin in the case of knowing polynomials.



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 did this in lines # through # in my code in `my_other_file.py`

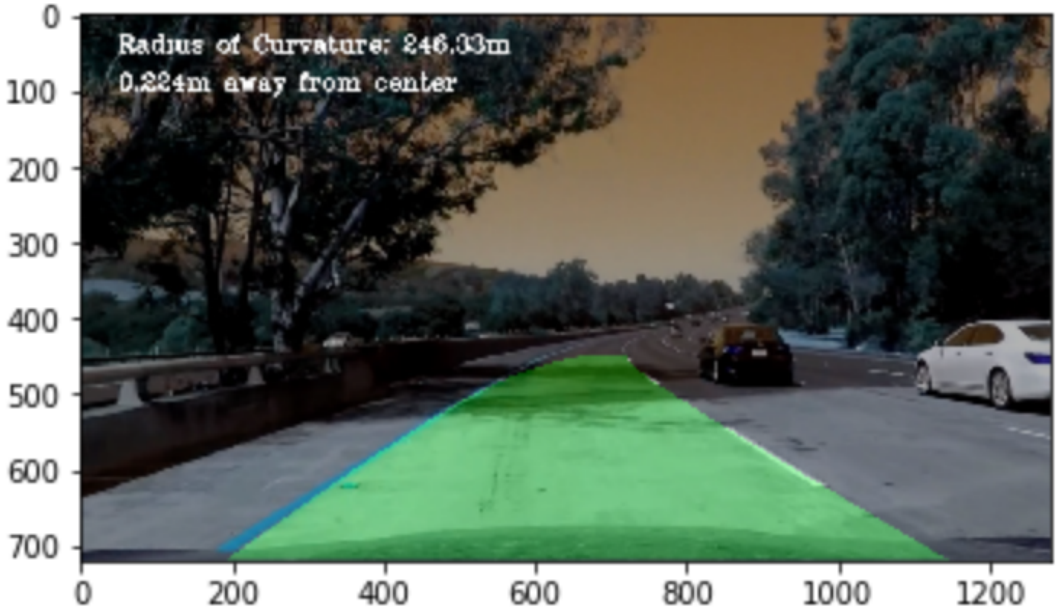
After multiplying by the ratio of meter per pixel, The formula below was applied.

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

After obtaining left_curverad and right_curverad, the average was obtained to obtain the curverad. Using the distance between the curriculum and the center of the screen, you can see how far out of the center.

6. 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 lines through applying the code called Pipeline in my ipynb file. Here is an example of my result on a test image: After inverse perspective transform, the space between the polynomic was colored and combined with the original 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!).

The video will be attached together when submitting.

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 the curvature of the road becomes too small, only one lane may be detected. Therefore, it is necessary to apply a more sophisticated algorithm to detect this more accurately.

If there is a speed limit or arrow picture on the road, it may interfere with detection. This may be solved by elaborating the mask setting.