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.

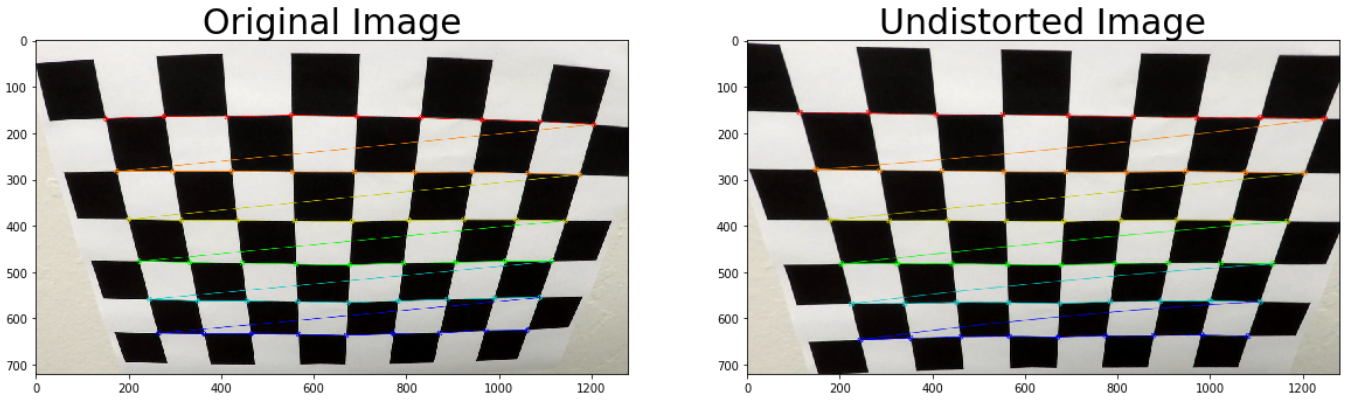
Camera Calibration

Computed the camera matrix and distortion coefficients

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

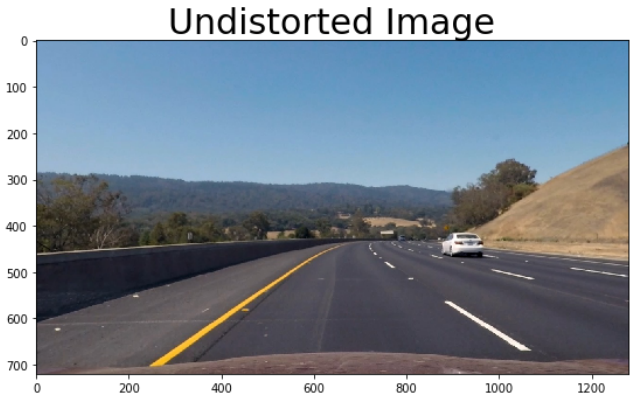
I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. The parameter aare saved in a pickle file for further use. Then I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:



Pipeline (single images)

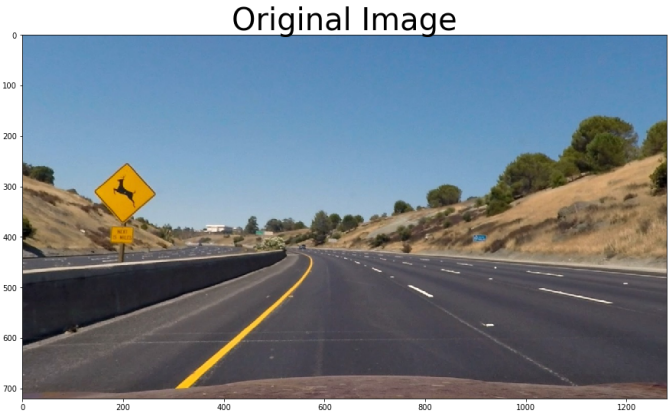
1. Example of a distortion-corrected image.

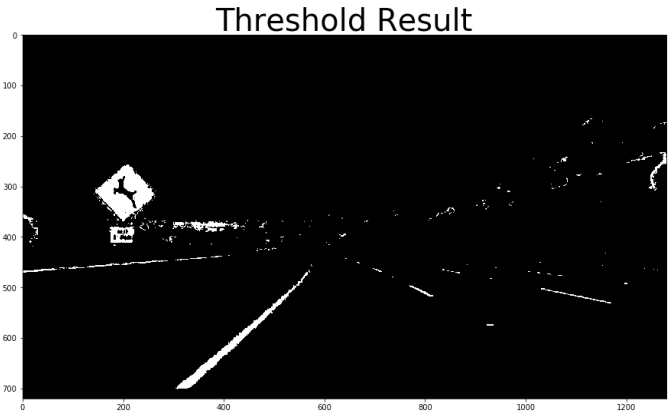
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:



2. Color and Gradient Thresholds

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines 15 through 51 in the 4th code cell of the IPython notebook). Here's an example of my output for this step. (note: this is not actually from one of the test images)





3. Perspective Transform

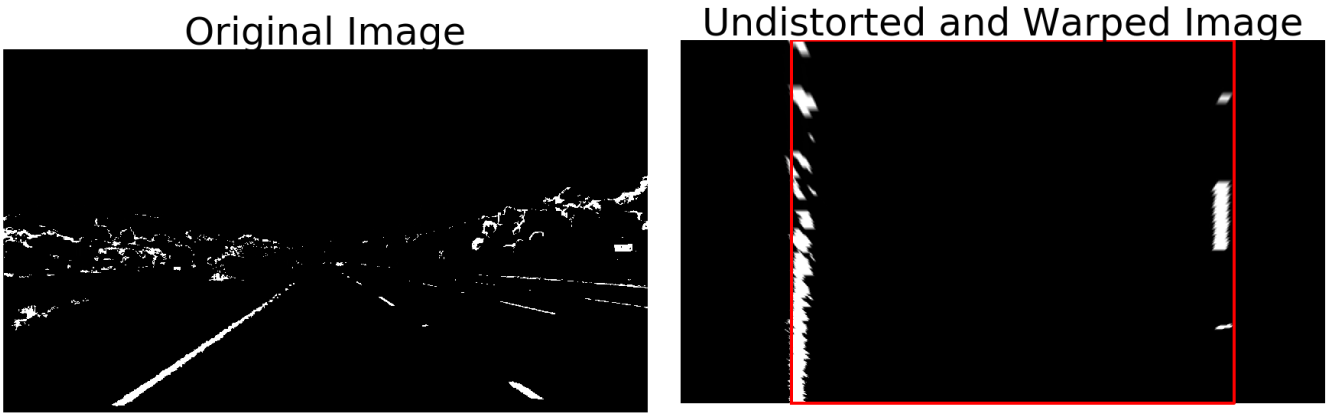
The code for my perspective transform includes a function called birds_eye(), which appears in lines 1 through 13 in the in the 4th code cell of the IPython notebook. The birds_eye() function takes as inputs an image (img), as well as source (src) and destination (dst) points. I chose the hardcode the source and destination points in the following manner:

```
src = np.float32([[0.1*imshape[1],0.9*imshape[0]],[0.415*imshape[1],
                  0.65*imshape[0]], [0.585*imshape[1], 0.65*imshape[0]],
                   [0.9*imshape[1],0.9*imshape[0]]])
dst = np.float32([[0, imshape[0]],[0, 0], [imshape[1],0],
                    [imshape[1], imshape[0]]])
```

This resulted in the following source and destination points:

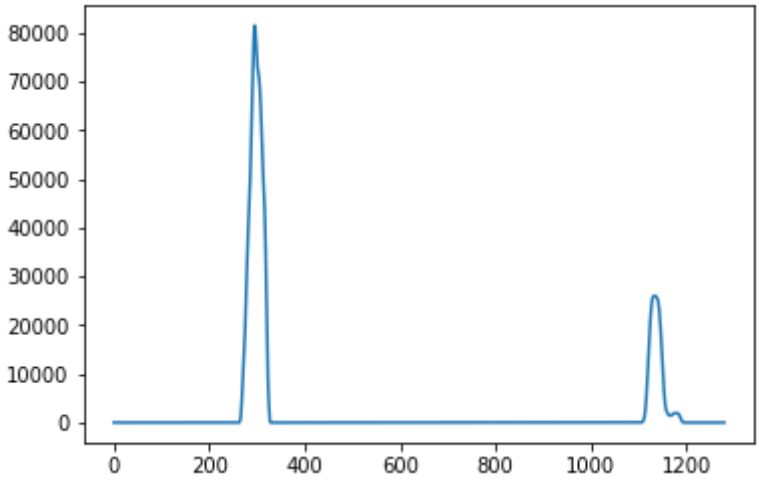
Source	Destination
128, 648	0, 720
531, 468	0, 0
748, 468	1280, 0
1152, 648	1280, 720

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.

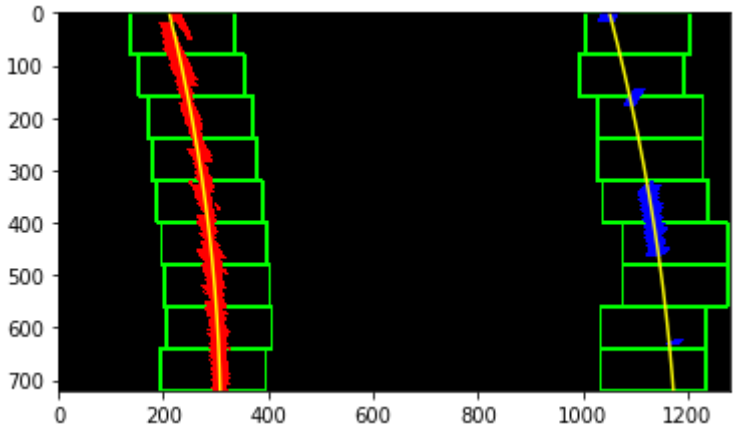


4.Polynomial Fitting

The histogram of image is show below. The two peaks in the histogram indicate the most possible position the lane line is.



I use sliding window algorithm(at lines 54 through 131 in the 4th code cell of the IPython notebook) to extract the pixels of interest from the bottom to the top of the image. Then all the pixels of interst are using to fit a 2nd order polynomial curve with polyfit() function. The yellow line in the image shown below is the fitted curve.



5. Measuring Curvature and Vehicle Position

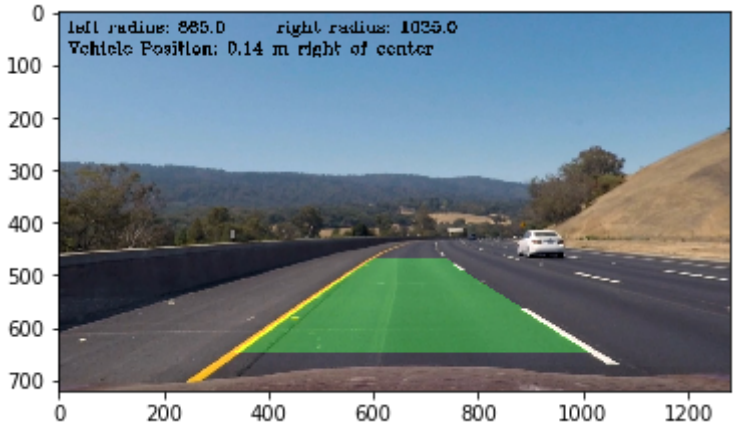
I did this in lines 144 through 163 in in the 4th code cell of the IPython notebook. The formula I use is.

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

And I assume that camera is mounted at the center of the car, so I calculate how many pixels are in between lanes and center of the car then convert it to meters to get the position of the car.

6. Pipeline Output

I implemented this step in lines 168 through 198 in in in the 4th code cell of the IPython notebook. I use inverse matrix to project the lane region I found back to the image. Here is an example of my result on a test image:



Pipeline (video)

Final Video Output

Here's my result: link to my video result (https://youtu.be/YaaAmK1cHQw)

Discussion

Problems Encountered

It's pretty hard to find a proper threshold value for getting a curve under every condition. This implementation may have problems when the there is no lane lines available or rainy weather occurs. I think the deep learning approach that I did in project 3 is much more robust than this one and with less coding efforts as well.