Example Writeup

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 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

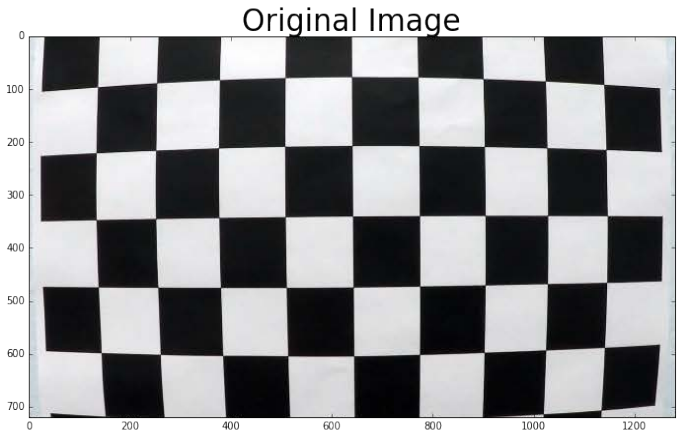
Camera Calibration

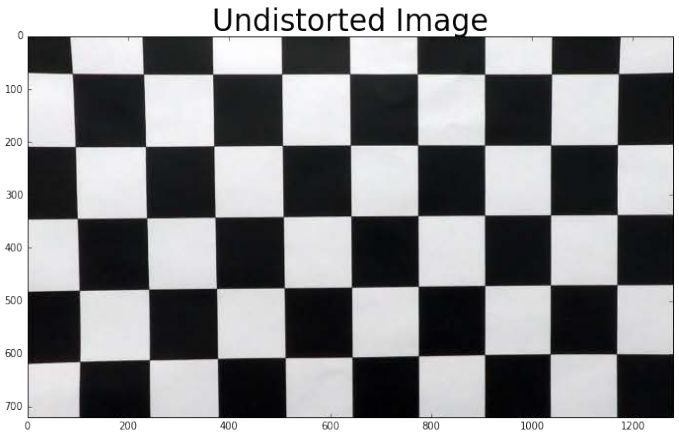
1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

Yes. First, Camera Calibration,use some chessboard images to catch the imgpoints on the chessboard. Then, according to the coordinate of the imapoints and objpoints, calculate the camera matrix and vector of distortion coefficients to run out the undistorted images. This step is contained in the first and second code cell of the jupyter notebook.

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.

Then, I used function cv2.calibrateCamera() with `objpoints` and `imgpoints` to compute the camera matrix and distortion coefficients. According to the matrix and coefficients, we can use function cv2.undistort() to run out the undistorted image, just as what shown below:

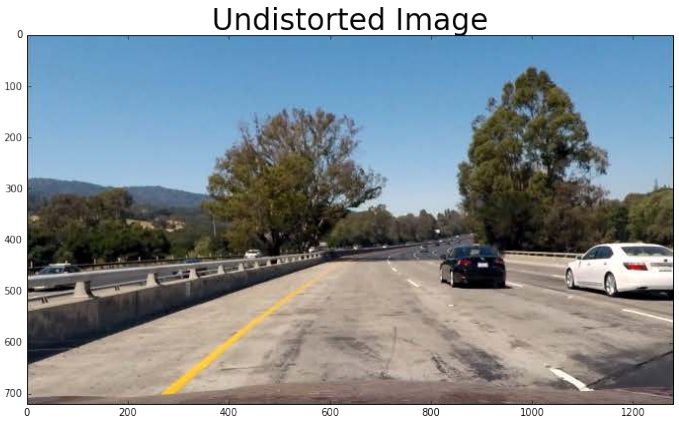




Pipeline (single images)

1. Has the distortion correction been correctly applied to each image?

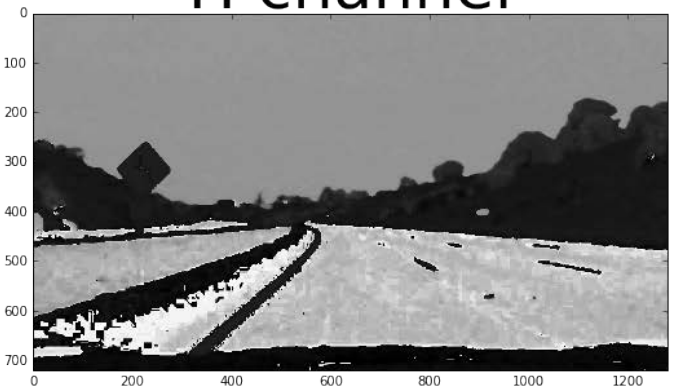
I just put the pipeline on a image for test, as what shown below:



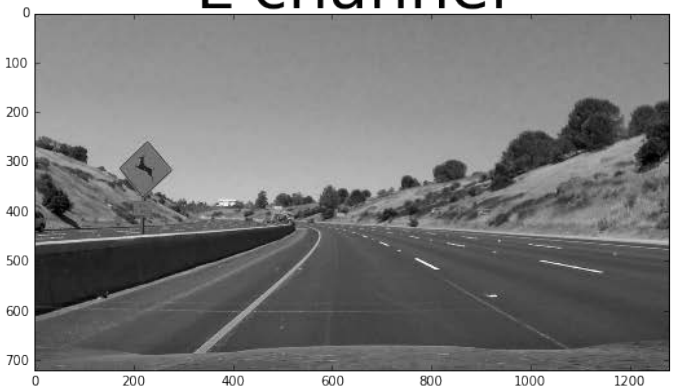
2. Has a binary image been created using color transforms, gradients or other methods?

Yes. Here, I will tell you how I achieve it step by step.
according to what said in the course, we know, the S-channel of image in HLS format has the better effect. Fisrt, I have to prove this fact. I randomly choice an image, use function cv2.cvtColor to transfer image's format into HLS and plots all channel images. Compared with each channel image of RGB, HLS, It's easy to prove the fact.
Second, we have to filter noise in the picture, some color or features we don't need. Here, I used sobel operation, with function cv2.Sobel, np.zeros_like, combined with different threshold, stacked these layers to get a good binary output. I choise the R-channel of image in RGB format, image in gray format through function cv2.Sobel with thresh=(40, 255), direction=x and image in HLS format through function cv2.Sobel with thresh=(50,255),direction=x to stack them into a complete binary output.
The code of this step is in the 6-9 blocks of jupyter notebook.

H channel



L channel

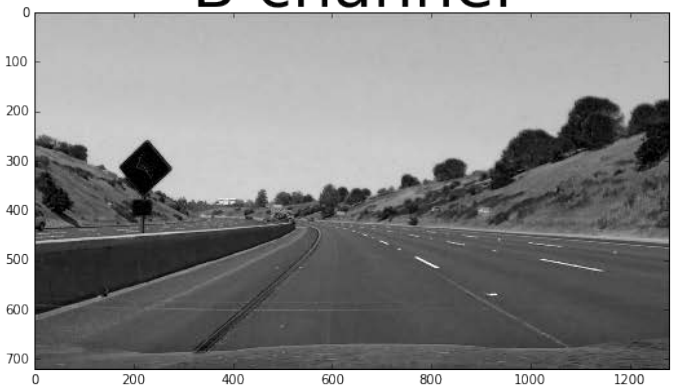


S channel

R channel

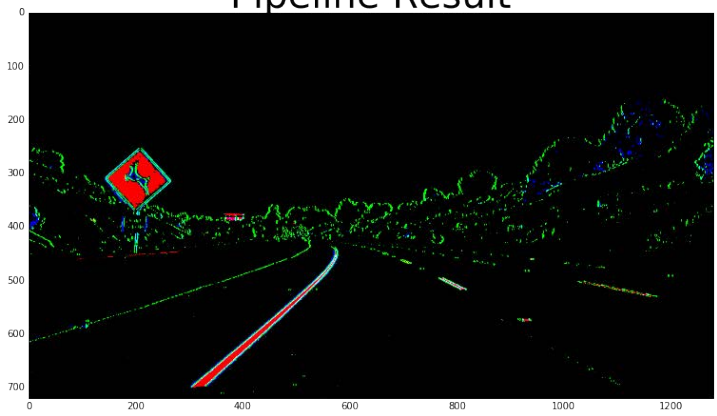
G channel

B channel



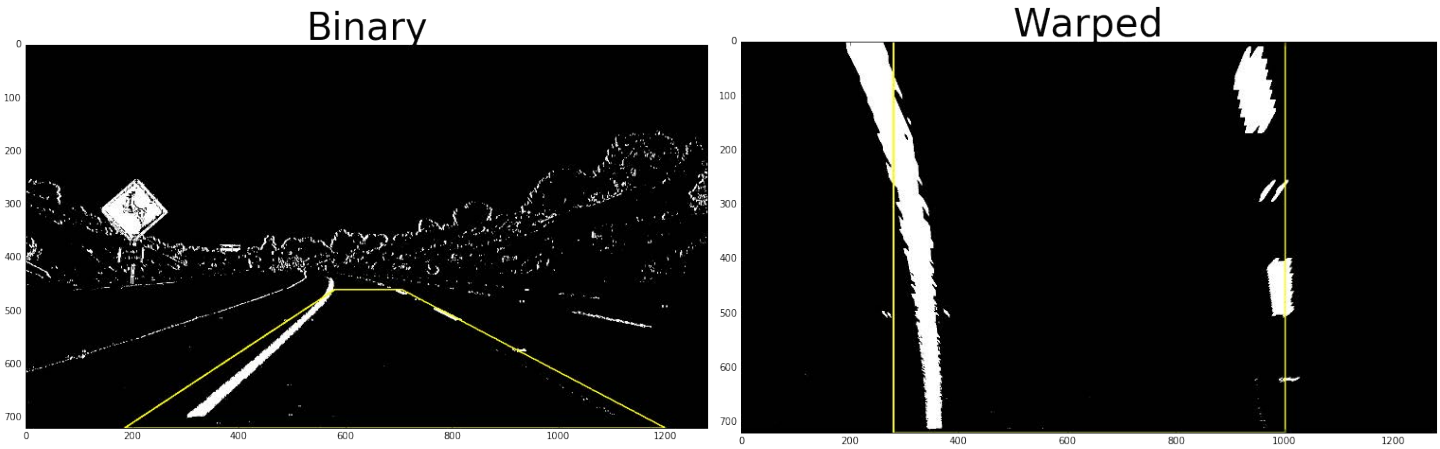
Original Image

Pipeline Result



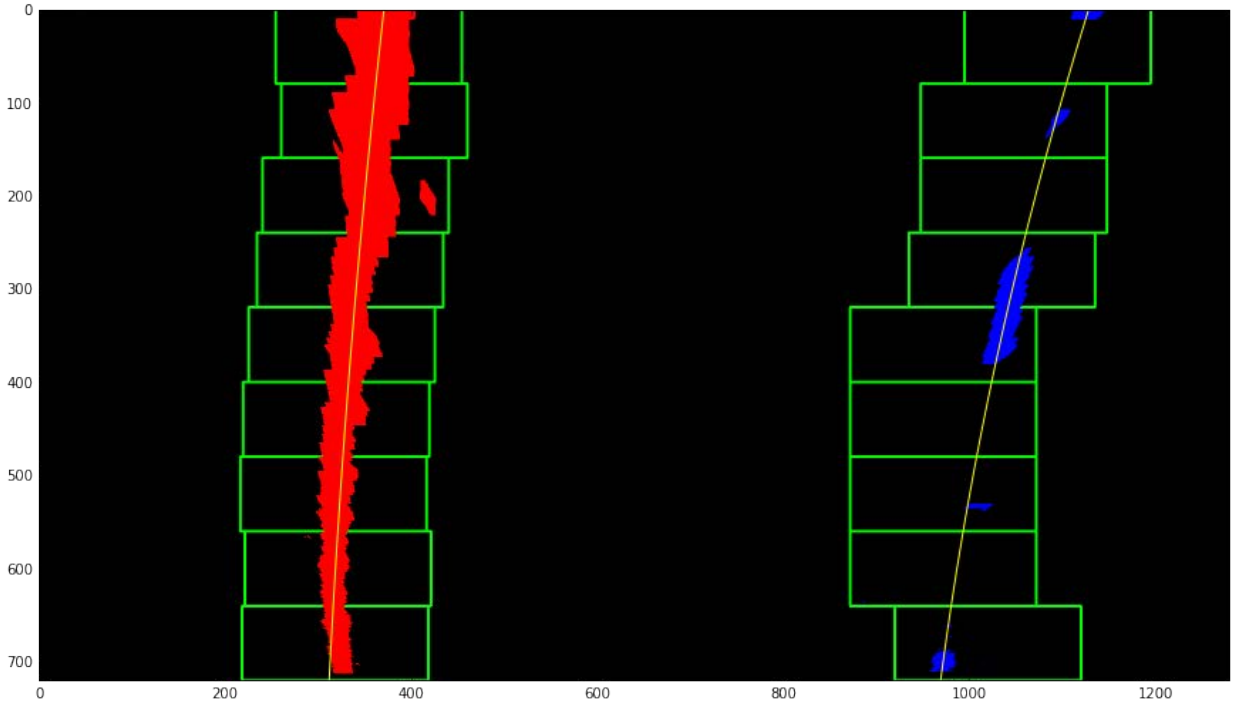
3. Has a perspective transform been applied to rectify the image?

The code for this part is in 10,11 blocks of jupyter notebook.
First, I point out the four points of the specific quadrilateral (The figure on the left below shows) to set the coordinate as src and the opposite coordinate as dst. Then, I used function cv2.getPerspectiveTransform with src and dst to compute the transfer matrix, M. With M and the binary output image, function cv2.warpPerspective will output the unwarped image.



4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

The code of this part is in the block 12,13.
Fisrt, we have to find the start point of the left/right curve. Use function np.sum, we can get the histogram of binary output which express the effective pixel density.
After get the end point of lane line, I set slide windows to split the curve.
Then, from every window, I can collect x and y index of left and right lane line, (leftx,lefty), (rightx,righty).
Use function np.polyfit with (leftx,lefty) / (rightx,righty), I can get the algebraic expression of the curve I want to draw. Finally, we get the curve drawn as shown below.



5. Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?

Yep, sure did! the code for this part is in the block 16. According to the algebraic expression, we can compute the radius of curvature and print them on the image.

Pipeline (video)

1. Does the pipeline established with the test images work to process the video?

It sure does! Combine all what we did before, we get the pipeline.!

README

1. Has a README file been included that describes in detail the steps taken to construct the pipeline, techniques used, areas where improvements could be made?

You're reading it!

Discussion

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

From the video.mp4 sent, it's easy to find sometimes the pipeline catch the wrong lane line. Although, it's few and did't affect the overall effect. However, it shows the problem in the pipeline. And from the challenge video, it's obvious that the pipeline is not robust.

Try to fix the problem, better binary output is the key.
1.Try other threshold case to get better output.
2.Try other channel of image in other format.
3.Try to stack more layers with effective value.