Advanced Lane Finding Project

The goals / steps of this project are the following:
1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
2. Apply a distortion correction to raw images.
3. Apply a perspective transform to distorted image ("birds-eye view").
4. Use color transforms, gradients, etc., to create a thresholded binary image base on the perspective image.
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

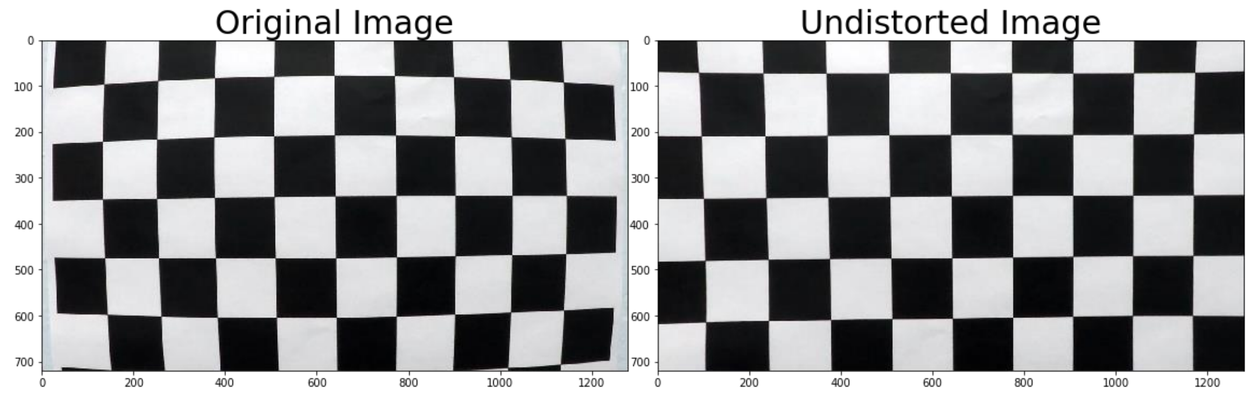
The code for above steps is contained in code cells of the IPython notebook file called “project.ipynb”.

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. This is in code cell 2.

Camera Calibration

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

In code cell 3, I used output `objpoints` and `imgpoints` to compute the camera calibration matrix and distortion coefficients using the `cv2.calibrateCamera()` function, and saved these parameters in a file named “calibration.p”. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



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

It is not easy to catch the un-distortion. Compare the difference between vehicle hood.

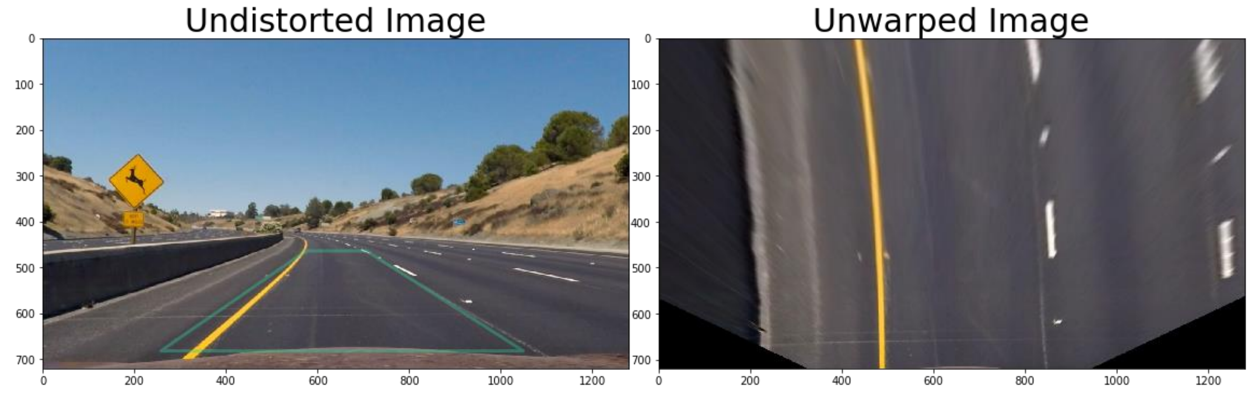
Pipeline (single images)

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

Yes, the distortion parameters are saved in file and packed in function. All the raw image will be undistorted in the first step of the image processing pipeline.

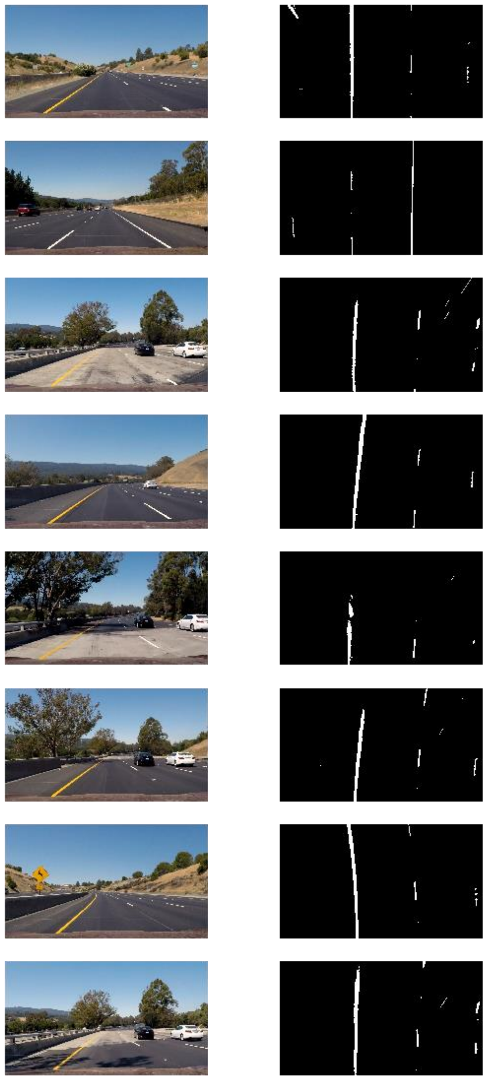
2. Has a perspective transform been applied to rectify the image?

In code cell 8 and 9, I did the perspective transform earlier on undistorted image, then applied color/gradients transforms to form binary image. The reason is that do perspective first can be more focused on the lanes other than the other objects in the image. I manually selected the vertices (source and destination) points, because these points by large chance is stable in the image of each frame, it is from the same source camera. I used `cv2.getPerspectiveTransform()` function to get the perspective view of the undistorted test image as:



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

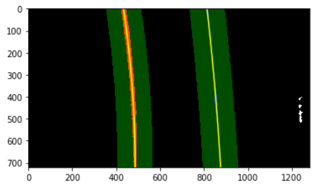
From code cell 10 to cell 30, I did the sobel absolute, magnitude and direction, and also magnitude + direction combination transforms to get binary images. The test code allows me to tune the threshold directly (kernel size/low/high) to get a best transformed binary image. I also did color space transforms used HLS and LAB color space channels. All the test output images can be found in the output_images folder, named by index from 04 to 11. I ended up with just using HLS L channel + LAB B channel, L channel gave the white lane and B channel gave the yellow lane. I tried sobel magnitude and direction, but it was very noisy. Once I decided use HLS+LAB, I added in the image processing pipeline and test on all test images as shown:



It is able to find the lanes for each test images.

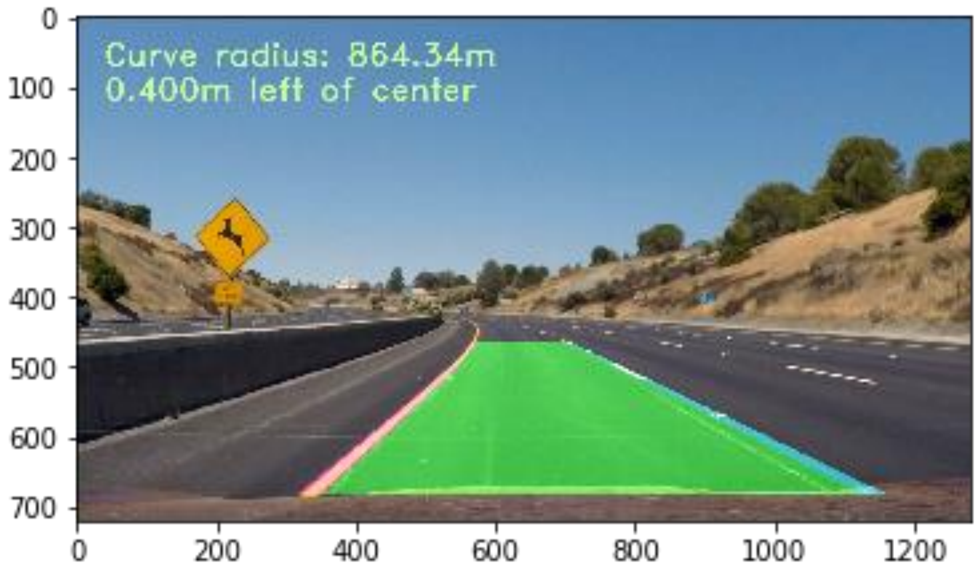
4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

In code cell 31, I used sliding window polyfit to get all lane pixels. The histogram is helpful to locate the left and right lanes. In code cell 34 and 35 did the polyfit using fit from previous frame and visualizing the output as



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?

Yes, the radius of curvature on the road had been estimated and the position of the vehicle with respect to the center of lane. The curvature calculation is referenced from this link: https://www.intmath.com/applications-differentiation/8-radius-curvature.php, and equation code is got from Udacity tutorial sample code. The results that warp back to the original image is:



Pipeline (video)

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

The final image processing pipeline worked well on the project_video. The output file is named as “project_video_output.mp4” in the project folder. For the challenge videos, it was not working perfect.

Discussion

The challenge is the light condition that affects a lot and the lane shape if it is varying very frequently. The pipeline is hard to find the correct lanes.
The improvement that can be made to complete a robust design: The binary transform should be more complex, the combination of sobel mag/direction and color space in HLS L and S channel, or even some other transforms. The threshold parameters tuning is an issue when combined several transform functions. There is no fixed setting that can work properly, a dynamic tuning maybe a solution.