CarND 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 threshold-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

The following points will be addressed in the write-up.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

This document serves as the project write-up

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 in the IPython notebook “CarND-Advanced-Lane_Lines.ipynb” in section 7 contains the camera calibration.
The process I used was to first prepare the object reference points in the x,y,z coordinate of the chessboard corners. It is assumed the chessboard is a flat plane and so the z value is always zero. The code detects the features programmatically and computes the object-points and image-points. Within the

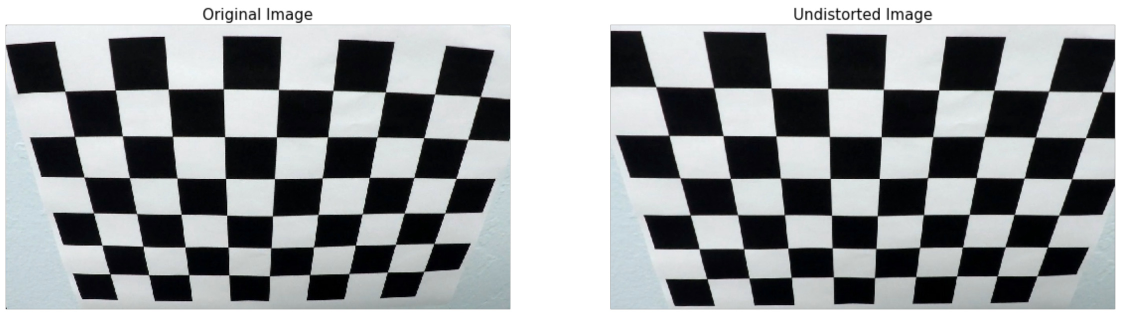
fuction undistort (section 6 of the notebook), the camera calibration and undistortion takes place using the object-points and image-points previously computed.

Pipeline (single images)

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

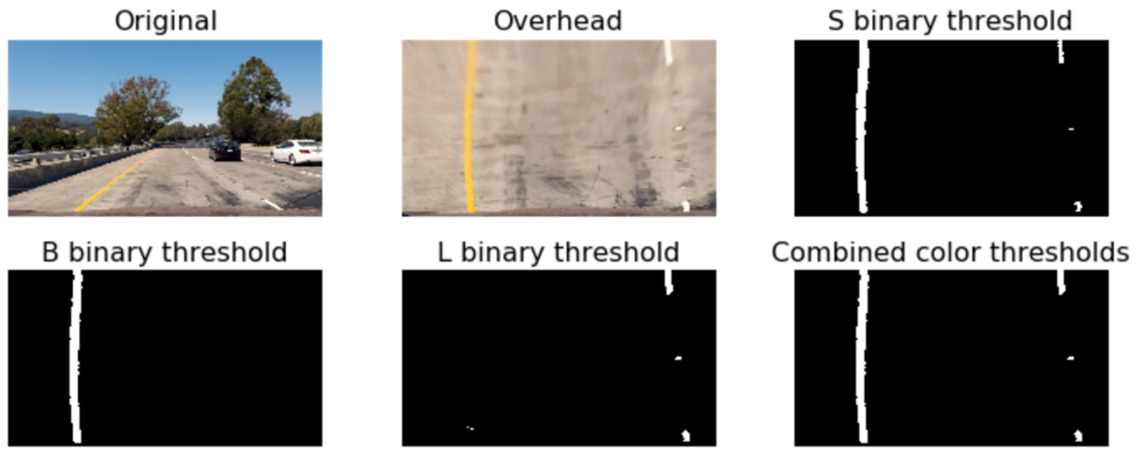
The following is an example of a before and after undistorted image. Please note

the elimination of the curve to the chessboard in the right image:



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.

In section 12 (function named: apply_thresholds()), both color and gradient thresholds were used to generate a binary image. Here is an example of the transformation.



3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code in section 9 and 10 of the notebook performs the overhead (aka birdeye) transform of the image. The function name is “overhead_view()”. The function undistorts and transforms the image into a bird-eye view of the road.

An important factor to the transform of the image is the source and destination points. The source and destination values were hardcoded to the following:

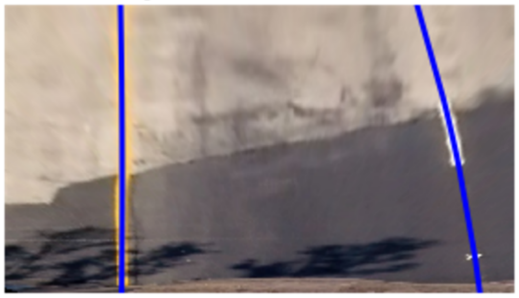
```
    src = np.float32([[490, 482],[810, 482],
                      [1250, 720],[0, 720]])
    dst = np.float32([[0, 0], [1280, 0],
                     [1250, 720],[40, 720]])
```

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.

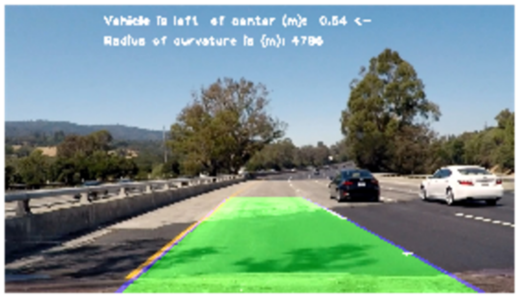
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

In sections 15 and 16 of the notebook there are functions for computing the left and right lane line curves. A histogram is used to determine the area with the largest number of pixels set to 1 from the binary threshold along the x-axis.

Fit Polynomial to Lane Lines



Filled Lane Lines

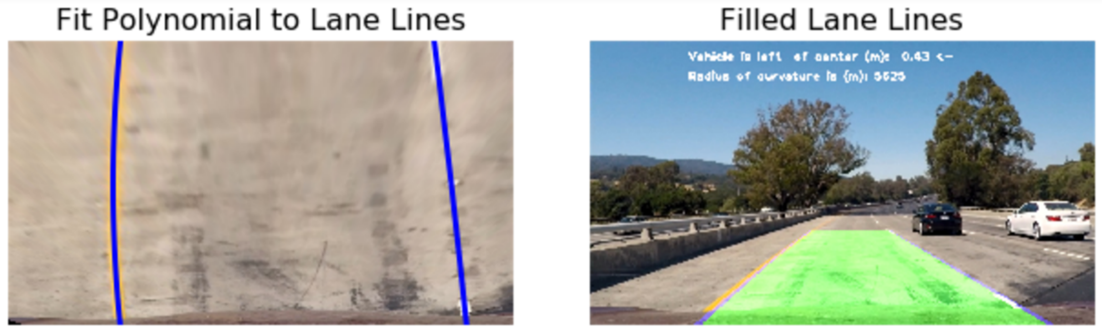

Vehicle is left of center (m): 0.54 <-
Radius of curvature is (m): 4786

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.

In section 22 of the Notebook, the radius of the curvature of the lane is computed. Also, the position of the vehicle relative to the center of the lane is computed in section 22 also.

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

The following an is example of the lines drawn also with the lane filled.


Fit Polynomial to Lane Lines
Filled Lane Lines
Vehicle is left of center (m): 0.43 <-
Radius of curvature is (m): 5525

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!).

Here is a link to the result.mp4

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?

The project posed multiple challenges. The first was to get the thresholds setup correctly for a variety of road images. There were trade-offs between the types of binary thresholds presented different results for the different road types.

One approach I would like to take is to search from the center of the image out to the left and to the right for the lane lines. That way if the first histogram spike would likely be the lane line. This approach should address any variation to the image alongside the edges of the road.   Additionally, I would also like to filter on just the yellow and the white lines by filtering the color. I believe this would yield a more robust lane detection system.