CarND-Advanced-Lane-Lines writeup

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

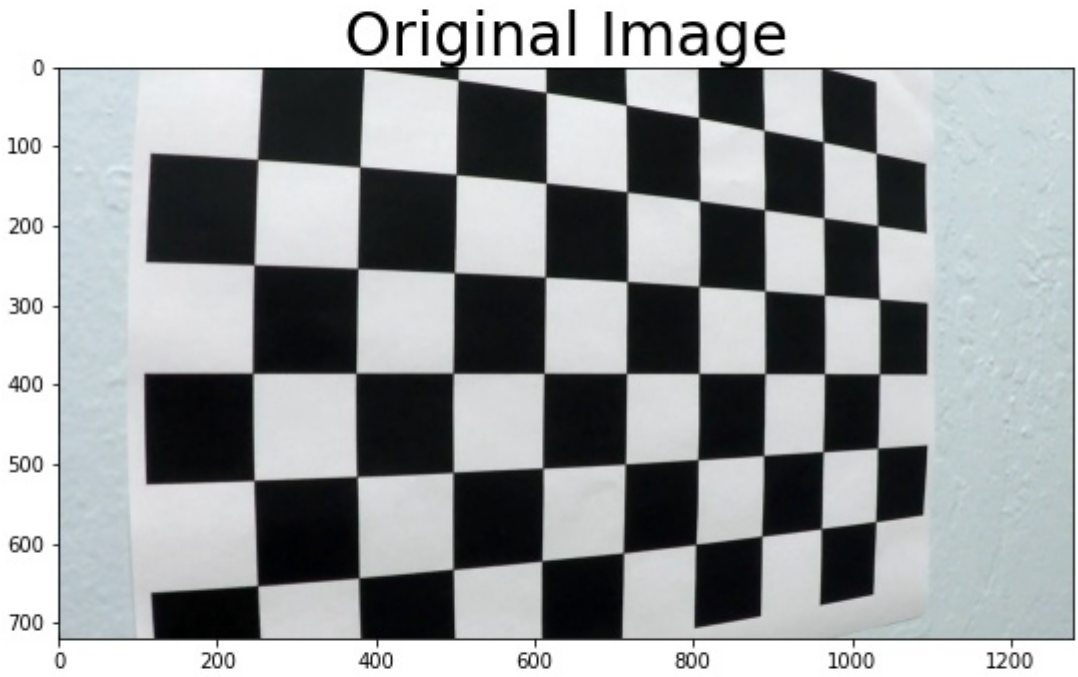
In this section, I will consider the rubric points individually and describe how I addressed each point in this implementation.

Camera Calibration

The code for this step is contained in the first code cell of the IPython notebook located in "./advanced_lane_lines.ipynb", section "Camera calibration matrix and distortion coefficients".

I started by preparing the "object subspace", which is 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. Every time the library found the chessboard corners in a test image, the object coordinates (i.e., `obj_subspace`) were appended to an array of coordinates called `obj_points`. Furthermore, the (x, y) pixel position of these corners were appended to the 2d points in the image plane, `img_points`. This step has been done with 20 calibration images to achieve good calibration results.

I then used the output `obj_points` and `img_points` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. This distortion correction was applied with `cv2.undistort()` in the following two notebook cells, the first one to see the distortion correction on a calibration test image, and the other to see this result on an image in the road. Below is the result of the chessboard with distortion correction:



Pipeline (single images)

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

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

The code to undistort an image is wrapped in a function called `undistort` and is used in the following way: `undistort(img, distortion_mtx, distortion_coeffs)`

Where `img` is the result of `img = plt.imread('test_images/test1.jpg')`.

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

The code for the binary transformation is in the section "Thresholded binary image" of the iPython notebook. For this, I used a combination of color and gradient thresholds to generate a binary image. In particular, I applied a threshold on the saturation channel of the image in HLS and a threshold on the sobel operator in the x direction. Here's an example of my output for this step:

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

Similarly to the previous sections, the code for the perspective transform is in its respective section, the " Perspective transform" section of the iPython notebook.

The code basically consists of a function called `warper()` that takes an image, source, and destination points, and returns the warped image. To select the source and destination points, I followed the steps below: 1) Extract the lane lines pixels from the test_image manually observing the pixel lane lines from the picture. 2) Draw the polygon in the picture 2) Fine-tuned the resulting polygon with slight corrections.

As a result, the code looks like this:

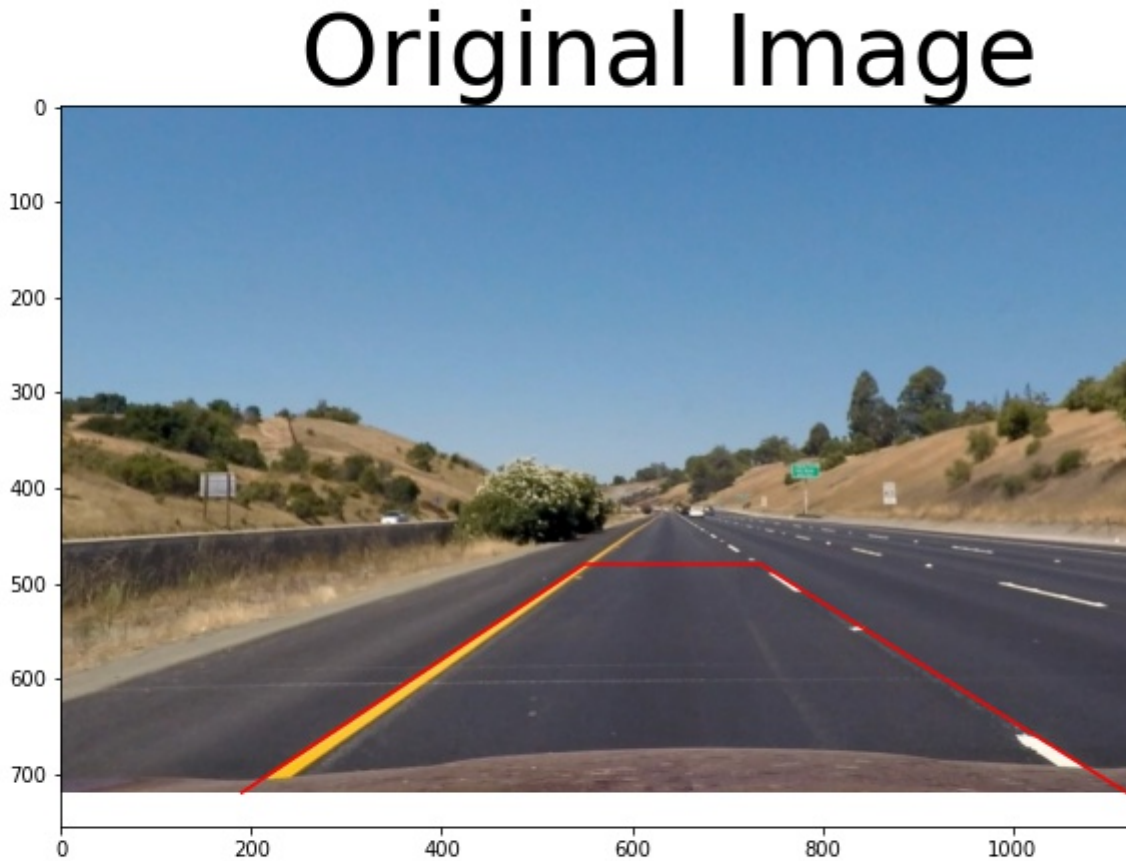
```
src = np.float32([
    [190, img.shape[0]],
    [550, 480],
    [735, 480],
    [1120, img.shape[0]]
])
dst = np.float32([
    [int(img.shape[1] // 4), img.shape[0]],
    [int(img.shape[1] // 4), 0],
```

```
    [int(img.shape[1] - img.shape[1] // 4), 0],
    [int(img.shape[1] - img.shape[1] // 4), img.shape[0]],
])
```

This resulted in the following source and destination points:

Source	Destination
190, 720	320, 720
550, 480	320, 0
735, 480	960, 0
1120, 720	960, 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.



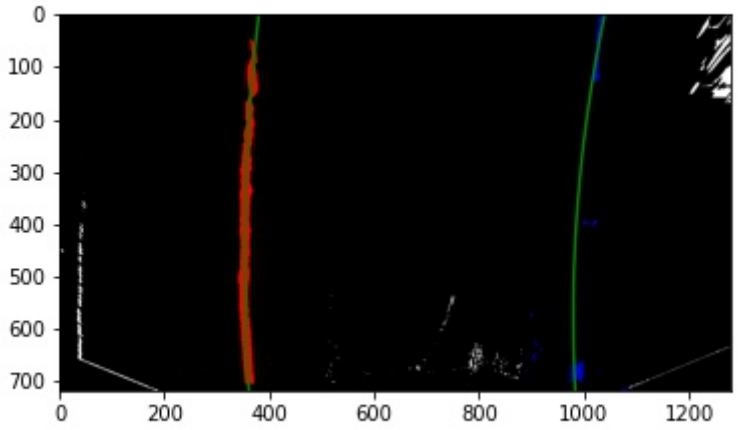
4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

The code for the polynomial fit resides in the "Fit lane line pixels with a polynomial" section of the iPython

notebook. Then, the code for finding lane lines (both through sliding windows, and using the previous polynomial) is in the next section, "Lane line pixels detection".

The code is then tested below in the "Steps so far" section, where I plotted the different transformations and the lane lines found through sliding windows and using the polynomial history. The following is an example of the polynomial found using the polynomial history:

```
img, left_line, right_line = lane_line_pixels_detection(binary_image, left_lin
plt.imshow(img)
```



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?

In section "Radius of curvature" of the iPython notebook there's a call to measure the radius of curvature of the "test_images/test1.jpg" image. The implementation of this formula has been done as a method of the Line class which can be found in the beginning of the "Fit lane line pixels with a polynomial" section.

As regards the position of the vehicle with respecto to center in the lane, it has been implemented in the " Measure distance from lane centre" section of the iPython notebook.

Examples can be observed along with the projected images in the next section.

6. Has the result from lane line detection been warped back to the original image space and displayed?

I implemented this step in the code cell in the "Project the measurement back down onto the road" section of the iPython notebook, and the it has been called in the following section "Pipeline projected". The following are examples of these computations over different test images:


Radius: 13443.0 m
Vehicle is 0.12 m ri


Radius: 18940.5 m
Vehicle is 0.17 m ri


Radius: 1157.8 m
Vehicle is 0.08 m ri


Radius: 2459.1 m
Vehicle is 0.08 m le


Radius: 2692.6 m
Vehicle is 0.08 m ri


Radius: 2791.6 m
Vehicle is 0.21 m le


Radius: 4276.4 m
Vehicle is 0.24 m le

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's a link to my video result

Discussion

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

In the first frame iteration, the algorithm finds lane lines using the slidding windows approach since it has no knowledge about the location of the lines. After that, it uses information of the polynomials of the last n frames to find lane lines in a more efficient way than the former approach.

For each iteration, if the fitted polynomial has a significant deviation from the mean of the last n polynomials, it means that the line has not been successfully detected and therefore this polynomial is not taken into account in the line's history. It may also happen that, in anomalous situations, no line is detected in k successive frames. If that is the case, then the line object is reseted and thus the following frame will find lane lines using the initial approach, i.e., with slidding windows.

The pipeline might fail in situations like the challengue video, where the lane consists of not only different pavement colors but also different pavement saturation and light conditions for a considerable amount of time.

If I were going to pursue this project further, I would try to find the best accuracy of the deviation measurement and the optimal number of frames to keep in the history of the line. Also, using more color channels to cover the situations illustrated in the challenge video.