Writeup Template

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 use some other method and 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

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Camera Calibration

1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

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. I applied 'cv2.drawChessboardCorners()' function and I got this picture:

I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

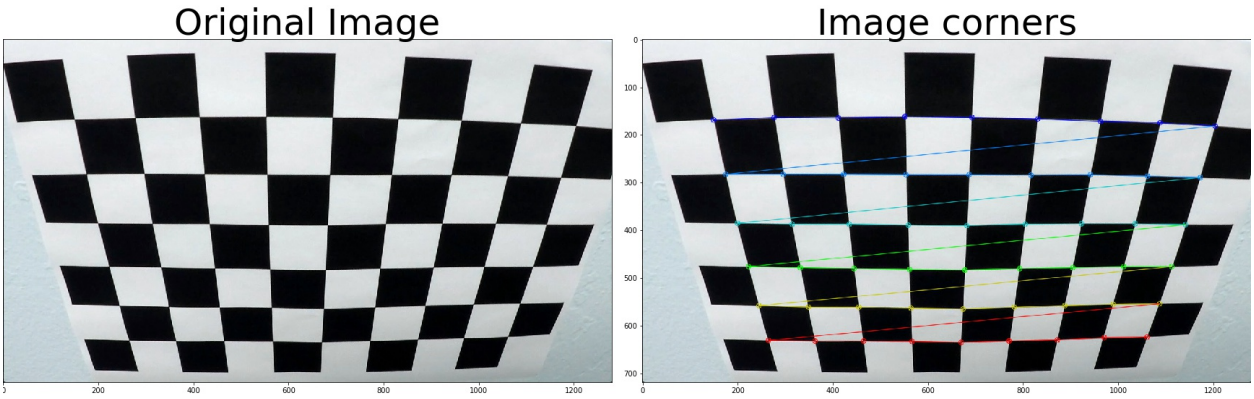

Figure 1: Chessboard corners

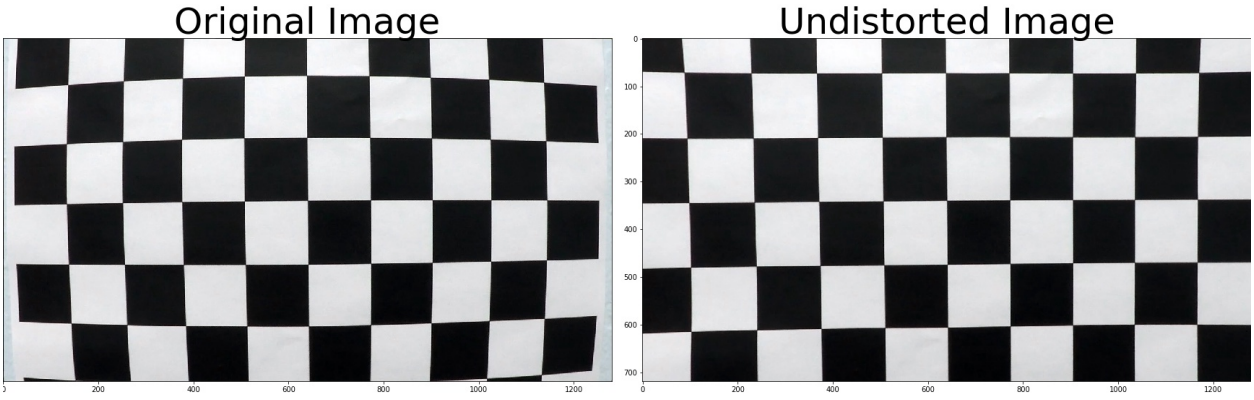

Figure 2: Undistorted chessboard

Figure 3: Undistorted road image

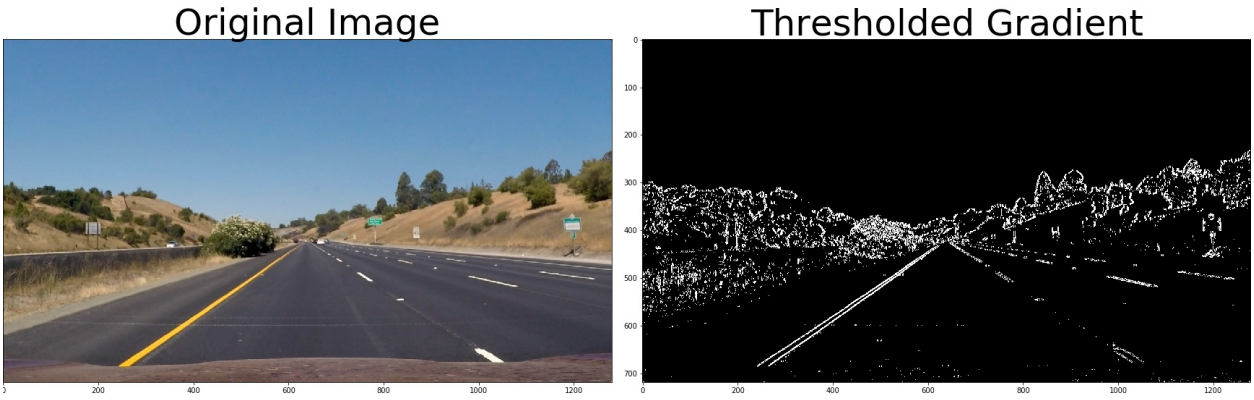

Figure 4: Threshed sobel

Pipeline (single images)

1. Provide an 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. 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.

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps at cells # 51 through # 55). The threshes were combined in cell # 55. Here's an example of my output for this step.

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

The code for my perspective transform includes a function called `pers_transform()`, which appears in cell

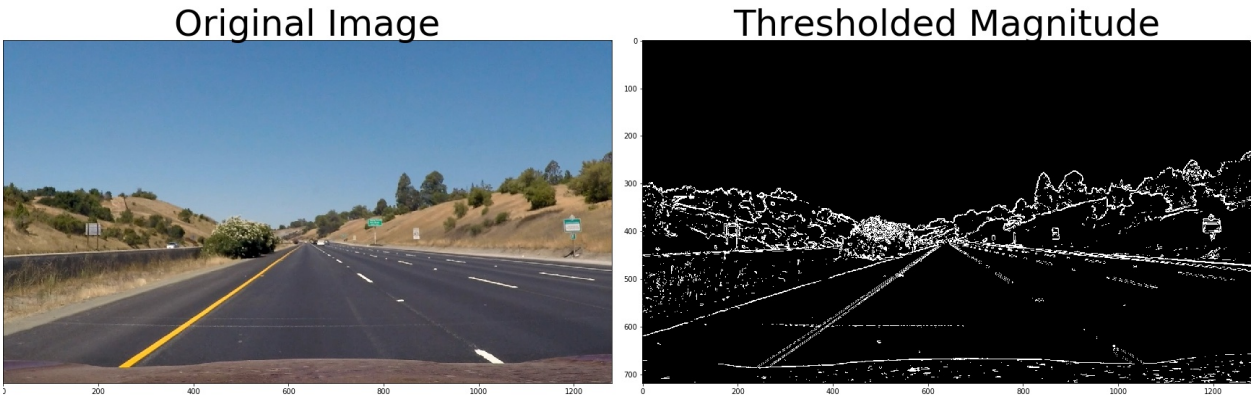

Figure 5: Threshed magnitude

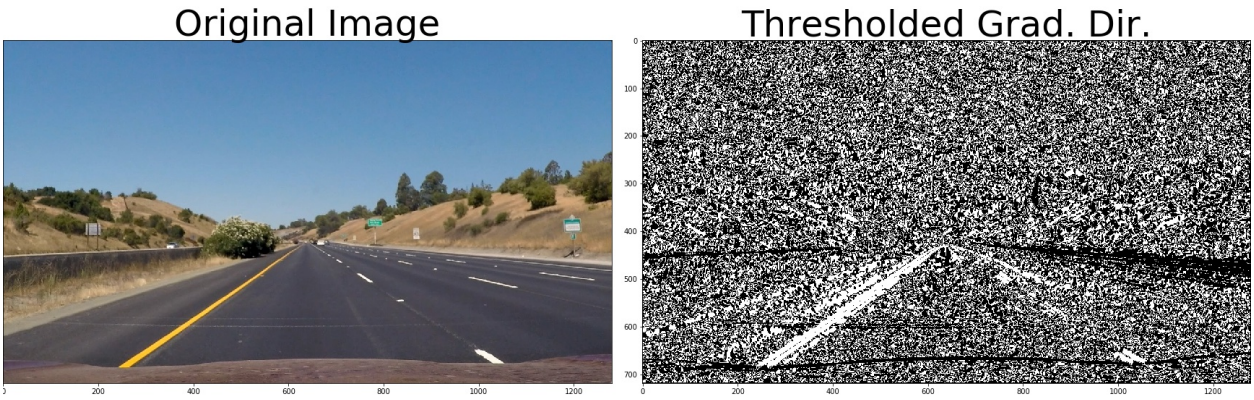

Figure 6: Threshed direction

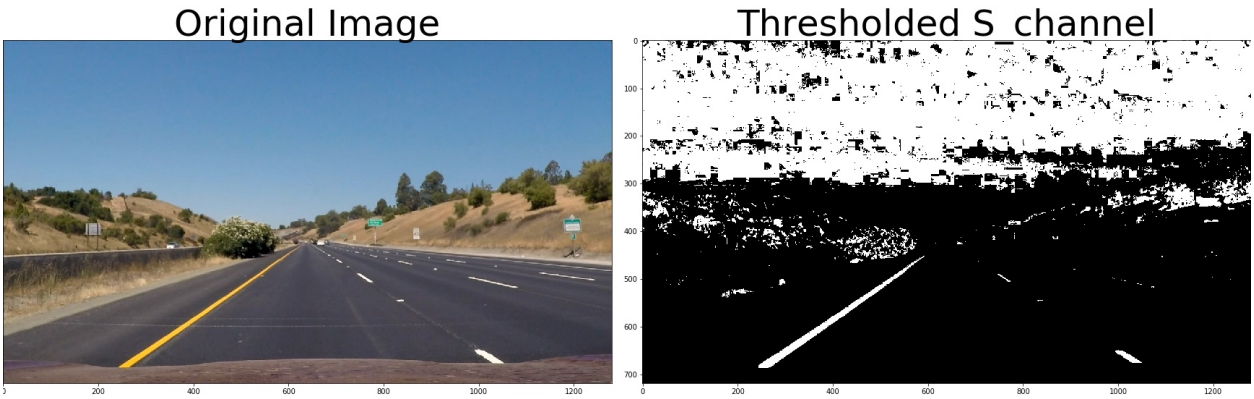

Figure 7: Threshed s_channel

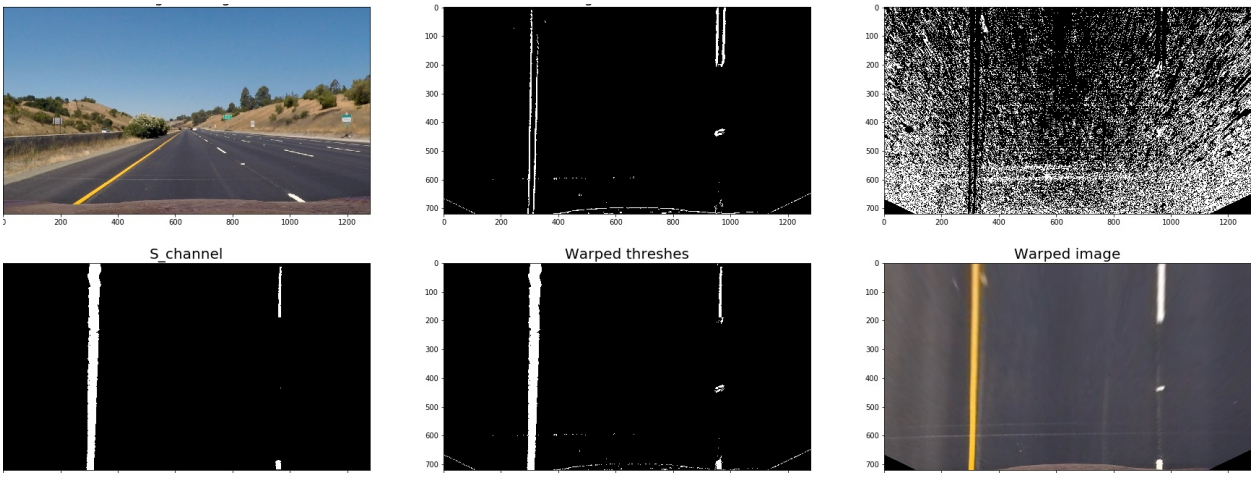

Figure 8: Threshed combined

50. The `pers_transform()` function takes as inputs an image (`img`), using the transoformation maxtirx got from the same cell by the source (`src`) and destination (`dst`) points. I defined the 'src' and 'dst' points manually:

```
src = np.float32([[470,490],
                  [758,490],
                  [1100,685],
                  [10,685]])
dst = np.float32([[100,0],
                  [1000,0],
                  [1000,710],
                  [100,710]])
```

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

The laneline was searched from scratch by using sum of histogram in 0 axis on the bottom side of the road to find a base of the lane. The road image was divided into 9 parts in y axis. The rest of lane was searched from the base with a margin of 100 pixels from both sides, once at a time to the top of the road image. The found index was store seperately in left laneline and right laneline. When the sum of pixels in a window is above a threshold of 50 pixels, the center of the points was taken as the laneline and stored seperately as well.

After windows search, the pixels index were concatenated and laneline pixels were returned as 'leftx' and 'lefty', 'rightx' and 'righty'.

A second order polynomial was calculated on each laneline. The left laneline and right laneline pixels were returned as 'left_fitx' and 'right_fitx'.

Another approach would be the next frame search, I can use it in video processing. When the last frame is good, I can use the second order polynomial from last frame for the next frame. Its base on the assumpation that the laneline of next frame should be near the laneline of last frame.

The indexs were search throughtout the nonzeroy of the frame fitted with the last frame second order polynomial with a margin of 100 pixels. The found pixels were use for the new polynomial and pixels of both lanelines.

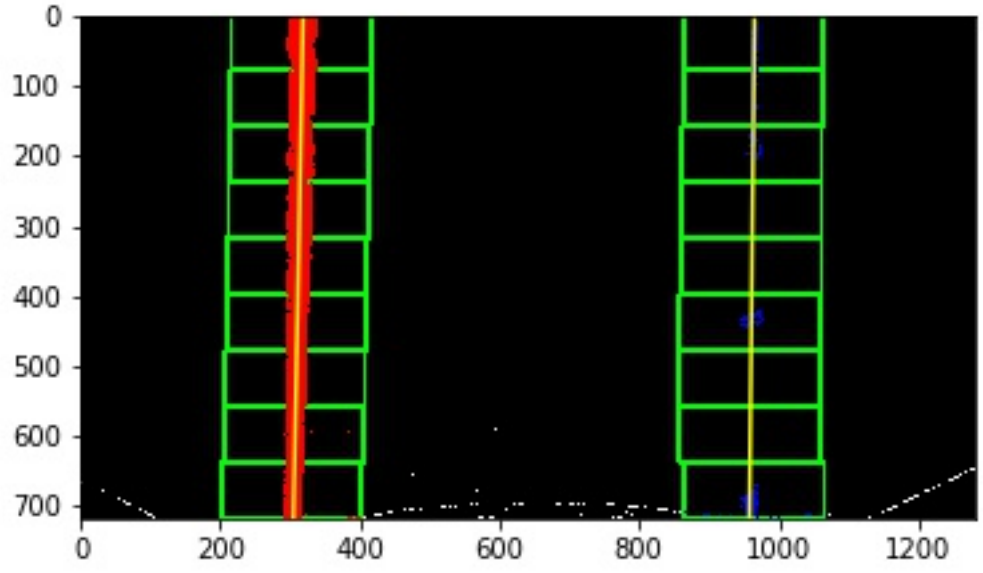

Figure 9: Blind search

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.

I coverted the pixel values into meters on the y axis by 30/720, and on x axis by 3.7/720, they both have units in meters/pixel.

The real world second order polynomial was got by coverting pixels into meter, with mutliply convertion formula.

The cuvature was caculated with Rcurve = ((1+(2Ay+B)**2**)3/2)/|2A| Where A and B are the second order polynomial, and y is the maximum y value in meters.

The car offset is the distance of image center from the lane center. The function is called 'get_param' in cell # 142.

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

I implemented this step in cell # 157 in the function `map_to_road()`. In cell # 158 I assembled a pipeline 'image_pipeline'for image processing. Here is an example of my result on a test image:

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!).

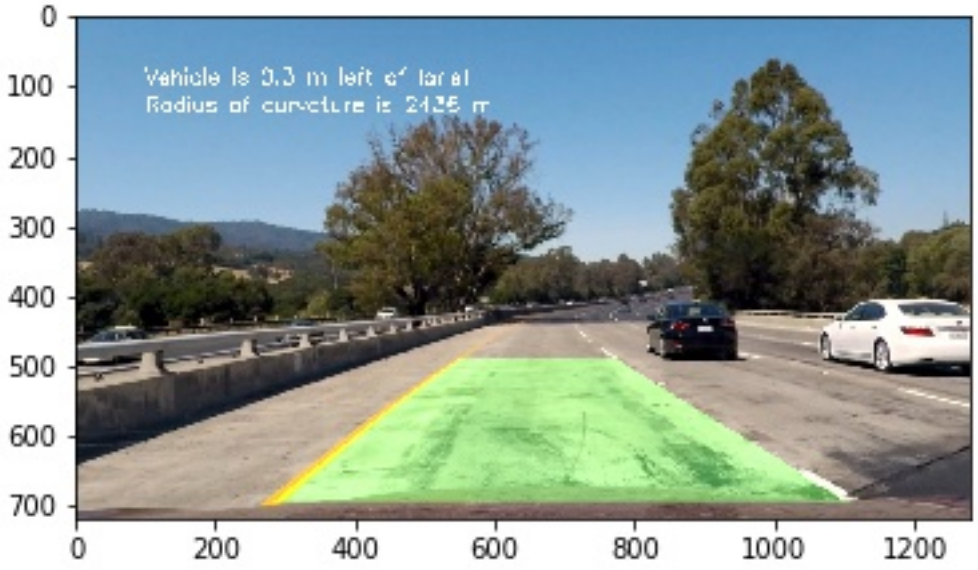

Figure 10: Marked road

The video is 'project_video_output2.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 video pipeline I used last frame second order polynomial for next frame lane searching. It could fail if one frame laneline is missing or not stable. To increase the robustness, a quality assessment should be applied to the lane finding and stored them in a good fit class. New frames should look on this class if they are good. Otherwise an averaged second order polynomial from recent frames should be used.

A hard coding can still fail in the severe shadow, sun glazing or wether conditions. I would prefer deep learning approches. Labeled data can be used to train the neural networks, and new image data can be used on the way to inprove the pipeline in general.