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!

I wrote the modified parts in red.

The points the reviewer requested to modify previously are

1. To provide a detailed explanation of how you identified the lane line pixels and fit polynomials to your detections.

2. To make sure that you are displaying a distortion corrected test image in the writeup.

3. To continue to work on the binary thresholding to make sure that both yellow and white lines can be identified throughout the conditions present in the video.

4. To correct the detection result of the lane line center.

Camera Calibration

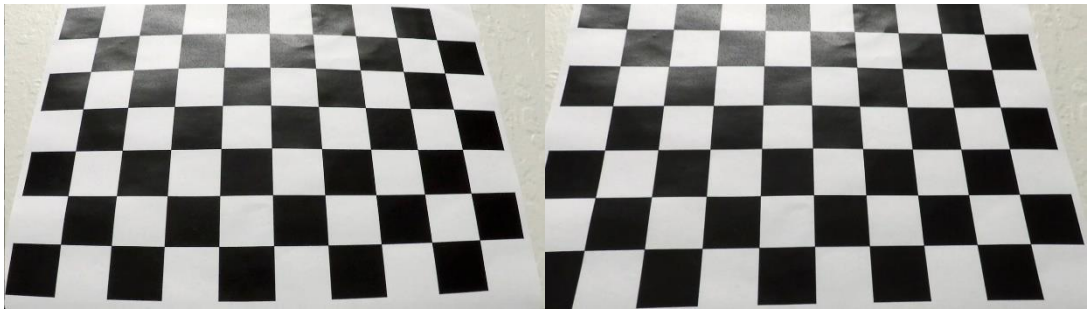
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the first code cell of the IPython notebook located in lines #29 through #76 of the file called `P4.py`).

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

Original image Undistorted image



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:

The original image

The undistorted image

I save the information of the camera calibration when I calibrated the camera by the checkboard images.(at line #76 in P4.py)

And I applied it when I correct the vehicle camera images. (At line #86 in P4.py)

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 lines #88 through #149 in `P4.py`). Here's an example of my output for this step. (note: this is not actually from one of the test images)

I modified the threshold treatment. (Actually I failed to implement the color threshold function.) The below images are the before modified one and the after one.

The previous image

after threshold treatment

The current one

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 is in lines #151 through #168 in the file P4.py. It takes as inputs an image (preprocessImage), as well as source (src) and destination (dst) points. I chose the hardcode the source and destination points in the following manner:

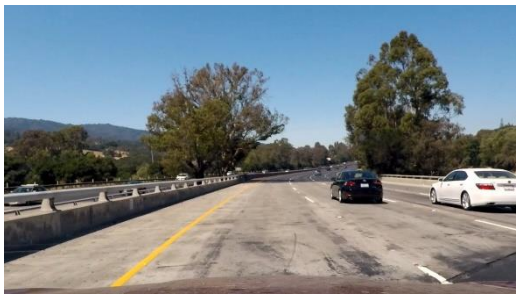
```
src = np.float32(
    [[(img_size[0] / 2) - 55, img_size[1] / 2 + 100],
    [((img_size[0] / 6) - 10), img_size[1]],
    [(img_size[0] * 5 / 6) + 60, img_size[1]],
    [(img_size[0] / 2 + 55), img_size[1] / 2 + 100]])
dst = np.float32(
    [[(img_size[0] / 4), 0],
    [(img_size[0] / 4), img_size[1]],
    [(img_size[0] * 3 / 4), img_size[1]],
    [(img_size[0] * 3 / 4), 0]])
```

This resulted in the following source and destination points:

Source	Destination
585, 460	320, 0
203, 720	320, 720
1127, 720	960, 720
695, 460	960, 0

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.

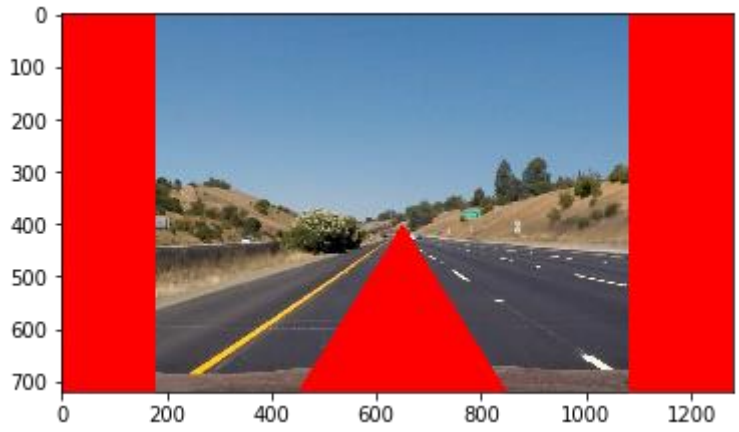
Undistorted image



Warped result

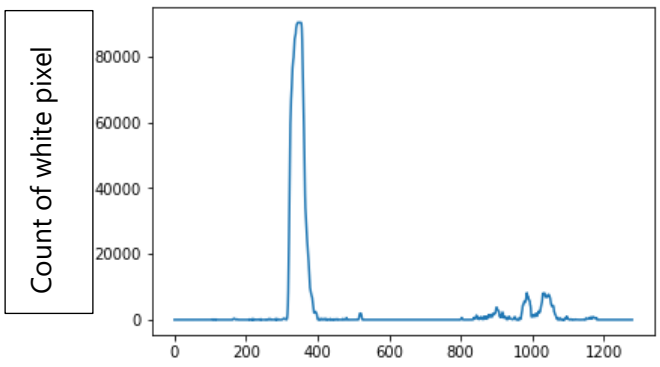
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Before lane line detection, I masked some part of the perspective image like below picture. (lines #346 through #364 in my code in `P4.py`)

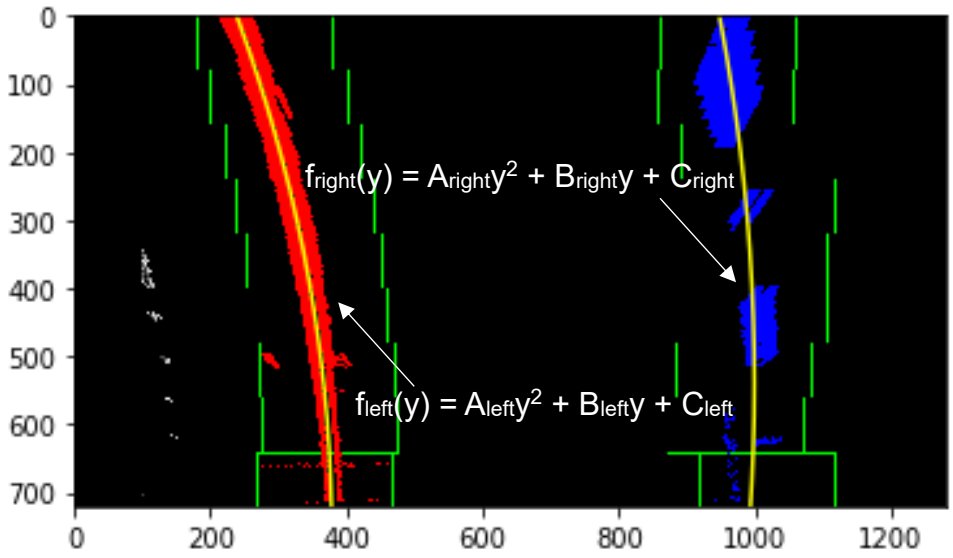


I made histogram of the binary image and then found the peaks of both of the left and right halves of it.

The histogram of the binary image



Then I fit my lane lines with a 2nd order polynomial along with the points of the peaks kinda like this:

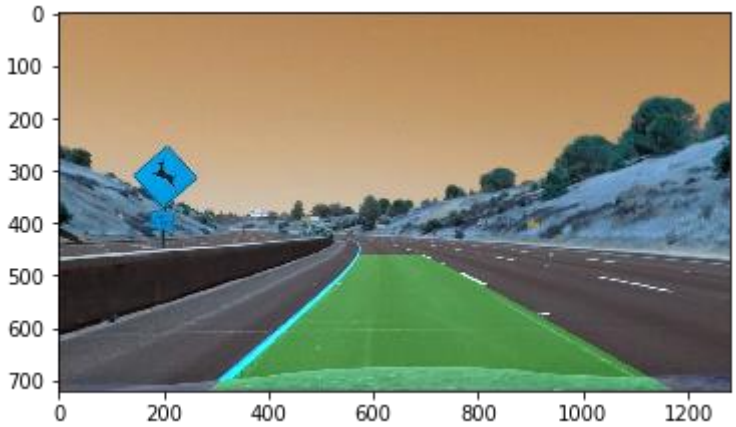


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I did this in lines #451 through #468 in my code in `P4.py`.

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 lines #470 through #485 in my code in `P4.py`. 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!).

Here's a link to my video result

I extracted just 50 secounds from the start of project_video.mp4 because the treatment is a bit heavy. If it's not enough to check my output, I'll submit the whole output video later.

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?

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.

At first my detected lane lines were wider than real ones. I guessed the problem was there was some points over threshold around real lane lines. So I implemented region mask not related to real lane lines. (lines #346 through #364 in my code in `P4.py`) After that, my code could detect lane lines better.

When there is shade of something like trees, my detected lane lines often have error from real lane lines. To match them closely I think I should tune the threshold of binary image. Or add condition such as other threshold.

My treatment of pipeline is a bit heavy. To make it lighter, there is a way to skip slidind window search if the function detects lane lines before.