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.

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

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 IPython notebook P4.ipynb.

Note: All that I am explaining here is part of P4.ipynb file. Every images will also be displayed in that notebook as well.

I have written a method find_points to find the object and image points corresponding to the chessboard pattern of 9x6 dimension given in our case.

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.

And I use the get_coeffs method 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:

Image before un distortion

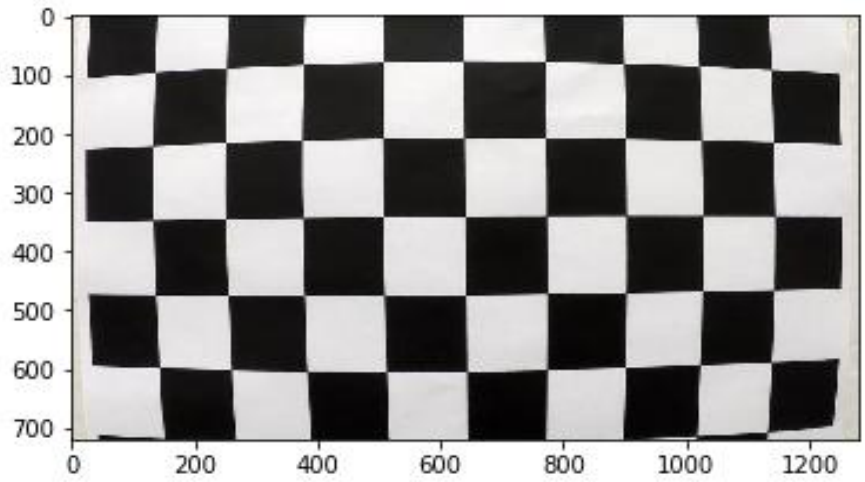
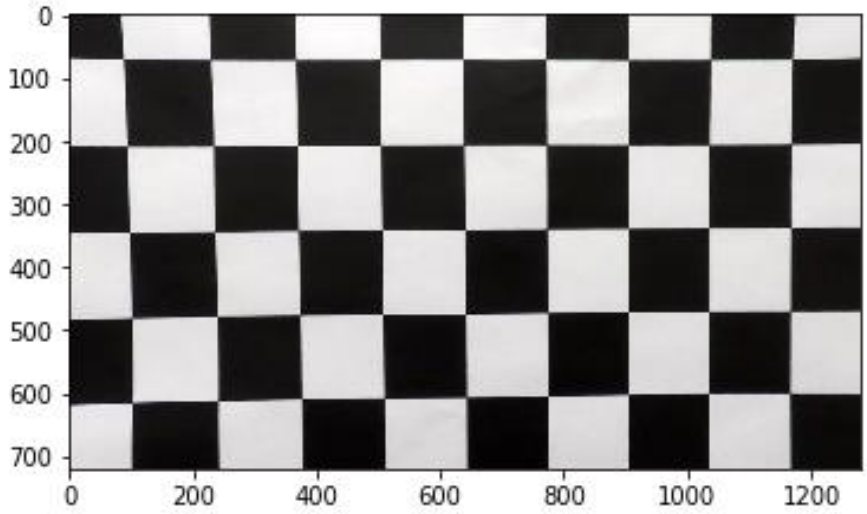


Image after un distortion



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. Now that I have the camera calibration and distortion coefficients with me, I can use cv2.undistort() method to undistort all the images that are taken using this camera at a particular angle.

Below is the example image before and after undistortion. The difference is minute. So you will be able to see the difference if you keenly look the difference will be seen in the hood of the car.

Image Before Un – Distortion:

Image After Un – Distortion:

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

I tried a lot of combination and finally made up mind to use this combo.
I have tried RGB, HSV, HLS, LAB color spaces and x,y gradient and direction threshold gradient transform. And also a combination of many of the above is also tried. Finally I have used X gradient transform with the combination of S and B color channels from HLS and LAB color spaces respectively.

But for re- submission I have changed my selection. I now use X gradient transform, HLS – L channel and LAB – B channel.

I am using methods gradx_thresh, hls_l_select and lab_b_select for selecting the separate channels output. And I am using get_binary_image method to combine all these model's output and return them.

Find below the image displays the image before applying color transforms and gradient threshold (left) and after applying them (on the right).

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 perspective_transform. This will take an image as input and returns it perspective transformed image with an Inv perspective transform matrix value to change it back to the original format.

I chose the hardcode the source and destination points in the following manner:
Source:
bottom_left = [220,720]
bottom_right = [1110, 720]
top_left = [570, 470]
top_right = [722, 470]
Destination:
bottom_left = [320,720]
bottom_right = [920, 720]
top_left = [320, 1]
top_right = [920, 1]

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 almost 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?

I have used the code provided in the lessons and also referred many blogs to identify the best approach.

I used new_fit and use_prev_fit methods to fit the polynomial for the lane-line detected.

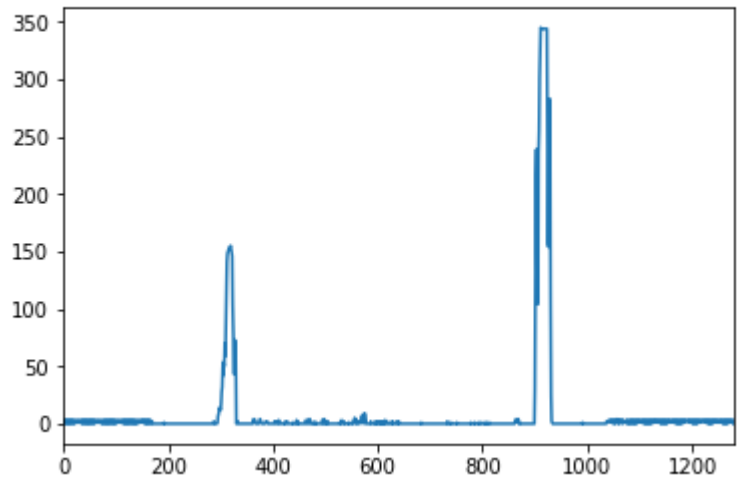
I use Line class to capture the best fit and current fit and compute other classification problems for both left and right lane lines.

The new_fit() function based on identifying peaks in a histogram for lowest portion of the frame in concern. Based on which, I separate the lane line to four parts. I will search for the peak of pixels in second part and the third part for left and right lane lines. Once I find a peak I will move my sliding window above to search through the top of the frame. And using those pixel values I will fit a second order polynomial for left and right lane lines and draw them.

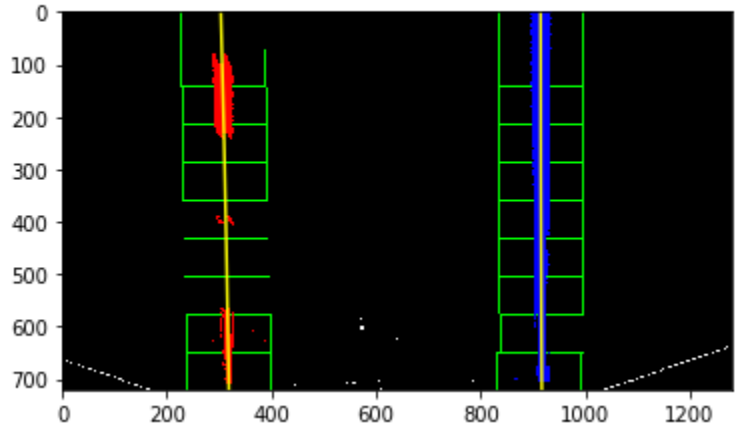
The line class also knows whether or not the lane line was detected in the previous frame, and if it was, it only checks for lane pixels in a tight window around the previous polynomial, ensuring a high confidence detection using use_prev_fit method.

new_fit:

The histogram for the frame :



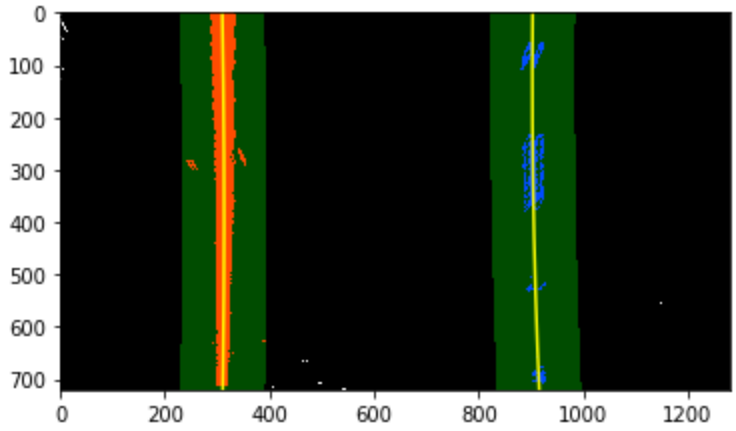
The sliding window details for the frame



The above are the visualizations of the new_fit approach. Find below the visualization for the use_prev_fit method.

use_prev_fit:

A sample image after using the existing fit from previous image



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.

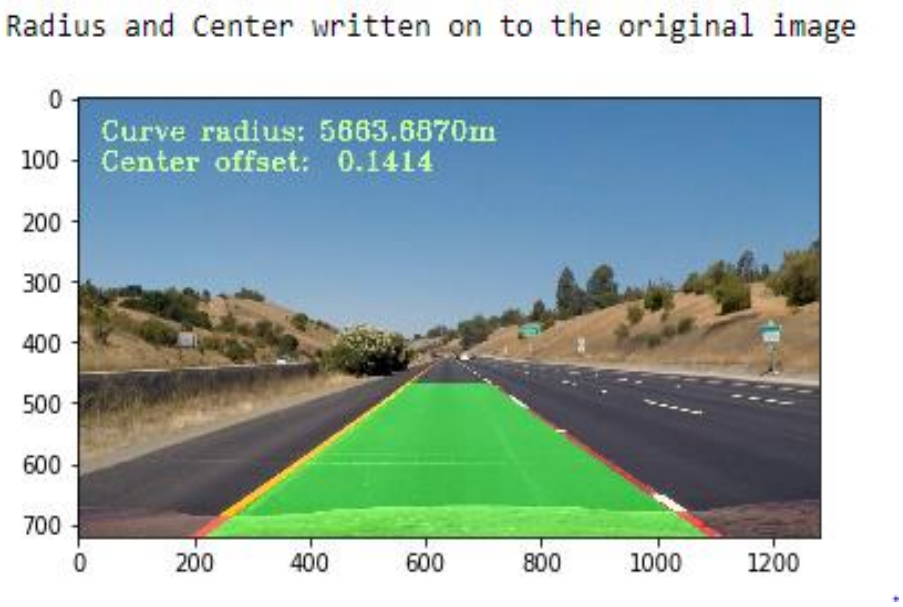
I did this using the method calc_rad_center.

First the radius is calculated using the x and y pixels values found for each of the left and right lane lines and fitting it to a polynomial equation. In this case, this is

not the end. We still need to make the pixel values to the real world meters value. I have used the method provided in the class room section to do this.

For center offset I use the left and right intercepts mean and concerting it back to meters as above.

And I also use method write_rad_center() to write the above identified contents back to the original image after finding the lane lines as written below.



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

I have written the method fill_color2original_image() to draw the lane lines and fill the polygon found with green and then transforming the perspective transformed image back on to the original image with all required tasks completed.

Please find below the image with original image (left) and the perspective transformed and binary image (middle) and the lane line identified, polygon filled with green, transformed back to the original image (right) for example.

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!).

As I have stated already, my pipeline is simple. I have used Line class and created two objects for left and right lane lines.

- Undistort the image, use color transforms and gradient thresholds to transform to binary image and use perspective transform. All these are done in my method basic_transformations.
- Check whether the best fit is available for previous n frames. If available use_prev_fit method to find the lane lines. Else, use new_fit method to find the lane lines.
- Use method fill_color2original_image to draw the lane lines and fill the polygon found with green and then transforming the perspective transformed image back on to the original image.
- Use methods calc_rad_center and write_rad_center to calculate and write radius and center offset back on to the image.

The project video output is available in the same zip folder as this file.

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?

I have taken into consideration the project video more than anything else and focused on it.

As a result, my model performs reasonably well on the project video and doesn't work so well on the challenge videos. Though it works better than my first submission in challenge video.

The reason behind is the color channels and threshold channels I used is not good enough under shadows and different lighting conditions. And also, I have a simple pipeline. My pipeline should also be improved make the model very robust to work on all conditions. I am working on it. And will improve my model.