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.

[Image References]

[image1]: ./output_images/Undistort_ChessBoardSample.jpg "Undistorted sample"

[image2]: ./output_images/Undistort_Test2.jpg "Undistorted test image"

[image3]: ./output_images/Warped_color_Test2.jpg "Warp the color test image"

[image4]: ./output_images/Threshold_binary_Test2.jpg "Binary Threshold test image"

[image5]: ./output_images/Warped _binary_Test2.jpg "Warp the binary test image"

[image6]: ./output_images/line_fitting_Test2.jpg "Polynomial Line fit image"

[image7]: ./output_images/line_fitting_prev_Test2.jpg "Window Line fit with previous detection"

[image8]: ./output_images/Warped_back_Test2.jpg "output image"

[video1]: ./output_images/Warped_back_video_output.mp4 "output video"

[Camera Calibration]

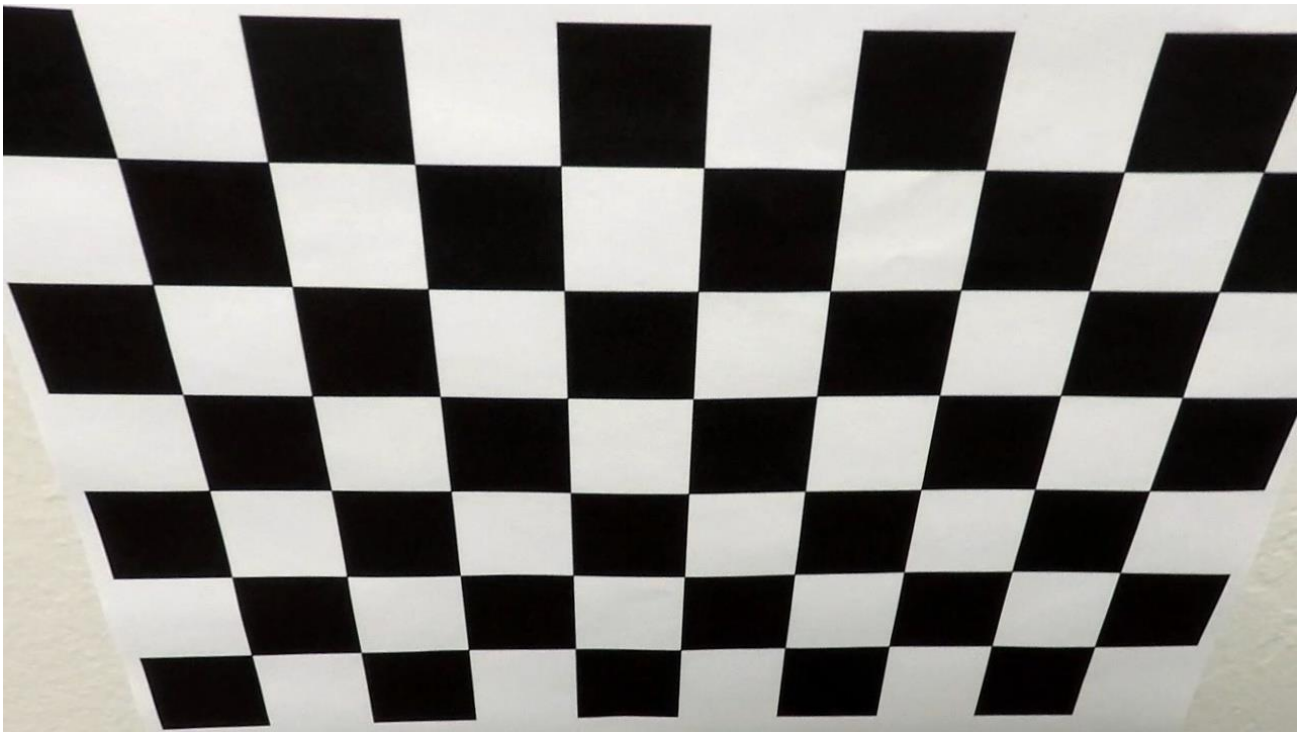
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the “1. Compute the camera calibration and Undistort the images” section of IPython notebook file.

I start by preparing "objpoints", 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 9x6 corners in the image plane with each successful chessboard detection. In order to refine the corners, I also used ‘cv2.cornerSubPix’ function, and it fine-tuned the location of corners.

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 chessboard image using the `cv2.undistort()` function and obtained this result:

[image1]

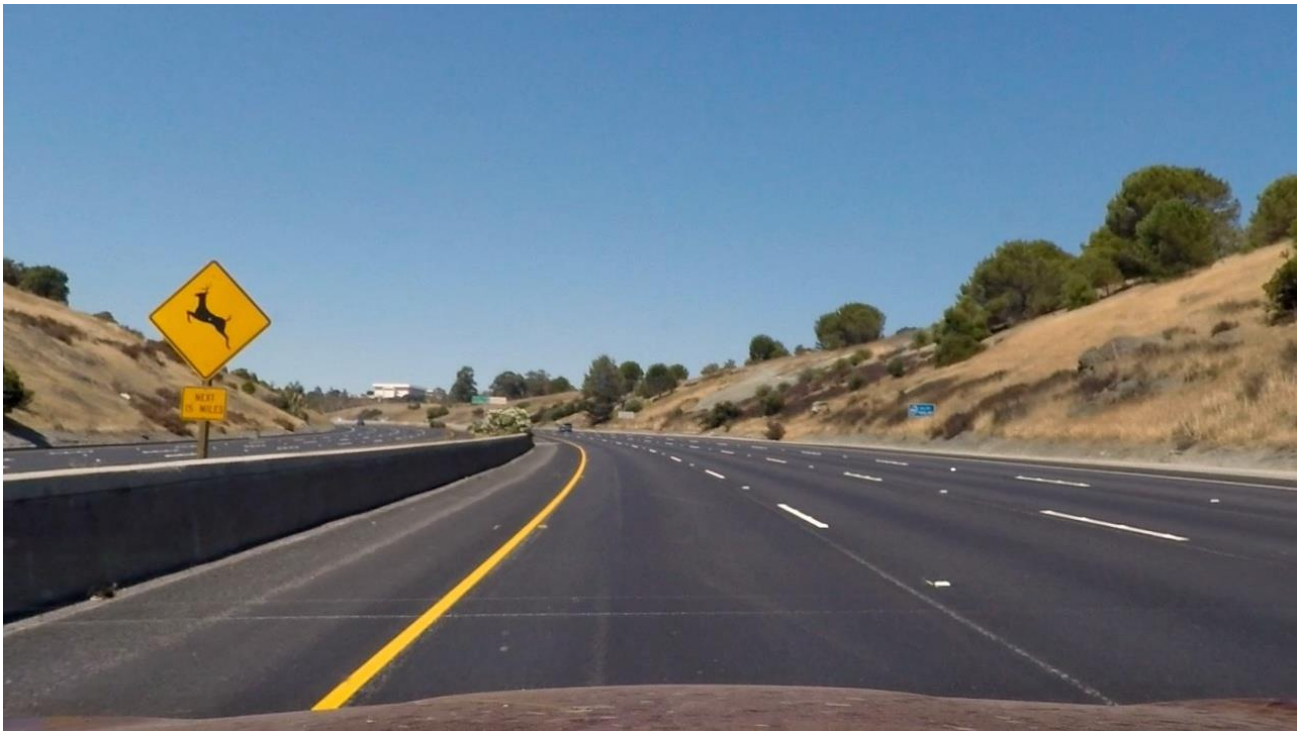


[Pipeline(IMAGE)]

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

I applied the distortion correction algorithem to ‘./test_images/test2.jpg’ image as below

[image2]



2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a threshold binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image. The code for this step is contained in the “3. Create Threshold Binary Image for Lane Detection” section of IPython notebook file. I tested abs Sobel (Both x and y direction), magnitude Sobel, directional Sobel and S channel of color thresholding to the test image. All the information was useful. For the right lane detection, abs Sobel and magnitude Sobel contributed a lot to find it. Otherwise, for the left lane detection, directional Sobel and S channel information was useful. However, in my algorithm, I didn’t use the directional Sobel because it includes too much noise information. I combined rest of them (absx, absy, mag and s-channel) and acquired the binary image as below.

[image4]

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 this step is contained in the "2. Perspective Transform for Original Image" and "4. Perspective Transform for Binary Image" section of IPython notebook file. I made "perspective transform" function which takes as inputs an image and returns as well as source and destination points(src, dst), perspective and inverse perspective matrix (M, invM) and warped image. I used a couple of cv2 perspective transform functions. I chose the hardcode the source and destination points in the following manner:

src=np.float32([[width/2-40,height/2+90],[300,height],[1150,height],[width/2+40,height/2+90]])

dst=np.float32([[width/2-350,0],[width/2-350,height],[width/2+350,height],[width/2+350,0]])

Because the src ,dst point should be generalized for any images, I used the width, height information of the images. However, (I will mention about this more in later) the choosing src, dst point is not easy task. It was theoretically impossible to make suitable src, dst points for all images, videos with hard code manner. I should think about better way to do that. 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.

[image3] [image5]

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

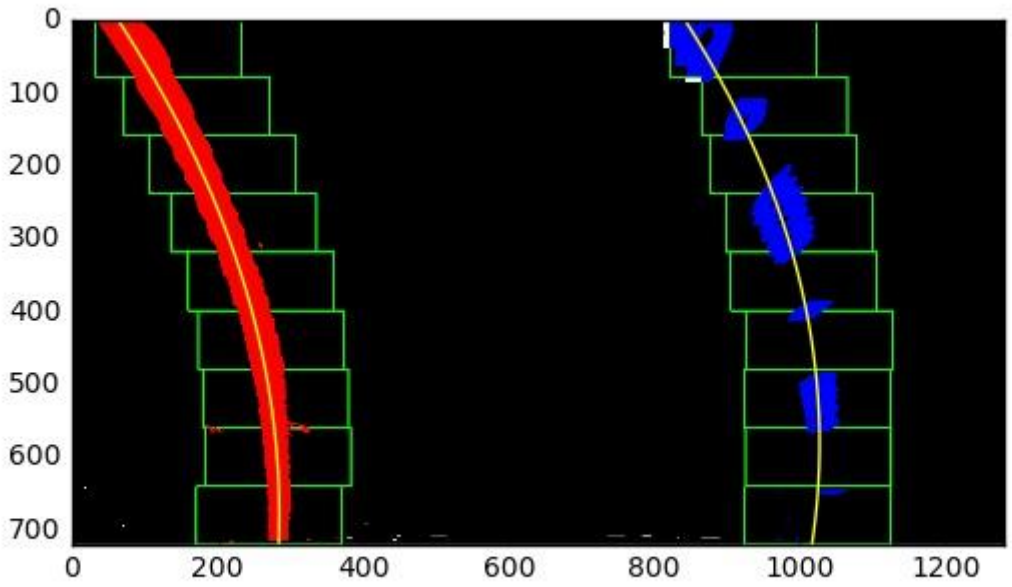
The code for this step is contained in the "6. Implement lane fitting function". In this section, I mostly followed Udacity's sample code and it worked well!! I made histogram and sliding windows to find the lane pixel and fitted them to 2nd order polynomial. Once I found the lane, I could find the lane with

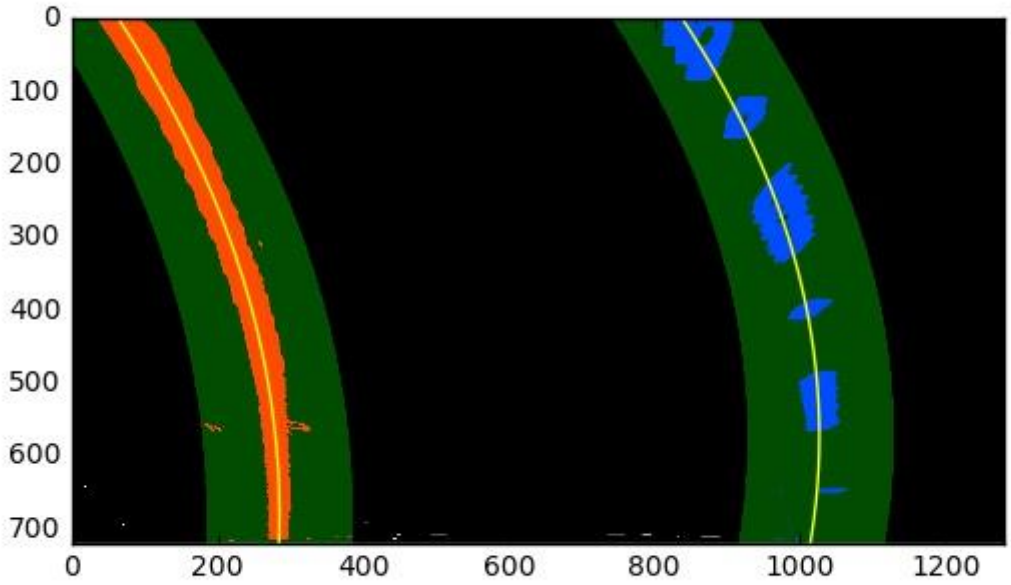
previous image frame information, which is organized at my “line_fitting_prev” function. One thing which I changed is that I added the conditional statement to the function as below:

```
if len(lefty) < 50 or len(righty) < 50:
        lanefind = False
    else:
        lanefind = True
```

This is for the video output. If we can’t find the lane within fixed margin (I assumed that it means failure to find the lane if length of y pixel is smaller than 50), we go back to initial “line_fitting” function and search it again (lanefind = False). Below images is the lane fitting examples

[image6] [image7]





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.

The code for this step is contained in the "7. Calculate Curvature & Vehicle Offset". For the curvature calculation, I also mostly followed Udacity's sample code. I convert pixel spaces to meter space based on the standard which Udacity suggests. Then, after fitting the polynomial in meter space, I calculated the curvature of left lane and right lane. Their curvature was almost similar and close to real value.

Left Lane : 342.85941609m, Right Lane : 301.051728318m

When it comes to vehicle offset, I assumed that the camera is located at the center of the image. Then, I calculated the left/right lane intercept with x axis using polynomial fitting. I calculated the difference between half of the image width (center of the lane) and half of the addition of left lane intercept and right lane intercept (vehicle position). The offset value of the test image was:

-0.0531616023977 m

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

The code for this step is contained in the "8. Lane Drawing Function", "10. Pipeline for Advanced Lane Line Finding", "11. Image Processing". Here is my image

[image8]

[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!).

The code for this step is contained in the “8. Lane Drawing Function”, “9. Line() Class for Smoothing the lane”, “10. Pipeline for Advanced Lane Line Finding”, “11. Video Processing”. My video is included in my GitHub project folder. In this video processing, I used ‘Line’ Class to prevent changing the lane abruptly. Even if my algorithm fails to find the lane or find the wrong lane, this can be prevented with the aid of smoothing algorithm in Line Class. I saved 9 lane information of previous frames and used the average value for the next frame. Because the location of lane line cannot be changed abruptly frame by frame, this is really robust method!!! If you see my video frame about 00:40~00:41sec, it looks like finding the wrong lane due to the shadow of the tree. However, the lane moves back to the original point smoothly (not changing abruptly) even with little sway. Without the smoothing algorithm, the lane will suddenly move the wrong position and back to the original position.

[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?

1. It is quite difficult and time-consumed to find the optimal source/destination point for perspective transform. It is very important task to identify the lane correctly with 'Bird's Eye View'. However, my source/destination point could not satisfy the all the frame of 'challenge_video', 'harder_challenge_video'. I should think about how I can find the source/destination point adaptively to every image/video.

2. When it comes to 'challenge_video', 'harder_challenge_video', my algorithm does not work well. It sways too much or even could not find the lane at all. With my frame by frame analysis, the first reason was that non-optimal source/destination point for perspective transform as I mentioned above. Secondly, my threshold binary image did not recognize the lane line correctly because of several noise in video frame (e.g. a lot of trees, shadows, light reflection, road pavement and so on) I am trying to make robust threshold binary image algorithm with additional algorithms such as masking method, R-channel information in RGB world, H-channel information in HLS world.