Advanced Lane Finding – Udacity self driving NanoDegree

Goals/Steps followed in this project are as follows

- 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.

Camera calibration measures the camera distortion inherent in the lenses.

I have used the OpenCV functions

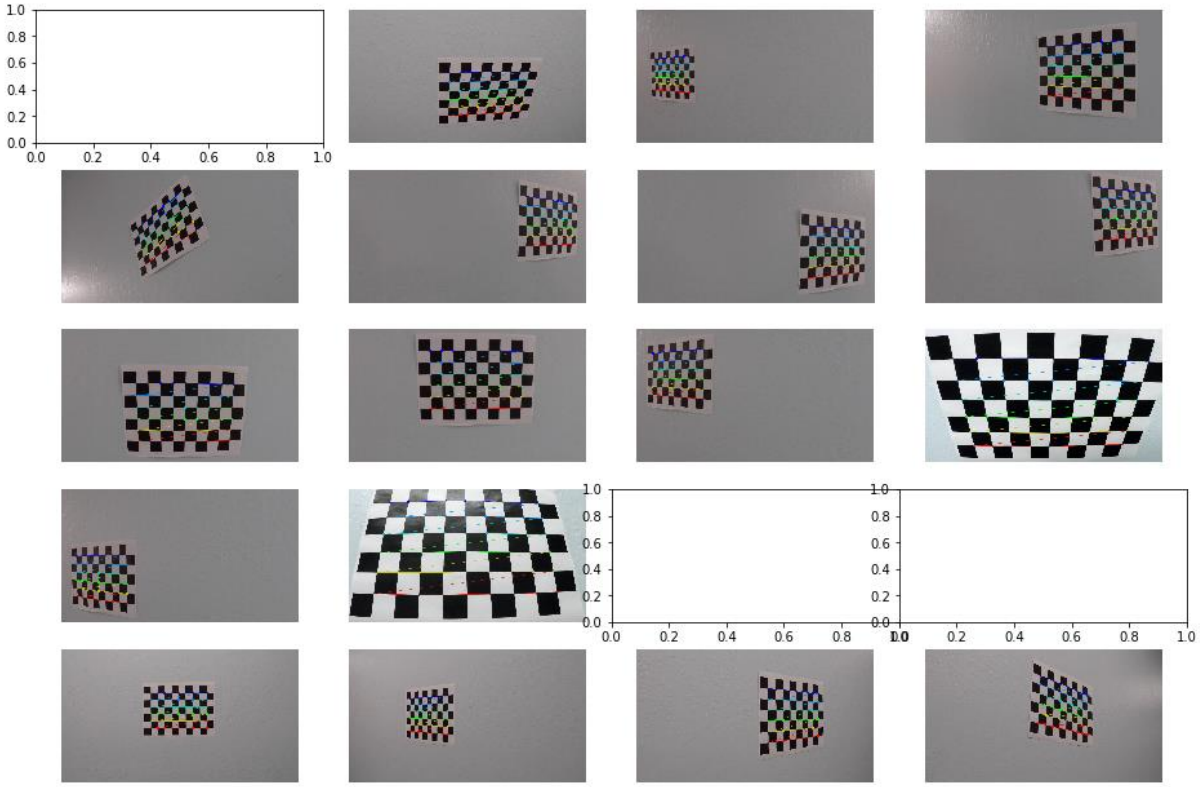
findChessboardCorners and drawChessboardCorners to identify the corners in the images and highlight the corners.

https://www.programcreek.com/python/example/89335/cv2.cornerSubPix

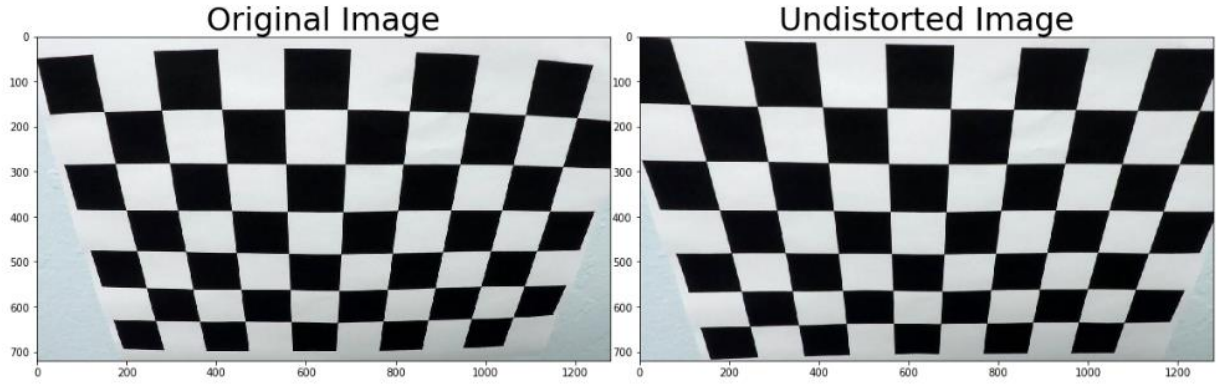
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

I have used the above material to learn more about calibration and increasing the accuracy of images. CaliberateCamera function measures distortion parameters.

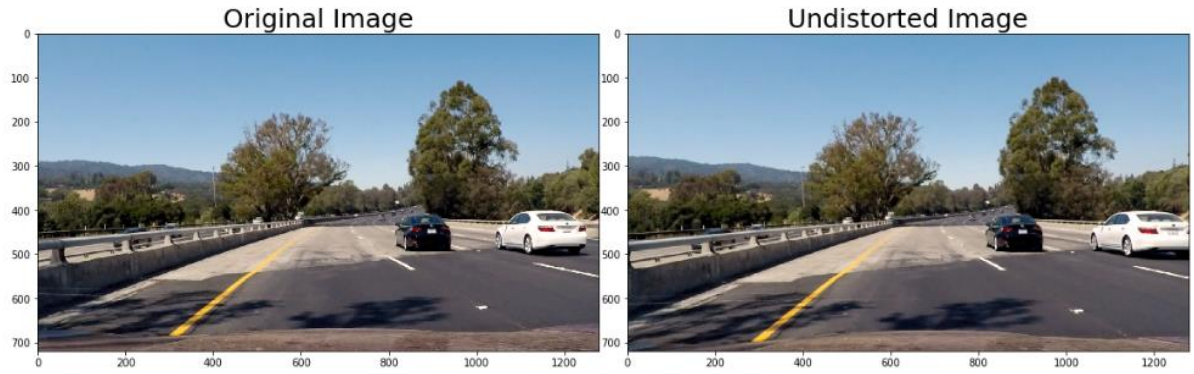
Chess board corners drawn were listed as below:



Undistorted image of Calibration2 is listed below:

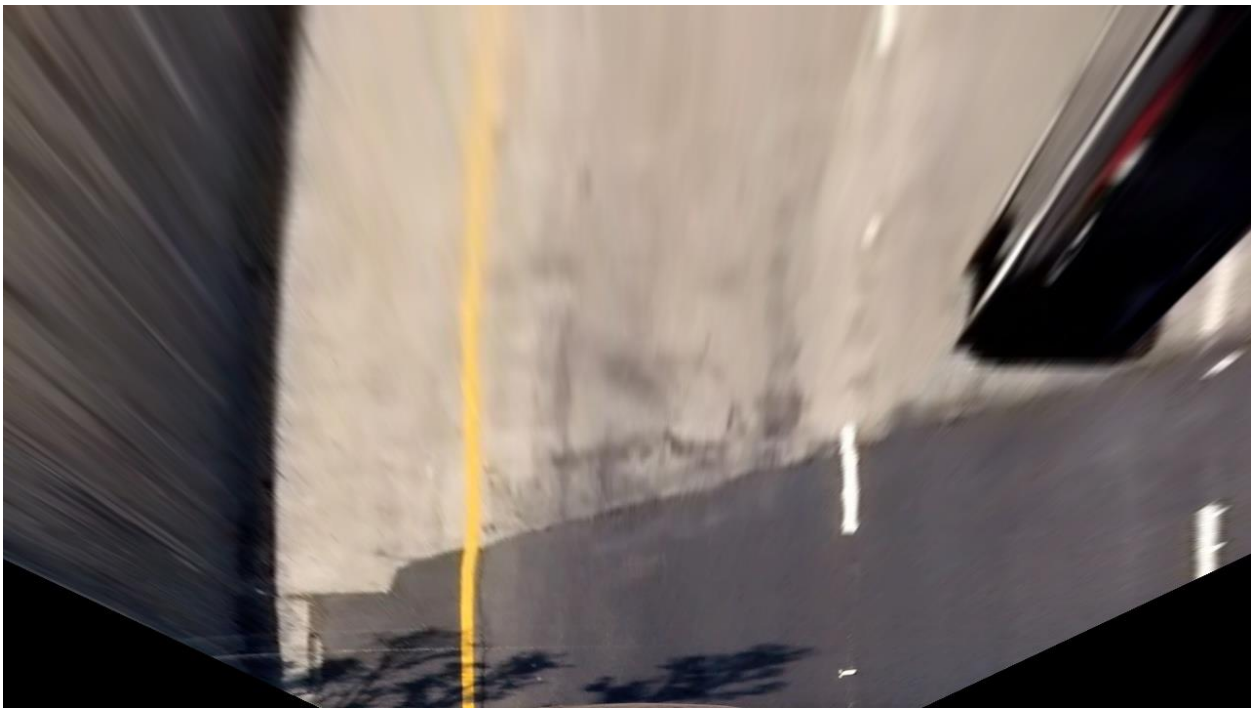


The undistorted image applied to a test image is listed below:



If it is not evident, please notice the hood of the white car(Toyota?).

Then I began applying the perspective transform to view the lane like a bird to identify the curvature of the lanes.



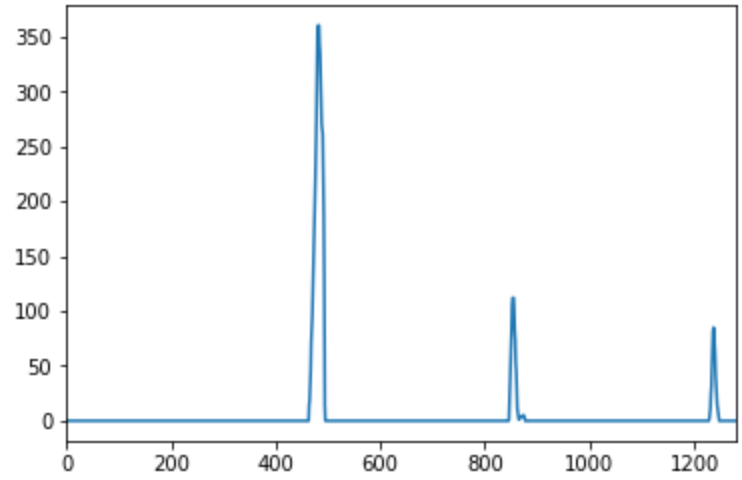
Extracting the lane information form the image. Here I had a very tough time in experimenting different techniques like sobel transforms both magnitude and direction and applying different thresholds.

When this is iterated through the test images, I got the resultant lanes identified as follows

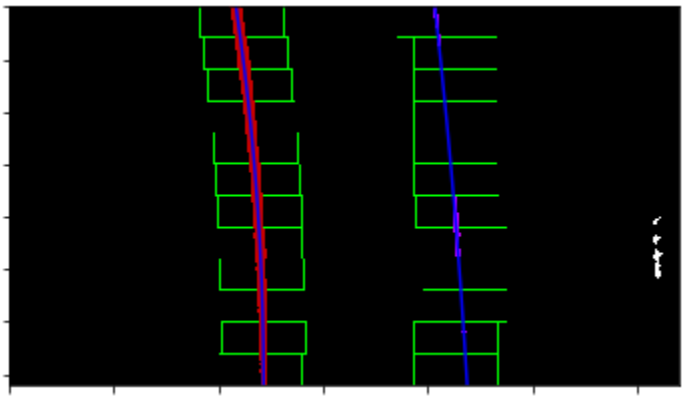
Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The lane line pixels were identified by image analysis techniques in sliding_window_polyfit function

1. Taking a histogram across the bottom half of image



2. Detect the left index and right index
3. Identify the maximum triggered edge of the histogram
4. I have sliced the area in to series of histograms as shown in the below picture



5. For each of the rectangles (or windows), the pixel positions where non zero pixels were found were identified
6. These pixels were qualified to be valid based on minimum and maximum calibration. (To avoid the noise factor)
7. Then those pixel positions are stored in to individual variables (To be used to draw them in the blank images later)
8. The polynomials were also stored in variables (Ax^2 + Bx + C)

Now it's time to test what has been exported out of this function

Now we got a good estimate of current lane line locations. We can thus implement the trimmed down or optimized version

Reproduce what has been extracted out of the image in to the new blank image. Later we can superimpose the extraction in to the original image and fill the region of interest

Calculation of radius of Curvature

1. In real world the values are in X and Z direction
2. In image world, the same values are in X and Y direction
3. Pixel coordinates need to be converted in to real world values
4. A simple conversion of pixel coordinates we gathered earlier will not work because in real world coordinates it differ
5. The radius of curvature of the curve at a particular point is defined as the radius of the approximating circle. This radius changes as we move along the curve.
6. The radius of curvature at any point x of the function x=f(y) is given as follows: Based on the equation A and B are the fit parameters

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

In the case of the second order polynomial above, the first and second derivatives are:

$$f'(y) = \frac{dx}{dy} = 2Ay + B$$

$$f''(y) = \frac{d^2x}{dy^2} = 2A$$

So, our equation for radius of curvature becomes:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

7. Calculating the center of the lane is by averaging the left and right lane edge fits

Drawing the lines back in to image

1. Here we create an image to draw the lines on
2. Reconstruct the accumulated polynomial fits
3. We only need to account for x coordinates here because warping accounts for z direction in the real three dimensional world but Y direction in the image plane. Thus we are generating x and y values for plotting
4. Then mark the left,right and entire area to fill by pts_left, pts_right and pts
5. Use the cv2.fillPoly to draw the lane in to warped binary blank image

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

Shown below is the example image plotted back in to the road

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!)

This project has been added to github and the video is under the following link

https://github.com/pradhapmoorthi/Advanced-Lane-Finding.git

Tried running the code under Challenge Video and Hard Challenge video. The performance in Challenge video is reasonable but the hard challenge video is not because of appearance of the lane positions keep dynamically changing 😊

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?

Challenges:

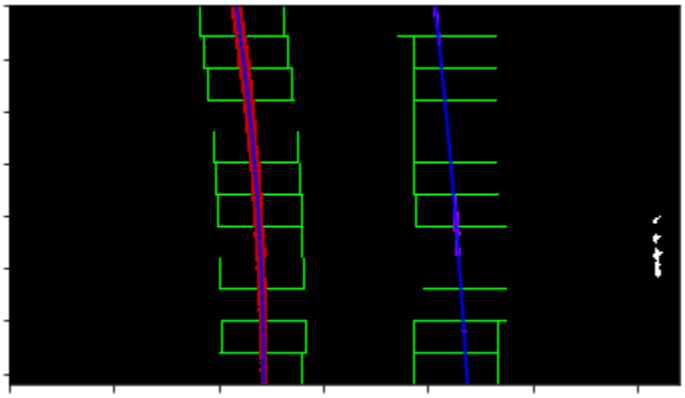
Alternative color spaces, Gradients and Thresholds

First I tried with RGB, then HLS and compared that with Lab color space

Numerous color space combinations made me so tired in the initial couple of days so finally I tried and found out a workable combination but not sure whether it is a perfect full combination or not

Camera calibration: Images 1,4 and 5 put me in a hard time while doing calibration because these images are not feeding in to the OpenCV functions and I need to search online for more helpful code apart from the lessons. There I learned about the termination criteria for the code and it worked

Sliding window rectangles: With number of windows as 12, and sometimes I am not getting the complete lines in the rectangles(Refer the sides of rectangles) and has been debugging for some time and finally convinced not to pull this further as it is not affecting the lane line drawing



Rooms for improvement:

Why the approach is failing on harder challenge video (Sharp turns) and our focus area is fixed. Need more calibrations and dynamic algorithms to fix this aspect which is beyond scope of advanced lane finding

Combining different functions, sometimes I use functions and some times in the common path, it is very difficult to maintain the functions. More granularity and reuse of functions may be needed as part of further refinement.

Simple averaging function: The simple averaging function is time effective but not performing well. In the Challenge video at 4^{th} and 5^{th} second has a wobbling effect and noise generated. With a advanced ring buffer of more time averaging, this could have been avoided(hopefully) if I have invested more time in searching for inbuilt libraries for lane line averaging.