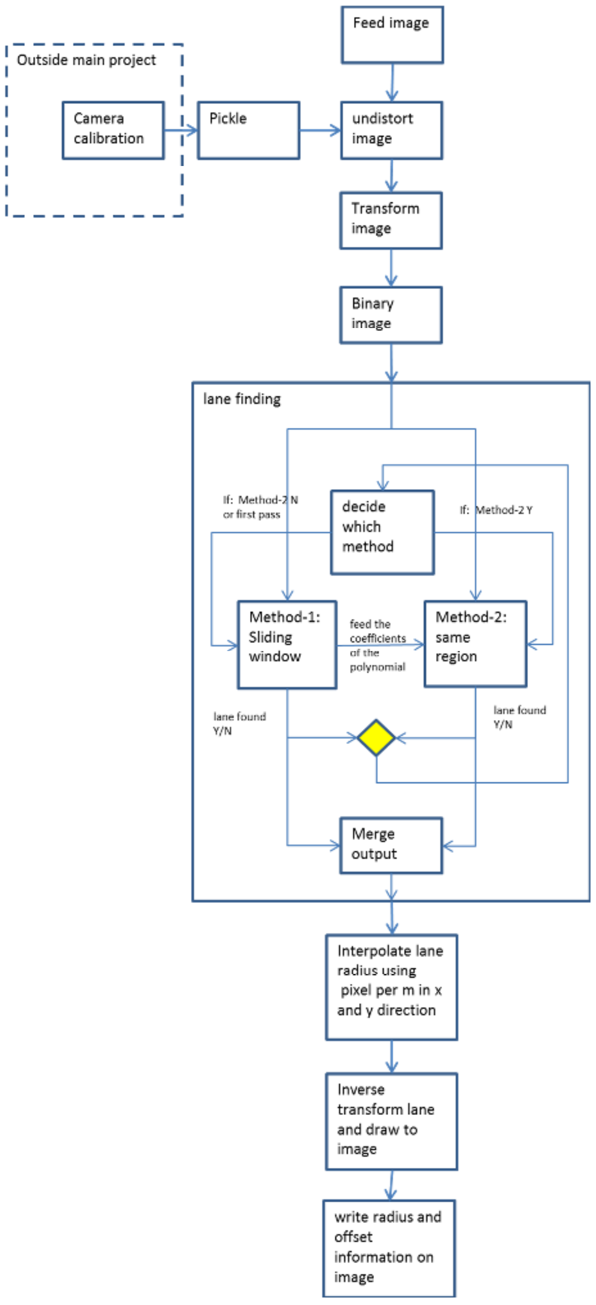
Write up: Advanced Lane finding

Author: Sushrutha Krishnamurthy

Preparation: As preparation all the code segments which were used in the classes and in the quizzes were copied locally to be used as templates in the project. The program itself suggested this with the phrase “feel free to use the codes used in the quizzes” therefore didn’t go to the trouble of inventing the wheel again. After every successful stage I copied the code to the next cell. This eased up debugging effort.

Please only refer to the notebook “Advanced Lane Finding Final version.ipynb” for evaluation the remaining were discarded after experimentation.

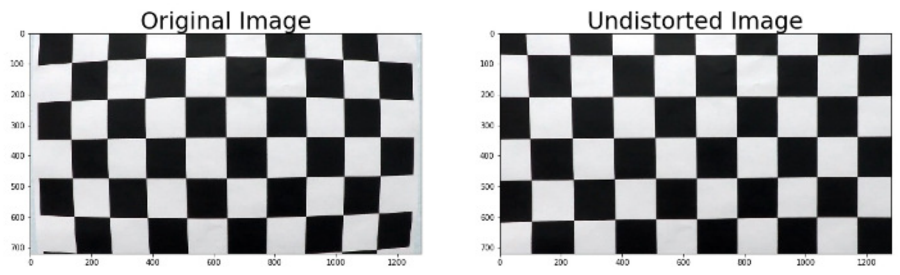
The overview of the pipeline is as follows:



Step- 1 Calibrate Camera:

The camera calibration has to be done only once to get the distortion co-efficient and the camera matrix. Therefore it will be inefficient to run this in every cycle and more over it needs to be feed with the camera calibration images rather than the image that has to be processed. For this reason it is run separately as a sub project under “chessboard_corners.ipynb” . The calculated distortion and the camera-matrix from all the calibration images are stored using the pickle.

The before and after distortion images of all the calibration images are stored under: output_images/Before_After_undistort_calibration_images

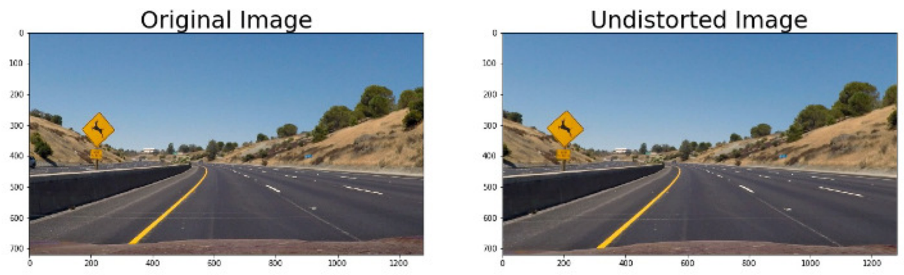


Sample image of calibration1

Step- 2 Read image and data from pickle:

The image is read as Image_read and the distortion-coefficient and Camera matrix are read using the pickle in this step.

The read image is undistorted. The before and after distortion images of all the test images are stored under: output_images/Before_After_test_images



Sample image of test2

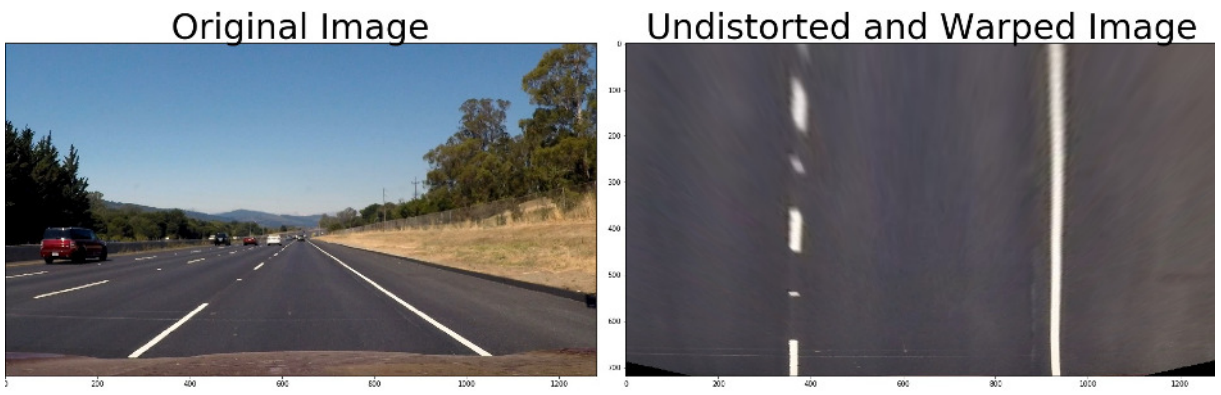
Step- 3 Transform Image:

The read image is undistorted and transformed into birds eye view in this step. I used the image straight_lines2 as reference to define source and destination. Taking a straight line as reference has the advantage that the transformed image has to be in the form of a rectangle. After several trail and errors the source and destination points seemed to match.

I then tried out to transform all test images using this function and the results looked plausible

Also return the value of M_inv by interchanging source and destination. This is required at a later stage to reverse transform the lane polynomial onto the original image.

The read image is undistorted and transformed. The before and after distortion images of the test images are stored under: output_images/Before_After_undistort_transform



Sample image of straight_lines2

Step- 4 Create a binary image:

The image is converted into a binary image using the following methods:

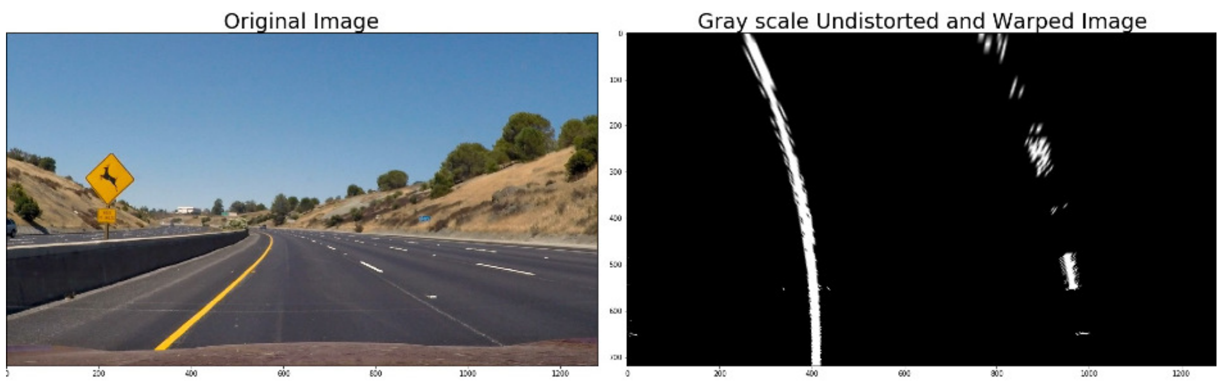
1- Gray scale(refer to line 17)
2- HLS channels fed to sobel derivative in X (refer to line 22)

Individual thresholds can be applied for each channel. This was tweaked with on a trial and error basis to arrive at the best solution.

All the individual binary images derived from the different methods are combined to form a single binary image (refer to line 47).

Bug fix: the image is converted into just one channel (refer to line 56) because the starting point detected latter on in the sliding window method would return values out of range.

All test images were tested and found to be satisfactory except for image test5. Differentiation of the shadow area was difficult to master even after several trials with the thresholds. The before and after binary conversion of the calibration images are stored under:
output_images/Before_After_undistort_transform binary image.



Sample image of test2

Step- 5 Lane finding:

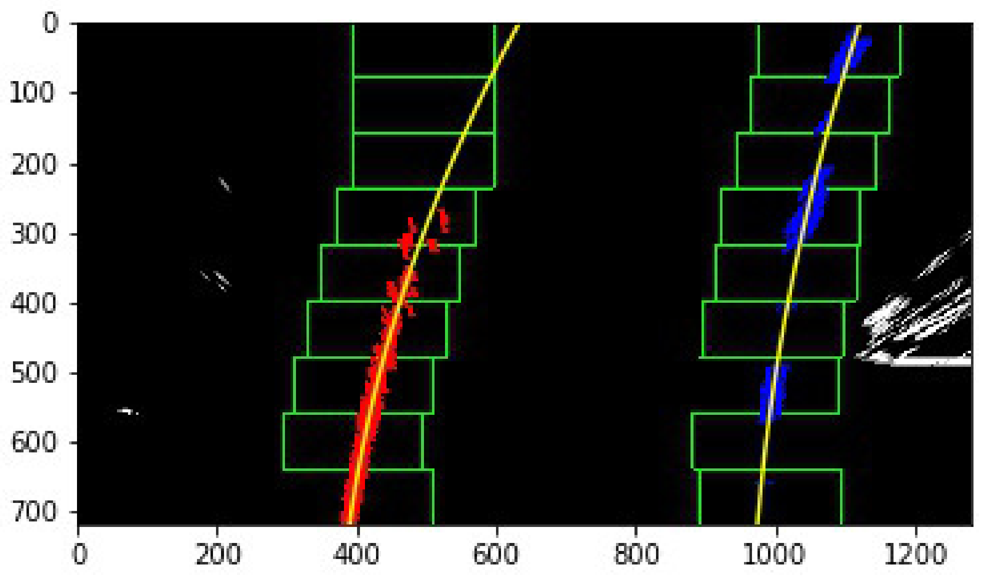
The main goals in this step are to identify the lanes on the transformed binary image and to draw a 2nd degree polynomial onto the identified lane.

I first used the sliding window method (lines 95 to lines 209) to identify and draw the polynomial. The code processes the image as follows:

1- Calculate the right-lane and left-lane starting points by plotting a histogram to the bottom half of the image (refer to line 98).
2- define empty arrays to append as left lane (left_lane_inds[]) and right lane (right_lane_inds[])

3- Generate equally spaced rectangles from the starting point (refer to line 132).
4- Re-center the rectangle if the found non-zeros exceeds threshold (refer to line 154).
5- Append the respective lane index when the rectangle re centers itself
6- Get the A, B and C coefficients for the polynomial for both lanes using the lane index
7- Return the lane index, starting points and the plotted image(this is for testing actually not required for the main code)

After this I tested it on all test images. I also deleted the code which draws rectangles as it is only meant for visualization but unnecessary for the main code and will only slow down the processing.



Sample image of test6

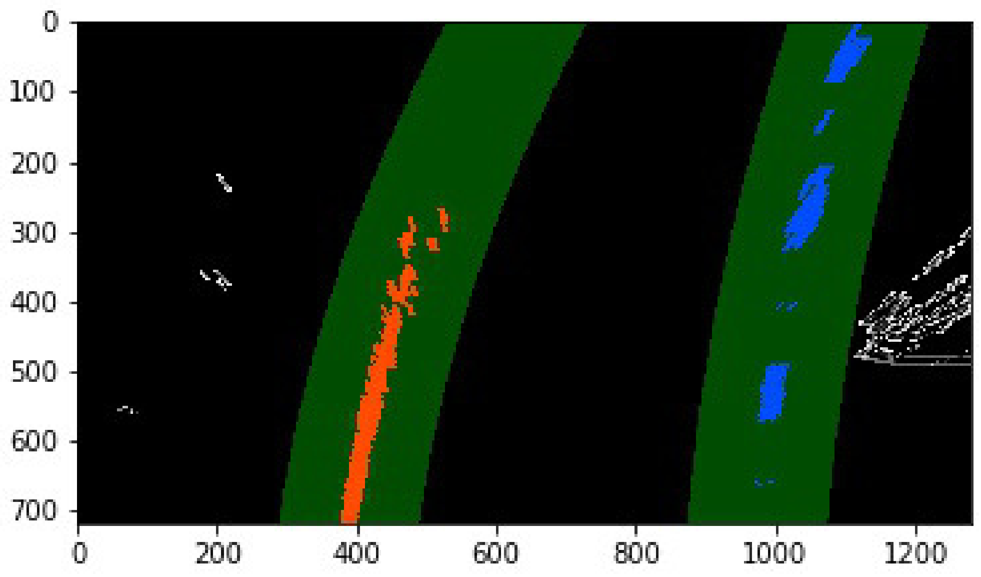
The sliding window method is inefficient and cumbersome especially when processing a video therefore ideally the identified region method has to be used. This method requires the A,B and C coefficients of the right and left-lane polynomials to start off. I printed out the coefficients from the sliding window method to test.

For the actual code the sliding window method has to run for the very first frame of the video and return the co-efficients of the polynomial to be used in this method (refer to line220).

In this method the algorithm looks for nonzero in the binary image within a defined margin of the polynomial and appends the left and right-lane index and fits the new polynomial. I tried this code on all test images and got plausible results.

The code lines that are meant for visualization are not actually required for the main code and are therefore commented out.

Sample image of test2



Sample image of test6

Step-6 Extrapolate real lane curvature radius:

The main goal of this step is to scale the polynomial on the transformed image and extrapolate it onto the real road. Once the coefficients of the extrapolated polynomials are derived the radius of curvature of the road can be derived.

Used the values ym_per_pix = 25/720 for y axis and xm_per_pix = 3.7/720 (standard lane width) as used in the exercises.

1- Plot the calculated left-lane and right-lane polynomials
2- Scale the polynomials in the x and y direction using the magnifications mentioned above
3- Using these indices derive the coefficients for the right lane (right_fit_cr) and the left lane(left_fit_cr)
4- Calculate the radius of curvature for left and right lane using the formula
   $R= [(1+(2AY+B)^2)^3/^2]/|2A|$

To calculate the vehicle offset, first the image mid-point is calculated and lane mid-point is calculated. The difference between the two mid points is the scaled up in the x direction to derive the real vehicle offset implemented in lines 303 to 346

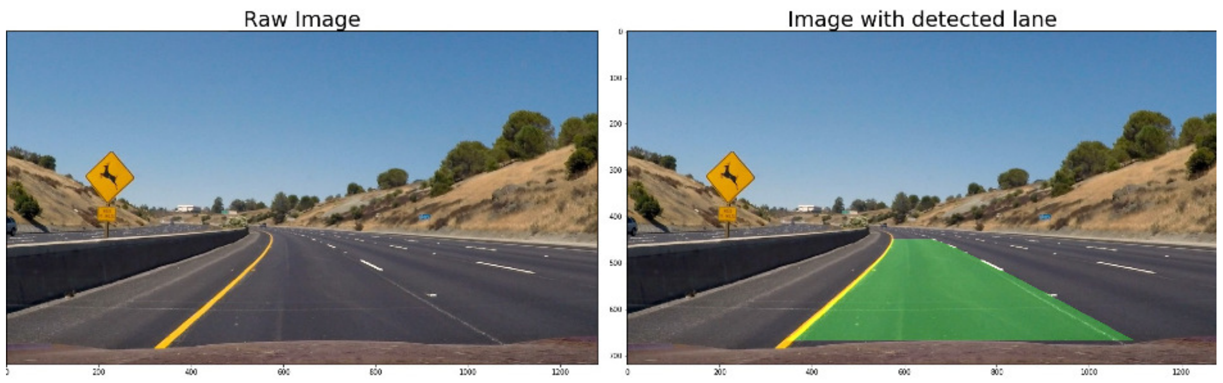
The radius of curvature and offset were calculated for all test images and found to be plausible

Step-7 Inverse transform lane and draw to image:

The plotted left-lane and right-lanes are reverse transformed (refer to line 357) to plot the left and right lanes on the actual image. The space between the plotted polynomials is filled using fillpoly. The actual reverse transformation and plotting is in line 375. The combination of the reverse transformed polynomial onto the original image is in line 378.

I tested this on all test images and got plausible results. The test-5 image though showed a problem that the lane veers off to the left even though the lane turns to the right. I traced this back to the binary image not filtering out the light and shade well enough resulting in the polynomial not being perfect. After tweaking with the threshold values the behavior improved but was still not perfect.

The before and after rewarp images of the test images are stored under: output_images/Re-transformation of lane onto original image



Step-8 Write radius and vehicle offset information to image:

The calculated radius and offsets are written on to the image using the cv2.puttext function.

Step-9 Implement code to video

The video implementation is through VideoFileClip as in the first project.