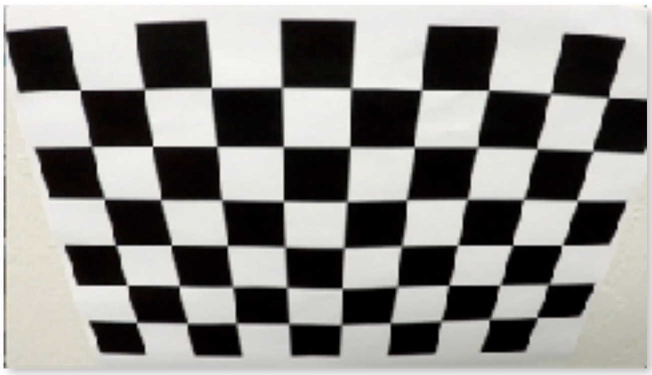
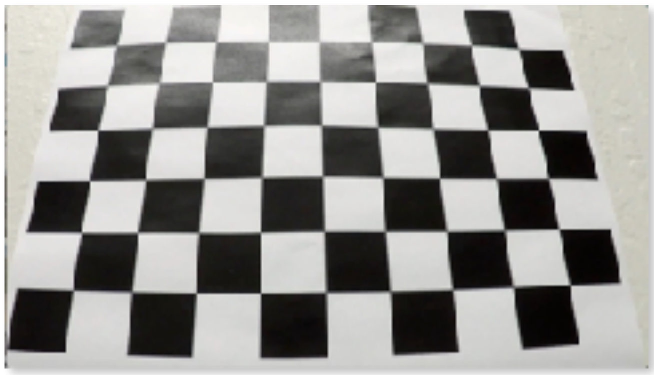
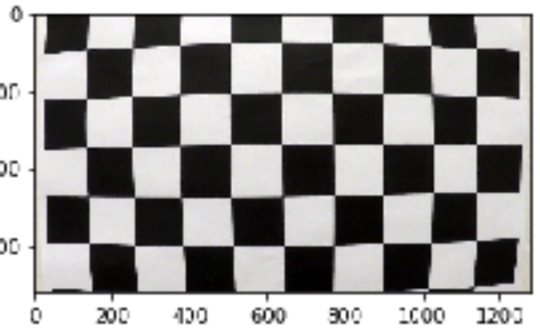
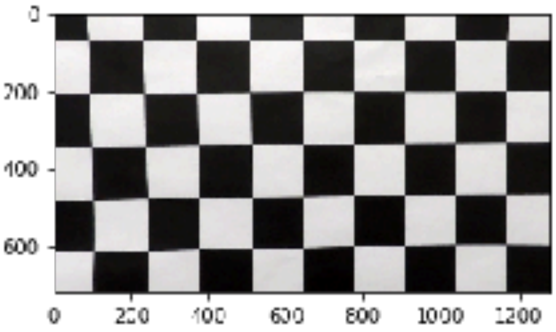
Camera Calibration:-

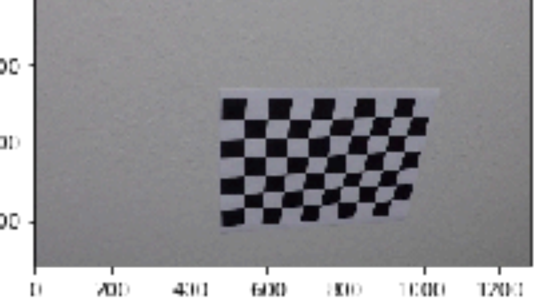
To calibrate the camera we need around 20 images of a chessboard patters. Few are shows below.

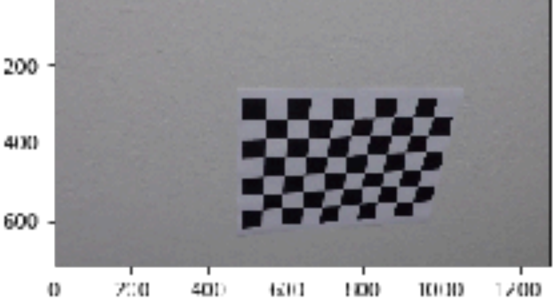



I used OpenCV function findChessboardCorners() for each of those 20 images. Then I used those found corners with OpenCV function calibrateCamera() to find the camera camera matrix (mtx) and distortion coefficients (dist) as taught in the Udacity tutorial and I implemented in code in jupyter notebook. Some examples of distorted (left) and corresponding un-distorted(right) images are as follows:



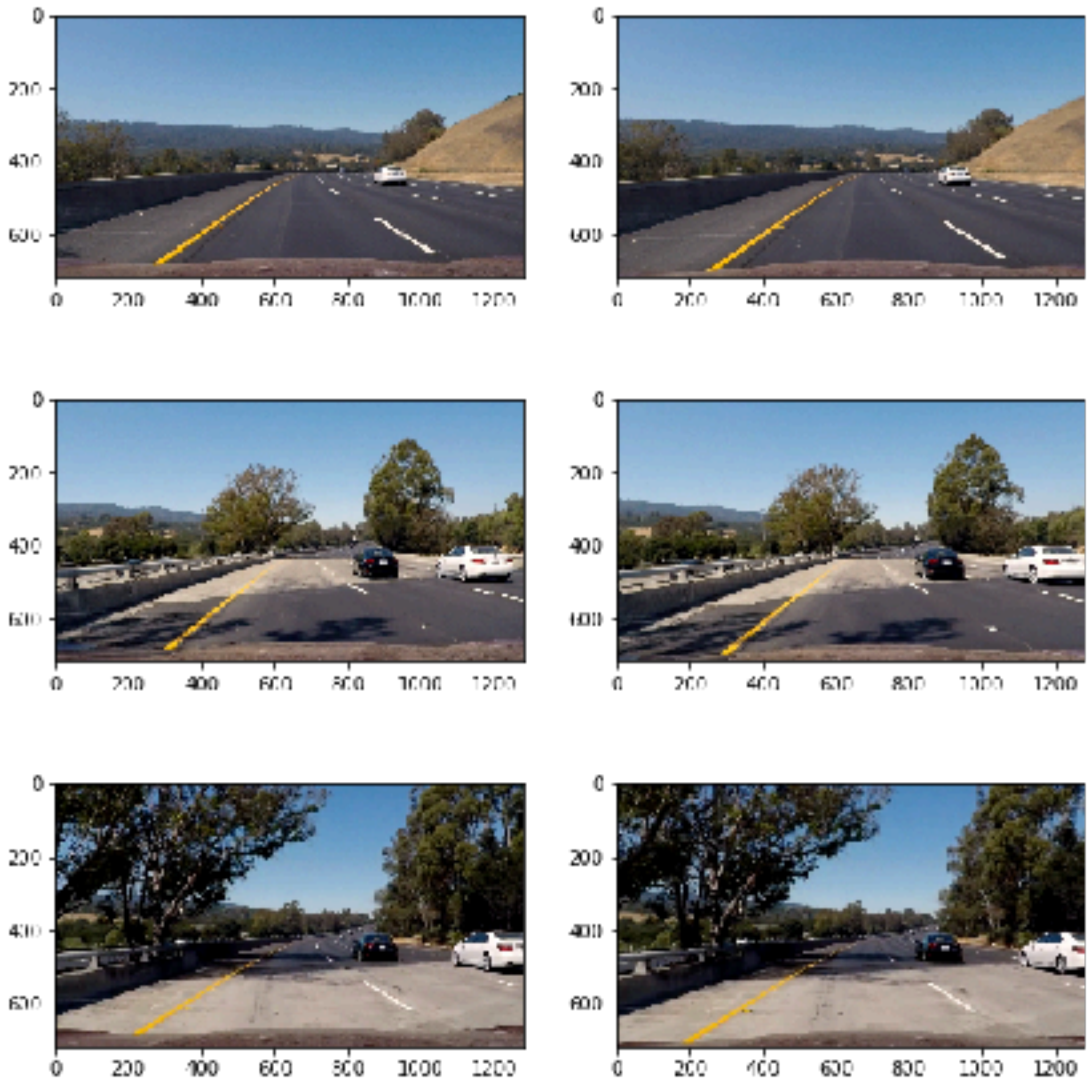






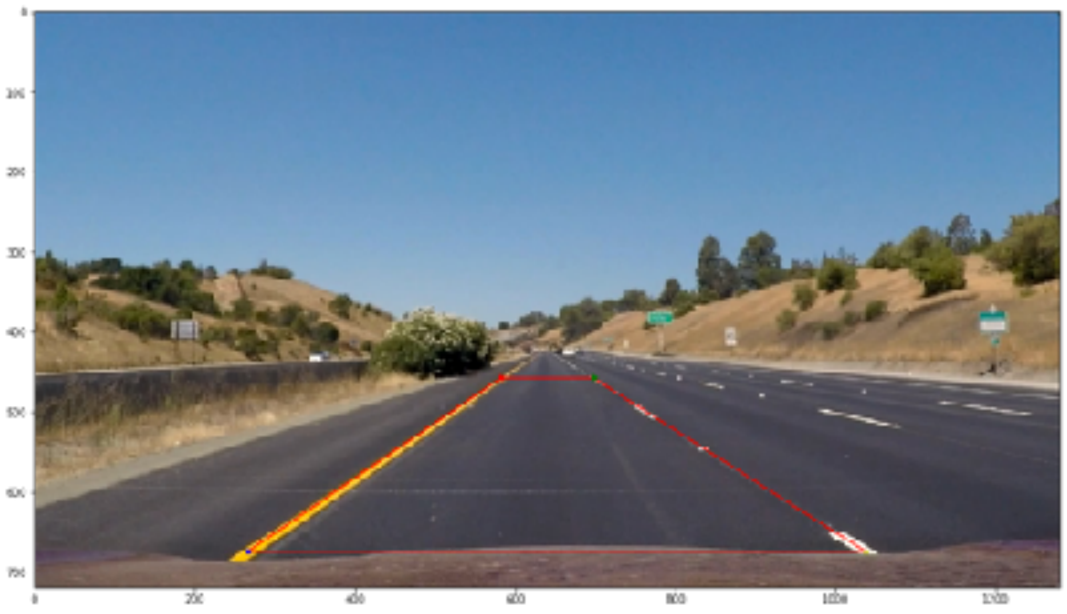
Examples of a distortion-corrected image(s)

Some road example of a distorted (left) and distortion-corrected (right) image are as follows:-



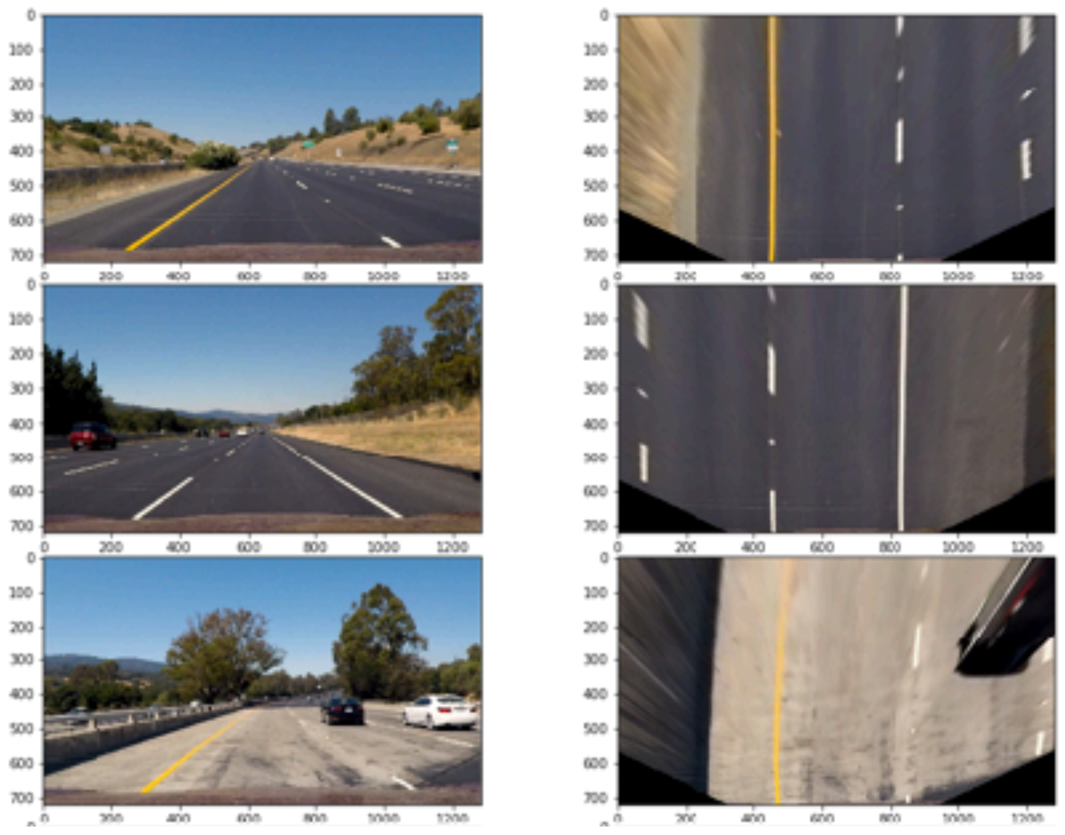
Perspective Transform

Please see the four source point on the images as point-1 (*), point-2(*), point-3(*) and point-4(*)



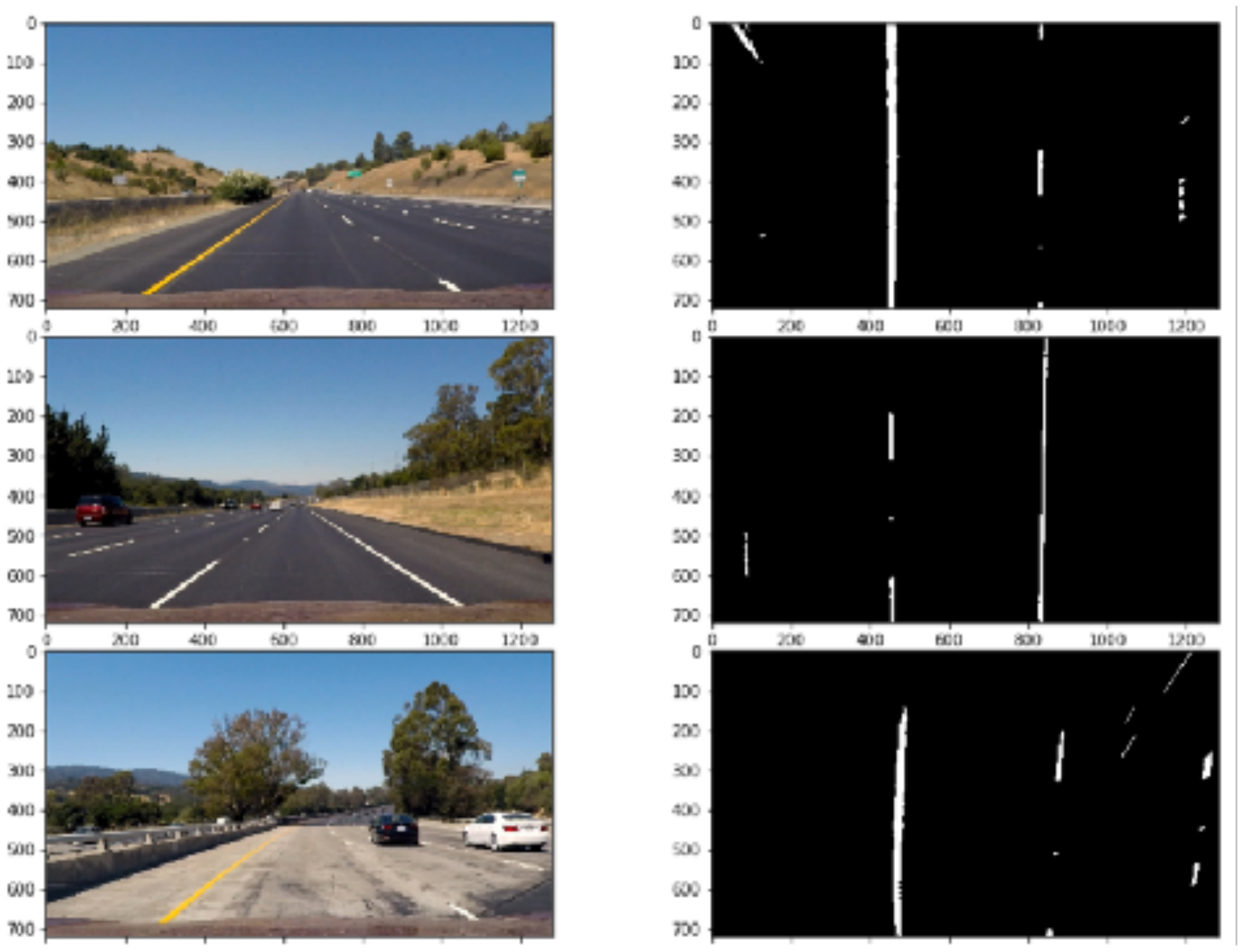
These point are used to transform image perspective and these 4 points (here as trapezoid) should map to a rectangle.

Some examples of transformation are as follows:-



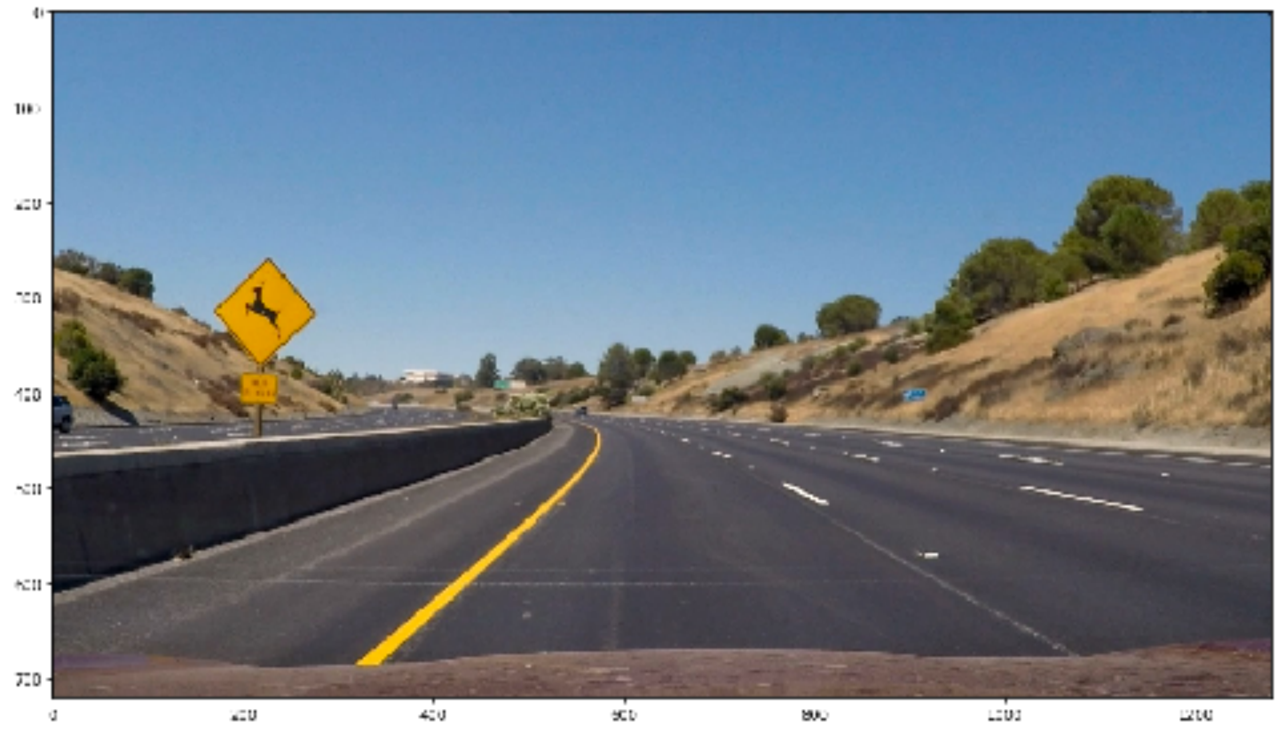
Thresholded binary image

HLS and and Lab are good enough to colour transformation for this. Take S-part of HLS and threshold it and take b-part of Lab and threshold it to get thresholded to binary images that corresponds to lane lines. Now combine these two binary images using OR logic, i.e. if any binary image contains lane line then combined should contains a lane line. I took each image from test images and passed each from process_1_cc_pt_thr() function which is 1st function in the pipeline. The results are as follows:-

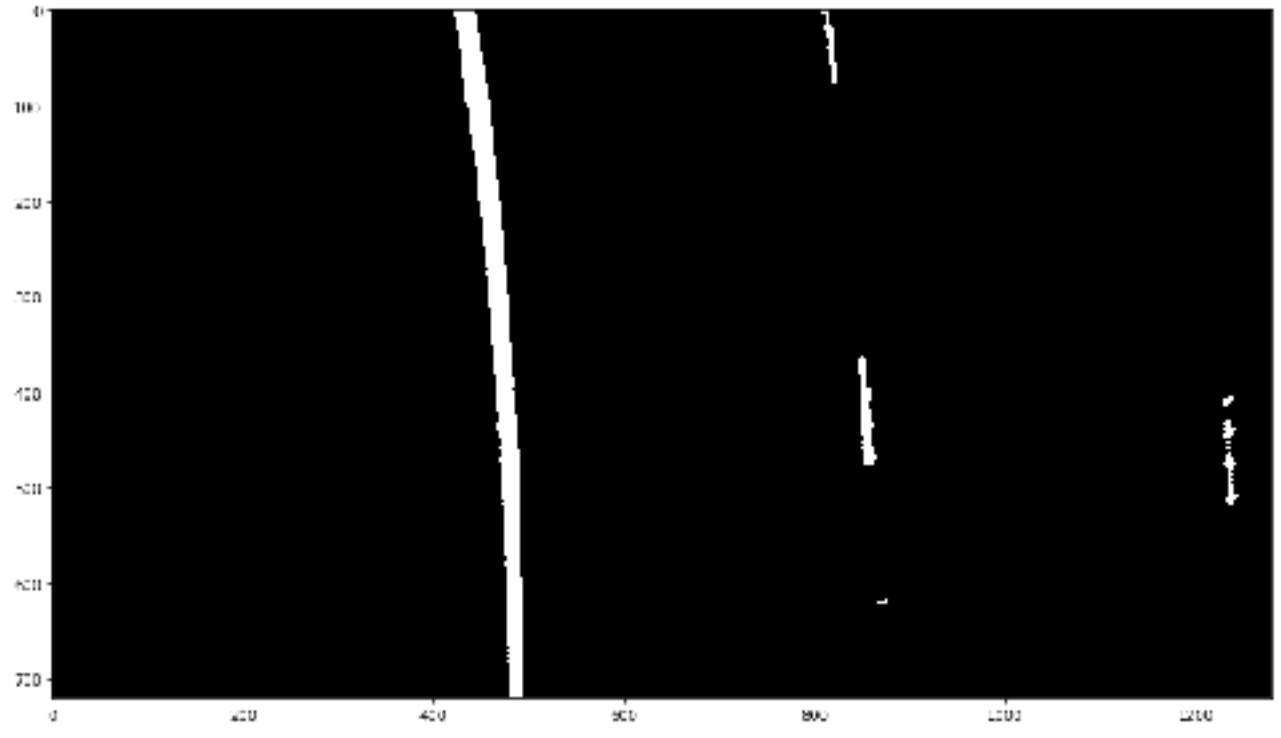


Identify lane-line pixels and fit their positions with a polynomial

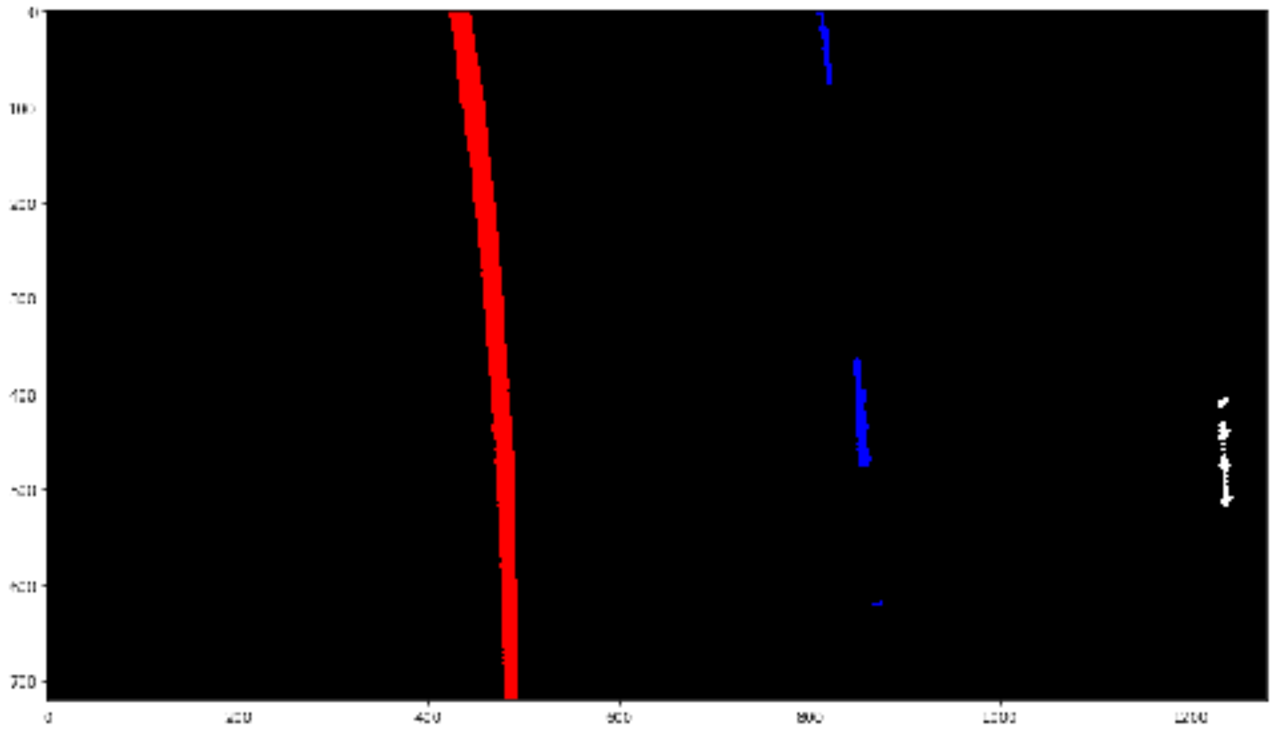
I started with following image



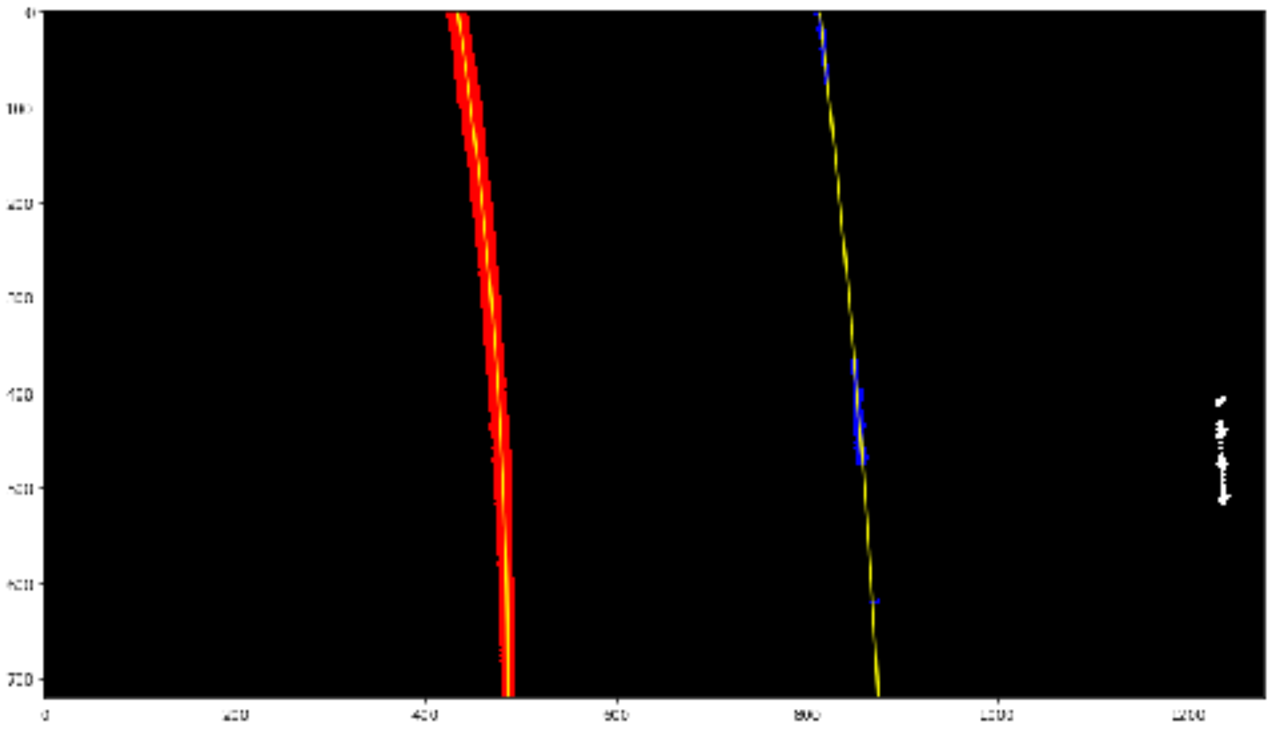
After that cam-calibrated, perspective transformed it and then binary thresholded it to get the following image.



First step is to find all the pixels with non-zero value, i.e. white pixels. Then as explained in the Udacity video tutorials, find maximum of the histogram of the bottom section of the image to find initial location of white part which is lane line bottom part. This is start of the lane lines from bottom to top. Now search remaining part of the lane lines upward in a windows around the expected lane lines locations. Next identify all the white pixels what are part of the lane lines. Using these indices find the pixels of the lane lines only. Left lane line pixels are in red and right lane pixels are in blue.



Now fit the 2nd order polynomial curve in yellow colour to these (x,y) pixels, left and right separately. After plotting the fitted line we get the following image.



Calculation of the radius of curvature of the lane

We got two 2nd order curves parameters for right as and left lane.

A = left_fit_coef[0]

B = left_fit_coef[1]

C = left_fit_coef[2],

Similarly right_fit_coef for right lane curve.

Now there is a very simple formula to calculate the radius of curvature using calculus.

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

$$f(y) = Ay^2 + By + C$$

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

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

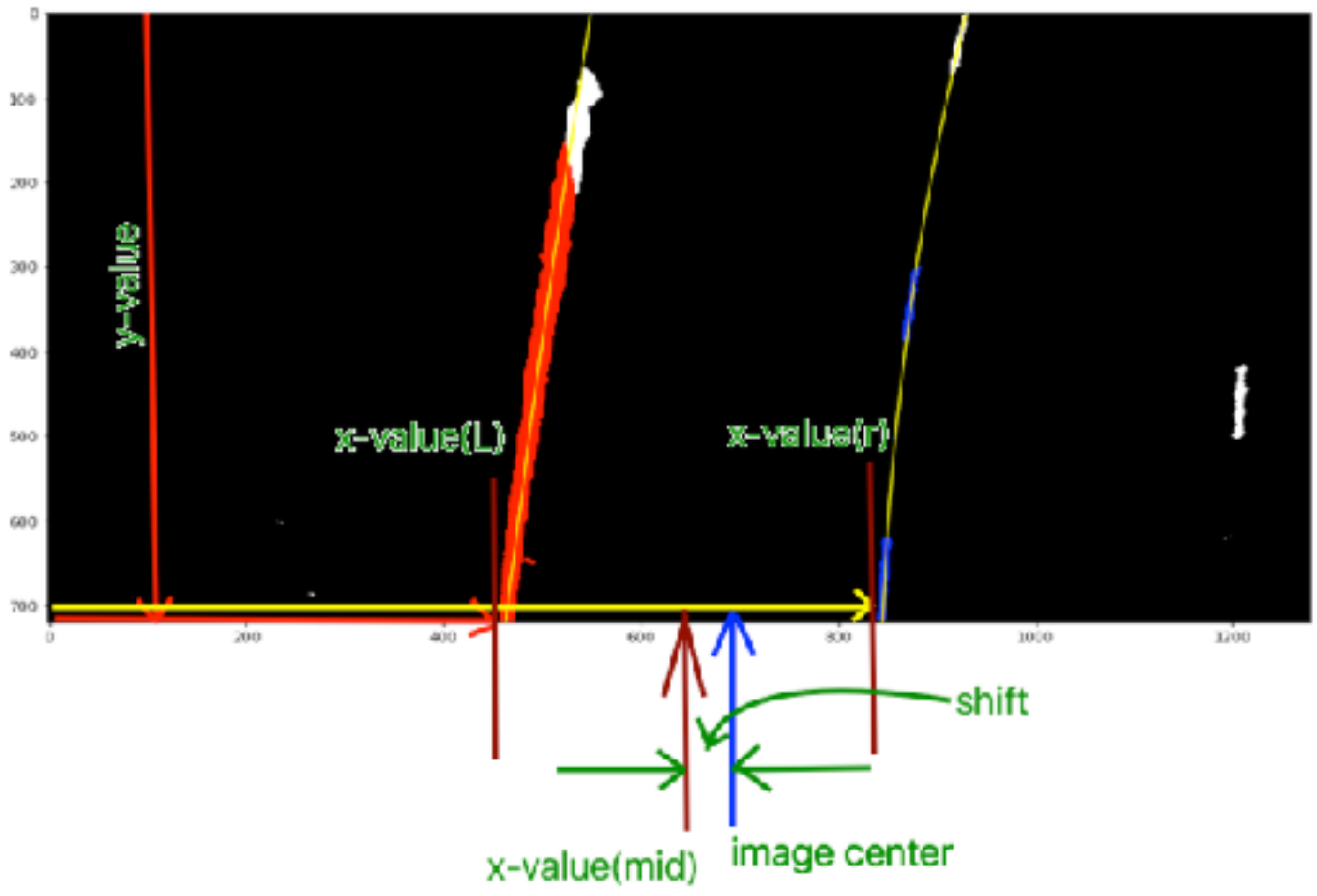
So radius of curvature becomes:

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

One point we need to keep in mind is that we need to use a conversion or scaling from pixel to actual road length in meters, as follows:-
y_scale_meters_per_pixel = 3.048/100
x_scale_meters_per_pixel = 3.7/378

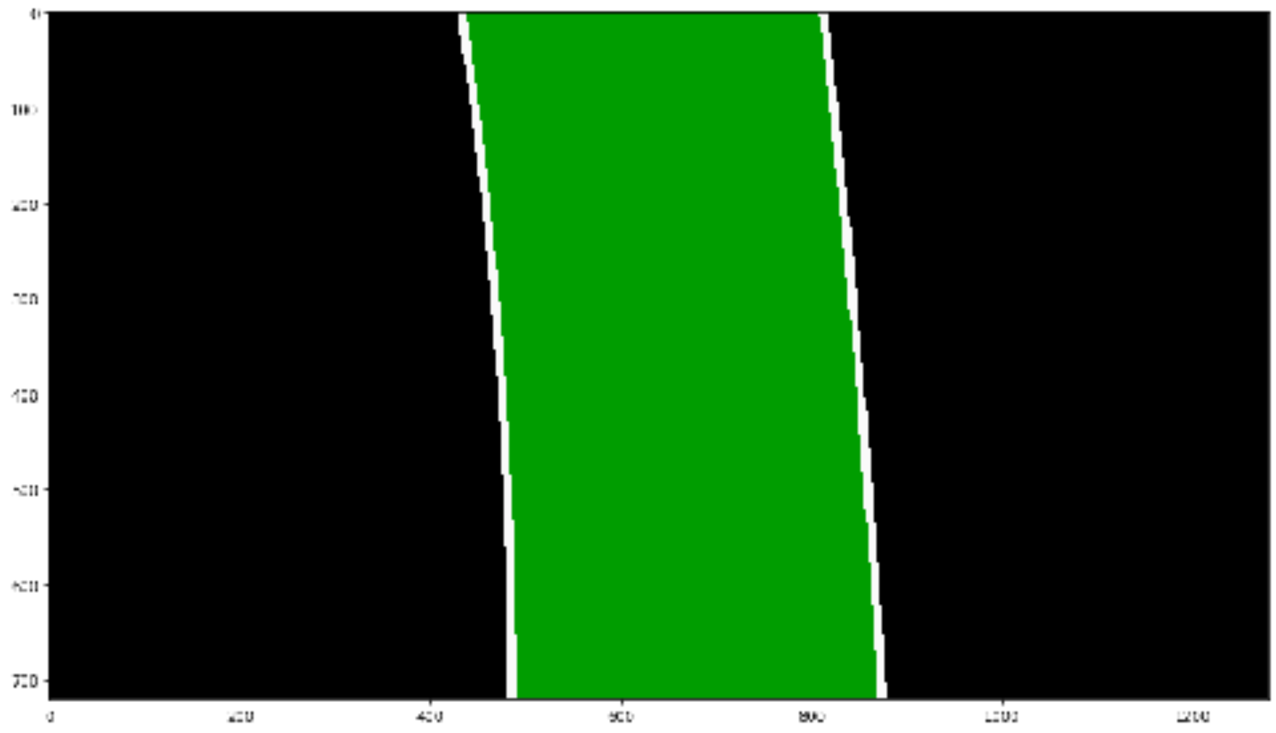
Position of the vehicle with respect to center

Because camera is fitted in the center of the car so image center is the center of the car. We also have both right and left lane equations. We can find the x-value using for x-right and x-left using y-value as height of the image. And then as shows in the code in function process_3_find_curvature_and_shift, we can calculate shift from center for the our car.

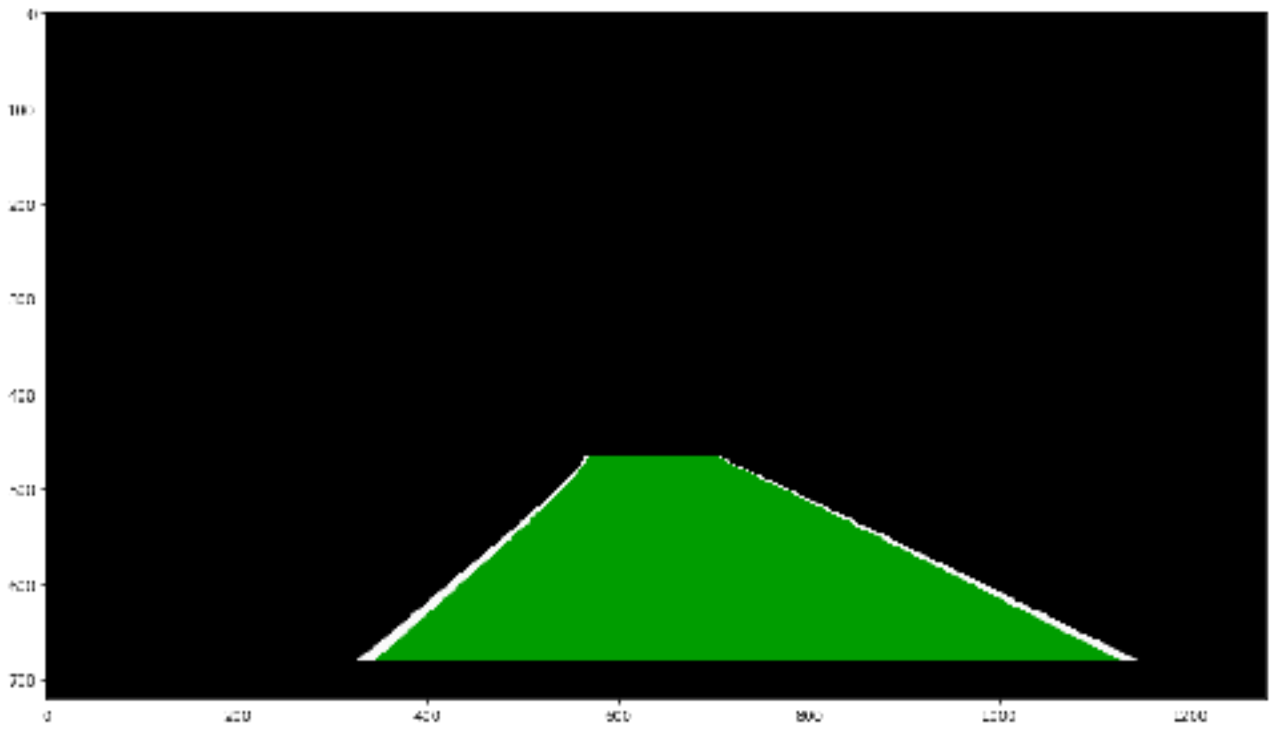


Plotted back down onto the road

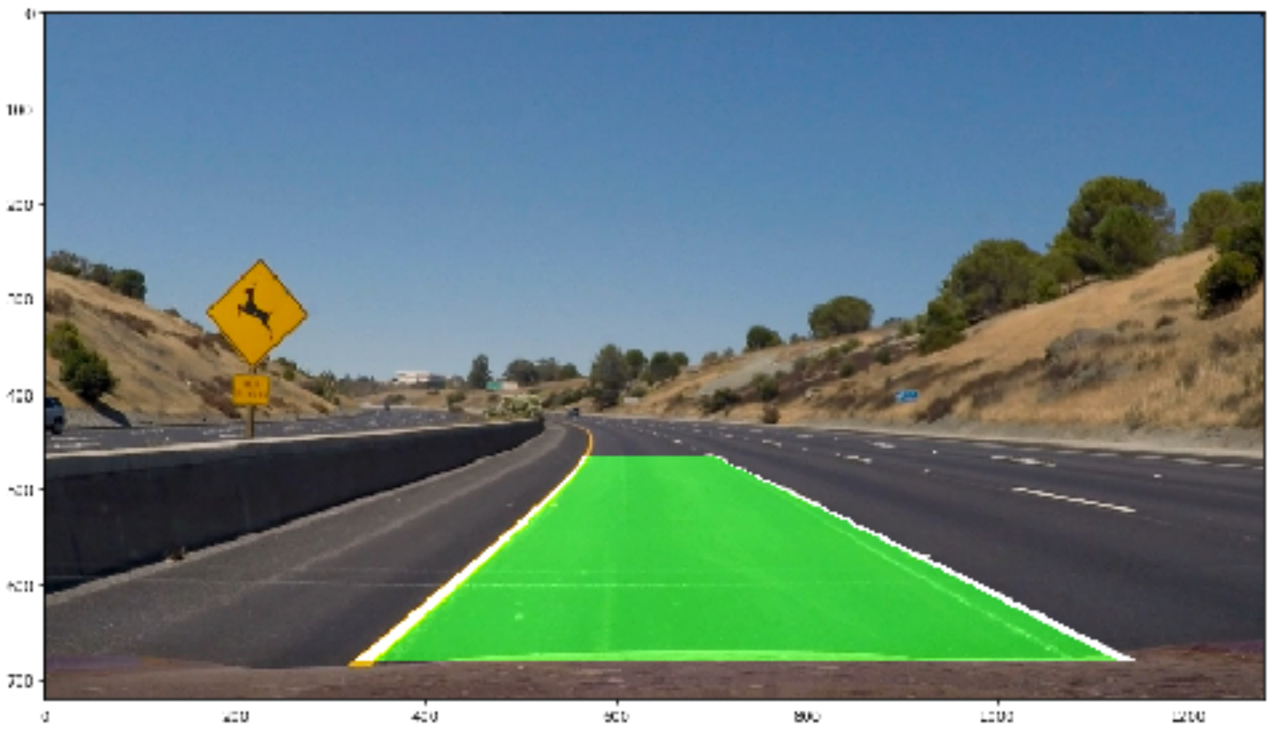
Now as shown in the Notebook fill the area between two fitted lane curve with a green colour as follows



Now use the function perspTransInverse() to inverse perspective transformation on the above image to get the following image.



After that use OpenCV addWeighted function to add these two images using blending parameters to get the following image.



Link to Videos

Project Video link	- https://youtu.be/Jh0We6WD9c4
Challenge Video link	- https://youtu.be/UbC3jJtVS_o
Harder challenge video link	- https://youtu.be/lGOPrNH-spw

Discussion

Apart from the methods used we can also use following ideas to improve the results

1. We can use optical flow and the direction of that optical flow to identify the lane lines.
2. Inter frame smoothing in video as there is a continuity in the lane lines along frames.
3. Probabilistic expectation where lane lines can be.