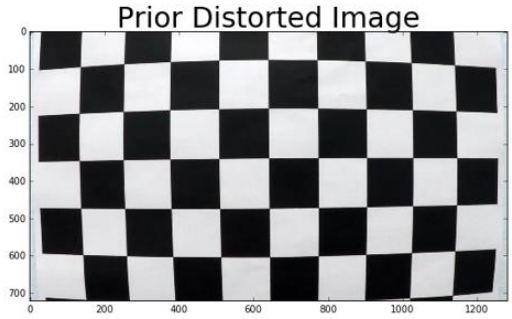
Advanced Lane Finding

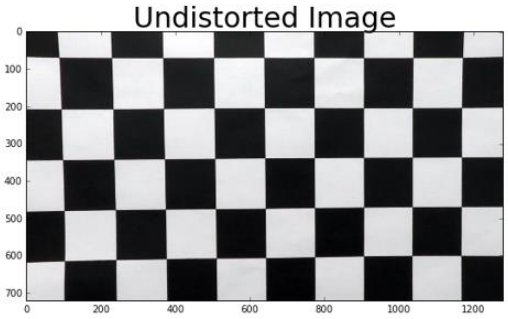
Camera Calibration

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

This was done in Step 1 of the p4-advanced-lane-lines.ipynb.

The calibration images were read, points to map were generated. The image points were found using cv2.findChessboardCorners. The camera was further calibrated and distortion coefficients were obtained using cv2.calibrateCamera





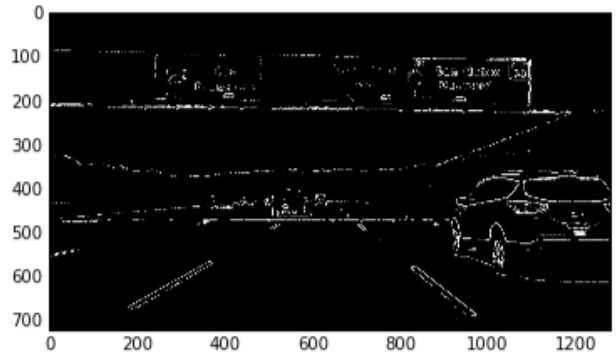
Pipeline (single images)

2. Apply distortion correction to each image

Cv2.undistort was used first

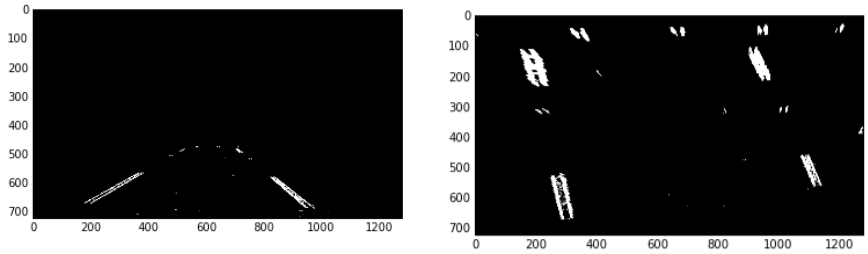
3. Binary Thresholding

The grayscale image and color channel were thresholded and the two binary thresholds were combined to generate a binary image. The parameters were determined by hit and trial.



4.Perspective Transform:

First binary mask was applied. Then the image was transformed into birds eye view perspective. The source and polygon coordinates were hardcoded, and matrix M was obtained which maps them onto each other using cv2.getPerspective.



Before After

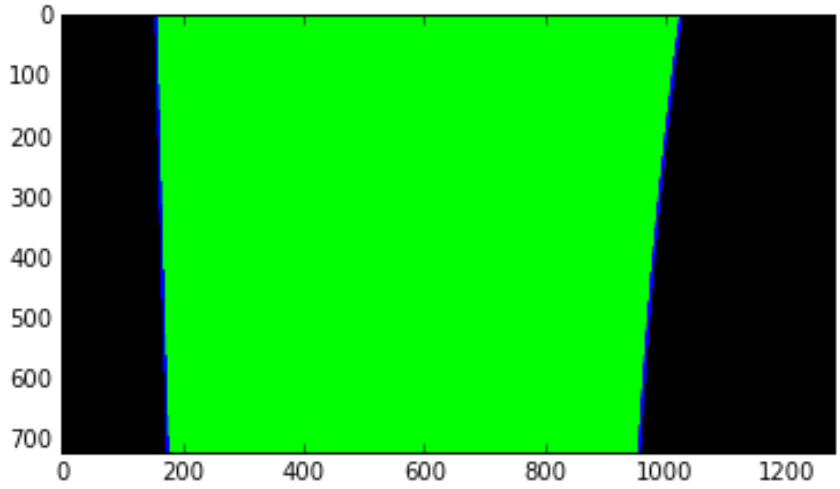
5. Identify lane-line pixels and fit their positions with a polynomial

Identify lane line pixels

• Divide the image into n horizontal strips (steps) of equal height.
• For each step, take a count of all the pixels at each x-value within the step window using a histogram generated from np.sum.
• Smoothen the histogram using scipy.signal.medfilt.
• Find the peaks in the left and right halves (one half for each lane line) histogram using signal.find_peaks_swt.
• Get (add to our collection of lane line pixels) the pixels in that horizontal strip that have x coordinates close to the two peak x coordinates.

Fit positions of lane-line pixels with a polynomial

• Fit a second order polynomial to each lane line using np.polyfit.



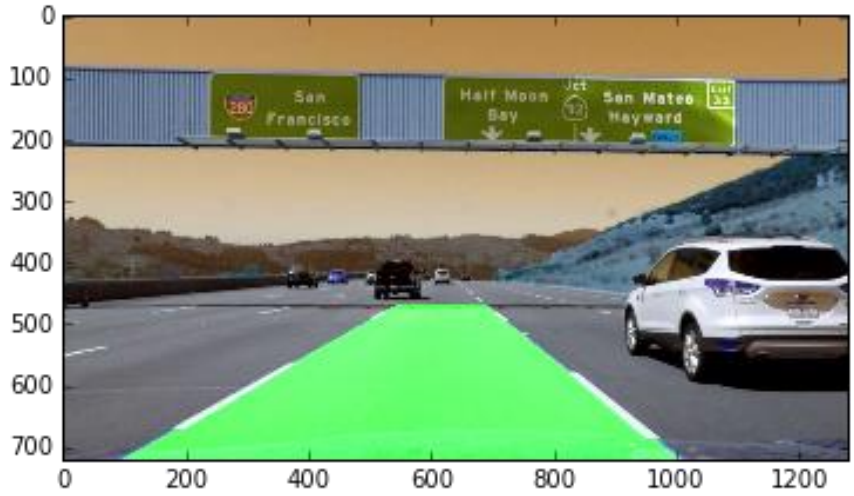
6. **Determine curvature of the lane and vehicle position with respect to center**

This was the code used to do the given task:

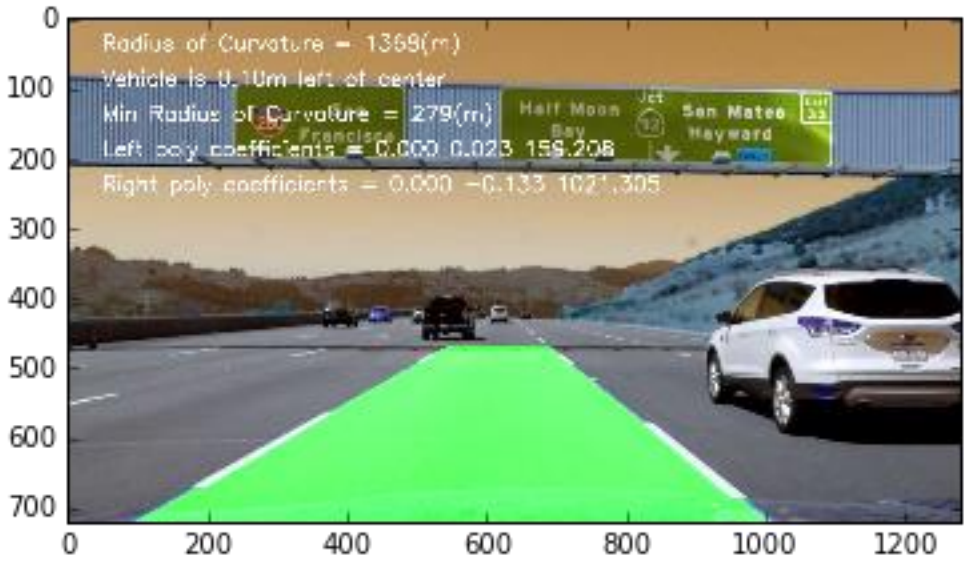
left_curverad = ((1 + (2*left_fit[0]*y_eval + left_fit[1])**2)**1.5) /np.absolute(2*left_fit[0])

7. Plot result back down onto the road such that the lane area is identified clearly.

The lane lines were warped on the original image using cv2.warpPerspective and lane lines were combined using cv2.add



8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.



Video Pipeline

All the required functions were put in a single function image_pipeline under useful.py.

The video has been attached as a part of the submission

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?

- There were instances of random noise interfering with the lane lines and making them wider. Increasing the threshold would help in this regard
- Sometimes when no lane lines are detected , overlapping data from previous frame over current frame might help