ADVANCE LANE FINDING PROJECT

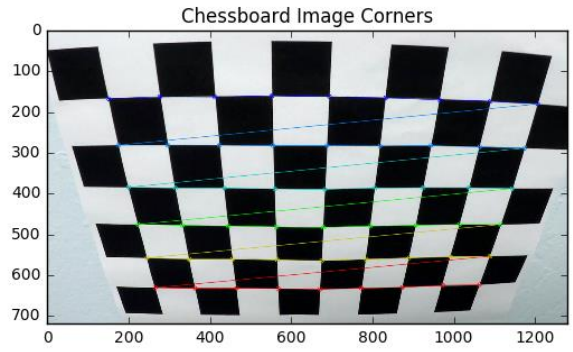
The objective of this project is to write an application pipeline to identify the lane boundaries in video from a front-facing camera on a moving vehicle and the result of the project is an annotated video that identifies the positions of the lane lines, the location of the vehicle relative to the center of the lane, the radius of curvature of the road

The following steps were taken to achieve these objectives.

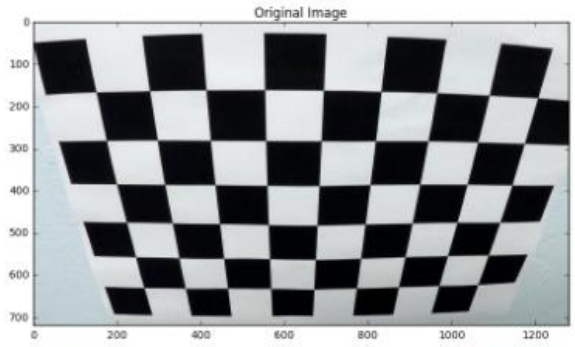
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a threshold binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to the 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.

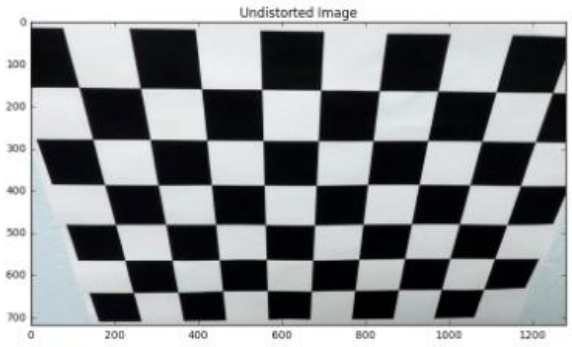
Distortion Correction

This step involves computing the calibration matrix and distortion coefficients of distorted images. We were provided with sample distorted chessboard images. For this step, I used the OpenCV functions findChessboardCorners to identify the locations of corners on the series of chessboard images with different camera angles and drawChessboardCorners to visualize the corners.



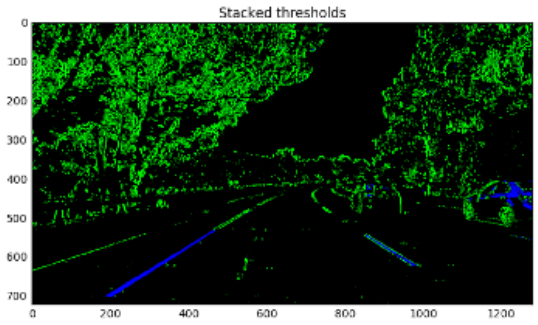
The calibrateCamera function is then used to compute the calibration matrix and distortion coefficients which were then fed as input to the undistort function to correct the distortion in the images.

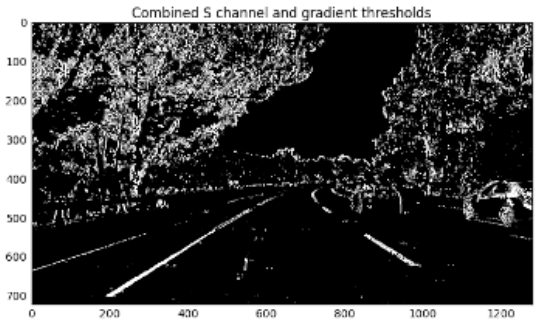




Create a threshold binary image:

In this step, I created a binary image using color transforms, which were a combination S channel of the HLS color channel and gradient thresholds. The OpenCV Sobel operator was used to compute the gradient. X and y axis magnitude and direction were computed and combined to isolate the lane line pixels.





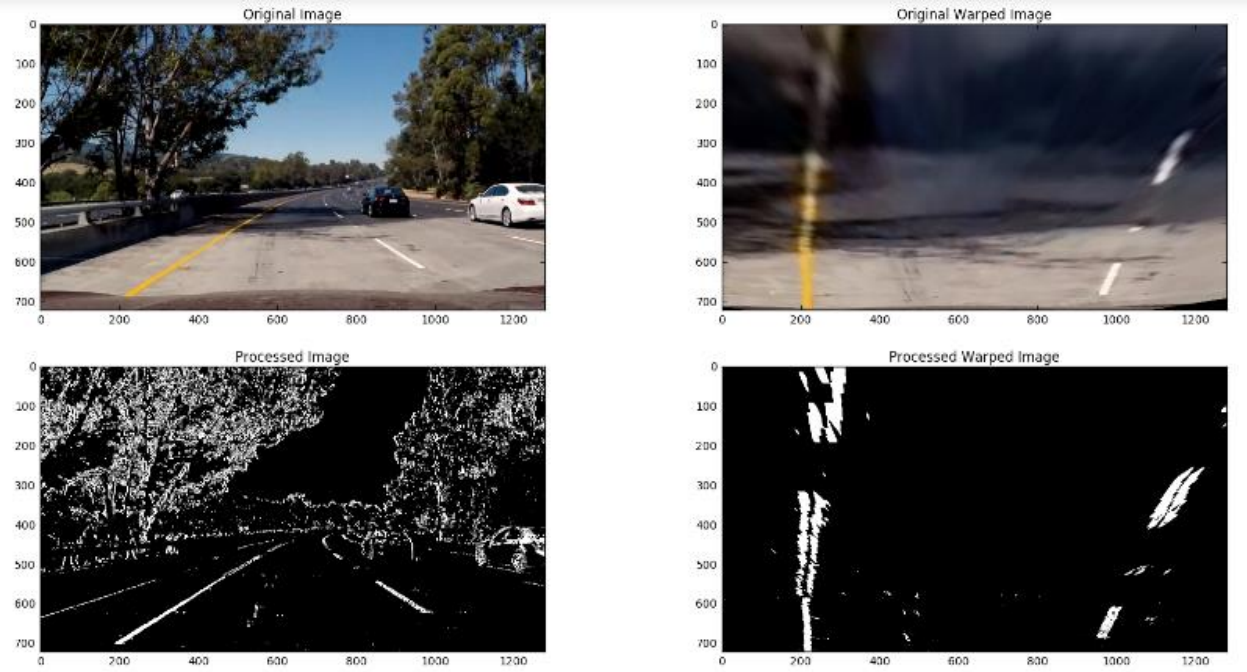
Perspective transform to rectify binary image

The step involves transforming the binary image to a birds'-eye view perspective which allows us to view the lanes from above. This is done by mapping the points in the binary image to our desired image points using the OpenCV getPerspectiveTransform and warpPerspective functions to produce a warped binary image. The source points used were [595,460], [680, 460], [1150, 700], [300,700] and the destination was computed as:

img_size = (img.shape[1], img. shape[0])

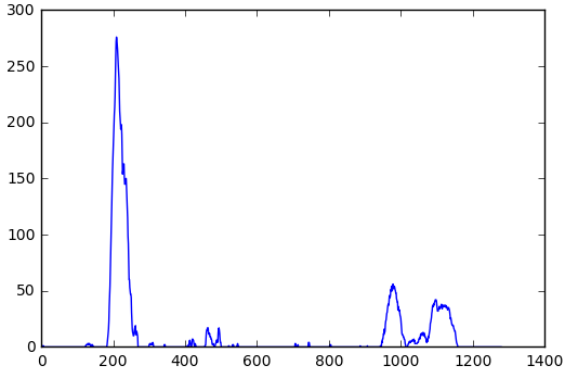
[offset, 0], [img_size[0]-offset,0], [img_size[0]-offset, img_size[1]], [offset,img_size[1]]

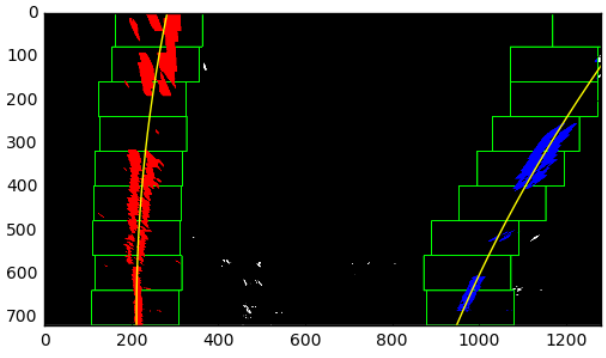
where offset = 300



Detect lane pixels and fit to find the lane boundary

This step involves determining which pixels are lane lines and which belong to the left lane lines and which belong to the right lane lines. The numpy Histogram function was used to determine the peaks which are indicators of the starting of a lane and then then used the sliding window to trace the lanes.





After detecting and marking all the pixels for the left and right lane lines, I then fit a polynomial to each of them and then use the polynomials to draw the lines and apply the green color in the area in between the lines.

Warp the detected lane boundaries back onto the original image.

The resulting image after all our computation has been in birds' view perspective so we need to transform the image back to its original image space. For this we compute the inverse of transformation matrix – Minv and used the warpPerspective function to transform the image back to its original image space.

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

Along with transforming the perspective of result back to the original image space I also annotated the resulting image with the resulting values from the computed right curvature, left curvature and lane deviation.

Discussion

The pipeline fails when the vehicle is driving segments of the road where the white lanes (right lanes) are not easily distinguishable because they have faded off the road or the road shade becomes lighter This can be improved by enhancing the robustness of the polynomial fit to accommodate situations where the lane lines might be absent due to lanes fading off.

The model also inaccurately detects the lanes lines briefly when there is sudden appearance of other continuously moving objects (moving vehicles) in the image. This is because the presence of the light colored tire rims appears like road lanes which makes the model draw the right lane beyond the actual road lane. This can be improved by enhancing the lane detection to exclude other light colored objects detected close to an already detected lane within the region of interest.

Occurrences of situations like these happen every day on most roads so the algorithm is not robust enough. A deep learning approach might produce a better result since the model would then be learning by the images it has seen which will include the different shade variations of the road and continuously moving vehicles detected while driving.