Advanced Lane Finding Project

The goals / steps of this project are the following:

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 thresholded 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.

Computation of the camera matrix and distortion coefficients.

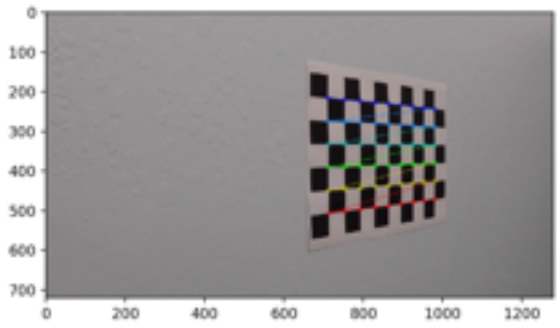
OpenCV functions were used to calculate camera matrix and distortion coefficients using the calibration chessboard images provided in the repository.

Following cv2 fn. is used to find chessboard corners (for an 9x6 board):

ret, corners = cv2.findChessboardCorners(gray, (9,6), None)

Detected corners on an image were drawn using below cv2 function:

img = cv2.drawChessboardCorners(img, (9,6), corners, ret). Below is an image with corner identified on chessboard.

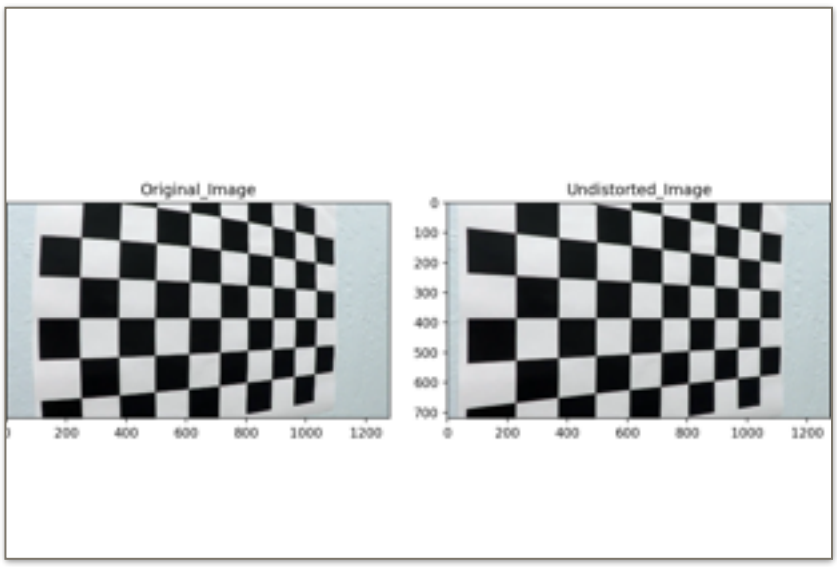


Note: corners were not detected for three images provided in calibration images set.

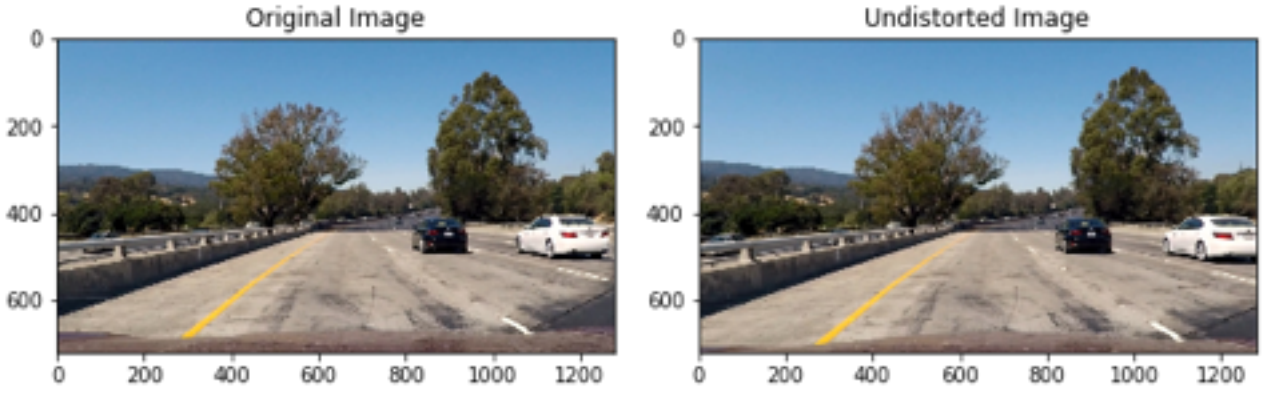
Camera calibration, given object points, image points, and the shape of the grayscale image is calculated as follows,
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

Undistorting a test image is done using below cv2 function,
dst = cv2.undistort(img, mtx, dist, None, mtx)

The calculated distortion matrix is used for distortion correction of following images as seen below. More examples shared in the output_images folder of the project and also in the jupyter notebook file.



An example of a distortion-corrected image.



how (and identify in code) color transforms, gradients or other methods were used to create a thresholded binary image.

HLS color space is used to detect lane lines. Saturation channel in HLS color space is chosen between min threshold of 190 and max threshold of 255 for lane line pixels detection. Saturation is a measurement of colorfulness. So, as colors get lighter and closer to white, they have a lower saturation value, whereas colors that are the most intense, like a bright primary color (imagine a bright red, blue, or yellow), have a high saturation value. Thus low contrast colours can be distinguished using HLS color space and distinction between lane lines and noises like tree shadow can be made.

Sobel x gradient thresholding is done to select lane line pixels. Min gradient threshold of 30 and max gradient threshold of 100 is combined with S Channel information from HLS Color space.

B-channel of Lab color space thresholded between 170 and 255 proved successful in distinguishing between yellow and light shaded region or white lines.

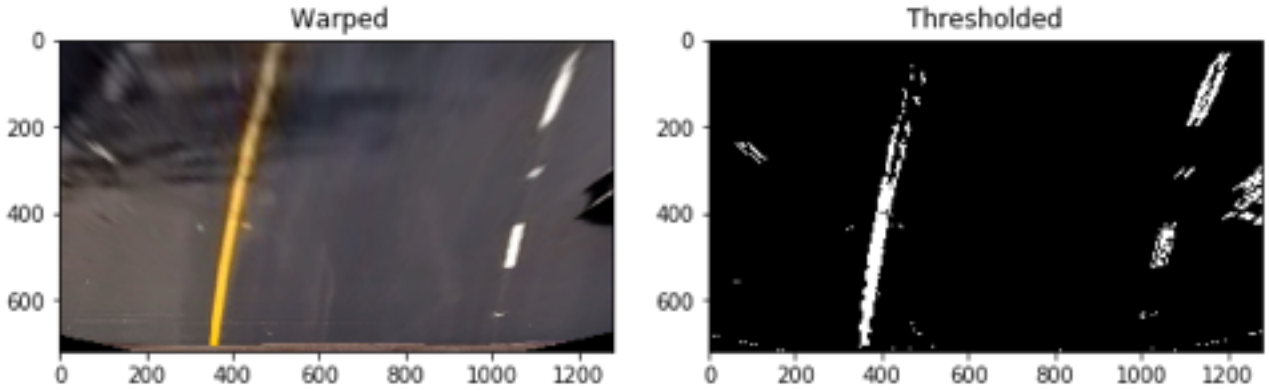
Combining the above three results provided lane line pixels required to plot polynomial for lane line detection.

Please find results obtained from various combination of channels from HLS and LBS color space in the jupyter notebook file of the project or the output_images folder of the project.

how and (identify in code) a perspective transform is achieved and provide an example of a transformed image.

Open CV function to calculate perspective transform matrix with inputs as source points on input image and destination points on the resultant image.

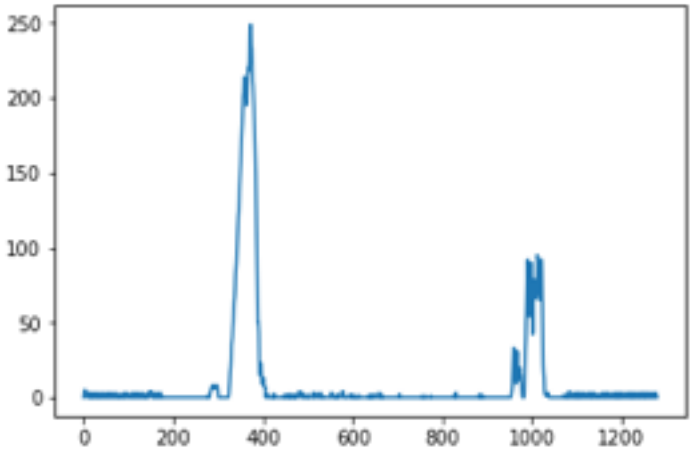
```
src = np.float32([[585,460],[203,720],[1127,720],[695,460]])

dst = np.float32([[320,0],[320,720],[960,720],[960,0]])

Minv = cv2.getPerspectiveTransform(dst, src)
newwarp = cv2.warpPerspective(color_warp, Minv, (thresholded.shape[1],
                              thresholded.shape[0]))
```



For more images, please refer jupyter notebook of the project or output images folder.

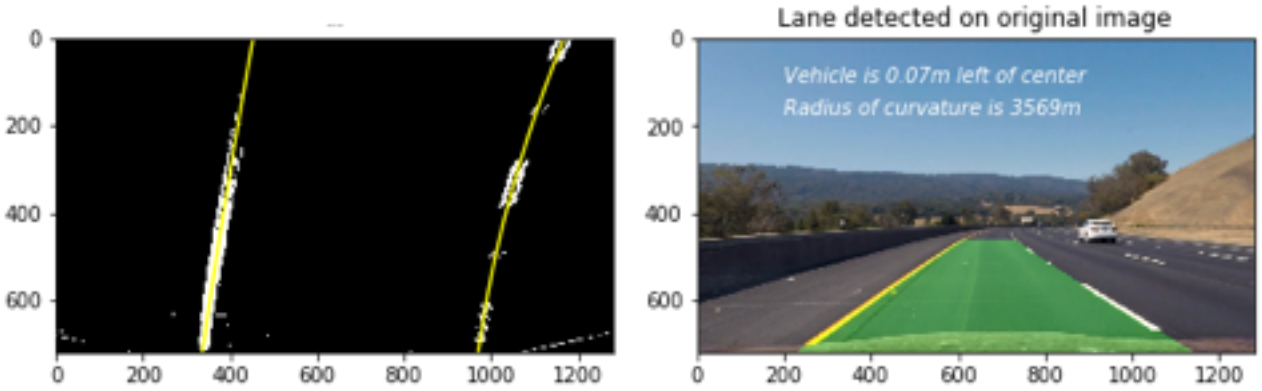
how (and identify in code) lane-line pixels are identified and fit their positions with a polynomial?

Line Finding Method: Peaks in a Histogram
After applying calibration, thresholding, and a perspective transform to a road image, lane lines stand clearly.I first take a histogram along all the columns in the lower half of the image like below:



Next Sliding Window is used to fit a polynomial for lane lines as below,

With the above histogram I am adding up the pixel values along each column in the image. In my thresholded binary image, pixels are either 0 or 1, so the two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. I used that as a starting point for where to search for the lines. From that point, I used a sliding window, placed around the line centres, to find and follow the lines up to the top of the frame.
Implementation of the same is present in function FitPoly() in the jupyter notebook file of the project.
Below image in left is seen after lane lines are plotted using sliding window technique.



Calculate the radius of curvature of the lane and the position of the vehicle with respect to centre.

Assume the camera is mounted at the center of the car, such that the lane center is the midpoint at the bottom of the image between the two lines detected. The offset of the lane center from the center of the image (converted from pixels to meters) is the distance from the center of the lane.

This is implemented in FitPoly() function in jupyter notebook file of the Project. Below are the excerpts from the code file,

```
ImageMidpoint = 640
XInterceptRightLane = right_fit[0]*720**2 + right_fit[1]*720 + right_fit[2]
XInterceptLeftLane = left_fit[0]*720**2 + left_fit[1]*720 + left_fit[2]
Diff = (XInterceptLeftLane + XInterceptRightLane)/2
Offset = abs(640 - Diff)
```

Example: in above image, offset is 0.34 m left of centre

For Radius of Curvature below is the implementation,

```
# Define y-value where we want radius of curvature
# I have chosen the maximum y-value = 720, corresponding to the bottom of the image and corresponding x value is evaluated.
```

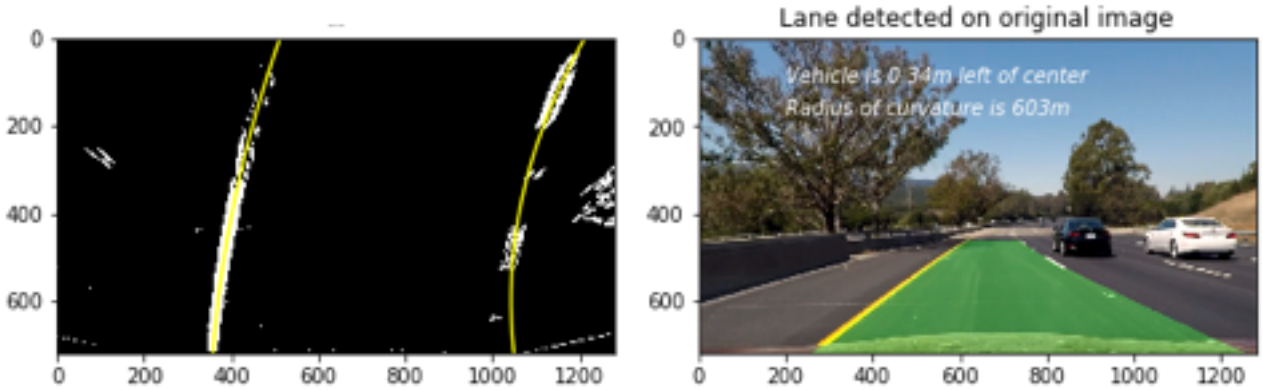
This value is further converted from pixels to metres assuming that lane length is 30 metres long and lane width is 3.7 metres long. Thus conversion ratio 30/720 and 3.7/700 is used to convert y and x points of lane line resp. from pixels to metres. Complete implementation can be seen in jupyter notebook file of the project.

```
#Measure radius of curvature for each lane line
    ym_per_pix = 30/720 # meters per pixel in y dimension, lane length is 30 metres
```

```
    xm_per_pix = 3.7/700 # meters per pixel in x dimension, lane width is 12 ft = 3.7 metres
    y_eval = np.max(ploty)

# Fit new polynomials to x,y in world space
    left_fit_cr = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
    right_fit_cr = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)

 # Calculate the new radii of curvature
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix  + left_fit_cr[1])**2)**1.5) /
np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix  + right_fit_cr[1])**2)**1.5) /
np.absolute(2*right_fit_cr[0])
```

Example values: 705meters in the above example

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



Link to the final video output.

Please refer below video or check output.mp4 present in the project for result of Advanced Lane Detection algorithm.

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?

I initially faced issue in removing the tree shadows and other discolouration/low shaded regions. This was overcome by using B-channel of Lab color space thresholded between 170 and 255. B-channel of Lab color space was further combined with S channel from HLS color space, thresholded between 190 and 255 and Sober x gradient thresholding with absolute value between 30 and 100. Once test results were confirmed on test images pipeline, the same was applied to the project video.

Next, best fit polynomial coefficients obtained by taking an average over the last n iterations is used in achieving smooth detection of lane lines.
For visualisation bestx, average x values of the fitted line over the last n iterations were used to plot detected lines from warped image onto the original image.

There were necessary resets required for cases where sanity checks failed on any frame. Example: when radius of curvature of lane lines differed drastically like of the order 10K or separation between two lane line difference was more than 700pixels (+/- some margin), then it was required to recalculate lane line coefficient again using histogram and sliding window technique.

Pipeline may fail on images with very less contrast available like snow condition when it may be difficult to distinguish between lane lines and wrong plotting may happen. This would require further work on different color space for thresholding and a robust pipeline.

Reference: Udacity