Advanced Lane Finding Project

The goals / steps of this project are the following:

1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
2. Apply a distortion correction to raw images.
3. Use color transforms, gradients, etc., to create a thresholded binary image.
4. Apply a perspective transform to rectify binary image ("birds-eye view").
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

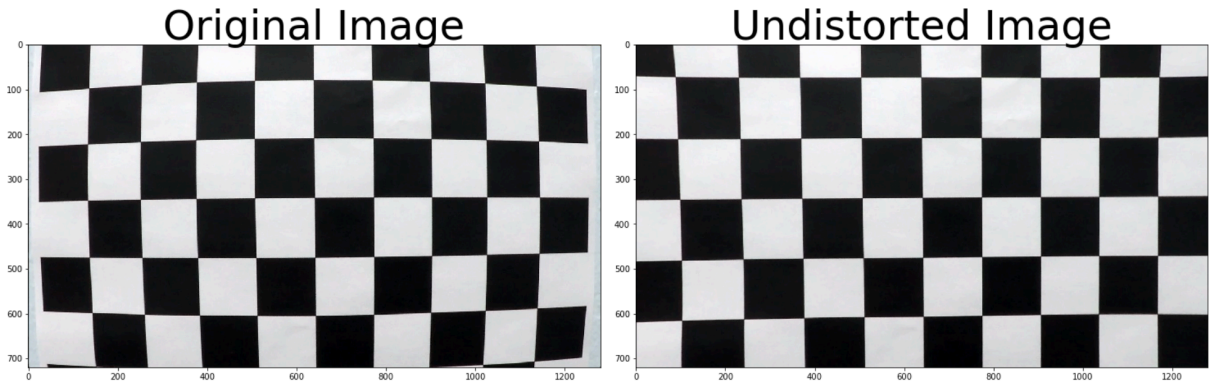
Camera Calibration

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

The code for this step is contained in the first code cell of the IPython notebook

I start by preparing "**object points**", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `**objp**` is just a replicated array of coordinates, and `**objpoints**` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `**imgpoints**` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output `**objpoints**` and `**imgpoints**` to compute the camera calibration and distortion coefficients using the `***cv2.calibrateCamera()***` function. I applied this distortion correction to the test image using the `***cv2.undistort()***` function and obtained this result:



Pipeline (single images)

1. Provide an example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image. I tried various techniques like HLS, Sobel and LAB-B colorspace. First, I applied HLS-S and Sobel-Abs combined on the image (that was undistorted and perspective transformed). This combination seemed noisy. Later I added LAB-B colorspace and combined LAB-B with HLS-L to get a cleaner binary image. The code can be found in "**Define Image Processing Pipeline**" cell. Here's an example of my output for this step.

3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `**unwarp()`,** which appears in the **"Perspective Transform"** cell of the IPython notebook. The ` **unwarp** ()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

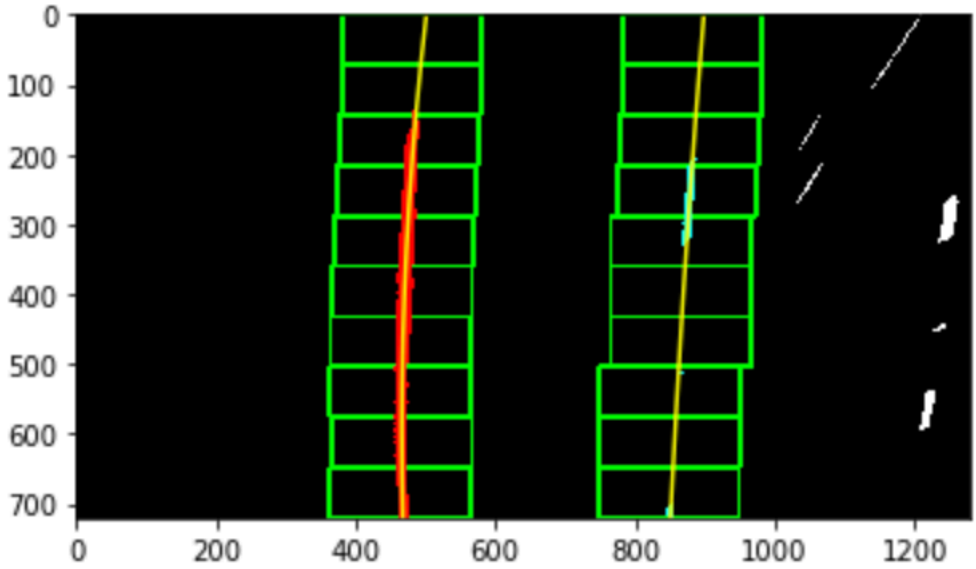
```
# define source and destination points for transform
src = np.float32([(575,464),
                  (707,464),
                  (258,682),
                  (1049,682)])
dst = np.float32([(450,0),
                  (w-450,0),
                  (450,h),
                  (w-450,h)])
```

This resulted in the following source and destination points:

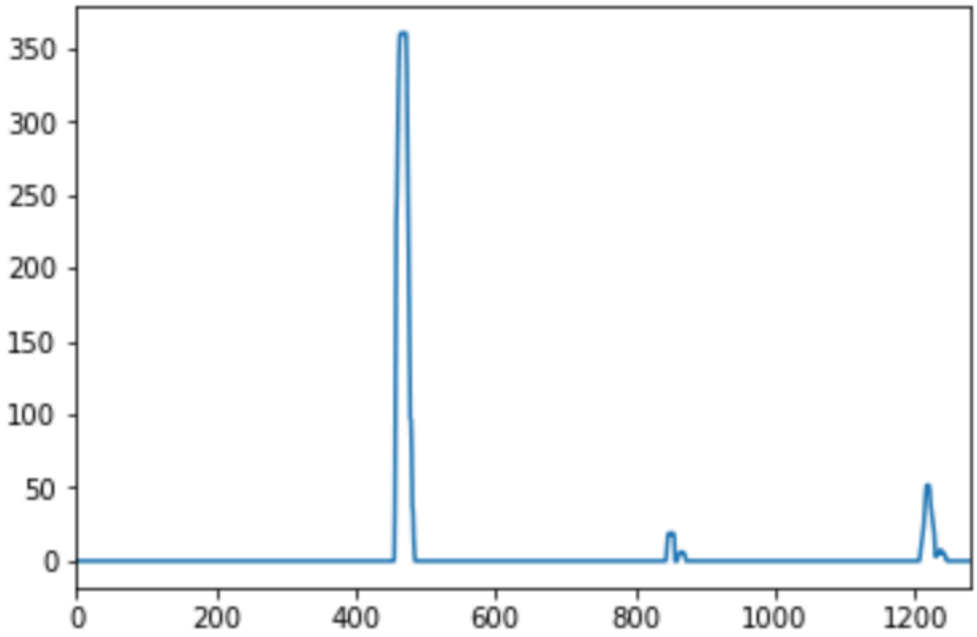
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

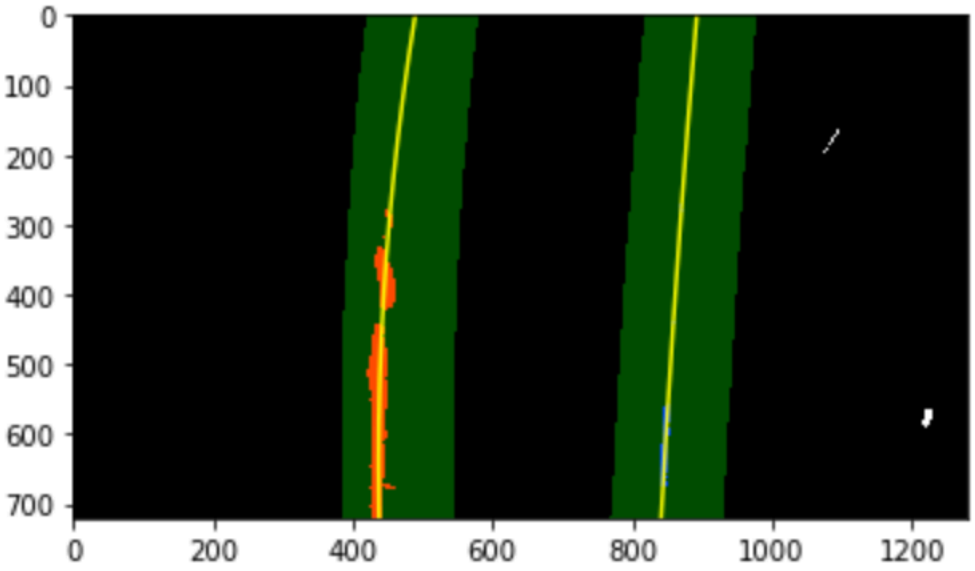
The functions **slidewindowPoly** and **polyfit_using_prev_fit**, identify lane lines and fit a second order polynomial to both right and left lane lines. I compute a histogram of the bottom half of the image and find the bottom-most x position (or "base") of the left and right lane lines. To fidd those left and right lane lines, I only consider a quarter of the histogram directly to the left/right to avoid the noise. The below diagram shows the output of **slidewindowPoly** function



The histogram generated by **slidewindowPoly**



And the diagram generated by the previous polynomial fit



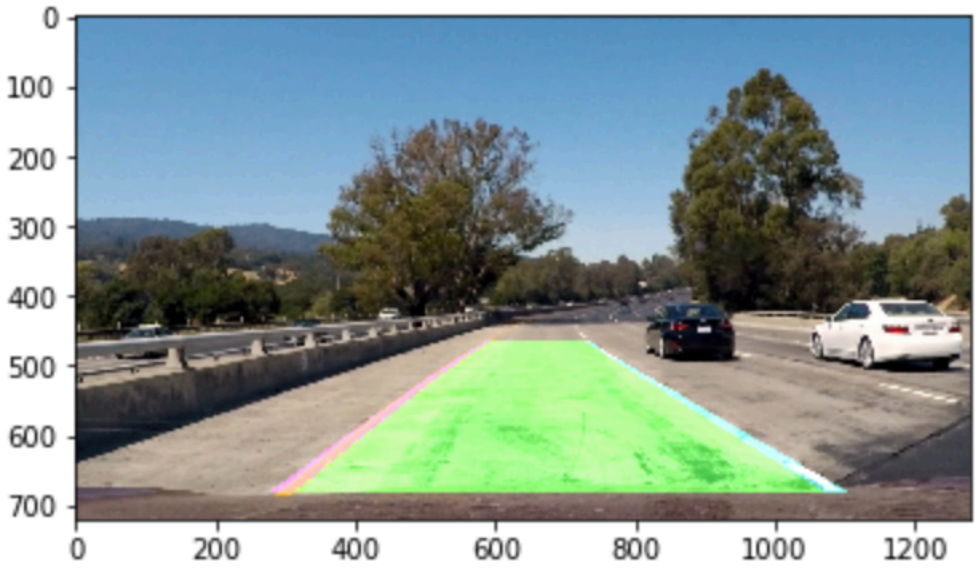
5. **Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.**

I did this step in "**Measure Curvature**" code cell using below steps

i. Define conversions in x and y from pixels space to meters

ii. Define y-value where we want radius of curvature
iii. Identify the x and y positions of all nonzero pixels in the image
iv. extract left and right line pixel positions
v. Fit new polynomials to x,y in world space
vi. Calculate the new radius of curvature in meter
vii. Find the distance from center is image x midpoint - mean of l_fit and r_fit intercepts

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



Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road.

Attached video **project_video_output.mp4**