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.

The main for the project is in "./Adv_lane_finding.ipynb" and all references in the below description refers to cell number in this notebook. There is another ipython notebook "./Experimental code.ipynb" which contains trial and error code which is used for debug purpose. Please ignore this ipython notebook.

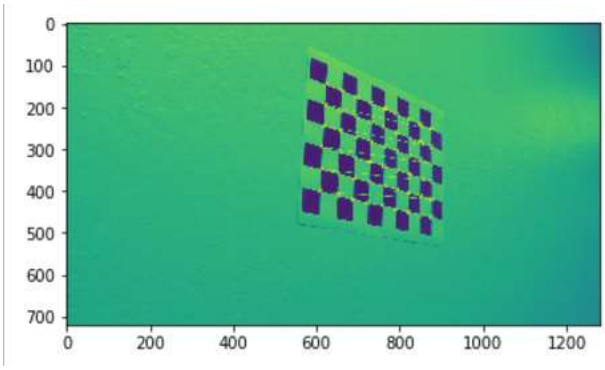
Camera Calibration

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

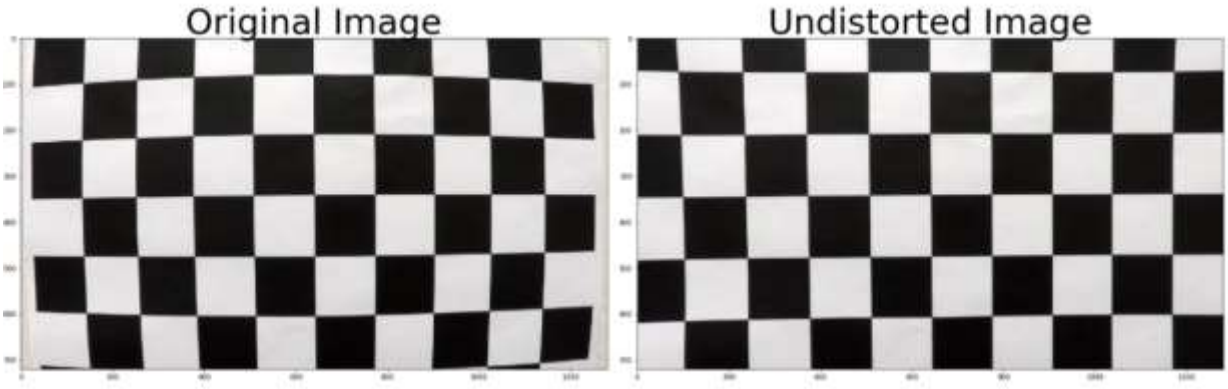
The code for this step is contained in the first code cell of the IPython notebook located in "./Adv_lane_finding.ipynb".

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.

Visualization of corners connectivity lines



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

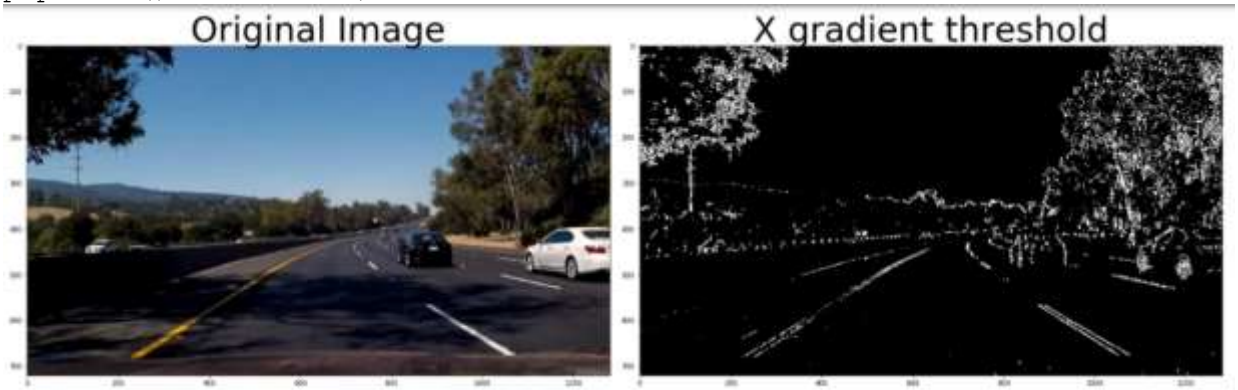
1. Provide an example of a distortion-corrected image.

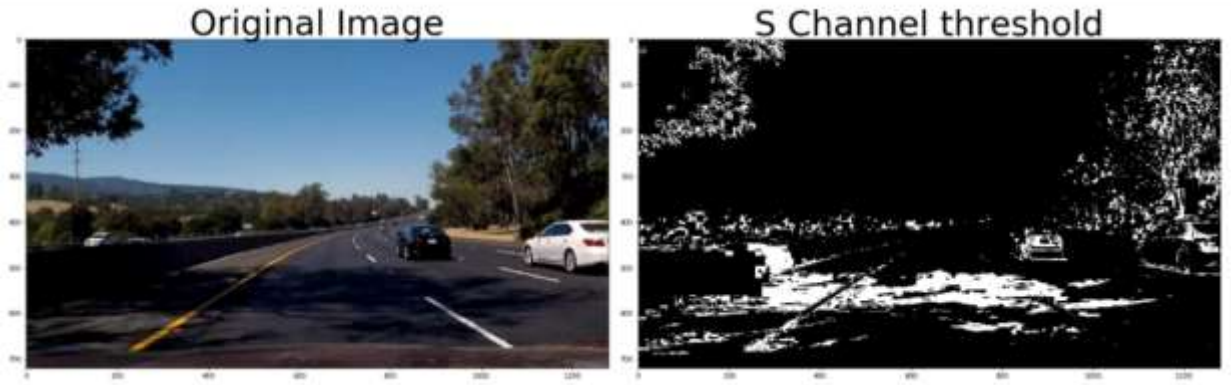
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

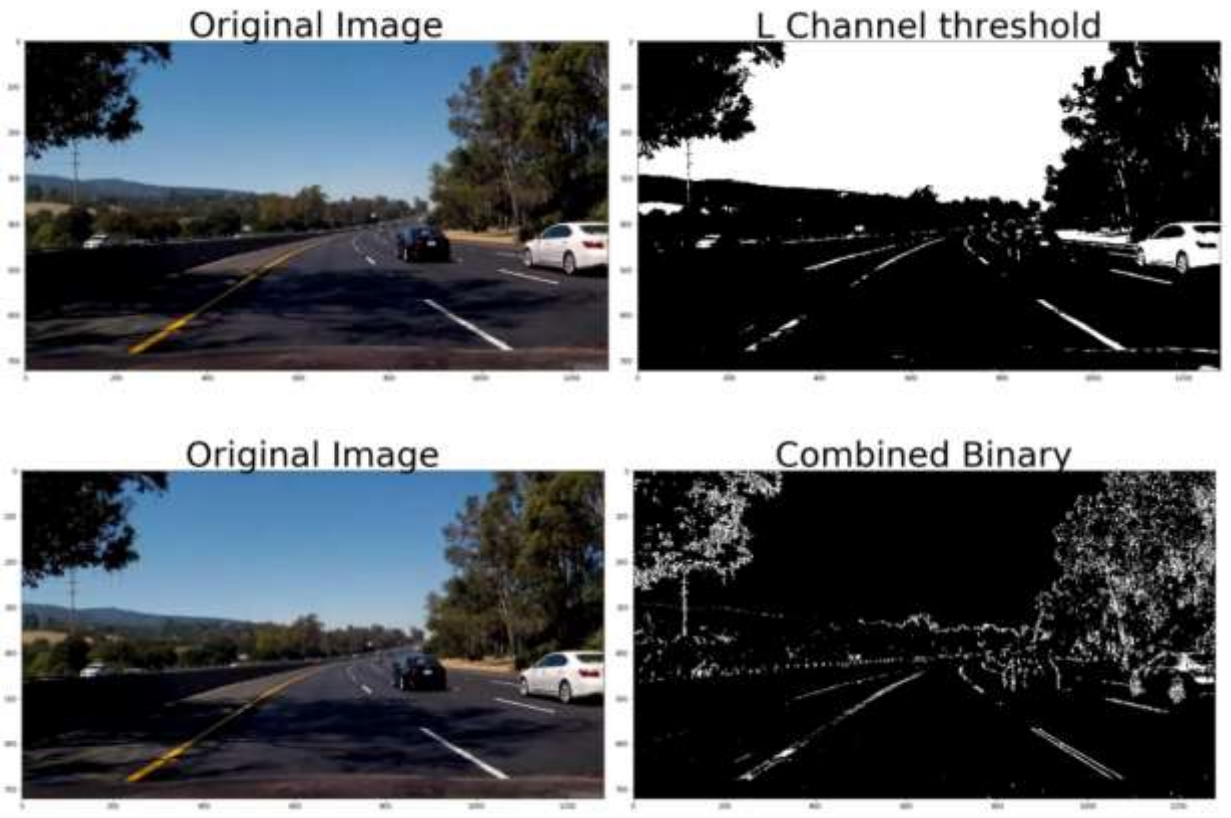
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps in function pipeline() in cell 6). Here's an example of my output for this step. (note: this is not actually from one of the test images. I captured this image from the video and used it instead. It contains shadow and was difficult to extract the lane line properly)

Initially I used s channel threshold and gradient threshold. With this combination lane line were detected for the test images. But if the image contains shadows in the middle of the lane and it was causing issues in identifying lanes. Then I added l channel threshold (ANDed with s channel binary) to remove shadows in the image. (Please see comments in function pipeline() in cell 39)





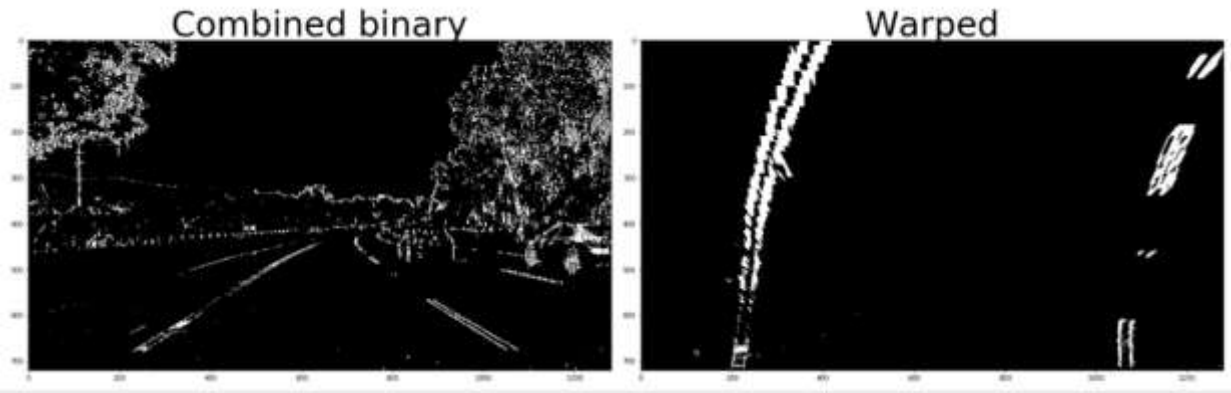


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `warp` in cell 8. The `warp()` 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:

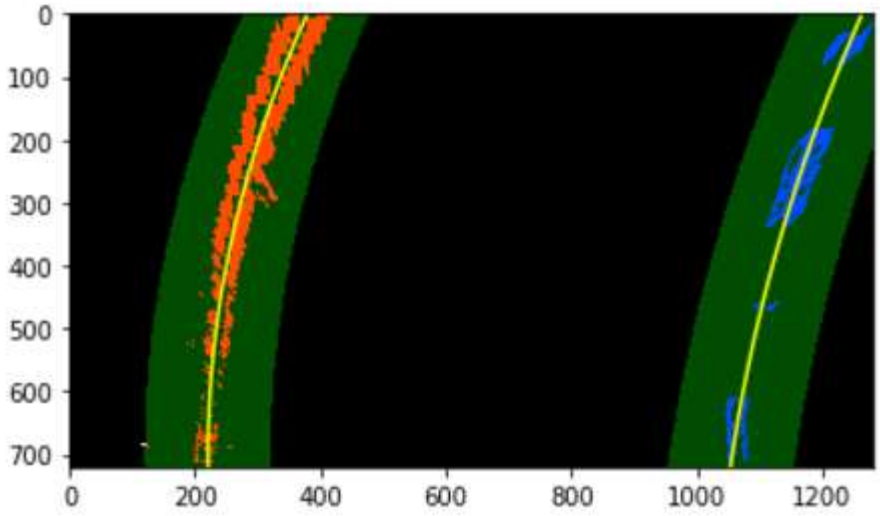
```
 # For source points I'm grabbing the outer four detected corners
src = np.float32([[701, 460], #top right
                  [1042, 675], #bottom right
                  [255, 675], #bottom left
                  [573, 460]]) #top left

# For destination points, I'm arbitrarily choosing some points to be
# a nice fit for displaying our warped result
# again, not exact, but close enough for our purposes
dst = np.float32([[1042, 0], #top right
                  [1042, 712], #bottom right
                  [222, 712], #bottom left
                  [222, 0]]) #top left
```

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?

I used the code given in the tutorial and created function get_poly() (in cell 13) and update_poly() (cell 15).
get_poly() – uses histogram taken in the bottom quarter of image to identify start of lane. Then image is scanned along the sliding window by splitting the image height into 9. The lane pixels are identified within the sliding window. Once the lane pixels are identified, polynomial fit is obtained.
Update_poly() – Searches for lane pixels around the polynomial identified before. Polynomial fit is obtained again using new lane pixels.

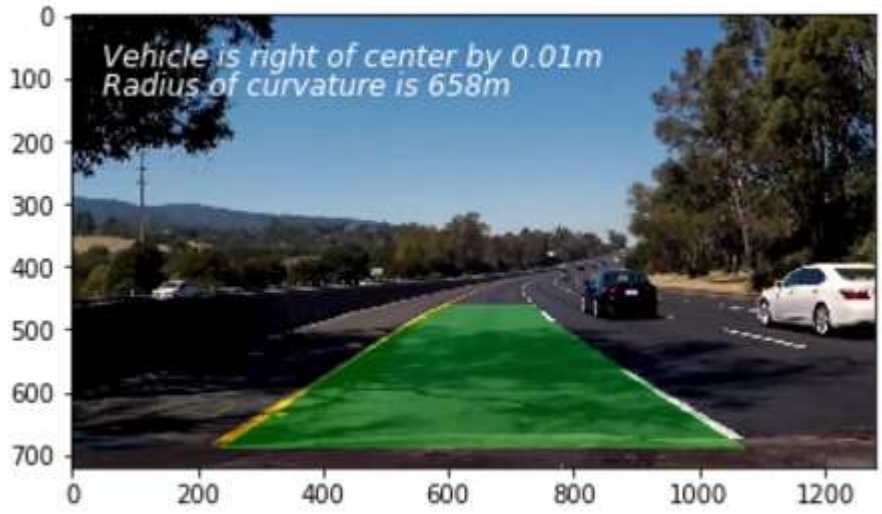


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

Radius of curvature is computed in method update_radius_of_curv() of Line class (in cell 12). Position of the vehicle is calculated within annotate_lines() (in cell 14).

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

I implemented this step in the function annotate_lanes(). Here is an example of my result on a test image:



Pipeline (video)

Here's a [link to my video result](./out_video.mp4)

Discussion

To eliminate odd frames with incorrect detection of lane lines, I used sanity_check() (in cell 16) to detect if the difference in percentage of radius of curvature between previous detected lane lines and current lane lines to be less than 50%. If the difference is high, then polynomial is calculated fully by doing sliding window(get_poly()) otherwise it is calculated by just looking lane pixels around the previously detected polynomial line(update_poly()). This check is done in the function process_vid() (in cell 17).

In order to avoid wobbly lane lines, I averaged the last 10 polynomial coefficient and used that to draw lane lines. Averaging is done in the method update_fit_q_and_best_fit() (in cell 12) of Line class.

Further Improvements

I didn't use direction threshold, it could be used to avoid any unnecessary lines being detected. Also additional sanity check can be

done by checking the distance between lane lines to be within certain limit.