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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

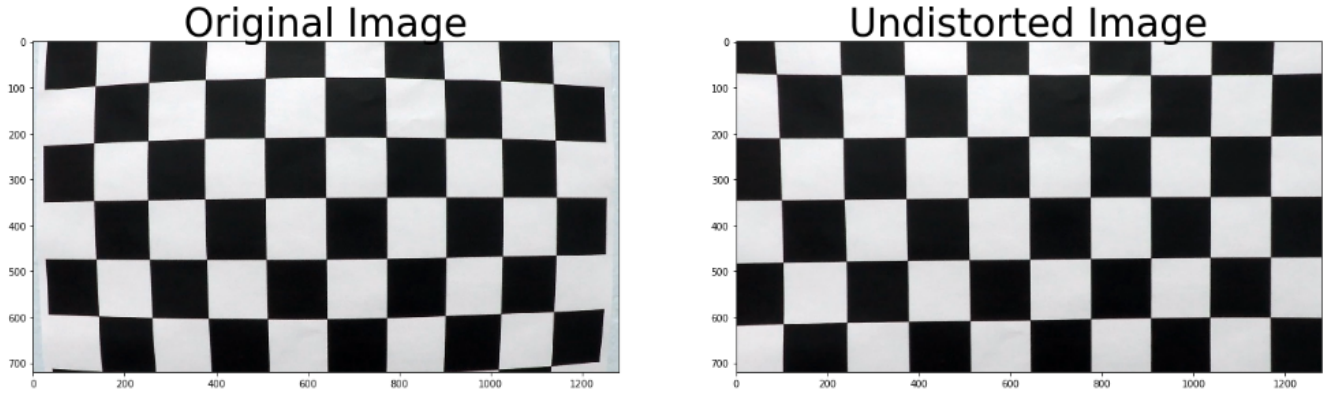
Camera Calibration

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 second code cell of the IPython notebook followed by the dependencies setup in the code cell 1. The procedure followed is same as what was discussed in the lecture with modified nx and ny (chess board corners as per the calibration images, nx=9 and ny = 8). 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 we 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.

Not all images returned the required 9x6 corners. Few images like calibration1 haven't detected all the corners. However, camera calibration is done with the images that were able to identify corners (relevant images were printed in code cell. The camera calibration and distortion coefficients are calculates using the **cv2.calibrateCamera()** function. The distortion correction to the test image (calibration1) 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. Note the distortion corrected image has car hood change (particularly at the edges and the change in position of cars at the edges)

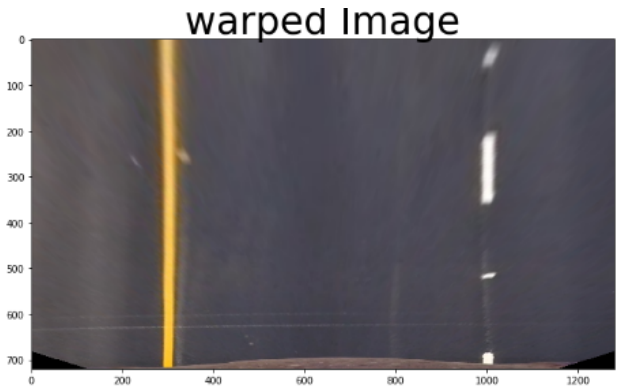
2. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called ***perspective*** which appears in the 5th code cell of the IPython notebook. ***The image has to be undistorted before picking the source points***. The source points are manually picked from a known straight lines image (straight_lines1) which is expected to be a rectangle from birds-eye view. The destination points map to a true rectangle. The destination coordinates are chosen such that the lane occupies most of the image with the lane center maps closely to image center.

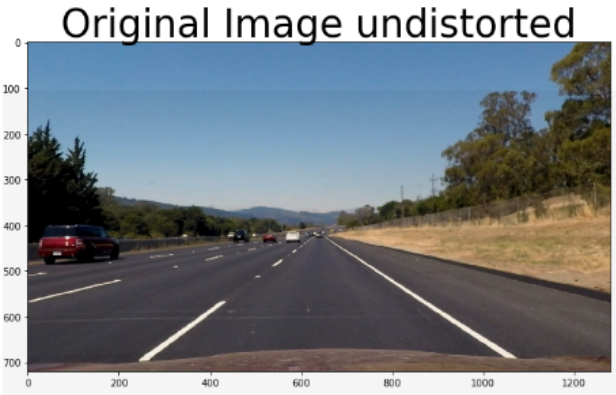
```
bottom_left = (265,680)
bottom_right = (1040,680)
top_left = [525,500]
top_right = [762, 500]

top_left_dst = [300,300]
top_right_dst = [1000,300]
bottom_right_dst = [1000,700]
bottom_left_dst = [300,700]

src = np.float32([[top_left, top_right, bottom_right, bottom_left]])
dst = np.float32([[top_left_dst, top_right_dst, bottom_right_dst, bottom_left_dst]])
```

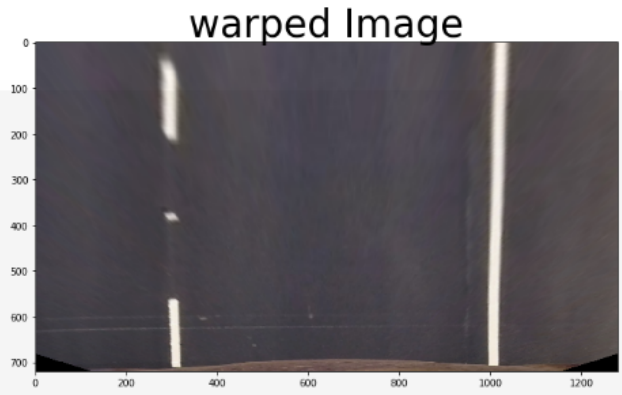
With the src and dst points, the perspective transform matrix is calculated
M = cv2.getPerspectiveTransform(src, dst).
Similarly inverse perspective transform is also calculated here, which would be used later in the code
Minv = cv2.getPerspectiveTransform(dst, src)

The perspective transform calculated works as expected; confirmed by the two straight lines test examples as below: Warped image shows parallel lines.



The perspective transform calculated in the above image is directly applied to straight_lines2 image, which also shows that the lane lines are paralle. Samething is confirmed with other test images.





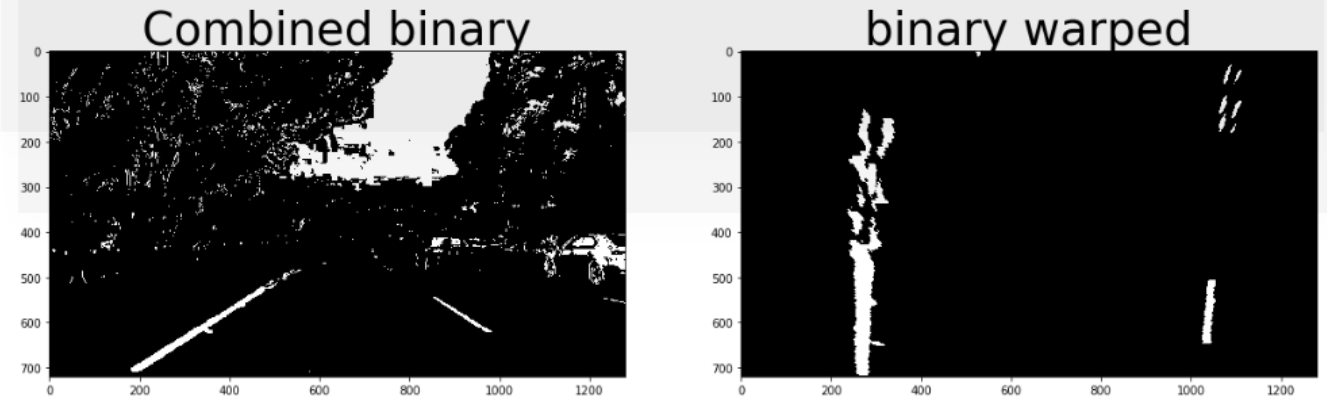
3. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image. Used the test images as a test set, the thresholds are modified to have cleaner lane marks. Initially, threshold (s-channel) and sx-binary (l-channel) from hls were used as discussed in the lecture. I have tried different thresholds, and I looked at

the udacity Q&A session on youtube where v-channel (from hsv) is used. I defined another threshold (v-channel). The logic and thresholds are chosen so that there are distinctive lane marks with minimal noise for all test images.

A function ***pipeline*** is defined with the color channels and thresholds as defined in the code cell 7. The logic for combined binary is:
(s_threshold & v_threshold) | (sx_threshold)

The pipeline function takes an undistorted image and results in a binary image. The relevant binary warped image is also shown here.

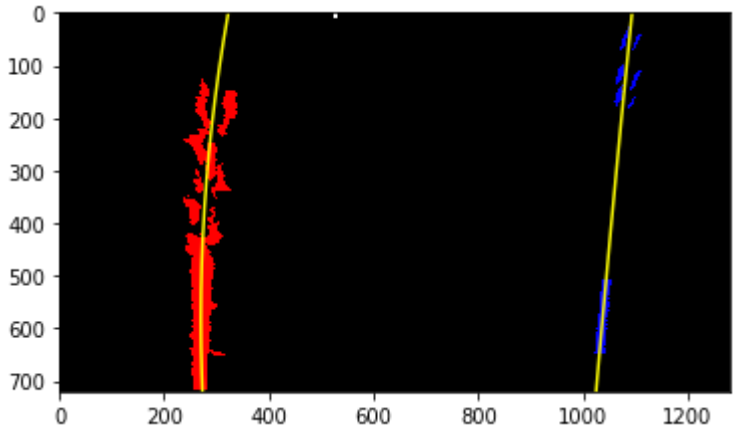


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

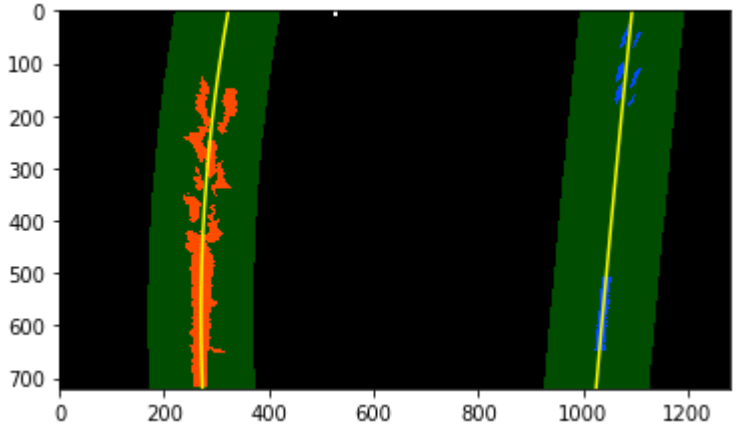
Finding lane pixels and fitting their positions with a second order polynomial was done in the code cell 9. The code is adapted from the lecture

After applying calibration, thresholding, and a perspective transform to a road image, resulted in a binary image where the lane lines stand out clearly. However, we need to decide explicitly which pixels are part of the lines and which belong to the left line and which belong to the right line. The two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. This is used as a starting point for where to search for the lines. From that point, a sliding window approach I used , placed around the line centers, to find and follow the lines up to the top of the frame.

The x and y positions of the pixels found in the window are extracted to be with a second order polynomial. This approach is applied to a test image, and the resultant image is as follows:



An image to show the selection window (in green overlapped to the above image)



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.

The radius of curvature is calculated in the code cell 12. Pixel to real world(m) conversion based on the lane width (x-direction) and lane length (y-direction). Radius of curvature is evaluated at the bottom of the image separately for the left and right lane lines. In order to do that, new second degree polynomials fits to x, y in real world space are done. The radius of curvature is determined for a fit:

$$f(y) = Ay^2 + By + C$$

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

The position of the vehicle with respect to the center of the lane is calculated by evaluating the x-coordinates at the bottom of the image for the left and right lanes separately. The mean of these coordinates gives the position of the center of the lane (also evaluated in the code cell 12)
The car position is at the center of the image, and the difference car position and lane center gives the position of the car with respect to the lane center.

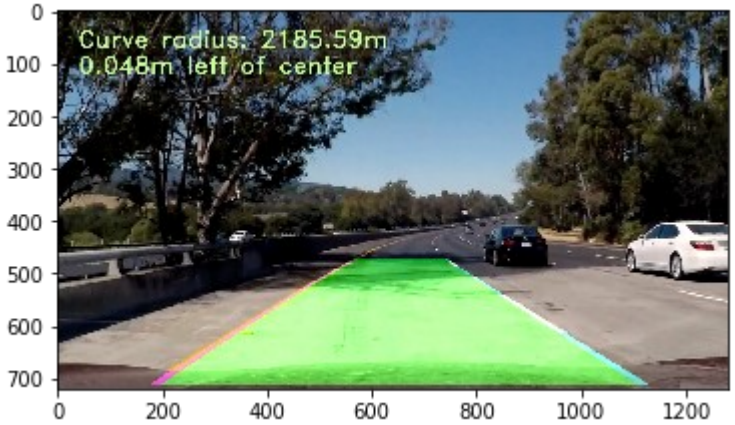
These are the results for the above image:
Radius of curvature for example: (l) 271.14 m, (r) 4100.03 m
Distance from lane center for example: -0.048 m

The radius of the curvature is the mean of the left and right lane lines, and it is 2185.59m:

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

draw_lane and draw_data functions are defined in the code cells 14 and 16 to clearly identify lane line with radius of curvature and the car-position clearly identified on the image:

Here is an example of my result on a test image:



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!).

The video is attached, and it performs really well on the project video. The entire pipeline is run through the process_video function in Code cell 19. Added an if statement to do sliding window approach if the current_fit is none else you don't need to do a blind search again, but instead you can just search in a margin around the previous line position like discussed in the lecture (implemented here)

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?

I faced two key issues. First issue in the implementation of color transforms and gradients to create a thresholded binary image robust for all test images and the video. Initially, I used HLS only as discussed in the lecture; later I included HSV to increase the robustness, and it performed well on all images and project video.

Second issue in making it work for the video. I used line class as discussed in the lecture. If the recent fit is none, sliding window approach is used. In cases where there there is a fit, a margin around the previous line position is used to determine pixels that belong to left and right lanes. If none are found, mean of the recent fits are used to determine left and right fits respectively. My video failed to run 100% because of lack of implementation like above. Eventually, it was implemented and it worked great.

However, it failed for challenge videos, as shown below.

Challenge video snapshot

In future, I would like to do health check for each fit to ensure that the fits don’t deviate too much. If one of them does, keep the other fit and make use of lane width in determining the other fit so that it won’t fail like above.