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

Writeup and Project files

This is the project writeup document. And other important files are:

1. The IPython notebook, which has all the code:
   https://github.com/khushn/carnd_p4/blob/master/advanced_lane_finding.ipynb

2. The output images folder:
   https://github.com/khushn/carnd_p4/tree/master/output_images

3. The lane marked output video:
   https://github.com/khushn/carnd_p4/blob/master/video_lane_marked.mp4

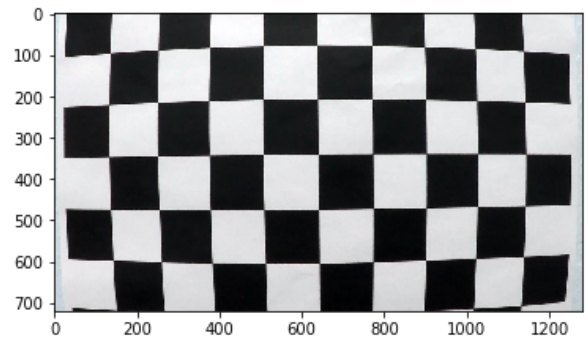
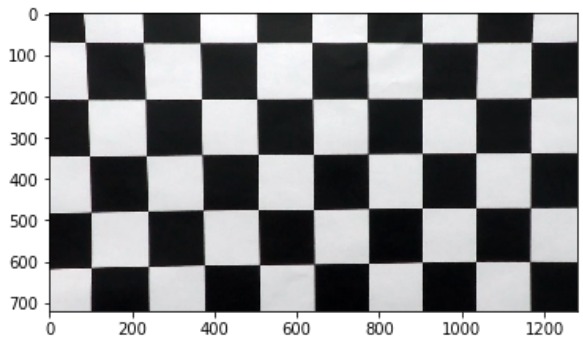
4. The lane marked first challenge video:
   https://github.com/khushn/carnd_p4/blob/master/challenge_lane_marked.mp4

Camera Calibration

The code for this step is contained in the first code cell of the IPython notebook referred above. After preparing the object points I got the camera calibration matrix and coefficients.

Then used them to undistort the images using `cv2.undistort`

The code for the same is in 3rd cell of the notebook.

Original Image	Undistorted image
	

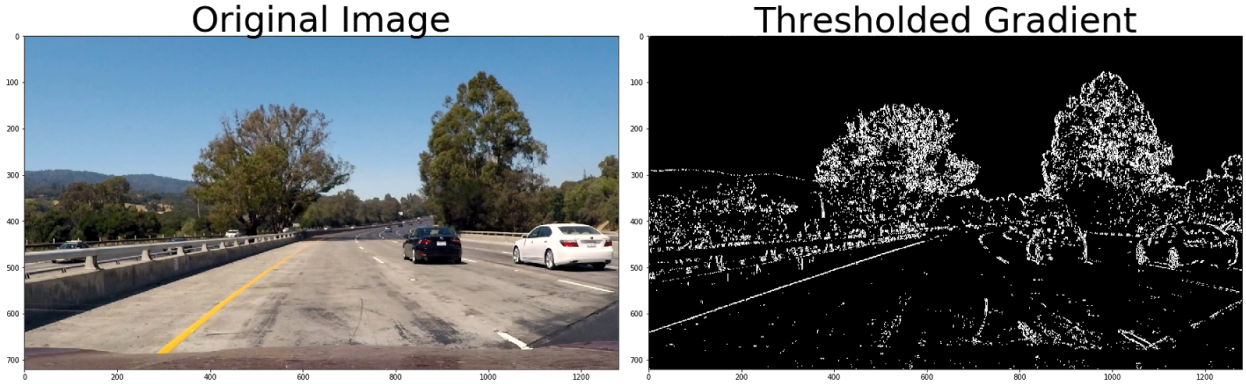
Pipeline (test images)

Distortion-corrected image

Using the same technique as used to undistort the chessboard, I use it to undistort a camera image of the road. The code for that is in 4th Cell of the notebook. Images below:

Gradient and S-Binary Transforms

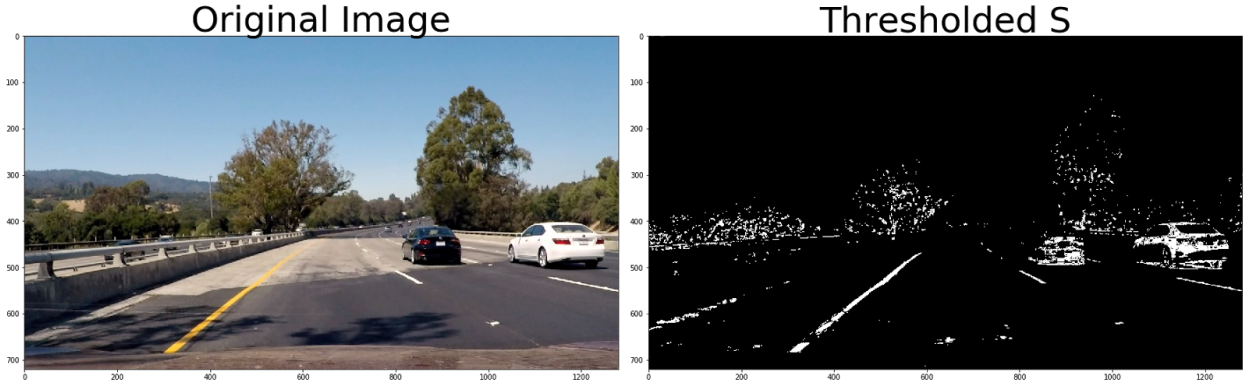
First, I used a Gradient transform, along x-axis using Sobel (code in 5th cell). Samples below:



In the above we can see that it can catch the white lines on the right well, but misses the yellow on the left.

Thresholded S using HLS transform

Then I applied the HLS transform on the same image, and picked up the S channel. And applied a threshold (code in the next cell). Sample below:



So now we can see that it picks the left yellow line, in an excellent way.

Combined Threshold

So, I combined the above two, using an OR condition (code in following cell of IPython notebook). To get a better result for both left and right lane lines.

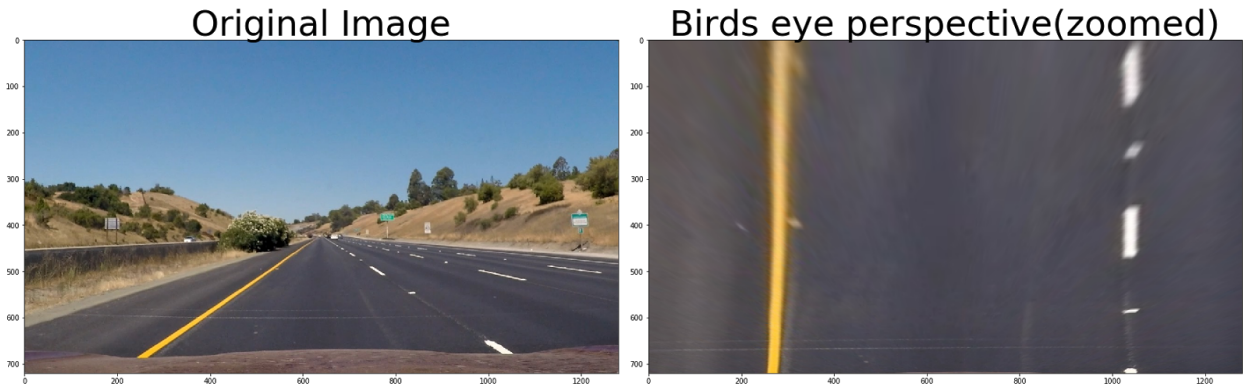
So now we can see this can pick up the left and right lane lines (yellow as well as white). So we use this technique of filtering the images in this project.

Perspective transform

As taught in the class for the project, we need to get the bird's eye view of the lane lines (road). So that we can do the math of lane finding using sliding windows etc. in a better way.
I used `cv2.getPerspectiveTransform()` and `cv2.warpPerspective()` functions for the same. The source and destination points are given in the table below:

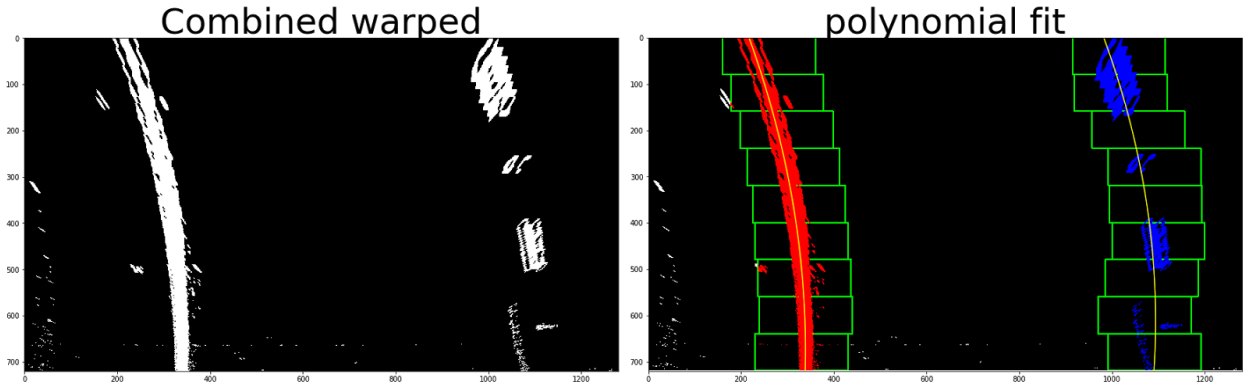
Source point(x, y)	Destination point(x, y)
578, 460	261, 10
700,460	1036, 10
1036, 678	1036, 720
261, 678	261, 720

Below is the sample:



Identifying lane line pixels and fitting with a polynomial

For doing this, I reused the code provided in the class, which uses a sliding window algorithm, on a frequency histogram of white pixels. These seems to work well, as shown below, on a curvy lane test image.

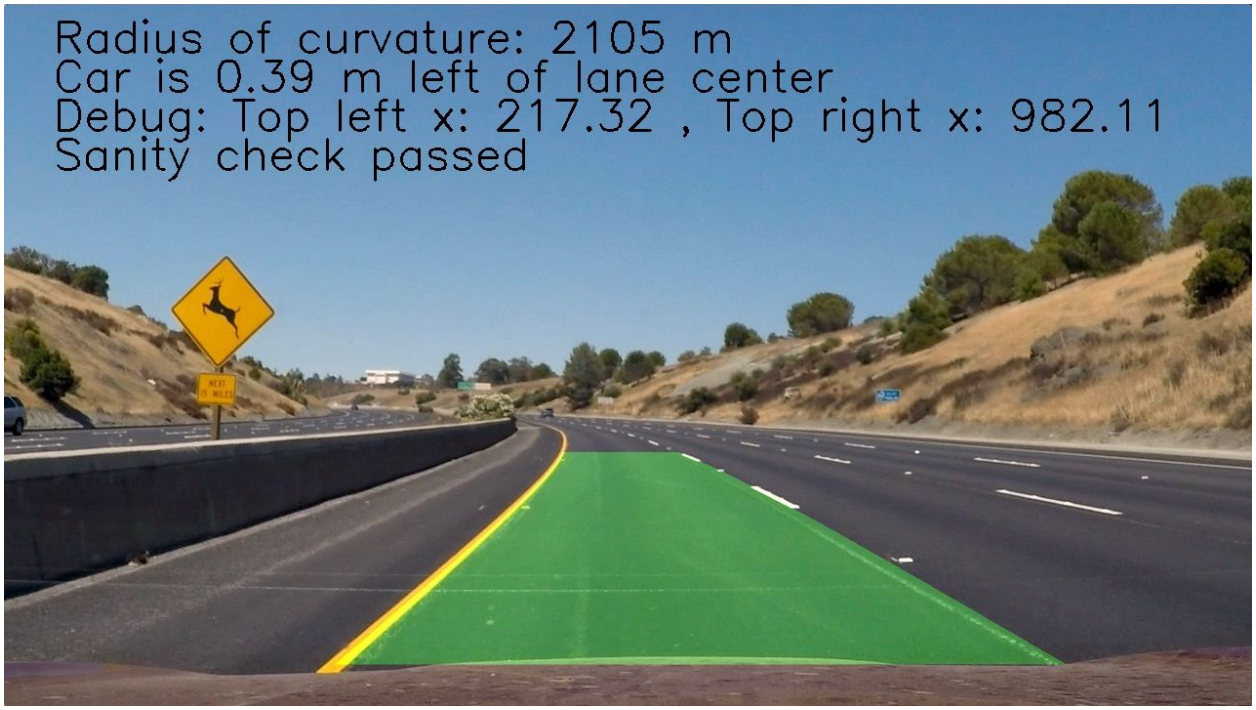


Radius of curvature

I then calculated the curvature on the polynomial fit function `radius_curvature()` in the notebook. Sample output image, having curvature shown below.

Output images

I tried the pipeline with the test images. All the images barring test image 4, worked well, and showed proper lane marked area.
Sample:



All the lane marked images are present in output images folder, here:
https://github.com/khushn/carnd_p4/tree/master/output_images

Pipeline (videos)

The link to the lane marked video is below:
https://github.com/khushn/carnd_p4/blob/master/video_lane_marked.mp4

I attempted the challenge videos as well. The first one, came out okayish. While the second one is quite embarrassing to include in this report(however, I have some ideas, on how to do it, which is discussed in Discussion section below).
Link to the first one (lane marked):
https://github.com/khushn/carnd_p4/blob/master/challenge_lane_marked.mp4

Discussion

The main video of the project was simple to get lane marked well, using the techniques taught in the project lessons, and described above.

How I got the first challenge video working

There are two problems which make this one difficult. 1) Shadows and 2) Tar repair/crack line in the center of the lane.
The detection got edgy when it saw shadows, picking up random areas. For the tar line, it was getting either the right half or the left half of the lane.
To solve this I included a sanity_check() function. Code for which is included below for easy reference.

```
def sanity_check(left_l, right_l, yshape):

    if left_l.radius_of_curvature < 1000:
        #Radius of curvature diff for curvy lines should be within range
        # Note: We can't do the same for straight lines as they have high values and diff
        #       may not reflect reality
        diff = abs(left_l.radius_of_curvature - right_l.radius_of_curvature)
        if diff > 300:
            #print('radius of curvature diff: ', diff)
            return False

    # we check if lines are parallel enough
    xl0 = get_xval_for_quad_fn(left_l.current_fit, 0)
```

```
    xr0 = get_xval_for_quad_fn(right_l.current_fit, 0)
    top_diff_x = (xr0-xl0) * xm_per_pix
    xln = get_xval_for_quad_fn(left_l.current_fit, yshape-1)
    xrn = get_xval_for_quad_fn(right_l.current_fit, yshape-1)
    bot_diff_x = (xrn-xln) * xm_per_pix
    #print('top_diff_x: ', top_diff_x)
    #print('bot_diff_x: ', bot_diff_x)
    if abs(top_diff_x - bot_diff_x) > 1:
        #if top lane width and bottom lane width are greater than 1 meter
        # its not a good detection
        #print('diff b/w top lane width and bot lane width: ', abs(top_diff_x - bot_diff_x))
        return False
```

This applies, some basic sanity on the detected lane, like enough lane width on top and bottom. The regulation threshold i.e. lane width within 3 and 6 metres. Also radius of curvature of the left lane line and right line should be within some limit for curvy roads. One interesting thing I found was that for straight lines, it can vary a lot e.g. million on one side and few thousands on the other lane line.

Also I applied a low pass filter by averaging the sanity checked lane detections to previous N values (I kept N as 10). I used the Line class, as advised in the project instructions and made suitable modifications to it.

With this the challenge video worked much better compared to the initial attempt.

Harder challenge video

Unfortunately, the improvements applied for the challenge video were insufficient for the harder one. Its too curvy, too noisy (very contrasting lighting ranging from dark shadows, to very bright light) and to add to it, some glass reflections on the camera images.
How to make it work?
I have some ideas on how to make it work. The left hand side has sold yellow lines (two of them). From initial good detections, I can find out the lane width. Then I can just follow the solid yellow lines and use it as a strong filter for all the detections. Using this technique, I am optimistic that it should work better if not well. I make this note here, for future extension of this project.