Advanced Lane Finding Project

The 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. Create a thresholded binary image using color transforms, gradients, etc.
4. Apply a perspective transform on the binary image to get 'birds-eye view'
5. Detect lane pixels using sliding window approach and fit a polynomial to find the lane boundary.
6. Measure the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Display the lane boundaries and numerical estimation of lane curvature and offset.

Reference:
I used the techniques and code provided in the class lectures and quizzes.

Rubric Points

1. **Provide a Writeup / README**
   This is the project writeup!

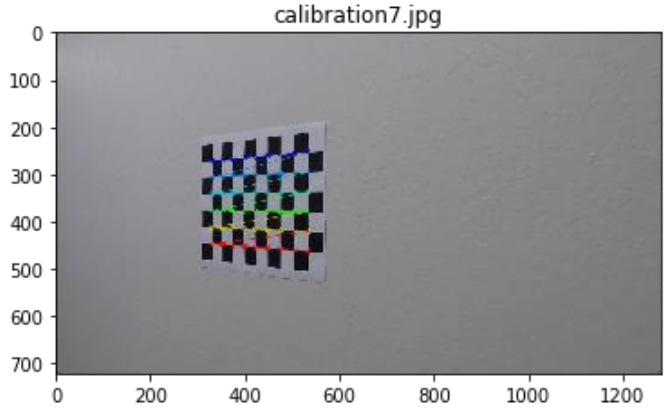
Camera Calibration

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

Implementation for this module is in 'calibrate_camera' function.

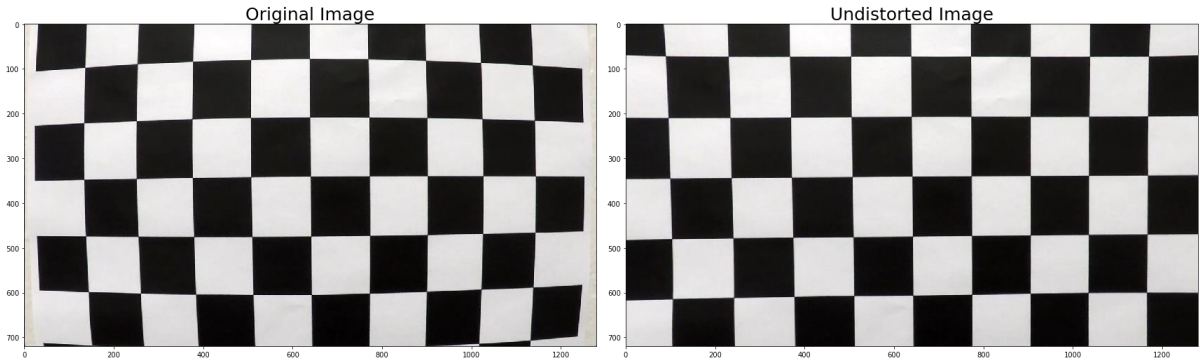
Here the real-world 3D points are referred as 'objpoints' and 2D points in plane are referred as 'imgpoints'. For 3D 'objpoints', z co-ordinate is always 0 as the chessboard is lying on a flat surface. I used 'findChessboardCorners ()' function of cv2 to find the corners in the chess board. I used 9 as the number of corners in x-axis and 6 as the number of corners in y-axis. A corner is defined as a point where 2 black and 2 white squares intersect. This (9,6) param didn't work in few images, as only a portion of the chessboard image is shown in those images. So, some corners didn't appear in the image. So I tried values like (9,5), (8,6), (8,5) etc... This worked, and I was able to find all the corners in the calibration images.

Then I used 'cv2.drawChessboardCorners' to draw the corners on the image.



Then I used ‘cv2.calibrateCamera ()’ and the detected ‘objpoints’ and ‘imgpoints’ to compute the Camera Matrix and distortion coefficients.
I also saved these values as pickle to load and use them in other functions.

Using the camera calibration matrix and distortion coefficients, I used ‘cv2.undistort ()’ function to correct the distortion in chessboard and test images.



Pipeline (test images)

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

Its not very clear to see the distortion correction on the test images, but it is clear on the chessboard images where there is a contrasting pattern.

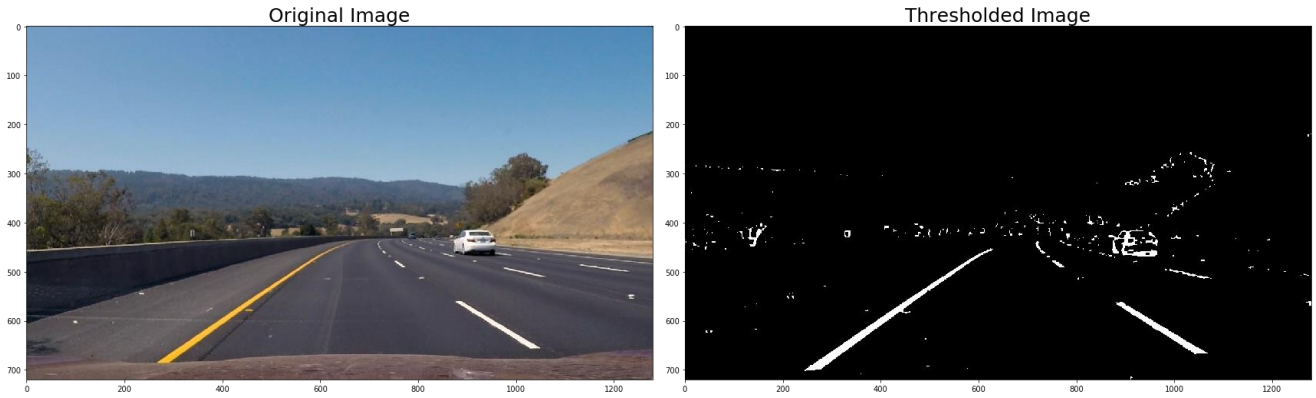
We can see that the sign board in this image has changed in size in the Undistorted Image.

4. 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

This was an important step as the thresholded binary image is the base for finding lane lines. Implementation is done in ‘apply_thresholding ()’ function.

Just like it was demonstrated in the class lectures, I used a threshold on V channel in HSV color space and then I used Sobel to get the x gradients, also applied a threshold on the gradients.

Then I combined the color threshold and gradient threshold images to get the binary thresholded image. I found that there were many gaps on the detected regions, so I used the ‘closing’ morphological operator to fill those gaps.



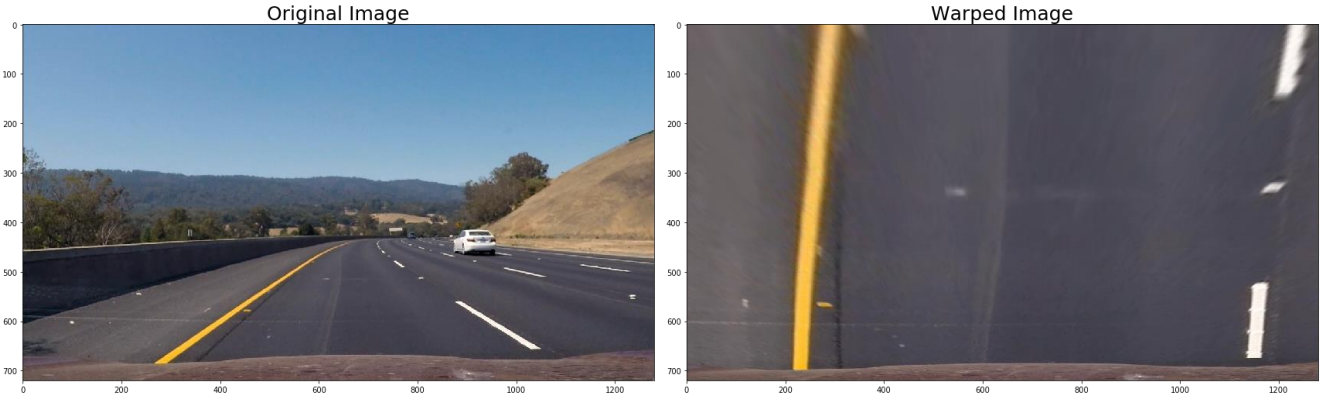
5. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

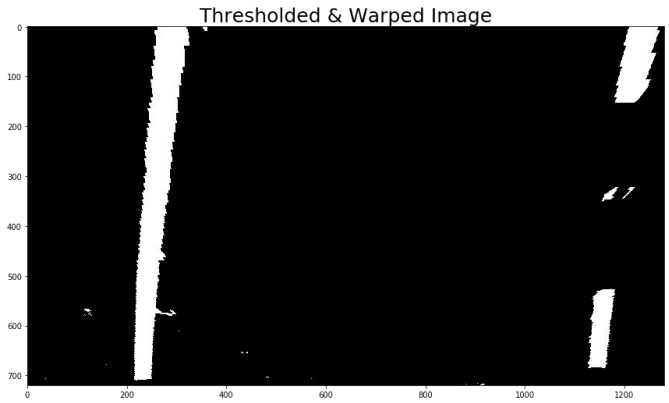
Implementation is given under section ‘**Apply Perspective Warp on thresholded images’.**

I defined a region of interest, where lane lines are located in the image, as ‘src’ points. I also defined the destination points where I expect these points to appear in the warped image. ROI is given below.

```
Source Points: [[  730.   480.]
 [ 1100.   720.]
 [  300.   720.]
 [  580.   480.]]
Destination Points: [[ 1100.     0.]
 [ 1100.   720.]
 [  300.   720.]
 [  300.     0.]]
```

I used ‘cv2.getPerspectiveTransform ()’ function to get the M and Minv matrices. Then I used the M matrix and ‘cv2.warpPerspective ()’ function to get the ‘birds-eye’ view of the original image. And then I applied the same function to get the warped thresholded image.



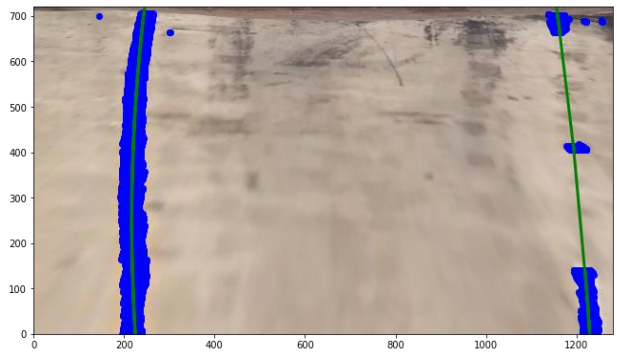


6. **Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?**

Two approaches were taught in the lectures, one using a sliding window and other using prior fits. I only used the sliding window approach.
Implementation is given in 'search_sliding_window ()' and 'find_lane_lines ()' functions.

In the sliding window approach, lane lines are found using an exhaustive search on every single frame. I took the histogram of the bottom half of the image to get the base points for left and right lanes. Then using a loop, I slid two windows starting at the base points to find non-zero pixels i.e. good left and good right indices.

Then I used the identified points to fit a second degree polynomial using Numpy's polyfit() function.



7. **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.**

Implementation is given in 'find_lane_lines()' function. I implemented the equation, to find the radius of curvature, as shown in the lectures. By using the left lane and right lane points at the bottom of the image and the image center along x-axis, I found the offset (deviation) of the vehicle from the lane center. I used the constants provided in the lectures to convert from pixel space to meters. The output is shown in the above image.

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

Pipeline (video)

9. **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!)**

I implemented a **Line** class, added a pipeline and put all the necessary code blocks within. I also did a sanity check on the curvature of the lane lines. I also averaged the fitted lines (the recent 10 fits), to get a smoother fit. I also applied sanity checks to check the detected lane width and adjust the lane fits accordingly.

Link to output video: https://youtu.be/PAP6v4AD330

Discussion

10. **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?**

These are the improvements I want to make.

1. Use ‘search from prior’ technique. The current approach searches exhaustively on every single frame, so it takes a few minutes to process the entire video. I believe that by searching around prior fits, the processing time will be reduced.
2. Better thresholding using other color spaces, histogram equivalization to detect lane lines in darker (shadow) regions.
3. Just like in the first project, I used a hard-coded region of interest for the polygon. This could be avoided.
4. Crack the challenge portion of this project!