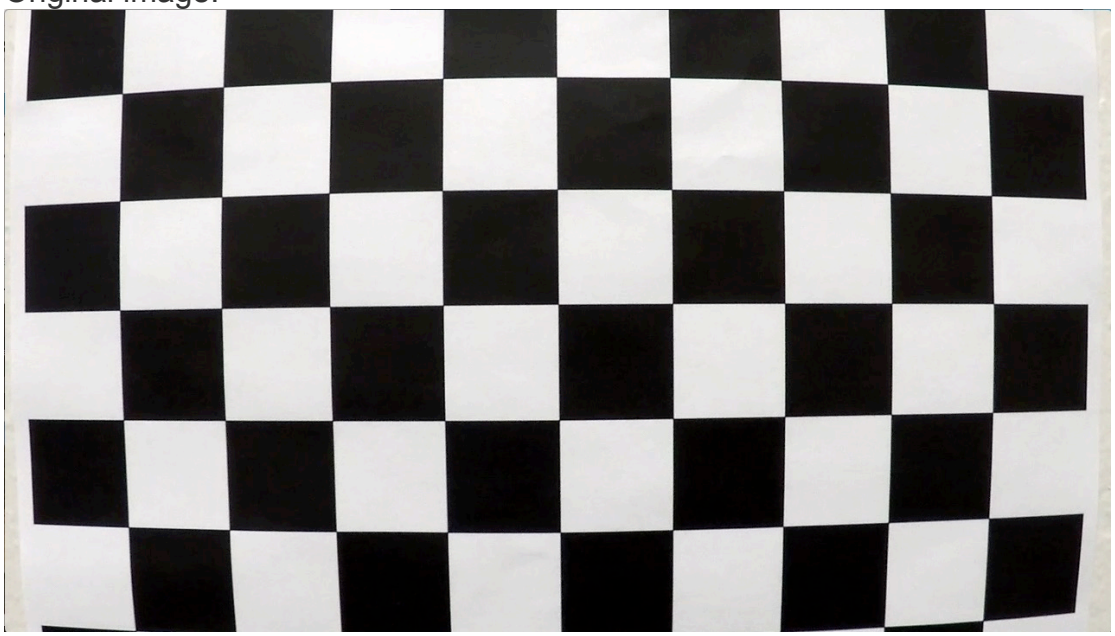
Camera Calibration

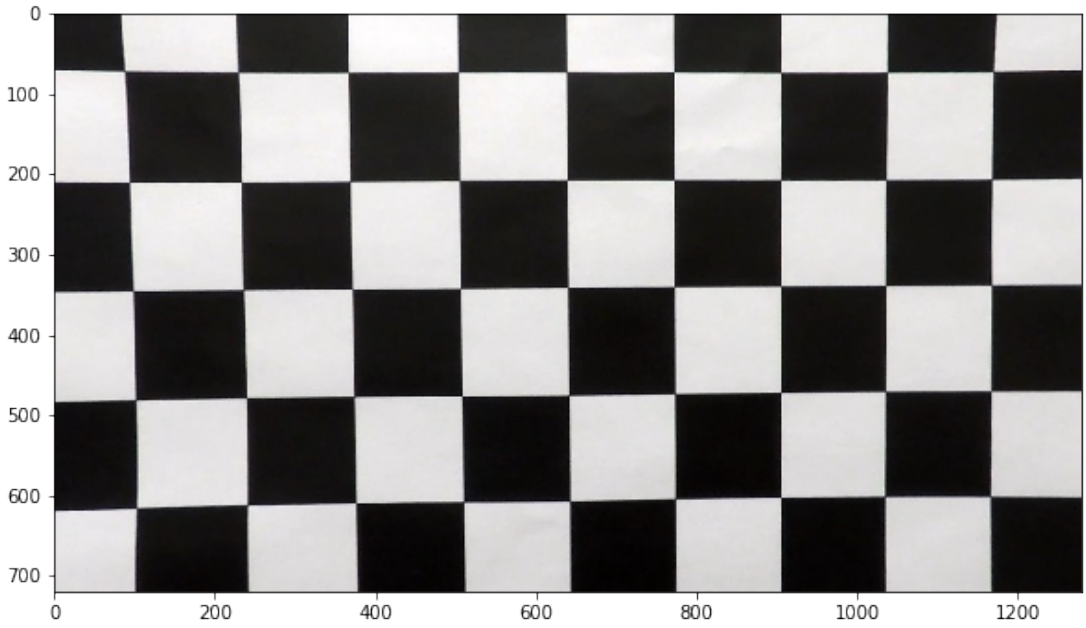
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the first code cell of the IPython notebook located in "p4.ipynb"
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. 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:

Original image:



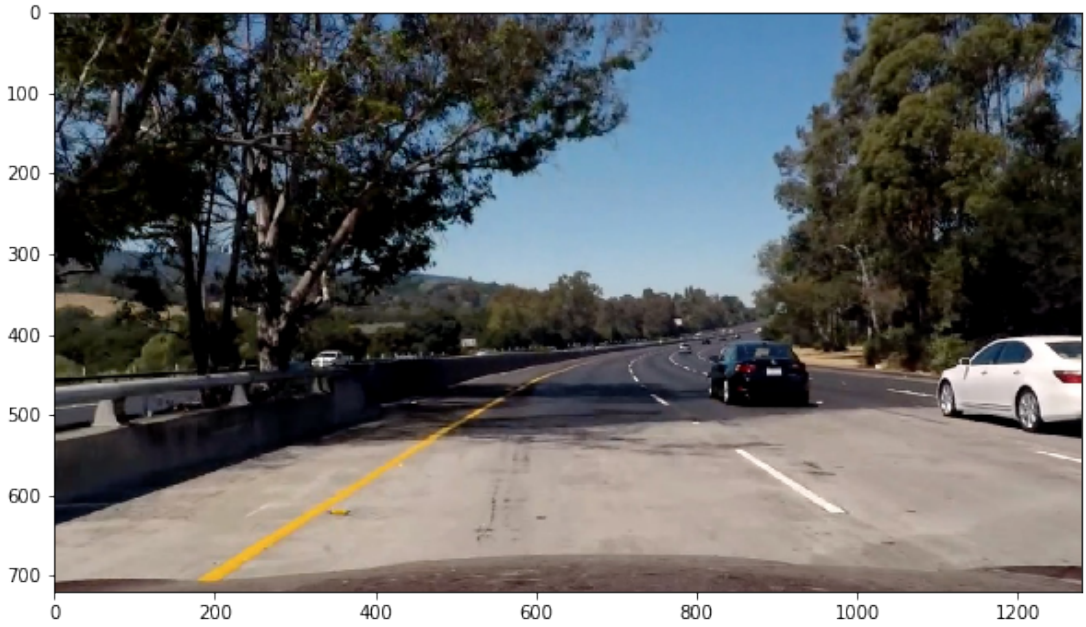
Undistorted image:



Pipeline (single images)

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

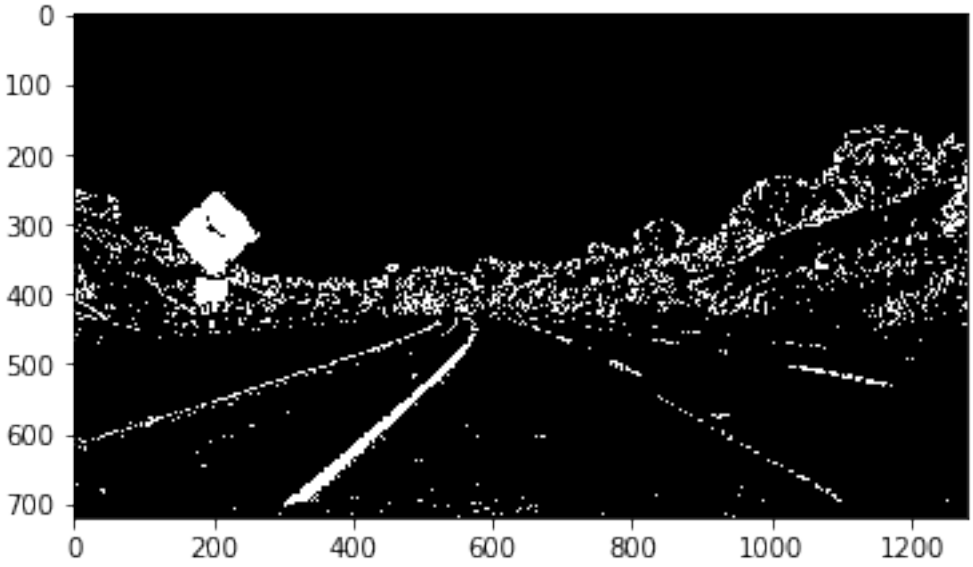
To demonstrate this step, I will describe how I apply the distortion correction to one of the t



est images like this one:

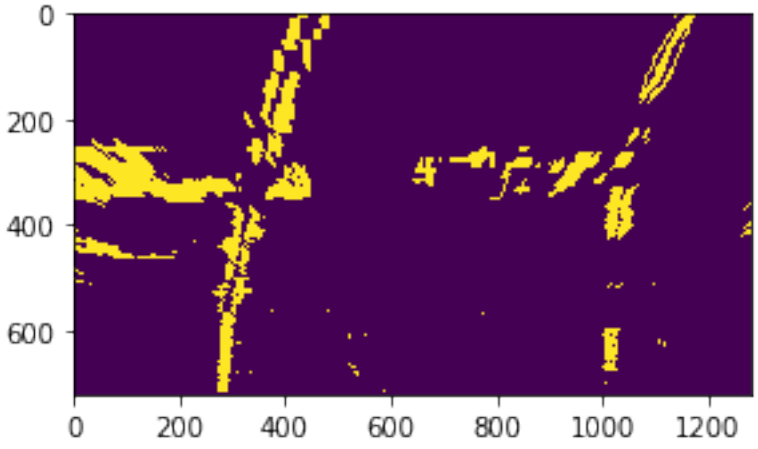
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image You can find the code in the second cell in p4.ipynb. 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 get_perspective_transform(), which appears in cell 5.
The get_perspective_transform () function takes as inputs an image (img) I chose the hardcode the 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?

Then I did some other stuff and fit my lane lines with a 2nd order polynomial. You can find the code in cell 8.

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 in cell 11 between lines 9 and 28.
I fitted a second order polynomial to pixel positions in each lane line. Then I chose the maximum y-value, corresponding to the bottom of the image. Then I made the calculation in real world space.

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 cell 10. 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!).

You will find the output video in the attached folder

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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

I tried to keep it simple. I mainly used the advices from the class.

I hardcoded the source and destination points on a trial and error method until I got the desired results.

After that I tried on test_images, and after I got good results, I changed my code to fit the video: I implemented the Line class and I improved the code by introducing the sanity check and by reusing the data found in previous frames.

I hardcoded the source and destionation points in the get_perspective_transform.

I would improve it by calculating those points based on the edges. I should be able to find the top of the lane. In some frames the brightness is so strong that the lines is not visible in some points(even for my eyes).

I would definitely add more sanity checks. I hardcoded the distance between the lines that would pass my sanity check. But I should calculate that based on previous frames.