Writeup / README

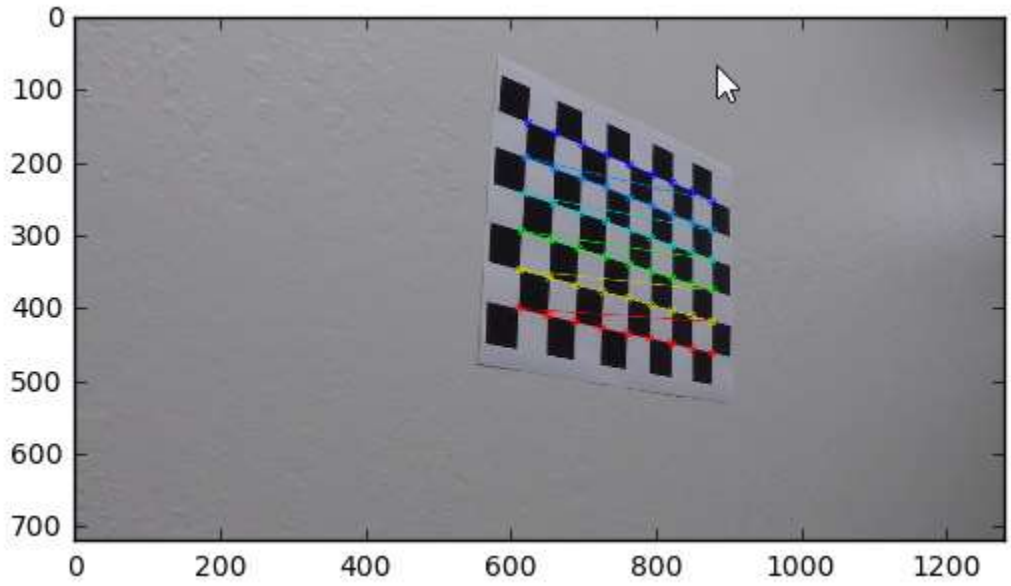
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

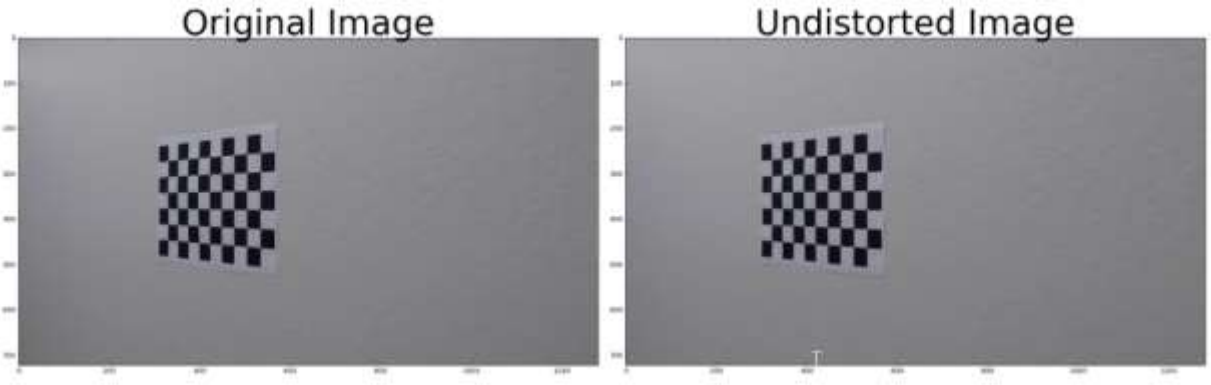
Camera Calibration

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

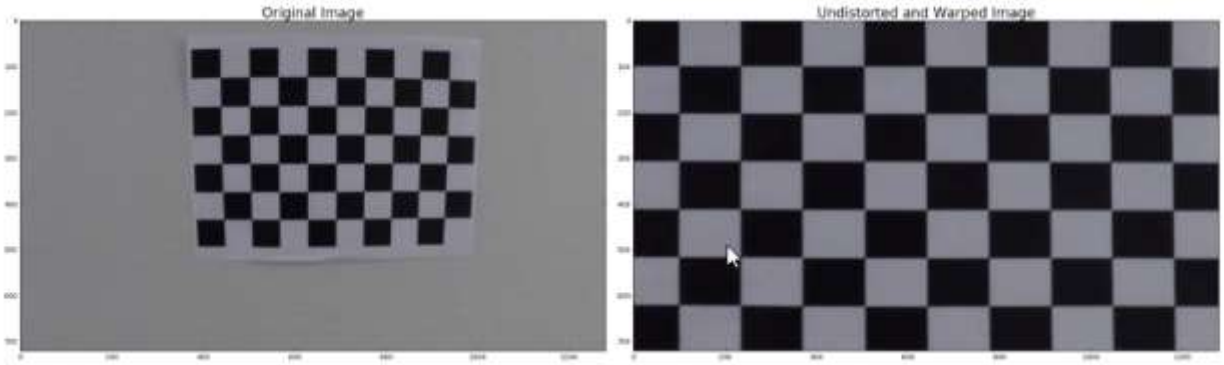
The code for this step is in code.ipynb .
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `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.



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:



After undistorting the image I used perspective Transform to to warp the image. The results obtained are as below:



Pipeline (single images)

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

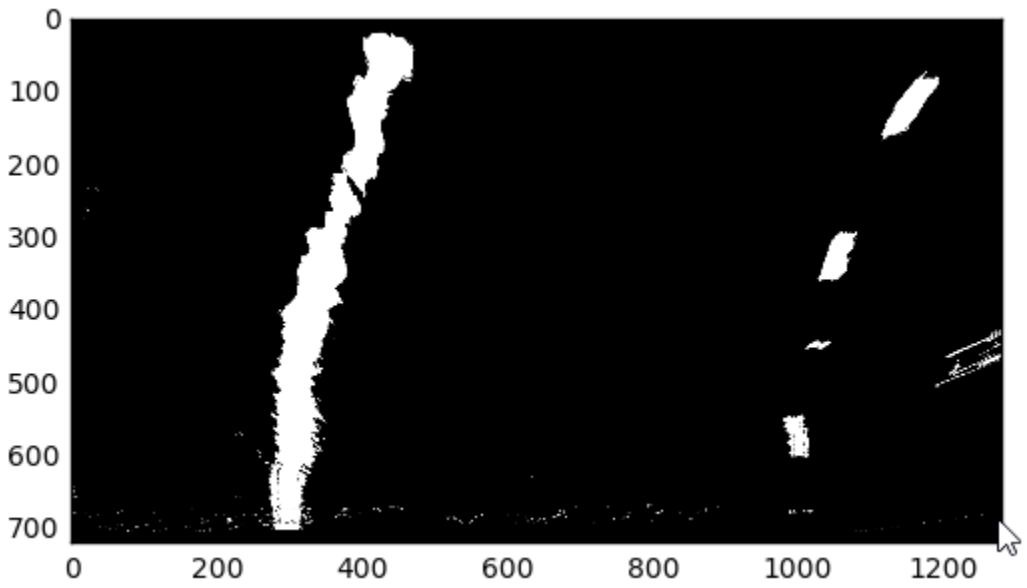
I used the objpoints and imgpoints matrix obtained from the chessboard calibration to calibrate the camera for the car images as well as undistort it. I did this on all straight line images as an example.

The results are as below:

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.

In final_code.ipynb, box 7, I have implemented an img_process function that combines a gradient as well as color threshold to give an optimum image with the lane lines. This function also undistorts the image as well as performs a perspective transform.
In img_process, I have implemented a pipeline function that calculates a gradient thresholding of the image using Sobel and also converts the image to HLS ans uses the saturation component to map the lane lines

An example of the output image from this function is shown below: This image is `test_images\test4.jpg`

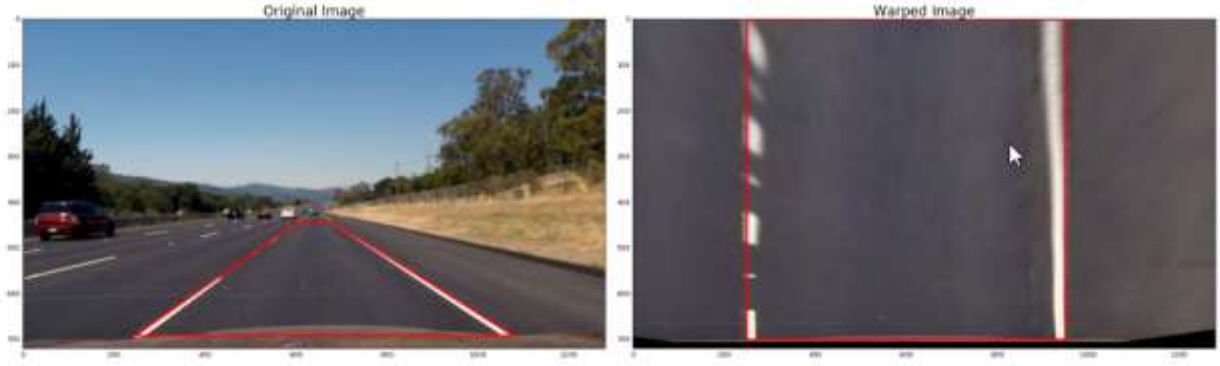


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

For perspective transform, I got four corner points from the lane image as my src and considered a destination rectangle of shape [250,0] , [950,0] , [950,700] , [250,700]. Then I used opencv function of getPerspectiveTransform to get the transform matrix and used warpPerspective function of opencv to warp the image

topL=[602.145, 443.433] topR=[681.948, 443.433] bottomL=[247.022, 691.62]
bottomR=[1076.97, 691.62]
src = np.float32([topL, topR, bottomR, bottomL])
offset = 200
img_size = (undist.shape[1], undist.shape[0])
dst1=np.float32([[250,0],[950,0],[950,700],[250,700]])

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.

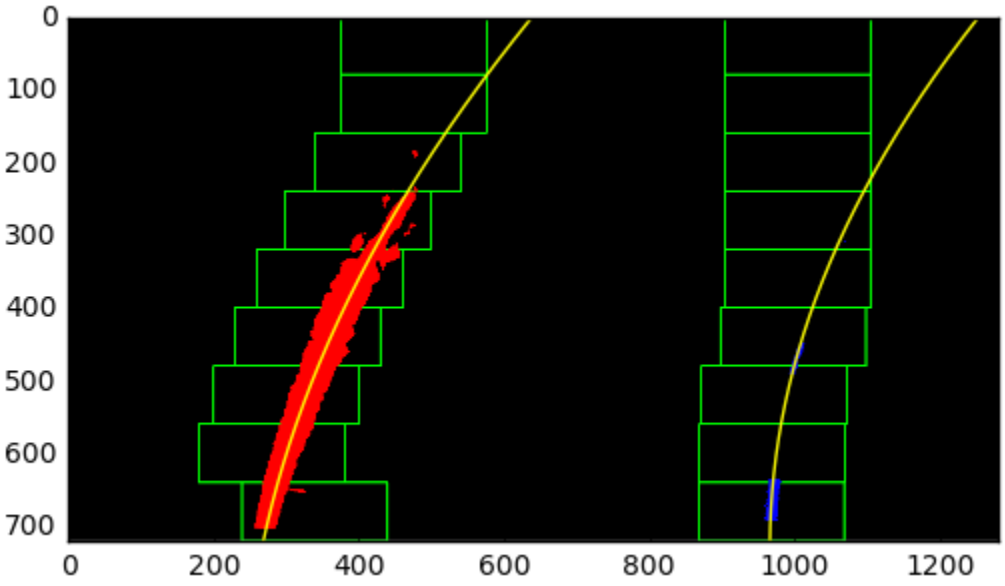


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The code is located in both final_code.ipynb and code.ipynb (box 304) I used a sliding window method to locate the lane lines. First, I took a histogram of the bottom half of the image and determined the base of the lane lines depending on the location of the peaks

Using 9 sliding windows, I stepped through each window close to the midpoint of the previous window to place the next one and locate the non zero points within the windows i.e the lane line.

At the end, I fit a 2nd order polynomial through the left and right lane to predict the lane line curves



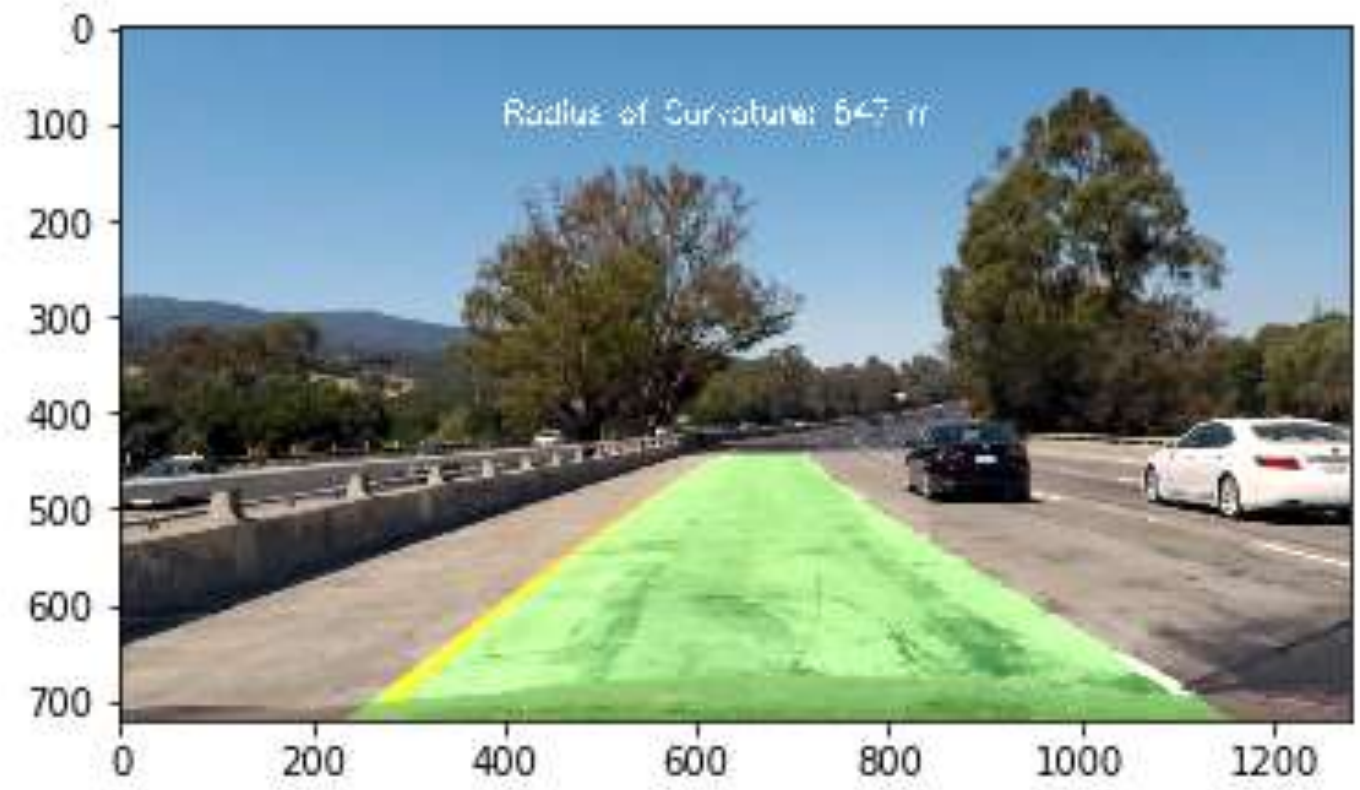
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I calculated the radius of curvature at the end of my video pipeline in final_code.ipynb using this formula

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

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 code at the end of the lane_map function in final_code.ipynb. I used inverse transform of the warped perspective image to plot the lane lines back on the road. Here is the result:



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!).

Here's a youtube link for it. The file is named project_video_soln.mp4 in github

https://www.youtube.com/watch?v=ObhuzwvHhhY&feature=youtu.be

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 have used good image processing techniques to leverage different color scales like RGB as well as HLS and also gradient thresholding to get the best of all worlds.

The algorithm I have currently stores 12 previous fit lines coefficients and takes a running average of them for each frame. This is good for the basic lane detection. But lacks sanity checks on whether the lane detection is good or not. The pipeline might fail on hard paths and might not give accurate results.

I could add a bunch of sanity checks for the lanes lines to get accurate results as well as compare the lane width to determine good or bad detection.