Advanced Lane Finding Project

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.

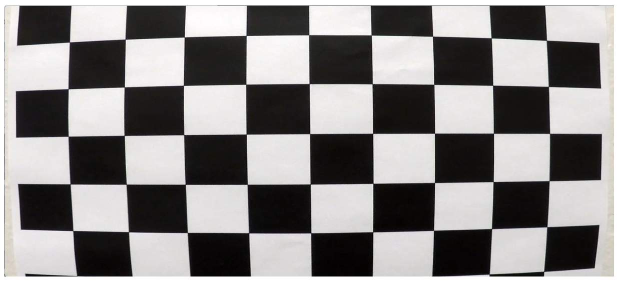
The code for this step is contained in the utils.py function from lines 13-32. I used opencv function find chessboard corners to find the corners in the example images. I create 2 empty list one for 3d real world coordinates of the chessboard corners called object_pts and another for corners found in the image called image_pnts.

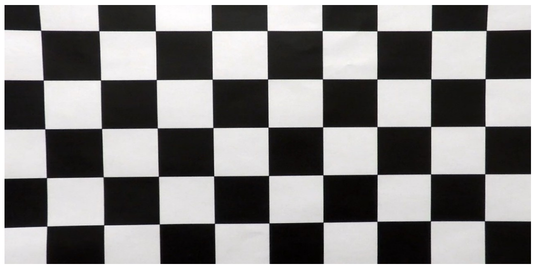
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of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

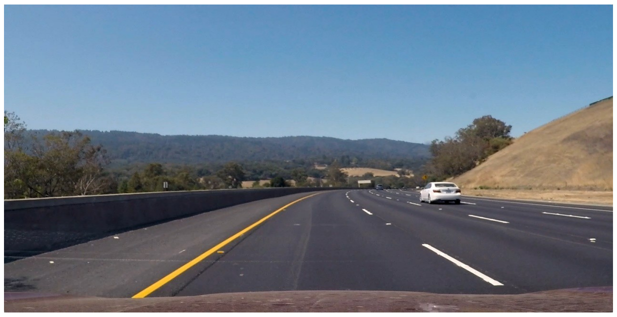
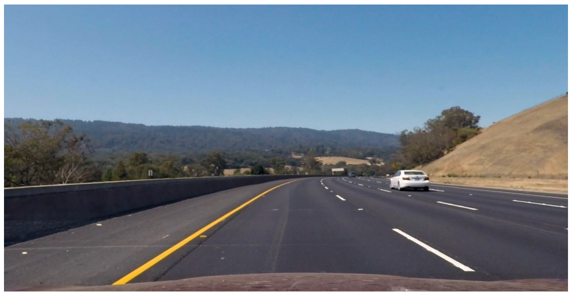
I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

The first set of images so correction in a chessboard image the second set of images shows the correction on one of the test image.

Original image: undistorted image:



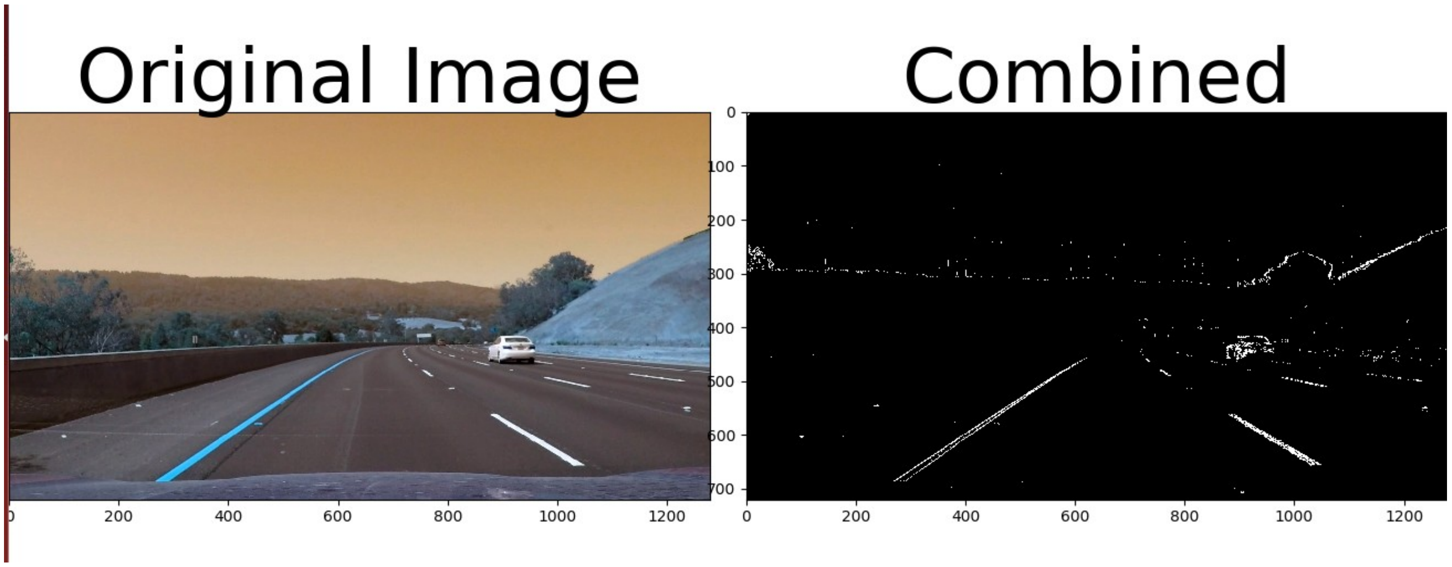


2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps are in ulits.py function from lines 102 - 138).The combined threshold image is generated with help of 2 functions namely abs_sobel_thresh and image thresh.
The L and S channels of a HSL image are choose and a sobel gradient is performed with lb and ub values to identify lanes .Later the 2 output binaries from sobel are combined with another binary to generate the final thresholded binary image.
 A lot of experimentation was done to come to a good value to give robust lane detection in varying conditions. Experimentation was done with hsv and gray image channels as well. The specific method for segmentation was chosen because it gave the best results.

Here's an example of my output on one of the test images.



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 is included in utils.py script as a function called get perspective which takes as input a image and a flag which tell the function either to return a perspective transform matrix or an inverse preceptive transform matrix(lines 74-99). The vertices's for the transform from src point to dst points or vice versa were chosen according to the image dimension and channels. Knowing the fact that the camera is in center of the vehicle and the vehicles are trying to stay in center of the lane. The points were choose as follows:

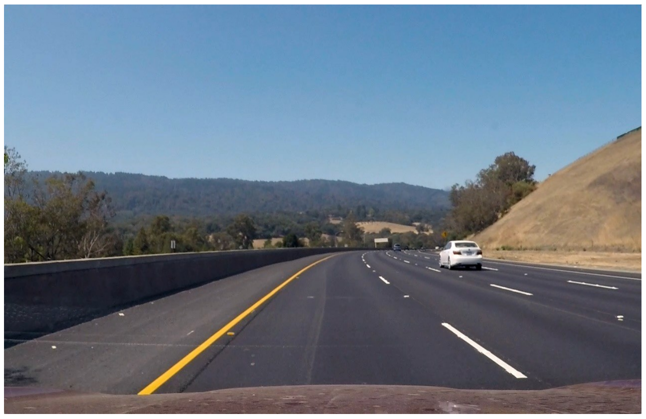
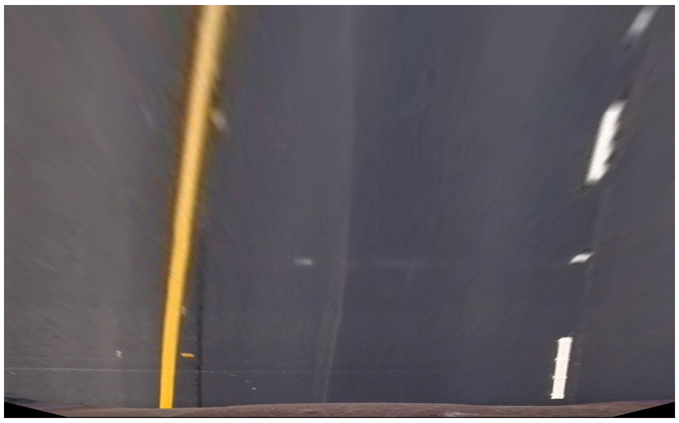
The way points were initial choosen was via visual inspection and the I converted those values into percent of image width or height.

```
# Source coordinates
src = np.float32([
    [image_width * 0.45, image_height * 0.6],
    [image_width * 0.5, image_height * 0.6],
    [image_width * 0.1, image_height * 0.9],
    [image_width * 0.8, image_height * 0.9],
])
# Destination coordinates
dst = np.float32([
    [image_width * 0.2, image_height * 0.02],
    [image_width * 0.8, image_height * 0.02],
    [image_width * 0.2, image_height * 0.97],
    [image_width * 0.8, image_height * 0.97],
])
```

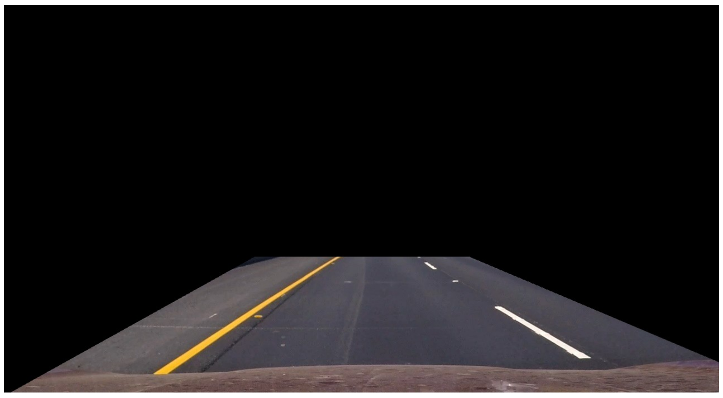
This resulted in the following source and destination points:

Source	Destination
572.79, 468	256, 18
707.20, 468	1024, 18
224, 684	256, 702
1065, 684	1024, 702

the ouput of a perspective transform is shown below:

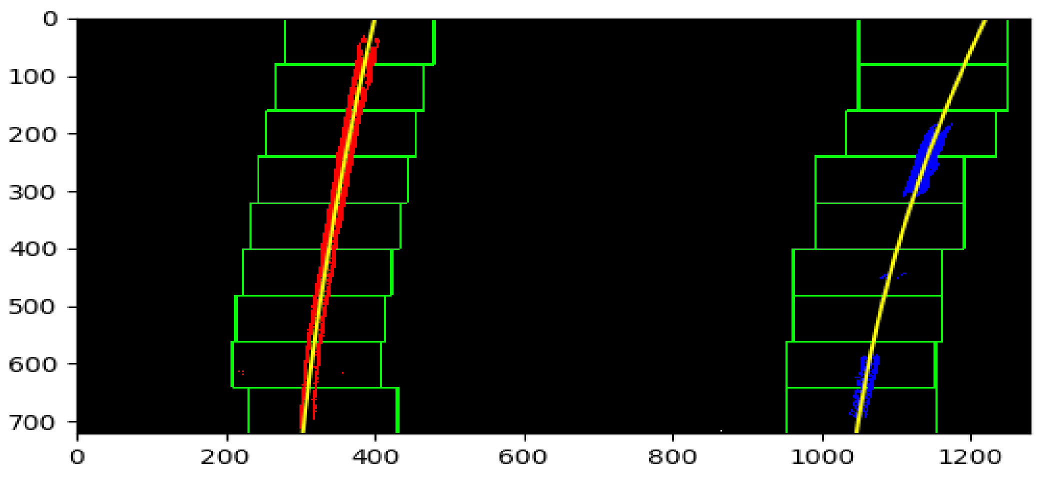



the output of masking operation for the same image is as shown :



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

Once I identified the lane line pixels using histogram peaks to identify the starting point followed by sliding window operation , I used np.polyfit function to fit the 2nd order polynomial function to the detected lane lines . I followed the same formula as shown during the lectures the result of sliding window is shown below for one of the test images. The code for the same is in script lane_detection.py from line (52-123)

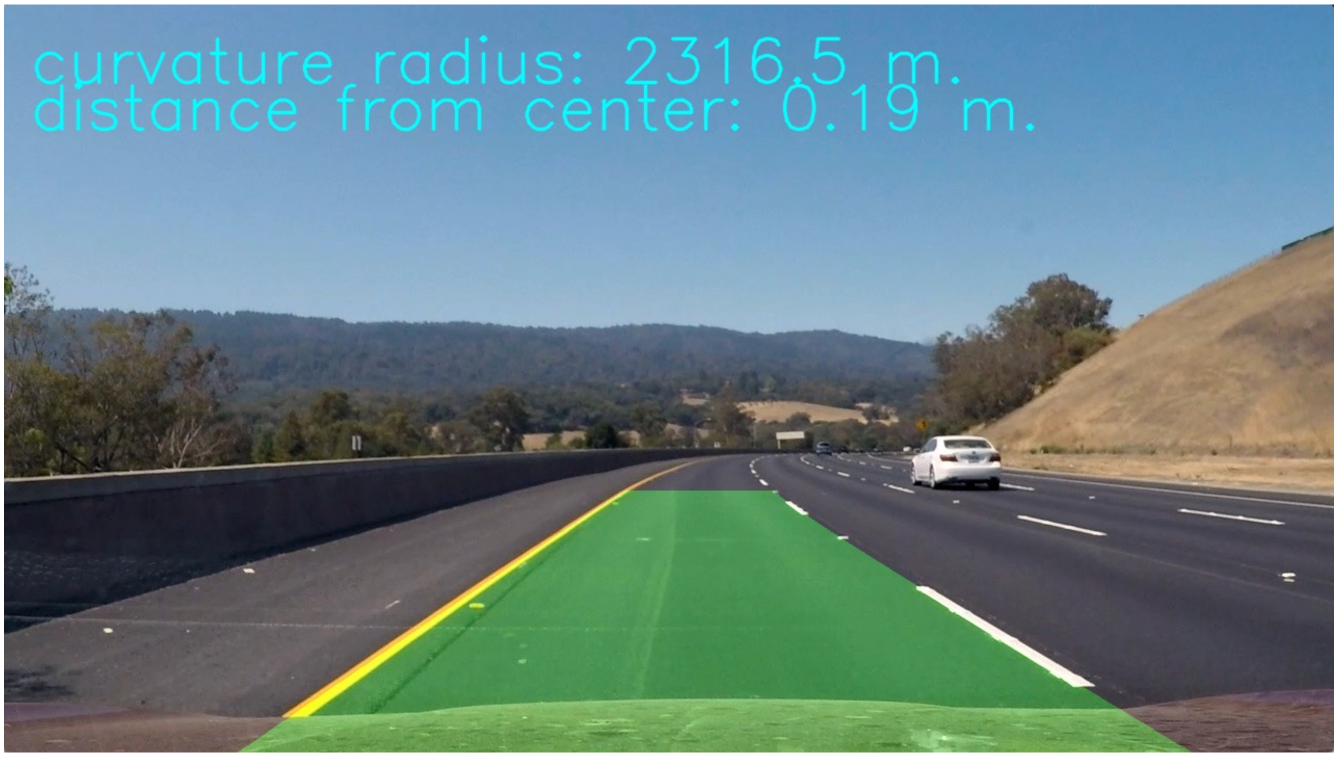


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

There are two functions in lane_detection.py from line 13-28 called lane curvature and dist_from_center to compute the lane curvature and and dist of vechile from the center of the lane respectively.
The radius of curvature was computed with formula
$Rcurve=|2A|(1+(2Ay+B)2)3/2.$
for both left and right lane respectively.

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 lines 130 through 146 in my code in lane_detction.py .
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 output video is named project_video_output.mp4.

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?

The most challenging part for me personally in this project is was coming up with right threshold values, which still need some improvement as I believe its not gone work in challenge video which I saw when I tested my pipeline on challenge video.

Further scope of improvement:

1. Make a class for lane and maintain the last best fit and last best curvature.
2. find a way to overcome when there is no lane found in the image. That is approximate the lane using n previous lane marking.
3. I world like to take some real world videos in cities because where I live the lane marking at say intersection cross each other which can be challenging.
4. try some night time lane detection and see if the thresholding works on them

too.