Advanced Line Finding

Project Report

Submitted files

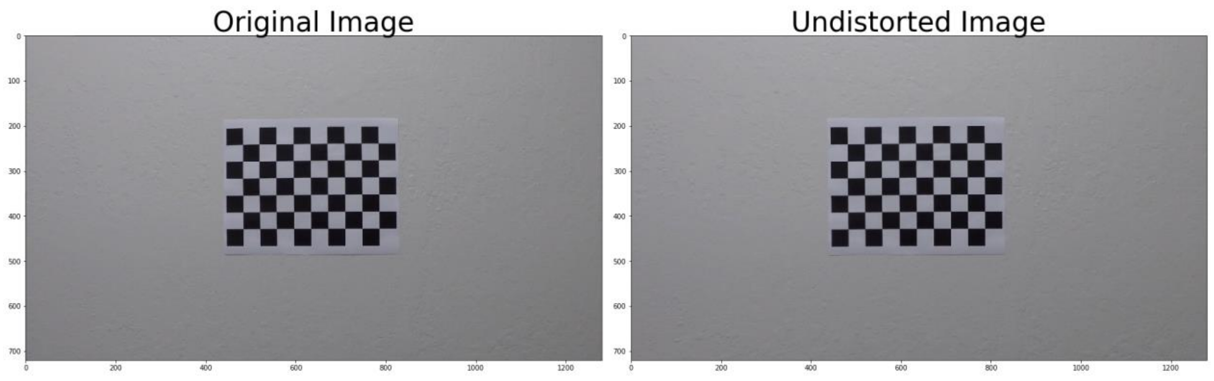
My project includes the following files:

- project4.py: the script for single image processing and line finding pipeline
- project4_video.py: the script for line findings in video
- project_video_output.mp4: output of the processing project_video.mp4 using project4_video.py script
- writeup_report.pdf summarizing the results

Camera Calibration

The code for this step is contained in lines 15 through 56 of the file called project4.py.

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 objp will be appended with a copy of it every time I successfully detect all chessboard corners in a calibration 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:



Pipeline (single images)

Pipeline for single image processing consists of the following steps:

1. Distortion correction
2. Image thresholding
3. Perspective transformation
4. Line finding
5. Calculating radius and position

Distortion Correction

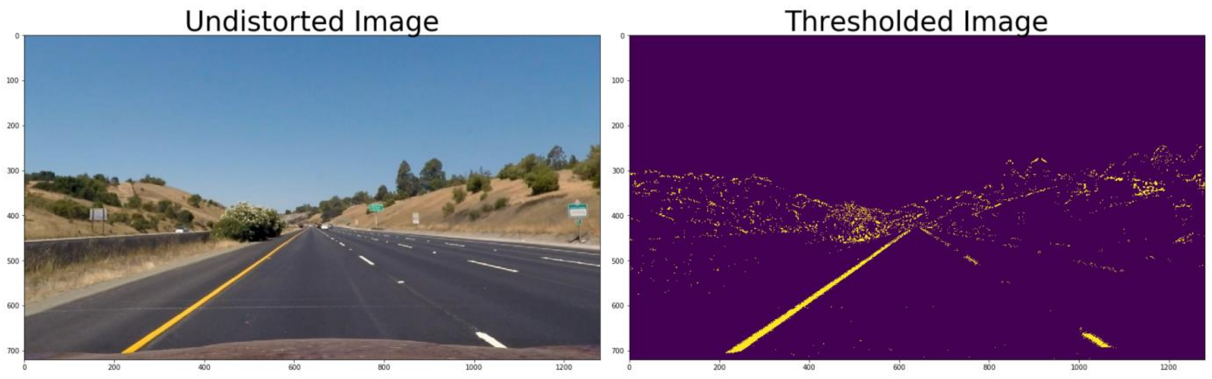
Distortion correction is performed on the new images using calibration parameters estimated during calibration process and stored in the wide_dist_pickle.p file.

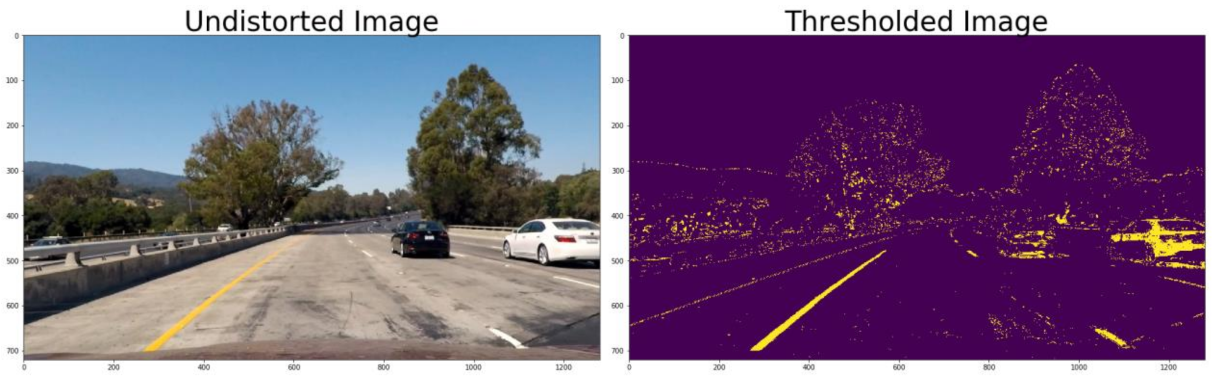
The following figures provides examples of original and distortion corrected image:

Thresholding Image

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines 68 through 113 in project4.py). I used HLS color space with S-channel for color threshold and L-channel for gradient thresholds over x and y direction as well magnitude gradient. I also used L channel threshold but it seems that it does not contribute much to the line detection performance at least with threshold values I tried.

Here's an examples of my output for this step:



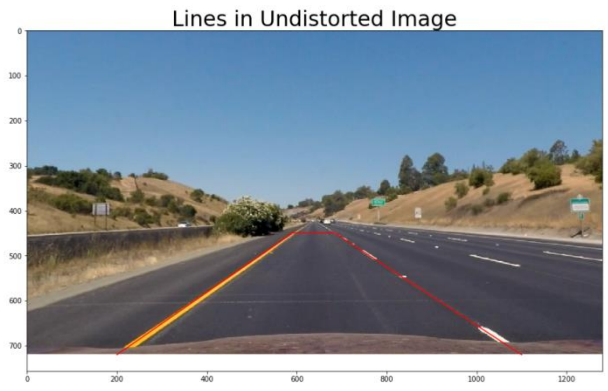


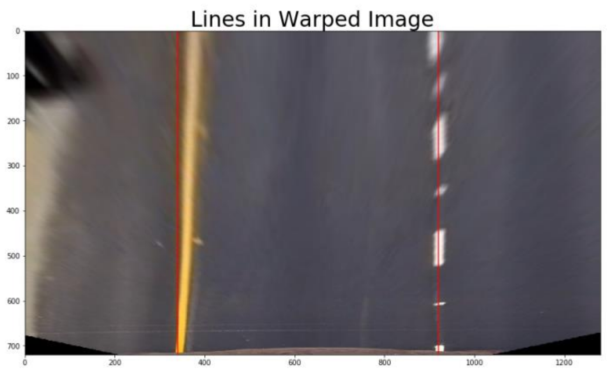
Perspective Transform

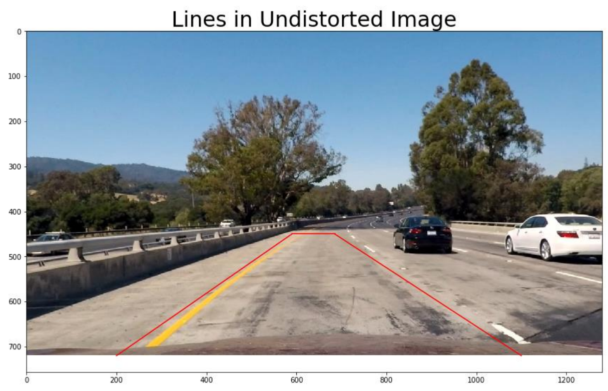
The code for my perspective transform includes a function called warp_image(), which appears in lines 117 through 124 in the file project4.py. The warp_image() function takes as inputs an image (img). The source and destination points are chosen so that the lines in the warped image appear approximately parallel. I chose the hardcode the source and destination points in the following manner:

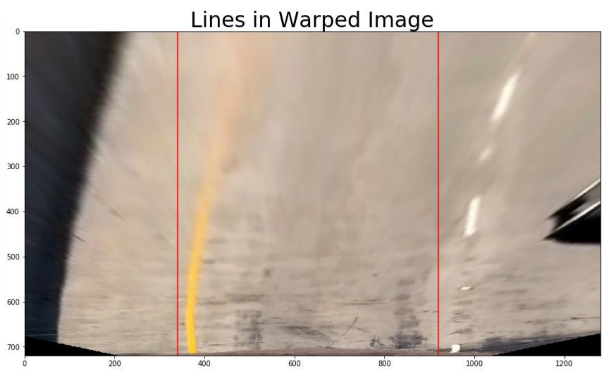
Source	Destination
200, 720	340, 720
590, 450	340, 0
685, 450	920, 0
1100, 720	920, 720

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 as can be seen in the figures below.







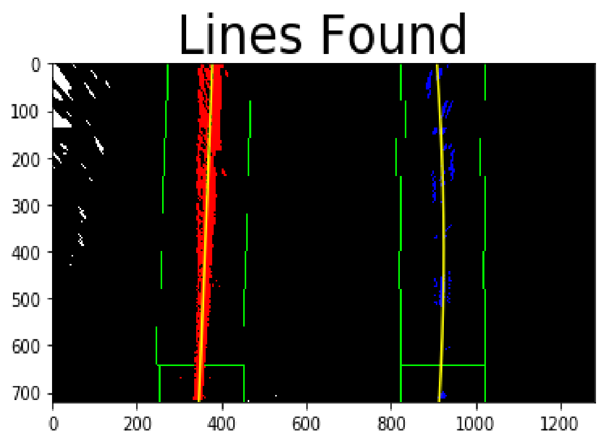


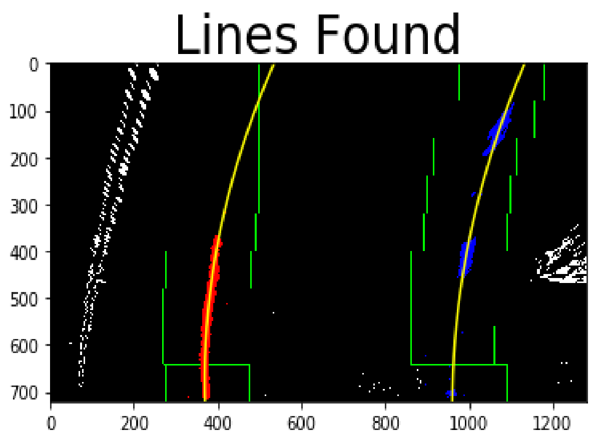
Line Finding

The line finding is implemented in the function called find_lines(), which appears in lines 126 through 207 in the file project4.py.

The search is performed in a sliding-window manner. At first histogram of the thresholded and warped image is generated. The search windows are initialized to the positions maximum in the first (left) half of the histogram for searching left lines and the maximum in the second (right) half of the histogram for searching right lines. Then 9 windows of the size 100 ("margin") points are used to search for the possible line points. In each of the window "good points" i.e. non-zero points within window are stored in arrays. If more than "minpix" (50) points in the thresloded image are found in a window, the mean coordinates of the points are used as the center of the next window. Finally, polynomial fit of order 2 is performed over found left and right points to find left and right lines respectively.

At the end of the function the found lines on the binary image are plotted as can be seen in the figure below:





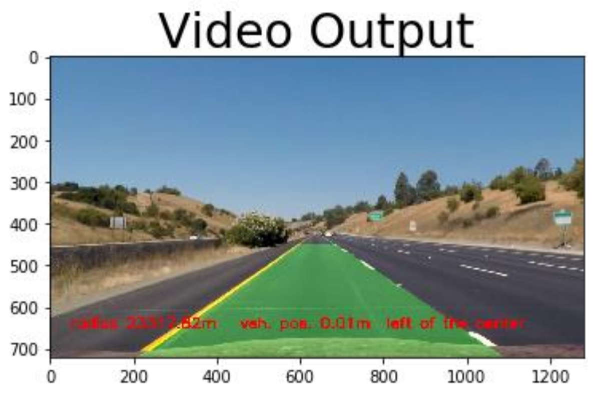
Note that for finding lines in video I don't initialize search using histogram but use previously found lines to initialize search in the new images. This is implemented in find_lines_video function in lines 209 through 264 in the file project4.py. (or lines 203 through 234 in the file project4_video.py). Only if more than certain number (10) of consecutive images in the video is not reliable detected or for the first video image the function find_lines is used in video processing. The estimation detection reliability is performed in the function sanity_check in the file project4_video.py lines 263 through 289.

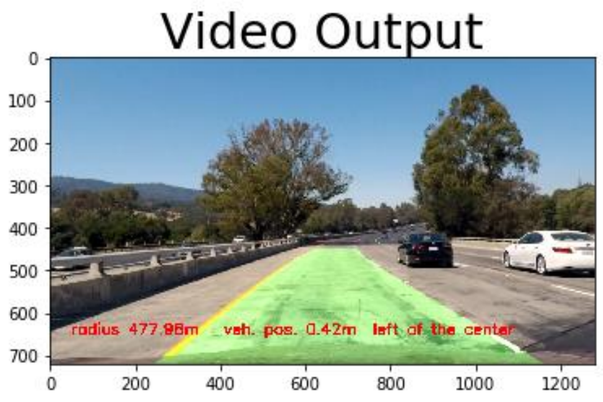
Calculating Radius and Position

Radius and vehicle position are calculated in the function called find_radius, which appears in lines 267 through 287 in the file project4.py. The radius is calculated using formulas provided in the lectures and position is estimated as the deviation of the midpoint of the lane from the center of the image. Both position and radius are calculated in meters using meters per pixel conversion constants.

Mapping Image Results to the Road

The following figure provides an example image of my image processing pipeline result plotted back down onto the road such that the lane area is identified clearly.





This is implemented in lines 388 through 414 in my code in project4.py. The same mapping is implemented for the video too (lines 388 through 428 in my code in project4_video.py).

The video is provided in the file: project_video_output.mp4 in my submission.

Discussion

My image processing pipeline seems to work reasonable well on the project video but not so good on the challenge video. I use averaging (moving average with constant 0.9) of found lines (using class Line() in project4_video.py) and plausibility check of the found lines in the function sanity_check. I tried different color and gradient thresholds as well as sanity check thresholds, but couldn’t improve performance much. Maybe I should refine the thresholds or even try quite another method?