CarND-Advanced-Lane-Lines-P4

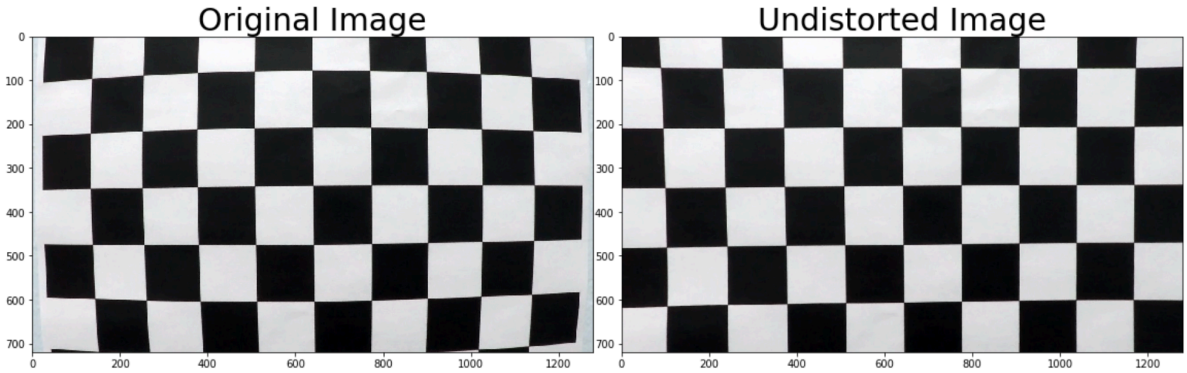
Writeup / README

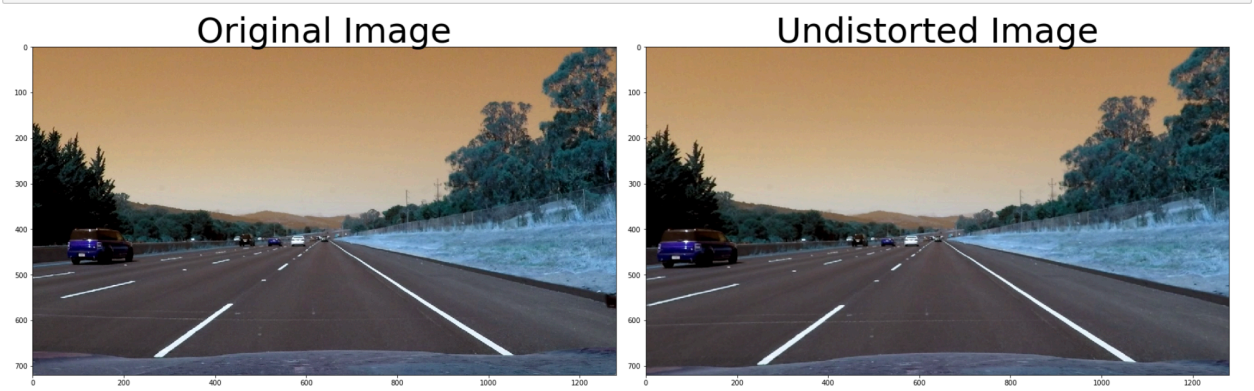
Camera calibration

I used `cv2.findChessboardCorners` to find the corner points that are stored in an array `img_points` for each calibration image where the chessboard could be found. The object points are stored in an array called `obj_points`. I then used the output obj_points and img_points to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera` function.

PipeLine

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

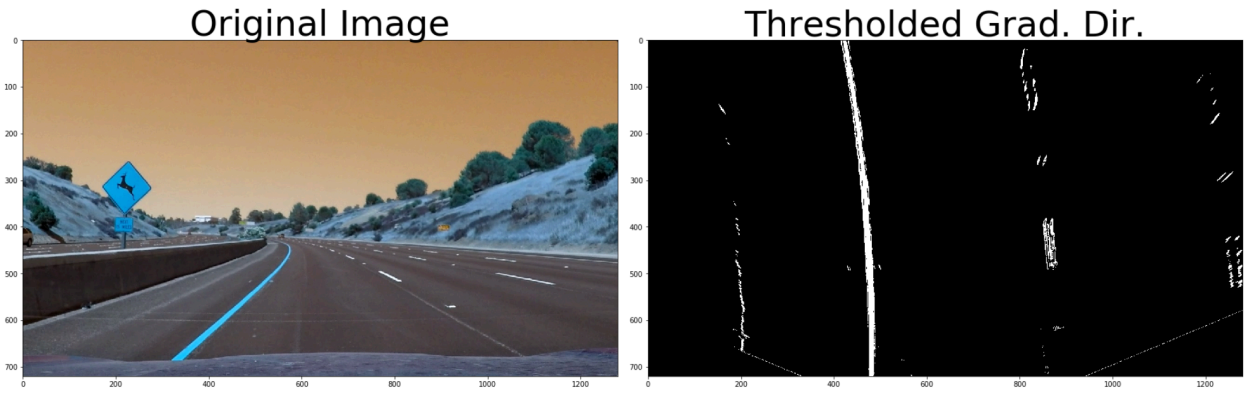




I used `cv2.undistort` to undistorted image. For some reason the image read from folder had weird colors.

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 experimented with different combinations but what worked for me was to calculate Sobelx and a binary threshold in s-channel. Then I combined the result from both to create a combined image.



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

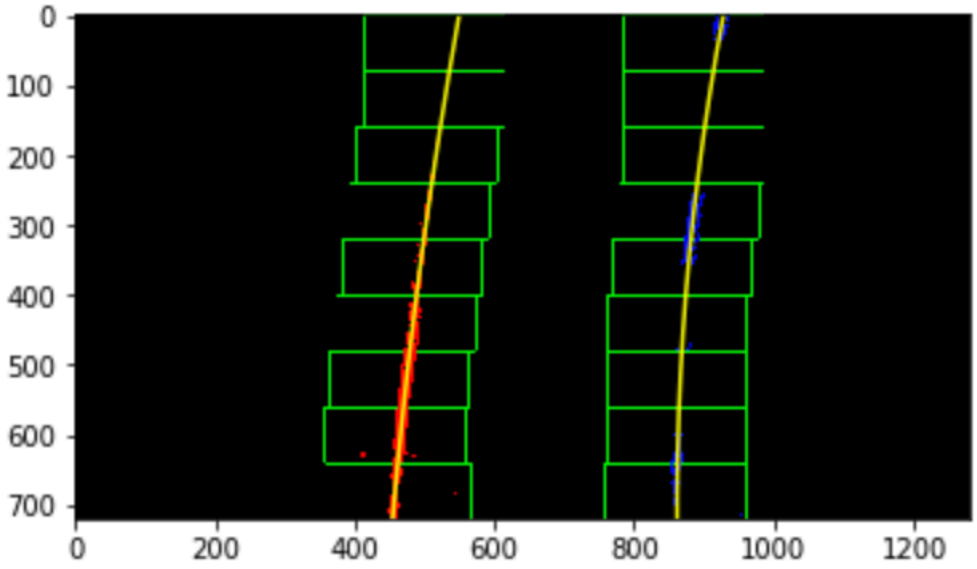
I did perspective transform before applying color threshold. This transformation happen in method in “image_unwarp” method.

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

This happens in “find_lane_pixels” function. I’m using a 9 windows to fit the lane line. First the sum was calculated of binary image obtained in previous step. Then we identified 2 max values - one on left of midpoint other on right of midpoint. These 2 points act as centre of our base window. Then for each window we identify the non zero pixels. If more than “minpix” pixels found we recenter our window to the average of their values.

We then extract the left lane and right lane pixel position and call “polyfit” to get left lane fit and right lane fit

In this I experimented with lot of different methods and threshold values. I also converted the pixel values to meter and stored in “left_fit_m” and “right_fit_m”

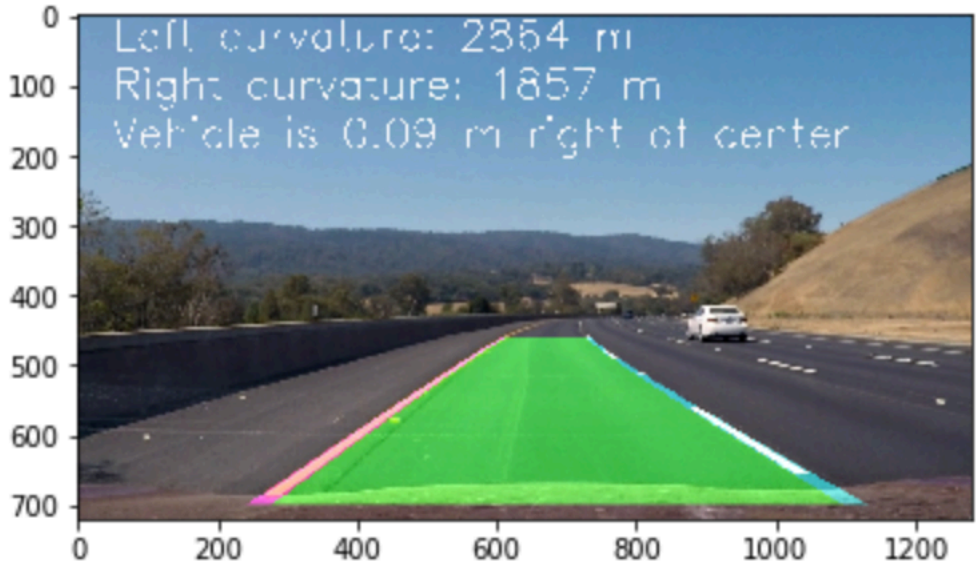


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

“measure_curvature_pixels” calculates radius of curvature in my code. It was a simple formulae given in the lectures.

Vehicle position is assumed to be the centre of image along x axis. First I calculated portion of left lane then right lane. Then line middle can be calculated using - “lineMiddle = lineLeft + (lineRight - lineLeft)/2”. Now we have both values we just subtract vehicle portion from lineMiddle to position of vehicle with respect to center.

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



Video PipeLine

Video can be accessed here https://github.com/Abhityagi16/Advanced-Lane-Detection/blob/master/project_video_ouput.mp4.

It is at the root level of project.

Discussion

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 problems I encountered were almost exclusively due to lighting conditions and shadows.

This pipeline will almost likely fail in snow or when a car with same color as lane line overtakes our vehicle.

I've considered a few possible approaches for making my algorithm more robust:

i) If we detect only one lane line in some frames we could approximately draw the other line based on last frame and using the fact that mostly lane lines are always at constant distance from each other.

ii) Dynamically vary thresholding parameters for different conditions.