UDACITY

PROJECT

Advanced Lane Finding

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Congratulations, your project meets all specifications!

I've enjoyed very much reviewing your project, you can be proud of the hard work you've done!
If you want to go a bit further, here you can find some very interesting papers and posts, hope you like:

- http://programmingcomputervision.com/downloads/ProgrammingComputerVision_CCdraft.pdf
- http://diml.yonsei.ac.kr/papers/Real-time%20Illumination%20Invariant%20Lane%20Detection%20%20for%20Lane%20Departure%20Warning%20System.pdf
- https://medium.com/@vivek.yadav/robust-lane-finding-using-advanced-computer-vision-techniques-mid-project-update-540387e95ed3#.n1ph5k1og
- https://medium.com/@heratypaul/udacity-sdcnd-advanced-lane-finding-45012da5ca7d#.c0ujnakj9
- https://medium.com/@heratypaul/experiment-using-deep-learning-to-find-lane-lines-c668a6e42070#.5flnqnf0a
- https://medium.com/@ajsmilutin/advanced-lane-finding-5d0be4072514

Keep learning and stay ***Udacious***! :D

Writeup / README

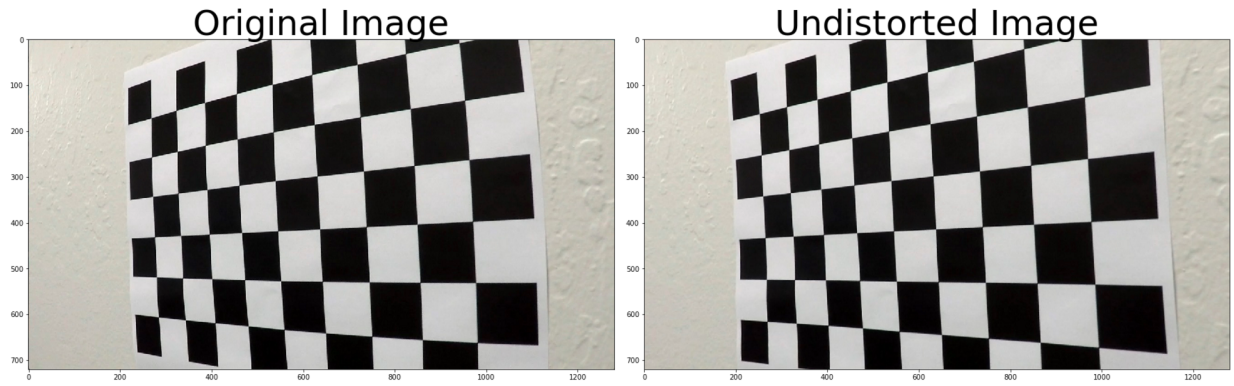
The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.

The provided README file includes a very detailed step by step explanation covering each rubric point, the provided examples images are very explanatory providing a quicker idea on how is the pipeline working. Very good job!

Camera Calibration

OpenCV functions or other methods were used to calculate the correct camera matrix and distortion coefficients using the calibration chessboard images provided in the repository (note these are 9x6 chessboard images, unlike the 8x6 images used in the lesson). The distortion matrix should be used to un-distort one of the calibration images provided as a demonstration that the calibration is correct. Example of undistorted calibration image is Included in the writeup (or saved to a folder).

OpenCV functions were properly used to calculate both the camera matrix and distortion coefficients. Un-distortion demonstration on the test image looks perfect, great job! :)



Pipeline (test images)

Distortion correction that was calculated via camera calibration has been correctly applied to each image. An example of a distortion corrected image should be included in the writeup (or saved to a folder) and submitted with the project.

All images/frames are properly un-distorted using the calibration and distortion matrices which were calculated in a previous step, excellent job!

A method or combination of methods (i.e., color transforms, gradients) has been used to create a binary image containing likely lane pixels. There is no "ground truth" here, just visual verification that the pixels identified as part of the lane lines are, in fact, part of the lines. Example binary images should be included in the writeup (or saved to a folder) and submitted with the project.

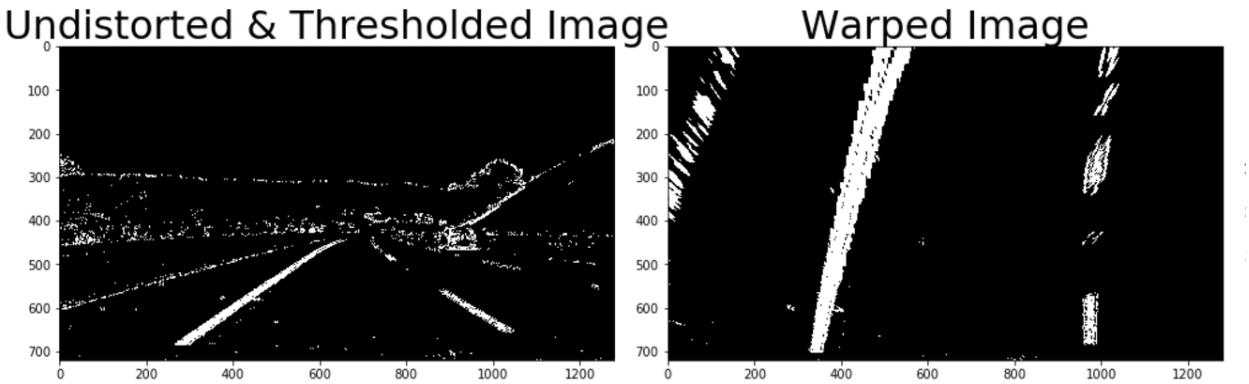
A very good job with the implemented image processing pipeline, using a combination of color transform, thresholding, and x/y-axis derivative (`cv2.Sobel()`) to accentuate the lane lines.

OpenCV function or other method has been used to correctly rectify each image to a "birds-eye view". Transformed images should be included in the writeup (or saved to a folder) and submitted with the project.

The image is correctly rectified (warped) to obtain a “birds-eye view” prior starting with the line detection procedures. Projected lane lines are very parallel, which will make measurements more accurate, great job!

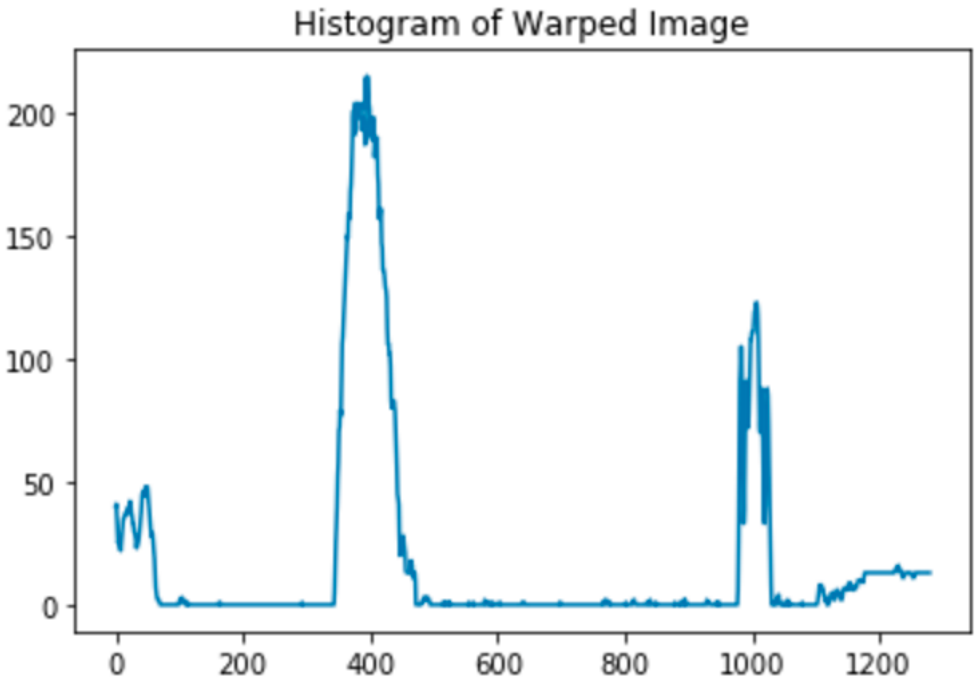
Suggestion

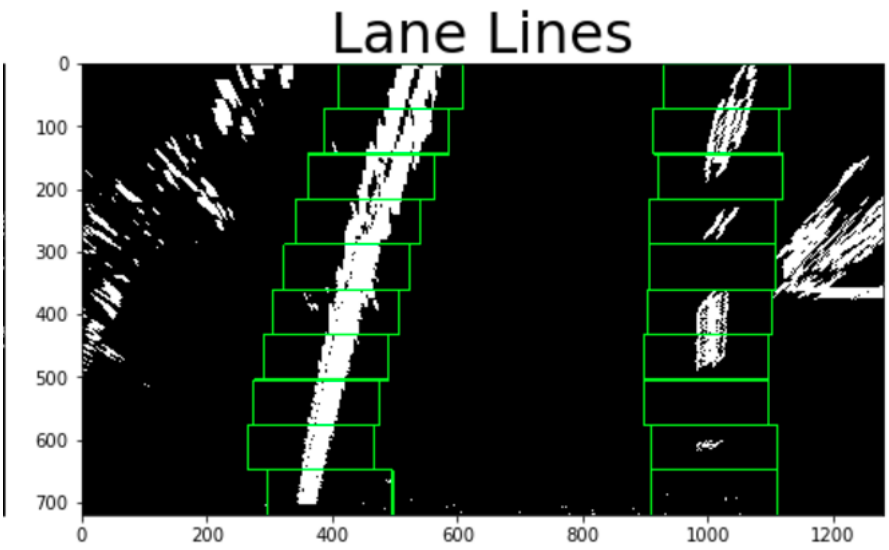
Please note that lines should be as parallel as possible in order to maintain a uniform horizontal scale (pixels/meter) in all the image (from top to bottom). It's recommended fine tuning the transformation matrices in order to get two lines as parallel as possible. :)



Methods have been used to identify lane line pixels in the rectified binary image. The left and right line have been identified and fit with a curved functional form (e.g., spine or polynomial). Example images with line pixels identified and a fit overplotted should be included in the writeup (or saved to a folder) and submitted with the project.

Left and right lane lines are perfectly identified from the rectified binary image, by means of a sliding window and a second order polynomial fitting (`np.polyfit()`), a very good job here as well!





Here the idea is to take the measurements of where the lane lines are and estimate how much the road is curving and where the vehicle is located with respect to the center of the lane. The radius of curvature may be given in meters assuming the curve of the road follows a circle. For the position of the vehicle, you may assume the camera is mounted at the center of the car and the deviation of the midpoint of the lane from the center of the image is the offset you're looking for. As with the polynomial fitting, convert from pixels to meters.

- Lane curvature and in lane vehicle position are properly calculated/estimated using an additional polynomial fitting, scaled to the real world space (meters).
- Returned values are consistent with the lane's curvature and car's position. In order to improve the accuracy of your estimations please consider fine tuning the perspective transformation matrices as proposed above. :)
- These estimated values are annotated onto the output video, nice job here as well!

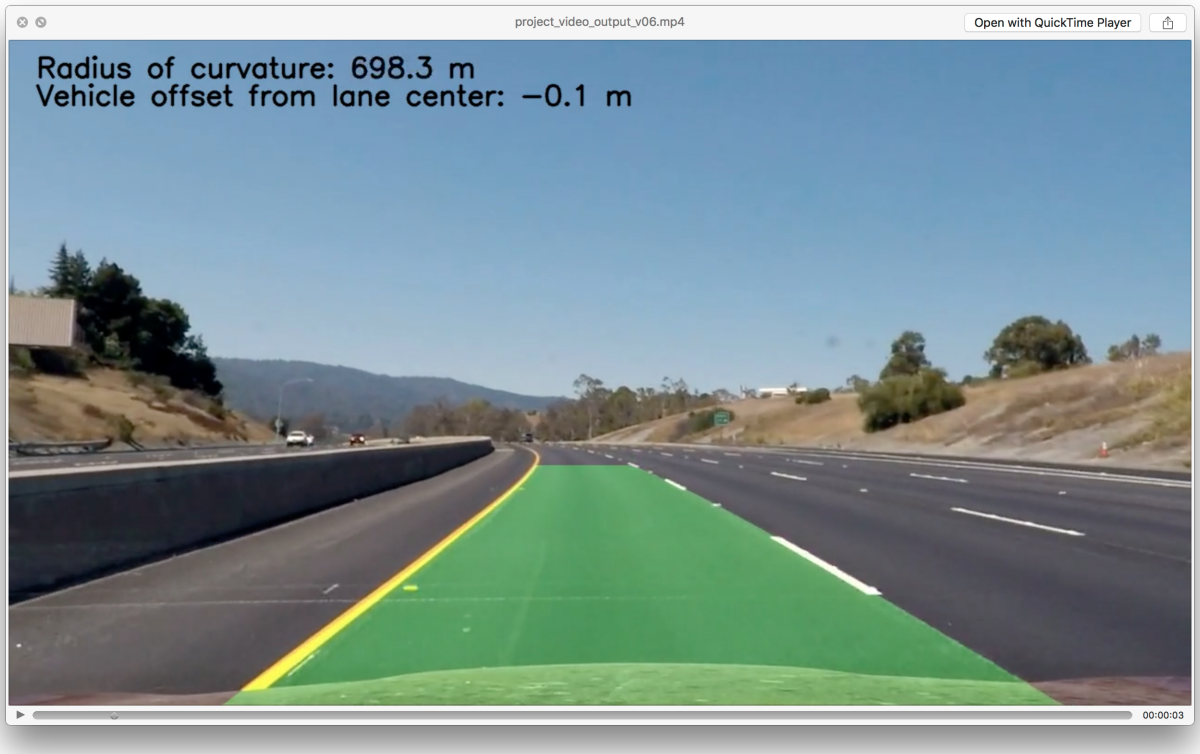
The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This should demonstrate that the lane boundaries were correctly identified. An example image with lanes, curvature, and position from center should be included in the writeup (or saved to a folder) and submitted with the project.

Annotated polygonal fill is properly “unwarped” and annotated on the original images. Great!

Pipeline (video)

The image processing pipeline that was established to find the lane lines in images successfully processes the video. The output here should be a new video where the lanes are identified in every frame, and outputs are generated regarding the radius of curvature of the lane and vehicle position within the lane. The pipeline should correctly map out curved lines and not fail when shadows or pavement color changes are present. The output video should be linked to in the writeup and/or saved and submitted with the project.

Implemented algorithm works really well finding and annotating lane lines on almost every video frame, however, I've found some issues in some frames with hard color and contrast changes, anyway, a good job since most of the video was properly annotated! :)



Discussion

Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

Very good job including your discussion in the README, explaining the main challenges you've faced and overcome during the design and development stages, pointing where the pipeline could fail, and also providing your thoughts about solving this project using deep learning techniques. :)