UDACITY

PROJECT

Advanced Lane Finding

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

2 SPECIFICATIONS REQUIRE CHANGES

Almost there, excellent work so far!
👏👏👏

I can see you put a lot of effort in your project and with just a few improvements you'll be good to go.

Keep going and good luck!
Paul

PS: Remember you have a number of support options to help you achieve success, such as mentoring, forums and Slack.

If you have further questions you can find me on Slack as `@viadanna`

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.

Good job submitting the write-up as required.

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).

Excellent work using `cv2.findChessboardCorners` and `cv2.calibrateCamera` to find the camera matrix and distortion coefficients.

Suggestion

You can store the shape of the chessboard as a variable so you don't have to keep repeating these numbers and the code can be easily updated for another set of chessboard images.

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.

Good job using `cv2.undistort` to apply the camera calibration.

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.

Awesome

Excellent work using a thresholded color channel and the Sobel operator to build the binary image 👍

Suggestion

You could also visualize each channel from a few color spaces such as LUV, HSV etc. to see which channels shows the lanes best and add those to your combined binary.

Check this lesson for further information. Remember that the idea here is to make the lane pixels dominate the warped binary.

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.

Good job using `cv2.getPerspectiveTransform` to convert each image to a bird-eye view while also getting the inverse transform to later convert the lines found.

Suggestion

I recommend testing the warp on a straight stretch of road as you'll be able to measure the warp success by seeing the lanes parallel to each other. Not that I foresee any problem, just a heads-up.

Remember to check this on sharp turns as well as this basically selects a region of interest that might cut those out of sight.

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.

Good job finding the lane lines and fitting a second order polynomial.

Suggestion

A good idea here is using the previous lines to make your pipeline more efficient, which can be implemented by keeping the results of the previous frame and search around that.

This can also be useful to implement a sanity check and refuse lines that deviate too much from the previous one.

Finally, the previous lanes could be reused for a small window of time when no lanes are found or accepted.

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.

Nice implementation of the technique used to calculate the radius of curvature and position of the vehicle.

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.

Awesome

Your pipeline is pretty good at identifying lanes on the test image, showing lanes right on target 👍

Required

Notice that the test images must show the curvature and deviation.

Also remember to submit the code used to generate the annotated video.

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.

Good job, your pipeline is pretty robust, detecting the lanes even when going through shadows and changing asphalt colors with just some minor deviations.

Required

As mentioned earlier, the output video must have the curvature and deviation properly annotated on each frame. For this you can use cv2.putText.

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.

Good evaluation of the current pipeline, its shortcomings and possible improvements.

I recommend running it on the challenge videos for more ideas. What possible modifications might improve these results?

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

Student FAQ Reviewer Agreement