UDACITY

‹ Return to "Self-Driving Car Engineer" in the classroom

Advanced Lane Finding

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Great Job!👏

You have worked hard and made all of the required improvements from your previous reviewers, and now you have a very impressive, complete project!😃

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.

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).

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.

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.

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.

I can see that you have adjusted the perspective transform to make the lines look much closer to parallel, nice job!

To get the best result I recommend testing this on one of the test images with straight lines because it is much easier to evaluate the result. In that case both lines should be vertical after warping.

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.

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.

It looks like the `xm_per_pix` conversion factor has become more accurate as a result of changes made to the perspective transform.😉

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.

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.

Excellent job making the improvements here! Now the video is not showing any of the major errors for which changes were required in the previous review.👏

Suggestion:

If you want to store information on the lanes over a series of previous frames you can try using a `deque` with a set max length to store the lines from a specific number of frames. In each frame you can append your detections to the deque and when the maximum length is exceeded the oldest entry will automatically be deleted.

For example:

```
from collections import deque
line_history = deque(maxlen=5)
line_history.append(something)
```

This way you can easily track the lines over a set number of frames and apply some smoothing/averaging techniques.

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.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review