UDACITY

PROJECT

Advanced Lane Finding

A part of the Self-Driving Car Program

PROJECT REVIEW

CODE REVIEW 5

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Excellent work, you nailed it!
👏👏👏

I can see you put a lot of effort in your project and advanced a lot, you should be really proud!

You have shown a firm grasp of the concepts presented here and are good to go.

Keep going and good luck!
Paul

PS. If you have further questions remember 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.

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 exploring multiple thresholded color channels and the Sobel operator to build the binary image 👍

Suggestion

You could also visualize other channels from a few color spaces such as LUV, LAB etc. to see which channels show 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.

Awesome

Testing the warping in a straight stretch of road is an excellent idea, as you'll know it's correct when the projection shows parallel lanes.

Suggestion

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.

Awesome

Excellent idea using the previous lanes to make the search more efficient.

Suggestion

The previous lanes coefficients can be used as a sanity check, as a wild interframe fluctuation might indicate a misrepresented lane.

✓

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.

Suggestion

The curvature and deviation values can also be used for sanity checking, as wild fluctuations in these values might indicate a misrepresented lane. Caching it and comparing with the previous values as previously described can be a solution here.

Another idea is estimating the distance between both lanes and making sure it's close to 3.7m but that's less flexible as you'll have to drop both lanes whenever this fails.

✓

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 in the test images, showing lanes right on target 👍

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.

Awesome

Your pipeline is pretty robust, detecting the lanes correctly even when going through shadows and changing asphalt colors.

Suggestion

Further improving the pipeline to tackle the challenge videos can be made easier by creating a small subclip where the pipeline has difficulties as processing that would be much faster. For instance, to extract a 3 second video from 41 to 43 you can use:

```
subclip = VideoFileClip("project_video.mp4").subclip(41, 43)
```

The subclip of the affected areas can be used to evaluate each step of the pipeline to find out where it might be failing (thresholding, warping, lane fitting etc).

Another idea would be implementing a side-by-side visualization of each step of the pipeline such as this so you can visualize for each mistake of the pipeline which step is the root cause. This can be achieved by concatenating the image arrays horizontally and vertically using numpy.hstack and numpy.vstack.

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.

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

DOWNLOAD PROJECT

5 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review

Student FAQ