UDACITY

登出

项目

Advanced Lane Finding

此部分属于 Self Driving Car Engineer Nanodegree Program

项目审阅 | 代码审阅 | 注释

Requires Changes

SHARE YOUR ACCOMPLISHMENT

还需满足 4 个要求 变化

You put a lot of effort into this project and what you achieved so far is awesome.
Based on the comments you will be able to continue this project and submit a successful project soon.
Keep up the good work.

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.
	You provided a writeup with the necessary information in it. Well done.

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).

You implemented what you learnt from the lessons. You applied the undistortion function on a calibration image. Great job.

However I recommend to choose a calibration image which has a bigger distortion (i.e. the stripes are not straight at all). After applying the undistortion function you should see that the stripes of the check board became straight lines.

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.

Though the first call is undistortion in function `pipe_line(...)`, its result (`undistorted`) is not used, instead the original image (`img`) is used in most of the function calls.

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.

You combined color based and gradient based methods to get the binary images. Nice work.

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 transformation looks OK. The lane lines are more or less parallel on the birds-eye view image. Well done.

In case of an ideal transformation the lane lines of a straight section will be mapped to vertical and parallel lines. In your case the mapped lines are slightly diverging downwards. This can influence slightly the accuracy of your measurements Here you might want to fine tune your transformation by changing `src` and `dst` values.

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.

You used the sliding window technique to identify the lane line pixels. Then you fitted a 2nd order polynom onto the pixels found.
Nice work.

However you used this only for the left lane line and you used fixed distance to get the right one. Unfortunately this does not work for several reasons:

- The perspective transformation is not perfect, so the lane lines are not perfectly parallel
- There are slight changes in the perspective (because the road is not perfectly flat), so the lane distance is not a constant.

Also the right lane line should be searched by the same algorithm. Here are some ideas if it is not robust enough:

- You should fine tune the `margin`, `minpix` and/or `nwindows`
- Try to detect a longer section from the lane (by changing the perspective transformation through the variable `dst`)

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 seems that the estimates are missing from the video. According this rubric point each frame should contain the estimation of radius of curvature and the position estimate from the lane center.

The code following code coming from the lesson is just for generating **fake data** (just for demonstration purposes shown in lesson). Instead you should use the variables `leftx` and `rights` which you calculates from the fitted 2nd order curve.

```
# Generate some fake data to represent lane-line pixels
ploty = np.linspace(0, pixel_in_y-1, num=pixel_in_y)# to cover same y-range as image
quadratic_coeff = 3e-4 # arbitrary quadratic coefficient
# For each y position generate random x position within +/-50 pix
# of the line base position in each case (x=200 for left, and x=900 for right)
```

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.

You warped back the detected lane segment onto the original image. Well done.

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.

You already have a working pipeline. Well done.

However the pipeline shows a lane segment which follows the bottom of the lane, but bending always to the right. The reason is that you used variable `test_image` instead of the incoming parameter `img` in the following code:

```
# combien the threshold
def combine_thresh(img, ksize=3, sobel_thresh=(30, 255), mg_thresh=(30, 255), dir_thresh=(0, n
p.pi/2)):
    gradx = abs_sobel_thresh(test_image, orient='x', sobel_kernel=ksize, thresh=sobel_thresh)
    hls_binary = hls_select(img, thresh=(0, 160))

    combined = np.zeros_like(hls_binary)
```

```
    combined[((gradx == 1)) | (hls_binary == 1)] = 1
    return combined
```

After fixing this you may consider applying the following techniques to make the pipeline more robust (if needed):

- Store the last **N** detections and use some averaging technique to smooth the result
- Implementing sanity checks (e.g. check if the two lane line are parallel, their distance make sense, ...). If the sanity check fails, you can skip the current frame
- If you have a reliable detection from the last frame, you can use it to narrow the search space during the current lane line detection

You can read more about these ideas in the lesson 35. Tips and Trick for the Project.

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.
	You provided a reflection on your own code. Well done. You might want to write about hypothetical cases which would cause the pipeline to fail. E.g. think of a video made at night or a road segment with significant slope on it. Would the pipeline work in such situations? Why?

重新提交项目

下载项目

了解 修改和重新提交项目的最佳做法.

返回 PATH

学员 FAQ