UDACITY

Logout

PROJECT

Advanced Lane Finding

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

Amazing job with this. I really loved seeing the use of separate python files, classes and methods implemented for this project. Your code was far more interesting than the typical iPython notebook. I hope that my suggestions are able to offer some improvements in the robustness of your algorithm!

Keep up the amazing work and stay Udacious!

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 quality README with appropriate images.

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. 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).
Looks good and thanks for providing visualizations in your README

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.
Looks good and thanks for providing visualizations in your README

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.

Good job extracting the lane lines! Some advice for more robust extraction is to try color thresholding in the RGB, HSV and HSL channels for your yellows and whites!

Here is some sample code to play around with, it should help with more tricky areas!

```
HSV = cv2.cvtColor(your_image, cv2.COLOR_RGB2HSV)

# For yellow
yellow = cv2.inRange(HSV, (20, 100, 100), (50, 255, 255))

# For white
sensitivity_1 = 68
white = cv2.inRange(HSV, (0,0,255-sensitivity_1), (255,20,255))

sensitivity_2 = 60
HSL = cv2.cvtColor(your_image, cv2.COLOR_RGB2HLS)
white_2 = cv2.inRange(HSL, (0,255-sensitivity_2,0), (255,255,sensitivity_2)
)
white_3 = cv2.inRange(your_image, (200,200,200), (255,255,255))

bit_layer = your_bit_layer | yellow | white | white_2 | white_3
```

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.

Transformation points yield a good lane extraction image.

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.

Really excellent job with this part!

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 and the position of the vehicle within the lane may be given as meters off of center.

Your measurements for the locating the car relative to the middle of the road are great! One suggestion would be to average your results over 10 frames and update the measurement each 10 frames. This way it appears smoother and the number is more easily absorbed by a user.

The same applies with the smoothing of your radius of curvature. When smoothing the radius of curvature be sure to reject outlier values. Like radius of curvatures above 5000 m!

Playing around with the method to calculate the radius of curvature, I found it could be improved by using the middle point of the polynomials fitting. So I encourage this change:

```
def calculate_curvature_meters(self, all_y, all_x, y):
        ym_per_pix = 30 / 720  # meters per pixel in y dimension
        xm_per_pix = 3.7 / 900  # meteres per pixel in x dimension

        if(all_x.size>0):
            fit = np.polyfit(all_y*ym_per_pix, all_x*xm_per_pix, 2)
            curvature = ((1 + (2 * fit[0] * y/2. + fit[1]) ** 2) ** 1.5) \
                        / np.absolute(2 * fit[0])
        else:
            curvature = None
```

```
        return curvature
```

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.

Your warp looks great. I really do like the additional yellow "boundary" lane lines.

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.

Video looks awesome. Maybe consider moving the placement of your metrics. Off the side may prove to be less distracting.

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.

Great discussion, one of the more in depth ones I have seen. I appreciate the extra effort here.

DOWNLOAD PROJECT

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Student FAQ