UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB

Advanced Lane Finding

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations 👏🏼👏🏼👏🏼
It is obvious that you have put a lot of thought and hard work into this project.
The pipeline does an excellent job of identifying the lane lines in the project video, and the writeup is very thorough.

This research paper goes into how to detect curves and will also help in detecting faded lanes. It uses an extended version of hough lines algorithm to detect tangents to the curve which can help you detect the curve.
Here is another paper which talks about real time lane detection.

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.

The writeup is very detailed and clearly documents how each rubric item was implemented. 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).

Well done with camera calibration process! Here is more info about this process in documentation if you are interested:
http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html?
http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
http://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html

And very detailed Microsoft research:
http://research.microsoft.com/en-us/um/people/zhang/calib/

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.

The undistorted test image shows that distortion correction has been properly applied to images of highway driving, and I can see that this was implemented in the final pipeline as well. Nice work!

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 job applying a combination of color and gradient thresholds to create a binary image with clearly visible lane lines.

If you're curious about how to improve the algorithm so that it can work better in different lighting conditions, there is a very efficient method called Contrast Limited Adaptive Histogram Equalization (or CLAHE for short):
https://github.com/openpnp/openpnp/issues/481
https://docs.opencv.org/master/d5/daf/tutorial_py_histogram_equalization.html

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.

Nice job with the perspective transform, the lane lines appear very close to parallel in the birds-eye view images.

Tips for video processing(useful for the harder challenge):

- It's useful to have different sets of points for perspective transform. When the line becomes more curved, we need to narrow the region of interest.
- The current slope of the line may be used to dynamically change bottom coordinates for destination points for perspective transform. A positive slope means that we can place the bottom destination points closer to the left to catch the lines more accurately.

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.

Great job using a histogram and sliding window search to find the locations of the lane lines

For the extremely curved lines, you can improve your sliding window algorithm by shifting the center of the next window in the direction of the curve. Currently the center of the next window is simply calculated as a mean of x-coordinates of the pixels inside the current window:

```
if len(good_left_inds) > minpix:
    leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
if len(good_right_inds) > minpix:
    rightx_current = np.int(np.mean(nonzerox[good_right_inds]))
```

Also, if the radius of curvature decreases too fast(meaning that the line is becoming more curved), you can dynamically increase the width of the next window. On the other hand, when RoC increases(meaning the line is straight and we don't need too many windows), you can decrease the width and increase the height(decreasing the number of windows).

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.

Well done here. The values in the video look like reasonable estimates of the curvature of the road and the position of
the vehicle in the lane. You are using appropriate pixel to meter conversion factors based on the perspective transform that you are applying to the images.

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.

The fitting results have been properly warped back and plotted on top of the original image.

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.

The resulting video looks great! The pipeline is able to accurately map out the true locations of the lane lines throughout the
entire video and doesn't fail on curving roads, or in the presence of shadows and pavement color changes.

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 job reflecting on the project and pointing out some areas where the current implementation could still be improved.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START