UDACITY

PROJECT

Advanced Lane Finding

A part of the Self-Driving Car Engineer Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Great Job!👏

It is obvious that you have put a lot of thought and hard work in to this project, and the results are very impressive! The pipeline does an excellent job of identifying the lane lines in the project video, and the writeup is very thorough, making your project a pleasure to review!😃

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 clearly documents the steps taken to complete the project with references to where in the code each rubric item 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).

Perfect, the chessboard images have been used to calculate the camera calibration matrix and distortion coefficients, and these were used to remove distortion from one of the calibration images, demonstrating the effect.

Suggestion:

I noticed that you have the camera calibration with `cv2.calibrateCamera` inside of the `undistort_image` function which is being used in the final pipeline. This means that it is being repeated at every iteration of the pipeline, and for every frame when processing the video.

```python
def undistort_image(img, imgpoints, objpoints):
    # Convert to gray scale
    img_size = (img.shape[1], img.shape[0])
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img_size,None,None)
    undist = cv2.undistort(img, mtx, dist, None, mtx)
    return undist
```

Actually, the camera calibration is a pretty slow process and since you are using the same object points and image points each time, the result will always be the same. I noticed that your pipeline currently processes the video at about 1.5 frames per second which is quite slow, if you take the camera calibration out of the pipeline I think you should see a significant speed increase.

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 based thresholds to create a binary image with clearly visible lane lines.

Suggestion:

If you want to continue to explore additional color channels, I have seen that the L channel from LUV with lower and upper thresholds around 225 & 255 respectively works very well to pick out the white lines, even in the parts of the video with heavy shadows and brighter pavement. You can also try out the b channel from Lab which does a great job with the yellow lines (you can play around with thresholds around 155 & 200).

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. 👍

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 lane line pixels in the warped binary images, and fitting polynomials to your detections for each line.

Suggestions:

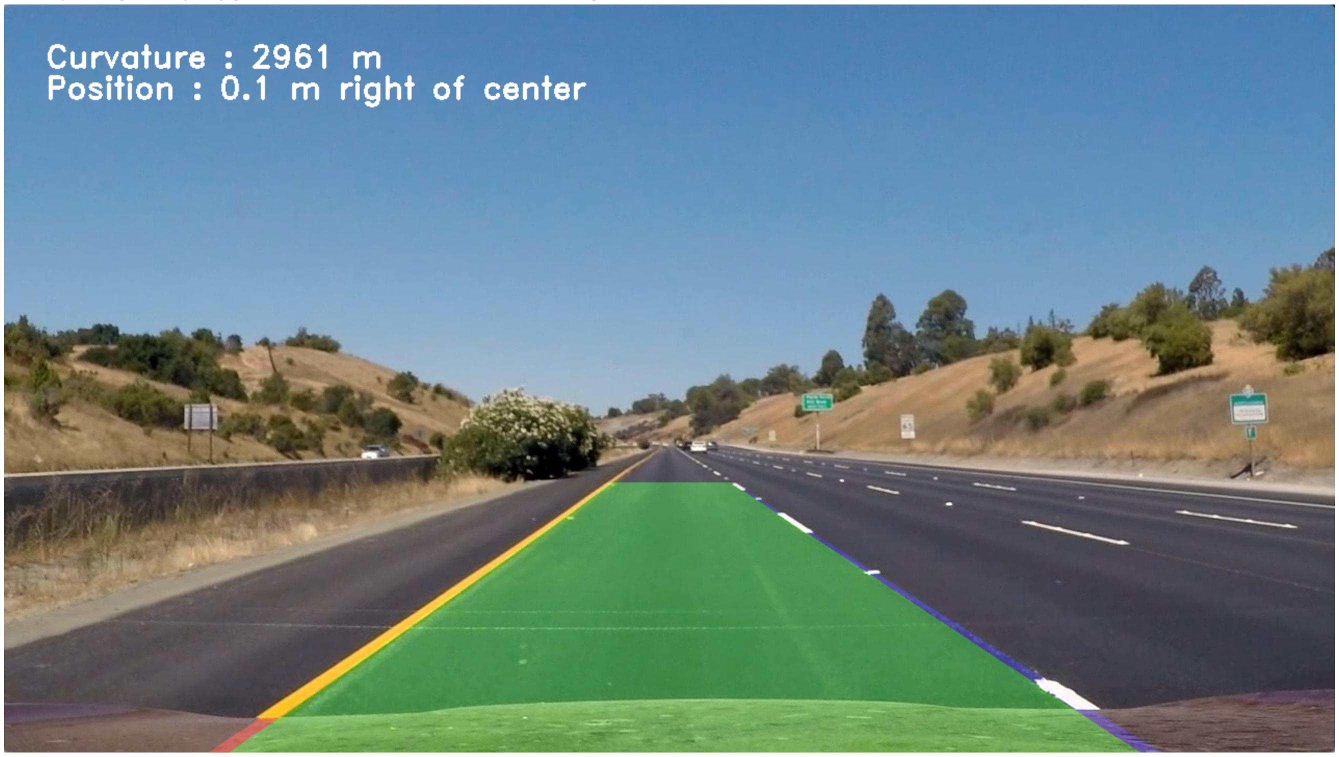
- After finding a confident detection in one frame, instead of repeating the sliding window search, you can restrict the search space in future frames to a close proximity around the previously detected polynomials. This way, if the warped binary image contains many positive pixels which do not belong to the lane lines, they would not be considered in the lane detection. A method for implementing this is given in this lesson from the classroom if you scroll down to the section titled "Skip the sliding windows step once you know where the lines are".
- You can also try applying sanity checks by making sure that the two lines are the right distance apart (based on the known width of a highway lane) and that their curvatures make sense in relation to each other. If a sanity check fails in a single frame, you can reuse the confident detections from prior frames.

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 very reasonable estimates of the curvature of the road and the position of the vehicle in the lane.

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 output images look pretty good and show that the lane detection is working.



Comment:

We really want to see that you are drawing the lane lines on to the undistorted version of the original image, but in the above example and in the output video I can see that the lane lines are drawn on to the distorted input image. You can tell this by looking at the hood of the car in the lower corners of the image.

To fix this, you should adjust this line:

```
result = cv2.addWeighted(input_image, 1, newwarp, 0.3, 0)
```

and `input_image` should be replaced with the undistorted 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 result 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. This is a very impressive result!👏

The only suggested improvement I have is what I mentioned in the previous rubric section, the video shows the distorted images with the lane line detections drawn on them, instead of the undistorted images.

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 discussing some ways that the current implementation could still be improved.👍

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ