UDACITY

‹ Back to Self-Driving Car Engineer

Advanced Lane Finding

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Nice work in using techniques including distortion correction, color transforms , gradient thresholding and image rectification to find and track the position of the lane boundaries in a real life video!

I left some comments/suggestions below for your reference. Happy Learning!

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.

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 here. Calibration chessboard images is properly un-distorted as a demonstration that the calibration is correct.

Pipeline (test images)

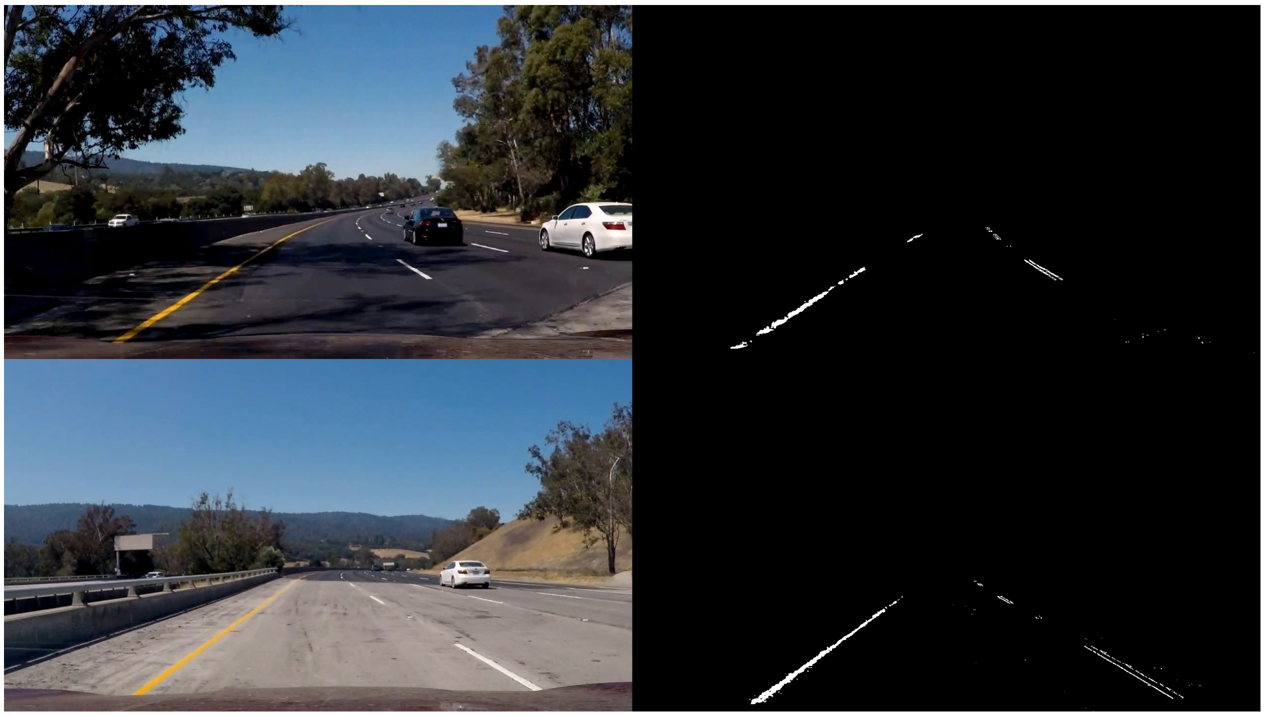
Distortion correction that was calculated via camera calibration has been correctly applied to each image. An example of a distortion corrected image should be included in the writeup (or saved to a folder) and submitted with the project.

Perfect! Good choice in selecting a test image whose undistortion effect is easily perceptible.

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 exploring different options and creating a binary image containing likely lane pixels.

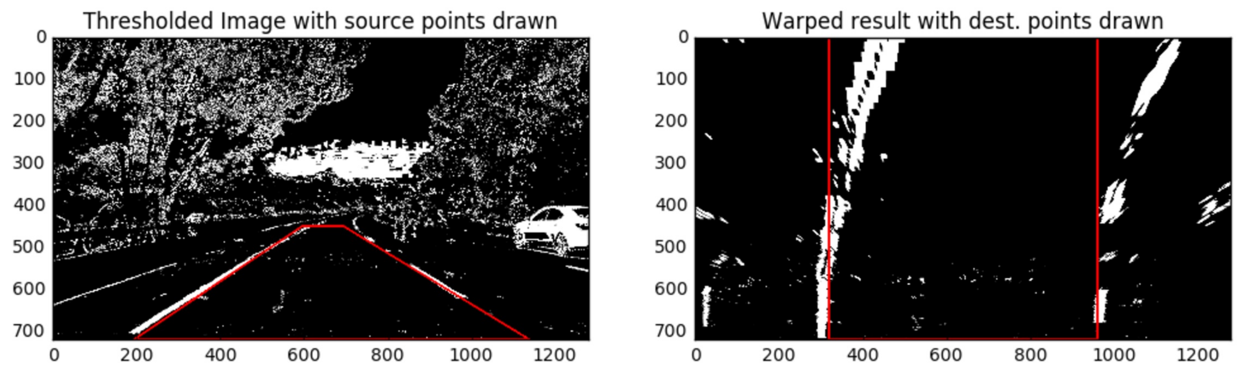
Suggestions: There are a wide variety of ways to performing thresholds that can achieve good results in this project. One very effective approach is to obtain lane pixels by color. The rationale behind it is that lanes in this project are either yellow or white. Below is a demo of using only YUV color space to do the thresholding, Pixels with a V component less than 105 are deemed white, while pixels with a Y component greater than or equal to 205 are deemed yellow.



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.

Perspective transformation procedure is correctly performed to obtain the "birds-eye view" of the images.

Suggestion, using an image like below can nicely visualize perspective transformation stage.



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. The left and right lane pixels have been identified and fit with curves.

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.

Your measurements for lane curvature and vehicle position look reasonable. Well done!
An idea for a cosmetic improvement would be to average your results over the previous 10 frames. This way the number would tend to be more smoother and accurate.

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.

From your example test image, I can see that the fit from the rectified image has been properly warped back onto the original image and plotted to identify the lane boundaries. Good job.

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 lane lines are nicely identified. Well done!

If you could extend the identified lane area to the bottom of the image, the video could have an even better visual effect.

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.

Project is well discussed in terms of known problems, as well as potential improvement opportunities.

Regarding the challenge video, my personal experience indicates two keys for successful lane identification:

1. Use only yellow color to detect the left side lane line

```
def hsv_thres_yellow(self, image):
    hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lowerb = np.array([20, 40, 100], dtype=np.uint8)
    upperb = np.array([30, 255, 255], dtype=np.uint8)

    mask = cv2.inRange(hsv_img, lowerb, upperb)
    mask[mask==255]=1
    return mask
```

2. Use a dynamic roi (basically reuse last frame's lane area as current frame's roi) to make sure the right side lane line is correctly detected.

DOWNLOAD PROJECT

RETURN TO PATH