A combined approach to single-camera-based lane detection in driverless navigation

Xin Zhang, Maolin Chen, Xingqun Zhan
School of Aeronautics and Astronautics,
Shanghai Jiao Tong University,
Shanghai, China
xin.zhang@sjtu.edu.cn

***Abstract*—A combined approach is presented with real-world examples to illustrate how to detect lane lines under several road conditions in driverless navigation. This approach is designed for single-camera perception and is achieved using camera calibration, un-distortion, color thresholding, perspective transformation, pixel identification and ploy fitting. Limiting factors such as illumination and radius of curvature of the testing road is considered. The results show that the method is effective in normal conditions but needs further improvement in extremely challenging environments.**

Keywords—self-driving; navigation; single-camera; lane detection; autonomous

I. INTRODUCTION

The fast evolution of driverless cars is driving a boost for car electronics and market of automotive electronics is expected to double its current size by the year of 2025. This article presents a case study to test the feasibility of using real data and general computing platform to develop lane detection algorithms [1-4], with an additional attempt to use Commercial-Off-the-Shelf (COTS) components and OpenCL as the programming language to test the algorithms in a simulated environment.

The paper is structured as follows. Background of the project is first introduced and previous work on lane detection used on driverless cars summarized. The second part introduces detailed procedures to compute the position of the lane lines, including (i) computing camera calibration matrix and distortion coefficients, (ii) applying distortion correction to raw images, (iii) using combined color transforms and gradient computation to obtain a thresholded binary image, (iv) applying a perspective transform to rectify binary image ("birds-eye view"), (v) detecting lane pixels and fit to find the lane boundary, (v) determination of lane curvature and vehicle position with respect to center of the detected lane, (vi) warping the detected lane boundaries back onto the original image, and (vii) outputting visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position. Among these steps, steps (iii) and (v) will be focused since these steps play a major part in ensuring the accuracy of the algorithm. In step (iii), different color spaces (RGB, HSV, HLS, YCbCr, and gray) will be examined. The objective is to find an effective combination of channels of different color spaces and corresponding thresholds to detect the pixels that indicate the existence of a segment of lane lines. This can be tricky since there are many disturbing effects. For example, shadows of trees, preceding cars, and even guardrails can alter the color of the lane lines. Some sections of a highway may have split colors within the same lane due to new pavements. All these factors count to the effectiveness of the algorithm. In step (v), a second order polynomial will be fit to the detected pixels that are assumed to form a lane line (either left or right). The collection of these pixels can present a problem because the preceding steps cannot ensure that each and every "good" pixel is detected (i.e. there must be noises). Sliding windows, smoothing, and a Look-Ahead-Filter (LAF) will be examined. The third part reports the processed results at each of the steps in section two. The implementation is tested on 3 video streams from different datasets with different road settings. We demonstrate that the algorithm can provide more than sufficient accuracy for lane detection. The fourth part reports an OpenCL and COTS framework for implementation and testing the algorithms in a simulated environment. The last section presents planned future efforts and concludes the paper. The datasets and source code are available at github. The sample code is provided using python and OpenCV (python).

II. CAMERA CALIBRATION

Image distortion takes place when a camera looks at 3D objects in the real world and transforms them into a 2D image; this transformation isn't perfect. Distortion effectively changes what the shape and size of these 3D objects appear to be. As a result, the first step in analyzing camera images, is to undo this distortion.

The following steps are taken:

A. Preparing Object Points and Image Points

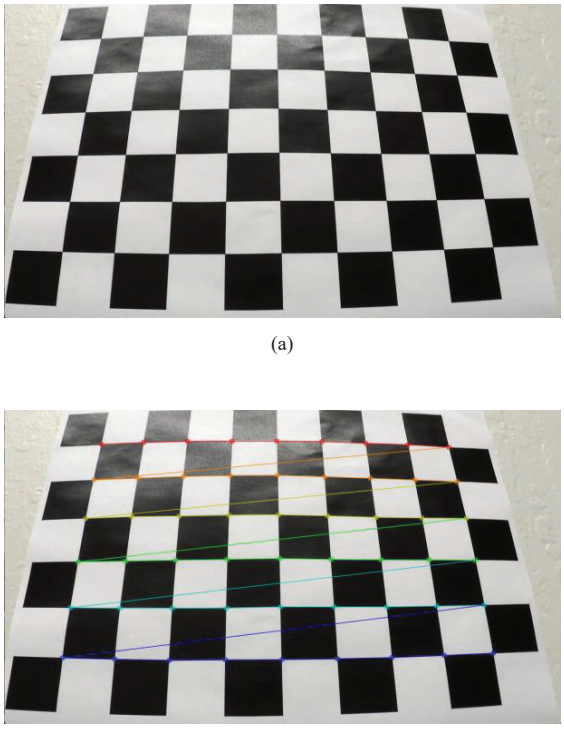
Here an example of calibration using chessboard images with 9x6 corners is provided. Suppose you already know there should be 9x6 corners in a chessboard image, then the object points should be created as:

```
# prepare object points, like (0,0,0), (1,0,0), (2,0,0), ..., (8,5,0)
nx = 9
ny = 6
```

```
objp = np.zeros((nx*ny, 3), np.float32)
objp[:,:2]=np.mgrid[0:nx, 0:ny].T.reshape(-1,2)
```

B. *Detecting Chessboard Corners*

We have 20 chessboard images at hand, so use cv2.findChessboardCorners to find their corners. Fig.1 provides an example usage of this function.



(a)

(b)

Fig. 1. Detecting chessboard corners. (a) original/distorted image; (b) detected corners

From Fig. 1 we can see that the 9x6 corners have been correctly identified in the distorted chessboard image.

C. *Un-distortion Using Camera Calibration Matrix*

Once the mapping between image points (detected corners) and object points has been established, use

(3.1) cv2.calibrateCamera to compute camera calibration matrix and

(3.2) cv2.undistort to correct the distortion of every image shot with the same camera and settings.

Here is an example.

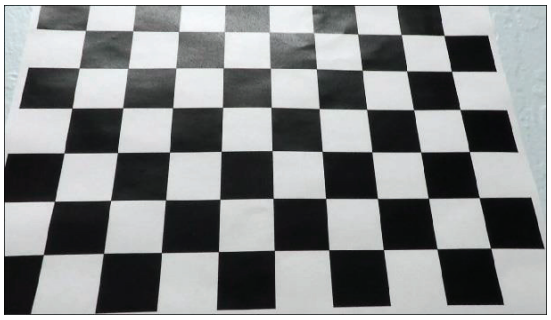


Fig.2 Undistorted version of Fig. 1(a).

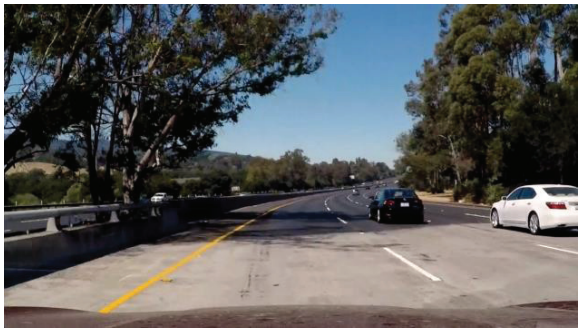
In Fig.2, the effects of un-distortion are easy to observe.

III. The Single-Image Pipeline

In this section, pipeline, i.e. a set of ordered code modules, will be described. The combination of these modules completes the task of lane detection of a single image. Detecting lane lines from a video file is simply a frame-by-frame calling of this pipeline.

A. *Preparation: from Distorted to Undistorted*

Like part C of the last section, for each image in an image sequence, we have to first un-distort it, so that we could correctly apply geometric transformation in later steps.



(a)

(b)

Fig. 2. Test image. (a) distorted; (b) un-distorted

Note in Fig. 2, differences between the distorted and undistorted versions can be discerned by the small changes around the four corners of the images.

B. Color and Gradient Thresholding

Lane lines can be considered as a combination of image features. Therefore, they can be detected using edge and color detection methods. Table 1 provides selected color and gradient thresholds after several trial and error.

TABLE I. COLOR TRANSFORM AND GRADIENT THRESHOLDING

Methods	Information	OpenCV Functions	Thresholds
Color transform	To detect whites : L channel of Luv To detect yellows: b channel of Lab	cv2.cvtColor	L channel: (210, 255) b channel: (143, 255)

Different color spaces (RGB, HSV, HLS, YCbCr, and gray) has been examined and finally, Lab color space is found to be the most effective in detecting yellow and white lane lines in different illumination conditions while rejecting noise such as shades from road guardrails and tree shadows. Gradient thresholding was also tested but it tends to include wrong edges (e.g. the intersected lines between guradrails and road surface) into the results so it is labeled unusable in this case.

(a)

(b)

(c)

Fig. 3. Test image. (a) before color thresholding; (b) after color thresholding of Table 1 methods; (c) after color thresholding and gradient thresholding with thresholds (20, 100). (b) is desirable and (c) is undesirable since it includes too many noises, in particular those from the intersection of guardrails and road surface, which will be a problem in subsequent steps.

Fig. 3 illustrates the result of color thresholding using the methods from Table 1. It can be seen that the resulting image is free from the various noises mentioned earlier. An undesirable result is also presented for better understanding of different thresholding methods.

C. Perspective Transform

In addition to perspective transform, the binary image is first filtered by a predefined region of interest (ROI). The result of perspective transform of this ROI-filtered image is shown in Fig. 4

(a)

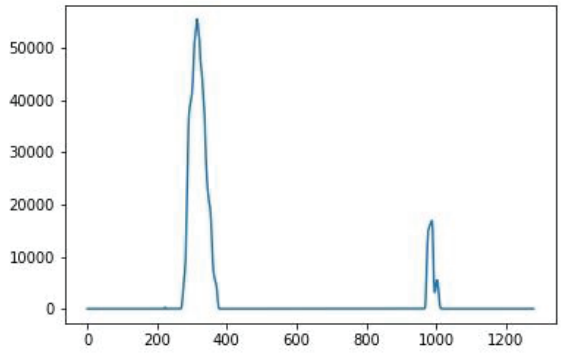
(b)

Fig. 4. Test image. (a) after cropping using ROI; (b) after perspective transformation.

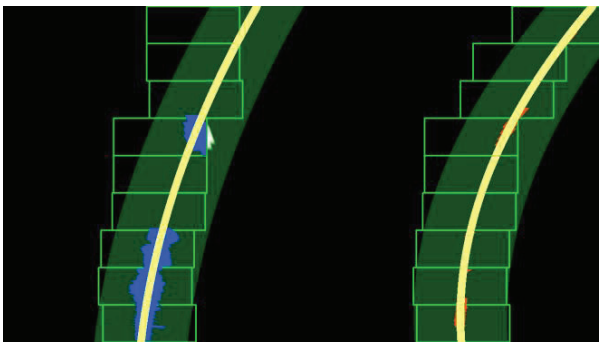
The selection of ROI is to minimize the inclusion of noise after cropping. The purpose of perspective transformation is to 'warp' the image so that as if the camera is looking at the road surface from above and the pixels in the image are now of the same scale and ready for lane detection.

D. *Lane Lines: Pixel Identification and Polyfitting Them with a 2nd-order Polynomial*

The sliding window method is implemented. The result is shown in Fig. 5.



(a)



(b)

Fig. 5. Result of the sliding window method. (a) Firstly, a histogram is used to determine the starting position of the left and right lane lines at the bottom of the image. (b) Secondly, a series of sliding fixed-sized sliding windows are applied to aggregate the pixels, whose mean position will be the points before polyfitting. The yellow curve is the fitted 2nd-order polynomial.

It can be seen from the results that the left and right lane lines are detected, with similar radius of curvature. This is can be verified by an estimation of the two radii using the method stated in the next subsection.

E. *Lane Lines: Radius of Curvature & Plotting the Lane Lines Back to the Original Images*

Note that this method is just an estimation of radius of curvature. Two assumptions are adopted:

Assumption 1: the lane center is the middle point of the image bottom. The position of the car center is calculated as an offset to this assumed lane center.

Assumption 2: conversion between pixels and world coordinates follows the rules of:

```
# Define conversions in x and y from pixels space to meters
ym_per_pix = 40/720 # meters per pixel in y dimension
xm_per_pix = 3.7/620 # meters per pixel in x dimension
```

Note here we assume that the lane line width is 3.7 meters for 620 pixels and each white line segment is 40 meters long. This combination of parameters are chosen according to U.S. government specifications for highway curvature [5].

Fig. 6. Plotting back the detected lane and overlaying it upon the original image.

In Fig.6, the fit from the rectified imaged has been warped back onto to the original image and plotted to identify the lane boundaries.

At this point, the task of detecting lane lines in a single image is complete. The source code can be accessed at: https://github.com/TMBOC/IEEEPLANS2018_LaneLineDetection. In order to run 'project.ipynb', which contains all the steps in a Jupyter notebook, you have to install relevant packages including OpenCV (version 2).

IV. Sanity Check for Video Processing

When processing a sequence of image or video, correlation between images should be taken advantage of to help detect unreasonable results (outlier detection and elimination) and smoothing. Here a simple yet effective sanity check method is implemented. The strategies taken include:

(1) (magnitude) checking the consistency of right lane curvature and left lane curvature

(2) (magnitude) checking the consistency of curvatures at epoch t and t-1, of either the left or right lanes

(3) (sign) checking the consistency of left and right polynomials

The most difficult case to check turns out to be the one when both lanes are detected straight (with very large radius of curvature).

A balance should also be struck between smoothing window length and rate of change of the radii of curvature.

The output of sanity check is the corrected polynomial coordinates, which is used to compute the corrected car offset w.r.t to the lane center and project the detected lane back to the undistorted images.

A test video output is also included in the aforementioned github repository.

V. Discussions

The presented method has some limitations and we will discuss how to improve it.

A. Known Problems of the Pipeline

- The curvature estimation needed to be improved, in particular, the curvature of the left lane.
- The projected back version of the two straight line pictures seem offset from where they are expected to be.

B. Framework for Fing Tuning

- More robust thresholding, this time possibly with gradient information should be tried.
- The projected back version of the two straight line pics seem offset from where they are expected to be. This can be overcome by fine tuning the coordinates of source and object in the perspective transformation step.
- The line fitting part (cal_polyfit function in the source code) can be improved. For example, each frame can be separated into the lower and upper halves. Once we succeed in fitting sensible lines in the lower half, we can 'look ahead' into the upper half.

This list is by no means exhaustive and various ways of decision tree methods are encouraged.

VI. Conclusions

A combined approach to detect lane lines under several road conditions in driverless navigation is presented. This approach is designed for single-camera settings and is achieved using camera calibration, image un-distortion, image color thresholding, perspective transformation, lane pixel identification and ploy fitting. Limiting factors such as illumination and radius of curvature of the testing road is considered. The real-world results show that the method is effective in normal conditions but needs further improvement in extremely challenging environments.

This work is expected to benefit the community in three ways:

(1) to show that a hardware-in-the-loop simulator can be useful in driverless car development if the current research does not allow a full hardware evaluation.

(2) to show the effectiveness of using only a front-facing camera in driverless car perception.

(3) to presents decent sets of parameters is the aforementioned procedures that will guarantee detection performance in some common yet comprehensive settings.

Acknowledgment

This work is supported by National Natural Science Foundation of China (No. 61403253) "A plug-and-play solution to All Source Positioning and Navigation problems based on factor-graph.".

References

[1] J.H. Yoo, S.W. Lee, S.K. Park, and D.H. Kim, "A Robust Lane Detection Method Based on Vanishing Point Estimation Using the Relevance of Line Segments," IEEE Trans. Intell. Transp. Syst., vol. 18(12), pp.3254-3266, Dec.2017.

[2] Y.N. Su, Y.G. Zhang, T. Lu, J. Yang, and H. Kong, "Vanishing Point Constrained Lane Detection With a Stereo Camera," IEEE Trans. Intell. Transp. Syst., vol. PP(99), pp.1-6, Nov.2017.

[3] A. Das, S.S. Murthy, and U. Suddamalla, "Enhanced Algorithm of Automated Ground Truth Generation and Validation for Lane Detection System by M^2BMT," IEEE Trans. Intell. Transp. Syst., vol. 18(4), pp.996-1005, Apr.2017.

[4] U. Ozgunalp, R. Fan, X. Ai, and N. Dahnoun, "Multiple Lane Detection Algorithm Based on Novel Dense Vanishing Point Estimation," IEEE Trans. Intell. Transp. Syst., vol. 18(3), pp.621-632, Mar.2017.

[5] http://onlinemanuals.txdot.gov/txdotmanuals/rdw/horizontal_alignment.htm#BGBHGEGC (Retrived on Feb. 2, 2018)