CS588 Final Project Report

Junzhe Wu, Sirui Wang, Weijie Lyu
University of Illinois, Urbana-Champaign
{junzhew3, siruiw4, wlyu3}@illinois.edu

Abstract

The problem of lanes detection and following is always challenging for modern autonomous driving system implementation. In this project, our goal is using vision-based approaches to detect and follow lanes in the highbay backlot. These lanes are quite different from existing dataset, so common approaches like pre-trained neural network would not achieve a decent performance. Our method based on traditional edge-detection, and it showed reasonable performance on both straight lanes and curved lanes. Also by fine-tuning the parameters, it shows good robustness in all kinds of weather and lighting conditions.

1 Project Design

1.1 Motivation

By learning topics about lanes detection in the class, we noticed that most cases shown are detection for road lanes on highway and wide city roads. Therefore, we were curious about the performance of the performance with narrow lanes and sharp turns. More specifically, we tried to implement the lane boundary detection with pid control for the route in the highbay backlot so that the vehicle would follow the route bounded by white lines.

1.2 Plan

We plan to use neural network and existed edge detection algorithms/packages to detect white edges for lanes in the backlot of highbay. Then, we intend to modify and utilize the code given for exercise 4 by Hang Cui for the pid control part for our project. We want to use the same logic as his code to chase some given waypoints that we found on the detected edges.
For our project, we decide to only use the front camera. By recording data bag of the mako topic, we will firstly test the performances of models and algorithms on these images and then adjust our parameters. We hope that we can compare their performances and choose the best method for our project. Then, when we test the performance of our project on the vehicle, we will use live images from the camera. Also, different data bag will be recorded under different weather and sunlight conditions to get a more general model or result.

2 Approaches

2.1 Convolutional Neural Network

Lanes boundaries detection is a important subject we learned in this course. The existed methods which Prof.Forsyth showed us with good performance are all using Convolutional Neural Network. So we first tried a few pre-trained network. We tried LaneATT[1] and LaneNet[2]. However, these

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

network do not have good performances on our scenario. Since most of the datasets are generated on highways or broad streets, it can not handle the sharp curve in the highbay backlot.

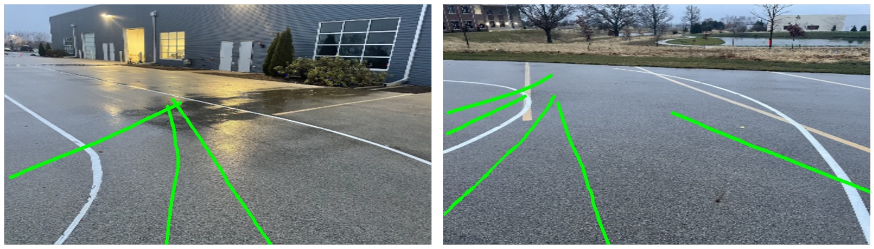

Figure 1: Result of applying LaneATT model to our image

2.2 Traditional Canny Edge Detection

As a traditional edge detection methods, canny edge detection can detect lane boundaries in any shape. On the other hand, it will also include other edges, like windows or parking space boundaries, which we do not need. We need to filter the edge detection result to get the lane boundaries only. To filter the lane boundaries, we use a mask to cover the upper part of images.

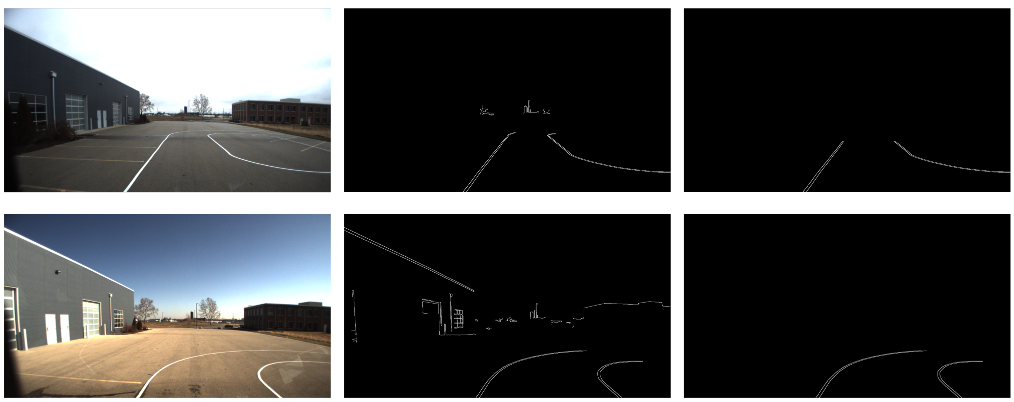

Figure 2: Canny edge detection and filtering results

2.3 Way Point Generation

For path planning, we want to generate a way point in the image frame coordinate based on the lane boundaries we detected, and then transfer this point to vehicle coordination and let the vehicle process to that point. For detailed implementation of coordinate transformation and path planning, see section 3.2 and section 3.3.

3 Implementation

3.1 Lane boundaries detection

As illustrated in the figure 2, LaneNet [2] was used to detect the pixels of lines in the original images received by camera on the vehicle, and the decteced pixels were shown in the first binary image, then DBSCAN [3] was used to cluster the detected pixels to get the second image where one color represents one cluster, finally, linear regression was used to calculate the line parameters of each cluster, and the calculated lines were shown in the third image. From Figure 1, we could find that the lane detection result from LaneNet is noisy and the detected pixels of each line is intermittent which is adverse for the stable lane following. Therefore, canny detector was utilized to detect edges in the original images. The low threshold and high threshold is quite important in this step, since we want the lane boundaries to be clear and as less noise as possible. After that, the same DBSCAN algorithm and linear regression method were applied to get the parameters of different lines. As shown in Figure 3, only two white lines were detected and other yellow lines were ignored which showed the accuracy of our method.

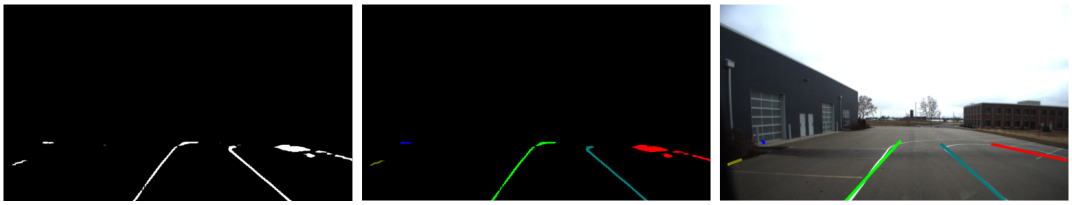

Figure 3: The lane detection result from LaneNet

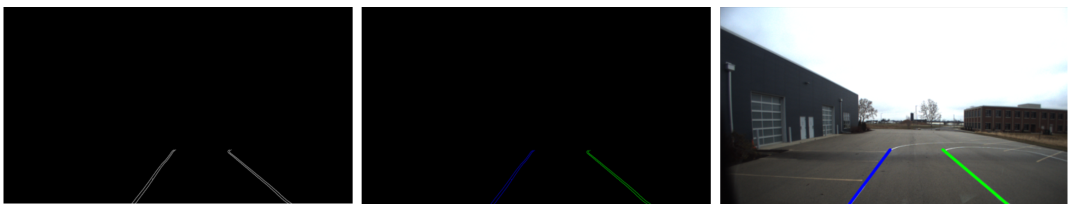

Figure 4: The lane detection result from edge detection

3.2 Path planning

As shown in the Figure 4, after the lane boundaries detection, we can get the coordinates of the top points of the two lines, then we take average of these two points and set the average point as the way point for this timestamp. However, the coordinate of this way point is in pixel coordinate system, we need to transfer it to the coordinate in the 3D coordinate system. As illustrated in the Figure 5, the blue x axis, y axis and z axis are the car coordinate system we set, and the orange x axis, y axis and z axis are the camera coordinate system. The origin of these two coordinate systems are the center of camera. Accoding to our measurement, the height of camera center h = 1.51m and the angle between blue z axis and orange z axis θ = 9.25 degree. Since the detected lines are all on the ground plane, the y coordinates of the lines are all h, we only need to calculate x and z according to the pixel value u and v. In order to project the points in the car coordinate system into pixel plane, firstly we need to calculate the coordinate of points in the camera coordinate system using the rotation matrix R, then we need to use the intrinsic matrix from the camera calibration to realize the projection.

$$
\begin{gathered}
P \cdot R \cdot \begin{bmatrix} x \\ h \\ z \end{bmatrix} = \begin{bmatrix} ut \\ vt \\ t \end{bmatrix} \\
\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ h \\ z \end{bmatrix} = \begin{bmatrix} ut \\ vt \\ t \end{bmatrix}
\end{gathered} \tag{1}
$$

$$
\begin{cases} u = \dfrac{f_x \cdot x}{-h\sin\theta + z\cos\theta} + c_x \\ v = \dfrac{f_y \cdot (h\cos\theta + z\sin\theta)}{-h\sin\theta + z\cos\theta} + c_y \end{cases} \tag{2}
$$

$$
\begin{cases} z = \dfrac{h\,(c_y\sin\theta - v\sin\theta - f_y\cos\theta)}{(f_y\sin\theta x - v\cos\theta + c_y\cos\theta)} \\ x = \dfrac{h \cdot (c_x - u) \cdot f_y}{(f_y\sin\theta x - v\cos\theta + c_y\cos\theta) \cdot f_x} \end{cases} \tag{3}
$$

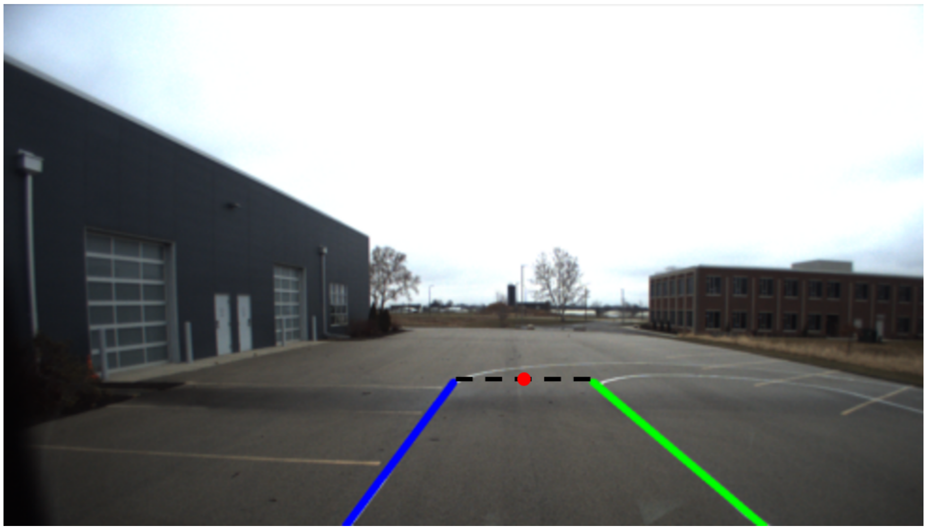

Figure 5: Illustration of designed car coordinate system and camera coordiante system

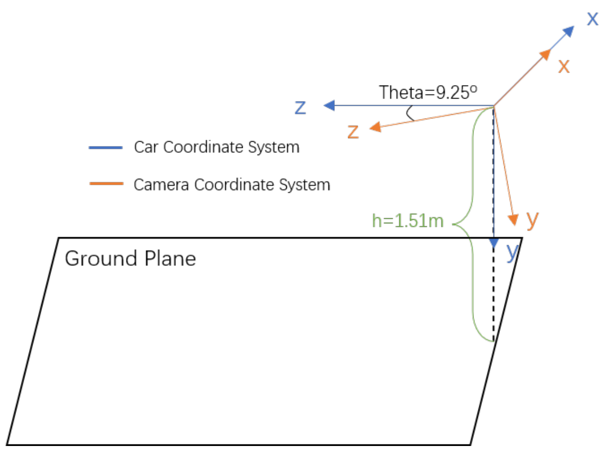

Figure 6: Illustration of designed car coordinate system and camera coordiante system

4 Result and Analysis

4.1 Result

You can check out our codes and video demo of the result via github repository (https://github.com/weijielyu/CS588_Final_Project).

Generally speaking, the vehicle can detect all the lane boundaries pretty well. It can follow the most parts of the route in the backlot. However, the vehicle sometimes cannot move between two boundaries of the lane, especially when it sees a sharp turn or circular lanes. For the straight lanes, the result is pretty good and the vehicle can always stay in the lane.

4.2 Challenges and Discussion

The unstable performance of the vehicle may attribute to a few factors.

Firstly, the camera. We tried to adjust the angle of the camera but it was still not optimal for us. We want to get a large acute angle between the ground and the camera so that we can extract more information from the images. Also, the small angle may cause the car cannot see the edges when having a sharp turn when the car's speed is too fast, thus, we set the speed pretty low. Moreover, due

to the angle of the camera, we have to let the car to turn a bit early so that the vehicle can always detect lane boundaries in the images. We think that changing the angle between camera and the ground or adding cameras on the front bumper may lead us to better results.
Furthermore, the weather condition and the layout of the backlot is another factor. When the sunlight is too strong, the reflection of the water on the ground can make our code unable to detect the lane boundaries. More importantly, the yellow lines of the parking space may cross the white lines of the route. Therefore, our detector is hard to find the correct edges, especially when sun is shining directly on those yellow lines as figure 7 shows. In this situation, those yellow lines in the images has no difference with the white edges of our proposed route. We tried to modify the threshold of our method to filter those irrelevant lines, but it does not work. Therefore, we choose to keep our original implementation and the performance of turning is not accurate. To solve this problem, we probably need to use better method or run the vehicle in a more simple environment.

Figure 7: When the sunlight is too strong, the yellow parking lots lines will be detected as lane boundaries

5 Reference

[1] Tabelini, L., Berriel, R., Paixao, T. M., Badue, C., De Souza, A. F., & Oliveira-Santos, T. (2021). Keep your eyes on the lane: Real-time attention-guided lane detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 294-302).

[2] Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M., & Van Gool, L. (2018, June). Towards end-to-end lane detection: an instance segmentation approach. In 2018 IEEE intelligent vehicles symposium (IV) (pp. 286-291). IEEE.

[3] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd (Vol. 96, No. 34, pp. 226-231).