The Architecture of a Driverless Robot Car Based on EyeBot System

Shuangquan Sun¹, Jingwen Zheng¹, Zihan Qiao², Shanqi Liu³, Zihan Lin¹ and Thomas Bräunl⁴,*

¹Dept. of Auto., University of Sci. & Tech. of China, China
²Dept. of EE, Zhejiang University, China
³Dept. of CSE, Zhejiang University, China
⁴Dept. of EECE, The University of Western Australia

*Thomas Bräunl: thomas.braunl@uwa.edu.au, +61 8 6488 1763

Abstract. This paper presents the system architecture of a driverless robot car, designed to participate in the Carolo-Cup, a competition regarding automated model vehicles. We describe an implementation that completed the different tasks in this competition on our EyeBot platform in detail. EyeBot has one RGB camera as well as three infrared distance sensors, and it's powered by Raspberry Pi 3B. We developed the lane detection algorithm using the OpenCV library and completed the traffic sign recognition task based on SVM which can be used offline. Experimental and simulation results recorded in real-time are also reported. The test result showed that our programs can run at high speed to achieve stable motion control in real time and complete all the tasks in the competition. The highlight of our work is that the whole system can run on a platform with limited computing resources.

1. Introduction

Autonomous vehicles have been the subject of some research words in fields including robotics, computer vision, and machine learning. The well-known DARPA Grand Challenge [1] has triggered a great deal of research by confronting academics with real practical situations. Other competitions have also been set up, such as the Intelligent Vehicle Future Challenge (IVFC) [2], the Audi Autonomous Driving Cup [3] and the Carolo-Cup [4].

In this article, we present the complete system architecture of the driverless robot car that completed the different scenarios in Carolo-Cup. In section 2, the model car platform EyeBot [5], the software EyeSim [6], the operating system RoBIOS [7], the test track and some preparation work are presented. Section 3 and 4 gives a detailed view of our system architecture, covering all the implemented modules: lane detection using OpenCV [8] and traffic sign recognition. These functions are discussed separately in the following sections, including algorithms and experimental results. Finally, Section 5 concludes the paper and provides an overview of future work.

2. Platform & preparation work

2.1. EyeBot & EyeSim & RoBIOS

EyeBot is the name of a robot family developed by UWA. We used SoccerBot, a member of the EyeBot family which is shown in figure 1. It is powered by a Raspberry Pi 3B in combination with a
self-developed IO-Board EyeBot7, which handles all motor and sensor control and communicates with the Raspberry Pi via USB. EyeBot7 and RoBIOS provide an API to assist in robot program design.

EyeSim is a multirobot, multi-tasking simulation software that allows realistic simulation of mobile robots on MacOS, Windows, and Linux. Its snapshot is shown in Figure 1. EyeSim uses the Unity [9] physics engine and also provides VR applications for Oculus Go [10] and HTC Vive [11].

![Figure 1. SoccerBot S4 and its sensors.](image1)

![Figure 2. Snapshot of EyeSim: Simulation Windows.](image2)

EyeBot's operating system is called RoBIOS (Robot Basic Input-Output System). It comprises a user interface for robot applications on the Raspberry Pi, a set of libraries for various I/O and sensor data processing functions, plus numerous demonstration applications.

2.2. Indoor testbed & simulation

We built an indoor testbed for research, and due to the smaller size of SoccerBot robots and limited space in the laboratory, the sizes of the track and traffic signs were halved. This track consists of a double-lane road, six curves, an intersection and two parking lots. We also built a similar traffic environment in the EyeSim simulator for testing. Figure 3 shows the test track in reality and simulation.

![Figure 3. Test track in reality and simulation.](image3)

3. System architecture

In order to perform all the tasks required in Carolo-Cup, we need to bring several sensors and processing modules into play. Figure 4 shows the global architecture of the system implemented on the experimental platform. A single computer running a Linux OS was used to handle the different sub-modules in multiple threads.

![Figure 4. Global architecture of the system.](image4)
3.1. Lane detection based on OpenCV

Lane detection is a fundamental task in autonomous driving. In this section, we discuss the method we used to detect lanes running on the EyeBot platform using OpenCV. The method is modified due to the limitation of computing power, while it maintains good flexibility for different scenarios like a straight road, curves, crosswalk, and intersection. Our algorithm can run at around 30fps, which is fast enough to achieve a satisfying lane following ability. Our lane detection algorithm contains the following steps:

- **Steps of lane detection.** Images captured from the camera [Figure 5] need to be preprocessed for features extraction. Several techniques are used:
  - Set Region of Interest (ROI): remove the top part of the original image that is useless for lane detection to accelerate the image processing speed. [Figure 6]
  - Greyscale Transform: Convert color image to grayscale image.
  - Gaussian Blur: This step reduces noise in the image.

Then we search for specific edges in the image to extract the road feature. We use:

- Canny Edge Detection. It is a computational approach to detect edges in an image [12]. We used the result of canny edge detection to find edges of the lane. [Figure 7]
- Hough Transform. It is a method can be used for line detection and general curve fitting [13]. We use it to find all possible straight lines to determine the two inner edges of the lane.

The final step is feature validation. Considering the limited field-of-view of the camera, we implemented a new way to determine the left and right edges of the lane. First, select the possible lines of the left edge and right edge by the slope of the lines. Then, for each part of the image, calculate the distance from the line to the top left or top right spot of the region of interest which is shown in Figure 8. The lines with the longest distance to the left or right corner spot are considered as the inner edge of the lane markings. Figure 9 demonstrated the result of lane detection.

### Detection of different scenes

As for a curved road, one of the most common methods is curve fitting. Curve fitting is a mathematical technique that is widely used in engineering applications. It consists of fitting a set of points to a curve using Lagrange Interpolation Polynomial [14]. However, it is computationally intensive and therefore cannot be applied to the EyeBot platform while maintaining a reasonably high frame rate.

In our scenario, when the robot drives on a curve, often only one edge is visible in the camera as Figure 10 and Figure 11 show. We consider the curve as a continuous polyline. The information of angle and position of the detected edge is saved each frame. When the robot drives into a curve and one edge of the lane becomes invisible, we assume that the current edge position is the same as in the last detected position.

![Figure 8. Edges (blue), distance to spot (red).](image1)

![Figure 9. Determined inner edge of the lane.](image2)

![Figure 10. Visible solid line for the right curve.](image3)

![Figure 11. Visible dash line for the left curve.](image4)
To detect the crosswalk, we use the number of lines detected in a frame. When the crosswalk is in the sight of the camera as Figure 12 shows, the number of detected lines will increase to around fifteen, and after passing the crosswalk, it will reduce to normal level.

To detect the intersection, as Figure 13 shows, when the robot arrives at the intersection (① and ②), the number of detected lines reduced to zero due to the temporarily missing edges on the two sides. Based on this information, we can decide if there is an intersection ahead.

In an experiment, we let the car drive from the blue spot to the red spot on the left in figure 14 and printed the number of detected left or right edges in a graph. From the graph, we can see a sharp peak when the robot reaches the crosswalk.

![Figure 12. Detected lines before the crosswalk.](image1)

![Figure 13. Image before the intersection.](image2)

![Figure 14. Test track and number of lines detected from blue spot to red spot.](image3)

3.2. Traffic sign recognition task

3.2.1. The HOG feature. There are many classical methods which can complete the traffic sign recognition task. The most common of hand-crafted features are SIFT [15], HOG [16], LBP [17], etc. After describing traffic signs, classifiers are learned based on the feature representing for traffic sign detection and classification, such as support vector machines (SVM) [18] and random forest classifier [19], [20].

For our application, we found the HOG feature can gain the acceptable accuracy for our driverless simulation task after particular optimization of the current scene. Moreover, its reaction time is short enough for real-time needs. The libraries that the algorithm relies on are limited, so it is easy to be installed on the Raspberry Pi. Based on these reasons, we chose the HOG feature to perform the traffic sign detection and recognizing task.

3.2.2. The influence of different parameters in SVM. To optimize the algorithm for our needs, we must change some parameters and need to add some methods for block normalization. This can help with generalization and the speed of the function. We mainly modified these parameters to adapt the model to our situation — the C for the SVM (penalty term), and thresholding the singular values of the filters and the size of the scan windows.

Firstly, for the C of the SVM, the below pictures tell how much it should be to avoid misclassifying each training example. In our situation, the parameter C can be large, because our training datasets do not have too many noises. The accuracy sensitivity to C shows in Figure 15.
As for the threshold of the filter, we keep it small, because we want to keep more details. The accuracy sensitivity to the filter shows in Figure 16.

The windows size is more complicated as it depends on the input image size. In our situation, the input image size is 320x240 (QVGA). So, the size of the scanning window should be smaller than 80. However, when the size of scanning windows is too small, the performance of the model deteriorates, especially in the presence of noise in the training dataset. Figure 17 shows window size sensitivity. Figure 18 shows the sensitivity to norm.

4. Experimental results and discussions

4.1. Traffic sign recognition

4.1.1. HOG feature models. The result images of our HOG feature models are shown in Figure 19. They primarily reflect the shapes and appearances of the given traffic sign images, so we presume the models we trained are correct.

| Traffic signs     | Precision | Recall | F  |
|-------------------|-----------|--------|----|
| Stop              | 82.61%    | 61.29% | 70.37% |
| Pedestrian        | 100%      | 73.33% | 84.62% |
| Park              | 100%      | 50.00% | 66.67% |
| Speed limit       | 100%      | 63.64% | 77.78% |
| Speed limit cancel| 100%      | 42.11% | 59.26% |
| Give way          | 100%      | 60.00% | 75.00% |

4.1.2. The recognition results. Moreover, the accuracy of our model is tested in a testing dataset we build based on our map; the main performance is measured by the accuracy, recall, and F-measure, the result of our model is shown in Figure 20 and Figure 21.
Figure 21. The recognition results.

4.1.3 Execution time. The most important part of this model is the executing time, as real-time performance is required for the robots. The average time of executing the main control loop is 340ms. However, the time varies from 240ms to 408ms, which is quite stable for most of the time. Combining the accuracy and executing time results, we can assume that this model meets our soft real-time and accuracy requirements for this application.

5. Conclusion and Future Work
The system architecture described in this paper was successfully implemented and demonstrated within the context of Carolo-Cup 2018, both on real EyeBot/SoccerBot robots, as well as in the EyeSim simulation environment. The implemented modules worked reliably and were capable of handling complex scenarios, despite the limited computational and memory resources of the chosen hardware.

In the future, we plan to improve the robustness and accelerate the speed of the recognition process which is crucial for racing competition. Besides, considering the basic functions we have, we can implement the communication function into our system. The possibilities offered by wireless communications for an automated distributed understanding of driving scenes are numerous and still largely unexplored.

Acknowledgments
This work was supported by The University of Western Australia. The authors would like to thank Alex Arnold, Kai Li Lim, Nicholas Burleigh, and Michael Finn.

References
[1] Buchler, M., Iagnemma, K., & Singh, S. (1973). The 2005 DARPA grand challenge: the great robot race. Industrial Robot An International Journal, 35.
[2] NovAtel Inc. Driverless in China | Velocity Magazine | NovAtel. [Online], Available: https://www.novatel.com/tech-talk/velocity/velocity-2015/driverless-in-china/
[3] AUDI AG 2018, June Audi Autonomous Driving Cup 2018 Rulebook. [Online], Available: https://www.audi-autonomous-driving-cup.com/wp-content/uploads/2018/06/2018-06-13_en_Rulebook_2018_V2.0EU.pdf
[4] Technische Universität Braunschweig, Carolo-Cup Regulations 2018. [Online], Available: https://wiki.ifr.ing.tu-bs.de/carolocup/system/files/Carolo-Cup%20Regulations_1.pdf
[5] Braunl T 1999 EyeBot: a family of autonomous mobile robots ICONIP’99. ANZIIS’99 & ANNNS’99 & ACNN’99. 6th Intl. Conf. on Neural Information Proc., pp. 645-9.
[6] Braunl Thomas, EyeSim VR - Unity Based EyeBot Simulator. [Online]. Available: http://robotics.ee.uwa.edu.au/eyesim/
[7] Thomas Bräunl, Remi Keat, Marcus Pham, RoBIOS - Mobile Robot Library. [Online]. Available: http://robotics.ee.uwa.edu.au/eyesim7/Robios7.html
[8] OpenCV team. (2018). OpenCV library. [Online]. Available: https://opencv.org/
[9] Unity Technologies. (2018). Unity. [Online]. Available: https://unity3d.com/
[10] Oculus. Oculus Go. [Online]. Available: https://www.oculus.com/go/
[11] HTC Corporation. (2011-2018) VIVE™ Australia | Discover Virtual Reality Beyond Imagination. [Online]. Available: https://www.vive.com/au/
[12] Canny J 1986 A computational approach to edge detection *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8(6) 679–98

[13] Duda, R. O., & Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1), 11-15.

[14] Yang Y 2013 Polynomial curve fitting and lagrange interpolation *Mathematics and Computer Education* 47(3) 224-30

[15] Meuter M, Nunn C, Gormer S M, Muller-Schneiders S and Kummert A 2011 A decision fusion and reasoning module for a traffic sign recognition system *IEEE Transactions on Intelligent Transportation Systems* 12(4) 1126-34

[16] Dalal N and Triggs B 2005 Histograms of oriented gradients for human detection 2005 *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR ’05)*

[17] Ojala T, Pietikainen M and Maenpaa T 2002 Multiresolution gray-scale and rotation invariant texture classification with local binary patterns *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(7) 971-87

[18] Greenhalgh J and Mirmehdi M 2012 Real-time detection and recognition of road traffic signs *IEEE Transactions on Intelligent Transportation Systems* 13(4) 1498-506.

[19] Zaklouta F and Stanciulescu B 2012 Real-time traffic-sign recognition using tree classifiers *IEEE Transactions on Intelligent Transportation Systems* 13(4) 1507-14

[20] Guo J, Lu J, Qu Y and Li C 2018 Traffic-sign spotting in the wild via deep features 2018 IEEE Intelligent Vehicles Symposium (IV) 120-5