Real Time Lane Detection and Lane Following Algorithms for Autonomous Driving with Quanser QCar

Khush Lalchandani
Liberal Arts and Science Academy
Austin, United States of America
khush.lal200@gmail.com

Aarav Paryemalani
BASIS San Antonio - Shavano Campus
San Antonio, United States of America
aaravpary@proton.me

Gideon White
University of Texas at Austin
Austin, United States of America
gideon.white@utexas.edu

***Abstract*—This research, conducted under the NSF Research Experience and Mentoring (REM) program, focuses on the development and implementation of real-time lane detection and lane-following algorithms. The Quanser QCar serves as the experimental platform for our work. We explore various methods for lane detection, including traditional image processing techniques and advanced clustering algorithms like DBSCAN, to accurately identify lane lines in real time. Our approach involves image preprocessing steps such as resizing, birds-eye transformation, and color masking to enhance lane detection accuracy and computational efficiency. Additionally, we incorporate side camera data to improve lane-keeping performance. For lane following, we implement the Pure Pursuit algorithm, optimized through extensive experimental trials to balance steering response and lane-following accuracy. The integration of a Proportional-Integral-Derivative (PID) controller ensures precise speed control to further enhance the vehicle's stability and safety.**

***Index Terms*—lane detection, autonomous driving, Quanser QCar, image processing, DBSCAN, Pure Pursuit, PID controller**

I. Introduction

The development of autonomous vehicles (AVs) has advanced significantly in recent years due to progress in sensor technology, control systems, and computational power. Autonomous vehicles aim to improve road safety, reduce traffic congestion, and reduce the almost 1.2 million deaths that happen every year on the road [1]. Lane detection and lane assistance technology are crucial components of autonomous vehicles to ensure a vehicle can maintain its position and perform safe lane changes [2].

Our research as part of the NSF Research Experience and Mentoring (REM) program focuses on creating an algorithm for lane detection and lane following. Our goal was to quickly process front and side camera image data from the car and provide steering commands to find and follow the lane lines. The testing platform for our experiments is the Quanser QCar. The QCar is equipped with four 2D 350 CSI cameras and LiDAR with 2-8K resolution and a 10-15 Hz scan rate. Its processing unit includes a 2 GHz quad-core ARM Cortex-A57 64-bit CPU, a 2 GHz dual-core NVIDIA Denver 2 64-bit CPU, and a 256 CUDA core NVIDIA Pascal™ GPU architecture, providing 1.3 TFLOPS (FP16) of processing power [3]. The experiments were conducted on the Quanser Self-Driving Car Studio model track.

National Science Foundation and University of Texas at Austin

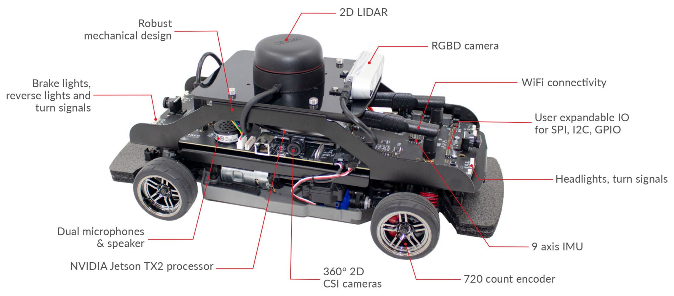
2D LIDAR
Robust mechanical design
RGBD camera
WiFi connectivity
Brake lights, reverse lights and turn signals
User expandable IO for SPI, I2C, GPIO
Headlights, turn signals
Dual microphones & speaker
9 axis IMU
NVIDIA Jetson TX2 processor
360° 2D CSI cameras
720 count encoder

Fig. 1. QCar Specifications

II. Related Work

A. Speed Control

Proportional-Integral-Derivative (PID) controllers are used in path tracking for controlling lateral position and heading. PIDs are implemented by calculating an error value and applying corrections based on proportional, integral, and derivative terms to keep as close to a set point as possible. The most common PID tuning methods are the Ziegler-Nichols methods. Their Oscillation Method involves adjusting system gain until oscillations occur, from which PID parameters are derived based on the system's response graph. The Cohen-Coon Method, on the other hand, uses the step response of an open-loop system to determine PID parameters [4].

B. Lane Detection

Traditional lane detection systems consist of multiple steps including image processing, feature extraction, lane fitting, and lane following. Vision-based lane detection is an important aspect of autonomous driving systems [5]. The two main categories of lane detection in computer vision are the model-based method and the feature-based method. The model-based approach has a predefined curve model to match the line features within an image, and the parameters of the model are then calculated to find the lane line. Y. Su et al. [6] introduced a vanishing point-constrained lane detection method using a stereo camera, which achieved high detection performance for both straight and curved lanes without assuming any parametric lane model. This method involves a computationally expensive graph-search procedure to find optimal paths for lane borders but was highly accurate. J. Wang and X. An [7] proposed a multi-step curved lane detection algorithm based on a hyperbola-pair model. Although these methods are accurate, they require significant computational resources, making them unsuitable for high-speed real-time applications on our model.

Feature-based methods focus on extracting road features such as color [8] or edges [9] from camera images. Several methods based on image characteristics have been proposed. In [10], the authors utilized spatiotemporal images to identify lane points and then fitted these points to a cubic curve. Aly [11] presented a real-time method for detecting lane markers in urban scenarios by generating a bird's eye view image of the road and using the RANSAC line fitting method to give an initial estimation for fitting Bezier Splines to the lane line.

The K-means clustering algorithm is an unsupervised algorithm used to divide an area of interest from the background. Previous studies have demonstrated that K-means can effectively group data into clusters to find a lane line [12]. However, K-means requires the number of input clusters to be predefined, which would not be possible for real-time lane detection scenarios. An alternative to K-means is DBSCAN (Density-Based Spatial Clustering of Applications with Noise). A large challenge when processing an image is noise, which is caused by shadow interference and inconsistencies in road and background colors [13]. DBSCAN is beneficial as it can handle clusters of arbitrary shapes and noise. Additionally, DBSCAN has adjustable parameters that allow you to fine-tune to specific requirements without compromising computational efficiency [14].

C. Lane Following

Path tracking is an important component in determining the steering angle once a lane line is detected. There are many methods to address this including the Pure Pursuit, Stanley, and MPC, each with its own strengths and weaknesses. The Pure Pursuit algorithm is widely used due to its simplicity and effectiveness in low-speed scenarios. This method involves calculating a target point on the path ahead of a vehicle and adjusting the steering angle to follow this point [15]. Myung-Wook Park et al. [16] proposed an adaptive controller based on the Pure Pursuit method, which adjusts the lookahead distance based on speed and lateral position deviation.

Developed by Thrun et al. [17], the Stanley controller was created for use in the DARPA Grand Challenge. His method uses a non-linear control strategy that minimizes cross-track error and heading error by adjusting the steering angle. This method is more effective in high-speed scenarios but can overshoot corners due to its correction mechanism. Model Predictive Control (MPC) is another advanced method used for path tracking. MPC involves predicting future vehicle states using a dynamic model and optimizing control inputs to minimize a cost function [18]. This method is useful for complex driving scenarios and high-speed maneuvers. However, the computational complexity of MPC can be a limitation for real-time applications in our algorithm [19].

III. Methods

A. Image Preprocessing

In this study, we used many image-processing techniques to improve lane detection.

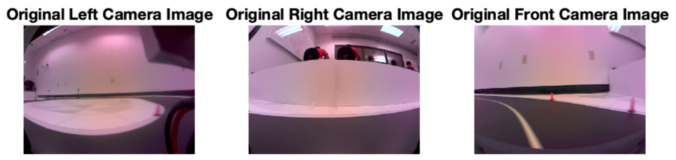
Original Left Camera Image
Original Right Camera Image
Original Front Camera Image

Fig. 2. Original Camera Images

The first step was to resize the image to have quicker processing times. The original image size of 480x640 pixels was reduced to 48x64 pixels through subsampling, where every 10th pixel was selected.

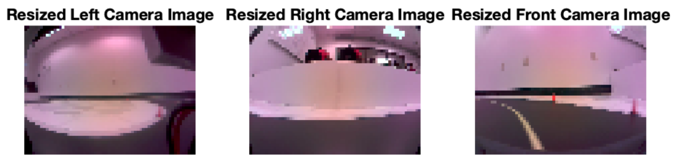
Resized Left Camera Image
Resized Right Camera Image
Resized Front Camera Image

Fig. 3. Resized Camera Images

The next step involved the birds-eye transformation to convert the front camera feed into a top-down view. To create an accurate birds-eye view without distortion, we measured the camera intrinsics and calculated the extrinsic parameters using the Camera Calibration app in MATLAB [20]. The intrinsics included the sensor specifications, such as height, pitch, yaw, and roll. The extrinsics were calibrated using a checkerboard to find the Focal Length and Principal Point.

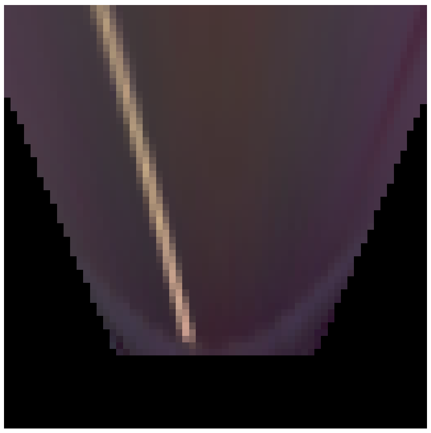

Fig. 4. Birds-eye Front Camera

Following the birds-eye transformation, a color mask was applied to convert the image to a binary format. Using the MATLAB color thresholder app, we generated a mask that assigned binary values to each pixel based on its RGB values. Pixels falling within a range that included lane line colors were indexed as one, and all others as zero, resulting in a black-and-white image [21].

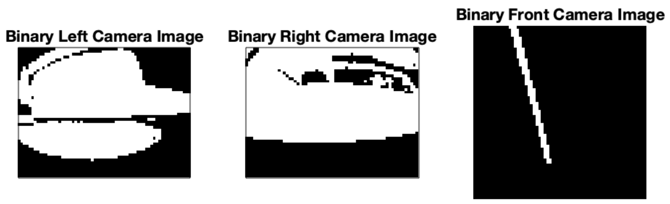

Fig. 5. Binary Camera Images

The final image processing step was binning to further reduce the image size and computational expense. Binning involved dividing the 64x64 image into multiple 2x2 clusters and selecting the highest value within each cluster to form a new array. This process reduced the front image size from 64x64 to 32x32 pixels while making sure to keep all of the lane line pixels.

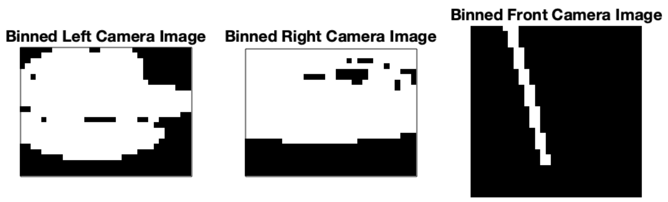

Fig. 6. Binned Camera Images

Once the image was resized to 32x32 pixels, the non-zero pixel coordinates were extracted for DBSCAN. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that we used to identify lane lines in our autonomous vehicle system due to its ability to find clusters of arbitrary shapes and handle noise in the data [14]. The Matlab DBSCAN algorithm identifies clusters by examining the density of data points which would make it effective in recognizing a lane line.

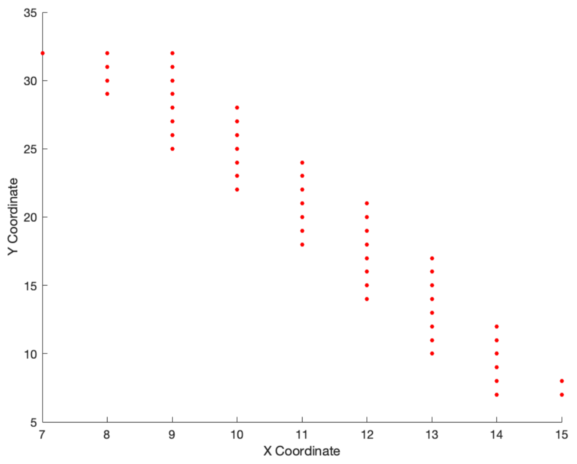

Fig. 7. DBSCAN Clusters for Front Camera

The DBSCAN parameters were chosen based on experimental analysis. Epsilon (), which defines the maximum distance between two samples for one to be considered as in the neighborhood of the other, was set to 2 based on our image resolution. The minimum number of points (minPts) required to form a dense region was set to 3. Our minimum cluster size was set to 10 points and our maximum to 80 to ensure that irrelevant clusters were filtered out. The remaining cluster with the largest y-coordinate range was selected as the correct cluster as this would most likely represent the lane line.

The coordinates were then extracted for the lane line cluster, rotated by 90 degrees, and translated up to align the vehicle's orientation with the x-axis and set the vehicle position to (0,0). A third-degree polynomial was fitted to the points. The third-degree polynomial was chosen for its ability to model complex curves without excessive oscillations, allowing the vehicle to follow the lane smoothly [22].

In addition to the front camera, we incorporated side cameras into our system to improve lane-keeping accuracy. The side cameras provided information about the vehicle's proximity to the lane edges. The lowest y-coordinate was identified from the binned image of the side cameras to find the distance to the lane edge. By comparing these values to a predefined threshold, we assessed whether the vehicle was too close to the lane edges. Based on how close the edge was to the vehicle, we applied a steering correction to ensure a consistent lane following with a smooth adjustment.

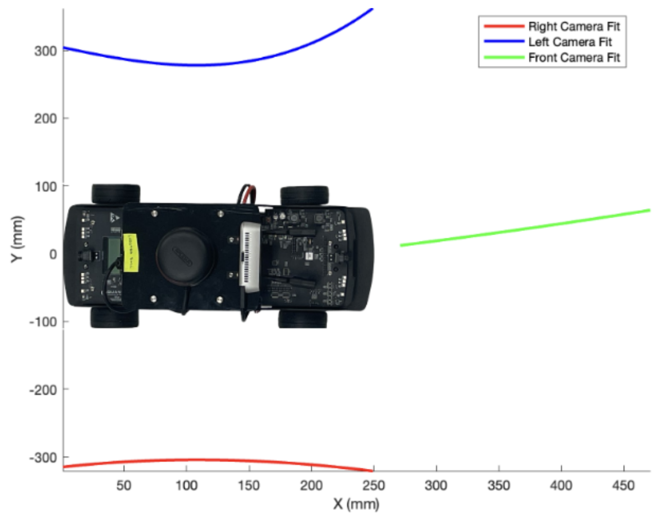

Fig. 8. Final Lane Lines

The Pure Pursuit algorithm was used to provide the vehicle with a steering input using the polynomial-fitted lane line as the reference path [15]. The lookahead distance, which is the distance ahead of the vehicle that the algorithm uses to calculate the steering angle, was determined through experimental trials. A short lookahead distance caused shaky movement which would not be safe for the driver while a long lookahead distance led to poor lane-following accuracy and eventually veering off the course.

$$\begin{aligned} L_{fw} &= \sqrt{\mathrm{dx}^2 + \mathrm{dy}^2} \\ \mu &= \arctan\left(\frac{\mathrm{dy}}{\mathrm{dx}}\right) \\ R &= \frac{L_{fw}}{2\sin(\mu)} \\ \mathrm{arcLength} &= 2\mu R \\ t &= \frac{\mathrm{arcLength}}{\mathrm{speed}} \\ \mathrm{steeringAngle} &= \frac{2\mu}{t} \end{aligned} \tag{1}$$

B. Speed Control

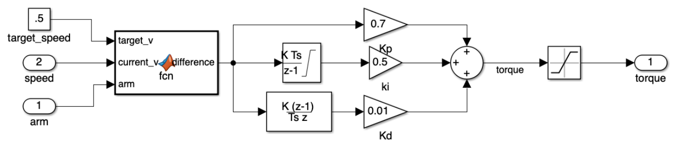

Fig. 9. PID Controller

To control the speed of our system, we implemented a Proportional-Integral-Derivative (PID) controller. To prevent the risk of windup from the Integral error, a separate function was used to start the calculations of torque value only when the vehicle was operational and ready to move. To tune the PID, we developed a plant model using an auto-regressive model with exogenous inputs (ARX) in MATLAB [23]. The ARX model was trained using historical torque and speed data from the vehicle. At each step, the previous velocity and torque were utilized to determine the current velocity without a time delay.

IV. Results

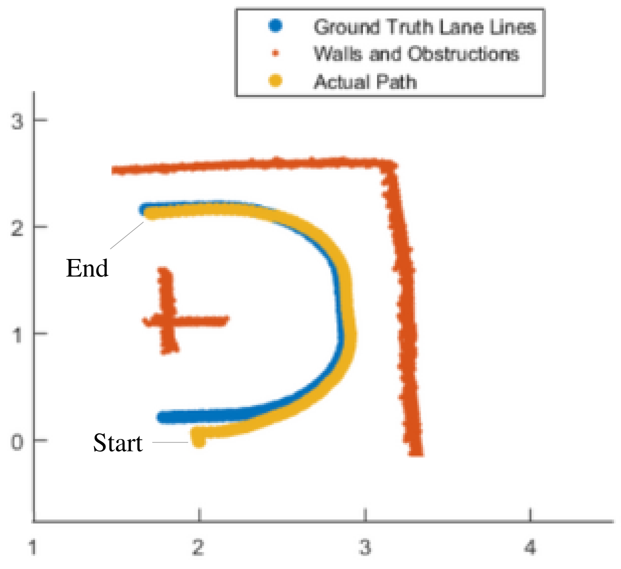

Fig. 10. Vehicle Path on Track

The lane-following algorithm for the QCar demonstrated high accuracy, deviating from the lane line by a maximum of 5 centimeters. The vehicle's speed was consistent and reached our target velocity of .5 m/s.

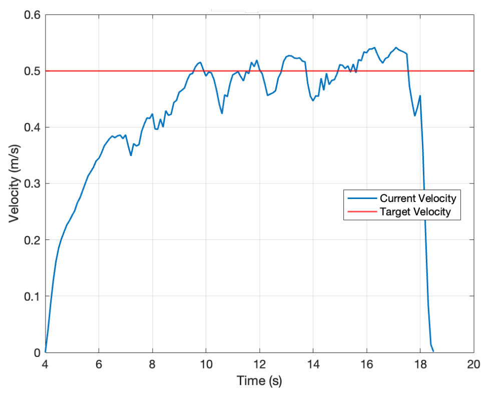

Fig. 11. Vehicle Velocity Graph

We spent the majority of our time troubleshooting the process for the birds-eye view. Some problems we encountered included the birds-eye images being initially warped due to poor calibrations, incorrect measurement units, and broken camera extrinsics (MathWorks). Finally, we discovered that the processing time of the birds-eye function on the full size image was leading to lag and the car was receiving improper steering commands. By resizing the camera input before the birds eye function was called, the average processing time for each image went from 0.4 seconds to .01 seconds. This led to the QCar being able to properly correct its course to follow the center lane line on the Quanser track.

V. Future Work

Further work would focus on enabling the QCar to consistently stay in the center of one lane on the Quanser track. We would then move forward with changing lanes, both from left to right and right to left, as well as choosing a path at an intersection. Because the intersections on the Quanser track do not have lines, this would lead us to process different types of lanes and roads, such as roads with a dotted lane line or faded/non-existent lane lines. Finally, we would begin to collect data and implement processes for traffic control infrastructure such as stop signs, traffic lights, and construction cones.

VI. Conclusion

The research successfully demonstrated effective image processing techniques for autonomous vehicle lane tracking. First, it was crucial to resize the QCar camera input to be small enough for our algorithm to process the image quickly. Our image was then converted from a fish-eye perspective to a birds-eye view and a binary mask of this birds-eye view was created. Clustering of the resulting points using a DBSCAN algorithm was completed at an average of 0.01 seconds per image. A pure pursuit algorithm used these clustered points to create and follow a line modeled after the lane. The final output of a steering angle was able to match the real-time location of the QCar and enabled it to follow the lane lines in a timely and accurate manner.

VII. Acknowledgements

This project would not have been possible without the financial support of the National Science Foundation and the Mobility Systems Lab at The University of Texas at Austin. Under the mentorship of Dr. Junmin Wang, Yung-Chi Kung, Robin Sambuis, and Ammaar Khatlani, we developed invaluable skills for our lives and future careers. We would also like to thank Shante Tyus for her coordination with the university and our fellow cohort members Shreehan Pippalla and Aayan Ghani for their work with speed control on the Quanser Qcar.

References

[1] W. H. Organization, “Road traffic injuries,” World Health Organization, 2023. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[2] J. Tao, B.-S. Shin, and R. Klette, “Wrong roadway detection for multi-lane roads,” *Lecture notes in computer science*, pp. 50–58, 01 2013.

[3] Quanser, “Sensor-rich autonomous vehicle - the qcar from quanser,” www.quanser.com. [Online]. Available: https://www.quanser.com/products/qcar/

[4] T. Yucelen, O. Kaymakci, and S. Kurtulan, “Self-tuning pid controller using ziegler-nichols method for programmable logic controllers,” *IFAC Proceedings Volumes*, vol. 39, p. 11–16, 01 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015325866

[5] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, “Recent progress in road and lane detection: a survey,” *Machine Vision and Applications*, vol. 25, pp. 727–745, 02 2012.

[6] Y. Su, Y. Zhang, T. Lu, J. Yang, and H. Kong, “Vanishing point constrained lane detection with a stereo camera,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2739–2744, 2018.

[7] *A multi-step curved lane detection algorithm based on hyperbola-pair model*, 2010.

[8] *Lane detection using color-based segmentation*, 2005.

[9] S. Yi, K. Li, J. Guo, and X. Gao, “Lane marking detection based on edge distribution and feature clustering,” vol. 36, pp. 1210–1215 and 1179, 10 2014.

[10] S. Jung, J. Youn, and S. Sull, “Efficient lane detection based on spatiotemporal images,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 1, pp. 289–295, 2016.

[11] *Real time detection of lane markers in urban streets*, 2008.

[12] *Lane detection based on straight line model and k-means clustering*, 2018.

[13] S. Srivastava, R. Singal, and M. Lumba, “Efficient lane detection algorithm using different filtering techniques,” *International Journal of Computer Applications*, vol. 88, pp. 6–11, 02 2014.

[14] *DBSCAN: Past, present and future*, 2014.

[15] O. Amidi and C. E. Thorpe, “Integrated mobile robot control,” *Proceedings of SPIE*, 03 1991.

[16] *Development of lateral control system for autonomous vehicle based on adaptive pure pursuit algorithm*, 2014.

[17] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, v. Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, “Stanley: The robot that won the darpa grand challenge,” *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20147

[18] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, “Predictive active steering control for autonomous vehicle systems,” *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[19] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, “Review on model predictive control: an engineering perspective,” *The International Journal of Advanced Manufacturing Technology*, vol. 117, pp. 1327–1349, 08 2021.

[20] H. Du, “Lane line detection and vehicle identification using monocular camera based on matlab,” *Academic Journal of Computing Information Science*, vol. 4, 2021.

[21] *HSI color model based lane-marking detection*, 2006.

[22] L. Tabelini, R. F. Berriel, T. M. Paixão, C. Badue, A. F. De, and T. Oliveira-Santos, “Polylanenet: Lane estimation via deep polynomial regression,” *arXiv (Cornell University)*, 01 2021.

[23] Y. Liu, S. Xu, S. Hashimoto, and T. Kawaguchi, “A reference-model-based neural network control method for multi-input multi-output temperature control system,” *Processes*, vol. 8, p. 1365, 10 2020.