


Research Article

Robot Obstacle Avoidance Controller Based on Deep Reinforcement Learning

Yaokun Tang ¹, Qingyu Chen,² and Yuxin Wei¹

¹Anhui Engineering Laboratory for Intelligent Applications and Security of Industrial Internet, Anhui University of Technology, Ma'anshan, Anhui 243032, China

²Institute of Intelligent Information Processing, School of Computer Science and Technology, Anhui University of Technology, Ma'anshan, Anhui 243032, China

Correspondence should be addressed to Yaokun Tang; youken@ahut.edu.cn

Received 11 August 2022; Revised 5 September 2022; Accepted 6 September 2022; Published 23 September 2022

Academic Editor: Sweta Bhattacharya

Copyright © 2022 Yaokun Tang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As the core technology in the field of mobile robots, the development of robot obstacle avoidance technology substantially enhances the running stability of robots. Built on path planning or guidance, most existing obstacle avoidance methods underperform with low efficiency in complicated and unpredictable environments. In this paper, we propose an obstacle avoidance method with a hierarchical controller based on deep reinforcement learning, which can realize more efficient adaptive obstacle avoidance without path planning. The controller, with multiple neural networks, contains an action selector and an action runner consisting of two neural network strategies and two single actions. Action selectors and each neural network strategy are separately trained in a simulation environment before being deployed on a robot. We validated the method on wheeled robots. More than 200 tests yield a success rate of up to 90%.

1. Introduction

The robot obstacle avoidance method is a comprehensive approach integrating multiple submethods. Generally, the completion of robot obstacle avoidance is separated into three parts: the leading part, the core, and the back-end part. Among them, the algorithmic models such as robot obstacle avoidance and path planning, equivalent to the human brain for decision-making, serve as its core. The leading part is used to obtain obstacle information (by a camera or a radar) to simulate human vision. As in the human nervous system, the back-end part is used for automatic control, which receives signals from the brain to control the body for actions, as shown in Figure 1. Progress in any one of these three parts will advance the robot's obstacle avoidance technology, allowing faster and better obstacle avoidance and expanding in its obstacle avoidance adaptability in various environments.

From the perspective of navigation and path planning, most traditional obstacle avoidance methods plan one or

more paths or navigation paths based on the location of obstacles in the current spatial environment for avoidance. The artificial potential field method, which was proposed by Khatib [1], tended to reach perfection following numerous improvements and expansions under the efforts of many researchers, allowing a wide range of applications. Han introduced kinetic conditions [2], which were used to generate short and smooth routes under aerial conditions for unmanned aerial vehicle (UAV) obstacle avoidance.

Proposed as early as 1968 [3], the A* algorithm also shows extensive applications in the field of path planning with various improvements, such as a data-driven A* algorithm proposed by Ryo [4]. Recent years have also witnessed efforts on hybrid path planning algorithms such as [5] and various types of bionic path planning methods applied to different scenarios [6]. However, in the obstacle avoidance methods based on path planning, the lack of environmental information in the unfamiliar practical application environment is often ignored. That is also the case for people walking, in which decisions can only be

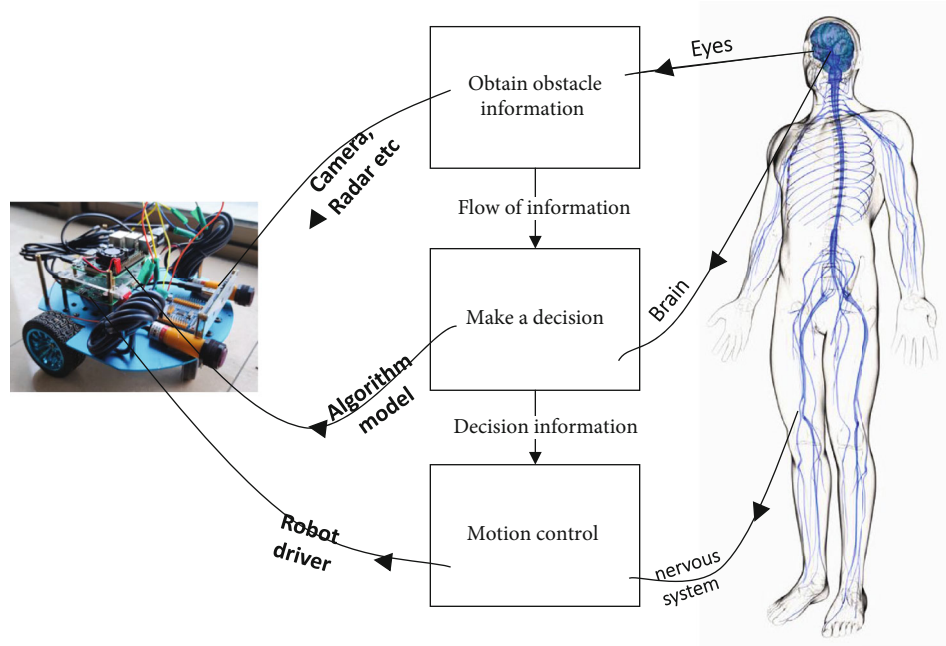


FIGURE 1: Three main components of robot obstacle avoidance technology and their corresponding robot components (our wheeled robot on the left).

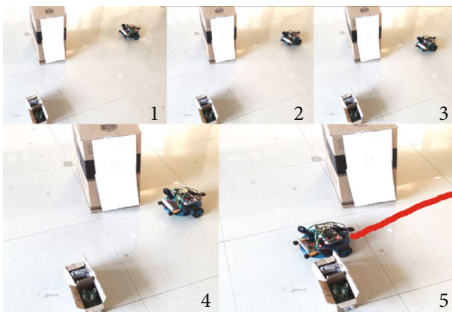


FIGURE 2: Robot completed obstacle avoidance task.

made according to current information acquired by vision. Therefore, we hereby consider building a controller specializing in dealing with obstacles without taking path planning into account.

2. Method

2.1. Deep Reinforcement Learning. Recent years have witnessed a widespread application of model-free deep reinforcement learning (DRL) in the field of robots, which can offer creative ideas and solutions for obstacle avoidance technology. The most distinguishing feature lies in that it enables the robot (program) to learn autonomously in the interaction with the environment, which is quite similar to the human growth and learning process. The concept “trial and error” functioned as the core mechanism of reinforcement learning, and “reward” and “penalty,” as the basic means of learning, were proposed by Waltz and Fu Jingsun in their control theory as early as 1965 [7]. Deep reinforcement learning has made tremendous progress in recent years. The DQN (Deep Q Network) algorithm proposed

and refined by Mnih et al. in 2013 and 2015 [8, 9] provided a powerful “weapon” for reinforcement learning. This deep reinforcement learning algorithm was employed in several components of our controller.

However, the “trial and error” learning approach requires a wealth of training for an effective model. Nonetheless, hardware systems such as robots cannot afford a mass of “trial and error” training for obstacle avoidance in terms of time and economy. As a result, training in a simulation environment emerges as a better option. Many recent studies have demonstrated that the experience gained from training in a simulation environment is also applicable to a real environment [10, 11]. It is critical to minimize the gap between the simulation environment and reality during training in a simulation environment. Neunert analyzed the potential causes of the gap in [12], Li solved some of them by system identification [13], and the DeepMind team enhanced fidelity by estimating model parameters [11]. Enlightened by these efforts, we rewrote the CarRacing-v0 environment based on the OpenAI Gym simulation environment [14] to enable the training of our selector.

Although training without any human guidance can yield a better model [15], training and learning with human guidance in their directions are more efficient. Furthermore, existing deep learning algorithms and neural networks are less effective when learning multiple strategies simultaneously [16]. The separate training of subaction strategies and selector strategies [17, 18] were a general approach in Behavior-Based Robotics [19], making the model training more efficient.

Based on the challenges discussed above and the efforts of previous researchers, an obstacle avoidance controller with a hierarchical structure was proposed in this paper.

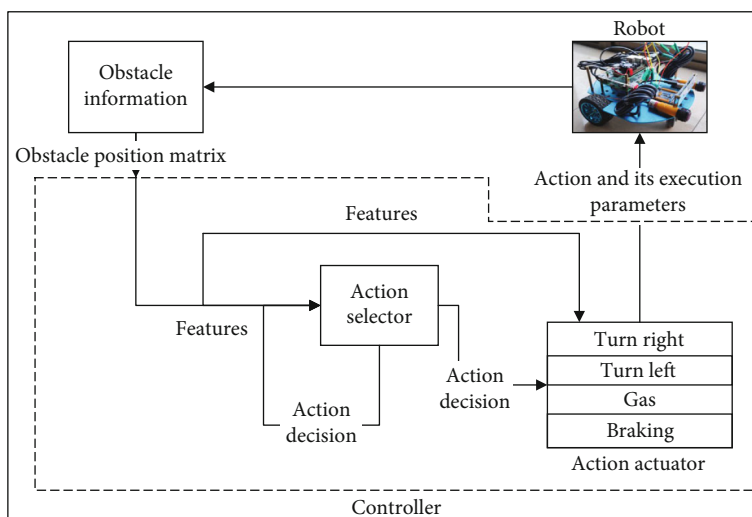


FIGURE 3: Framework of controller.

It only selected the information observable from the robot's current viewing angle as input, without considering path planning, allowing it to handle situations closer to people's progress in reality. To avoid obstacles, we decomposed an obstacle avoidance action into multiple subactions to train a DQN-based action selector to select and run appropriate actions, including Turn Left, Turn Right, Gas, and Stop. We deployed this method on a wheeled robot, as illustrated in Figure 2, and carried out more than 200 experiments in the real physical environment by arranging artificial obstacles, with the success rate of obstacle avoidance hitting 90%, demonstrating the effectiveness of the controller.

2.2. Overview. Unlike traditional path planning methods, we adopted a controller to avoid obstacles. Figure 3 presents the general framework of the controller. The action selector, along with the following four action runners, served as the core of the whole framework. Based on the DQN algorithm, the action selector made decisions based on the information in the environment and the ongoing actions in the current cycle. The action selector sent an action decision to the action runner for running in each running cycle.

We decomposed the obstacle avoidance action into four subactions: Turn Right, Turn Left, Gas and Braking. Among them, the subactions of Turn Right and Turn Left were also based on the DQN algorithm and determined the deflection angle in this direction based on the decision of the neural network. The subactions of Gas and Braking, as single actions, required no manipulation with the model and ran the selector's command directly upon receipt. The action selector and each action were separately trained in parallel in our rewritten simulation environment, making the design of complex loss functions unnecessary and parameter tuning and error finding more convenient.

2.3. Simulation Environment. In the CarRacing environment of OpenAI Gym [14] (Figure 4(a)), the designed task object

was for the racing car Agent to pass through the entire track as quickly as possible, without crossing the track or the boundary. In pursuit of perfection, the modeling for the simulation of the Gym involved the friction of the site, the wheels of the racing car, the ABS sensors, etc. On this basis, we improved the CarRacing simulation environment, which was called Car2D.

Inspired by [11–13], we readjusted the physical model of CarRacing (friction, mass of the wheeled robot, etc.) and added random noise to bring it closer to the actual conditions of our physical robot (Figure 4(b)) and the test site. The original road generation program is unsuitable for a single obstacle avoidance task training as it generates a complete circular road instead of obstacles. We also rewrote the part about road generation and added the obstacle generation function, with which the shape, number, and location of obstacles could be accompanied by random noise generation (Figures 4(c) and 4(d)). Randomization in Car2D could enhance the robustness of the training object, narrowing the gap between the simulation environment and the real environment, as has been demonstrated by studies [10, 20, 21].

For the performance of the four subactions of the robot in the simulation environment, we completed the design based on the physical model of the real environment. Each time running the Gas action command was run, the Agent would provide the robot with equal power to move forward, with the maximum speed reached when the action lasted for several consecutive cycles. The power would be gradually lost due to ground friction if the action stopped. The car would lose power due to running the Braking action command, with the braking distance determined by the car's mass and the elaboration of the physical model.

The task object for obstacle avoidance required no long runway or various curves; thus, only a limited straight runway was generated in Car2D. According to the set conditions, the environment sent a stop signal to stop this training and restart a new episode (Figure 5). Each cycle

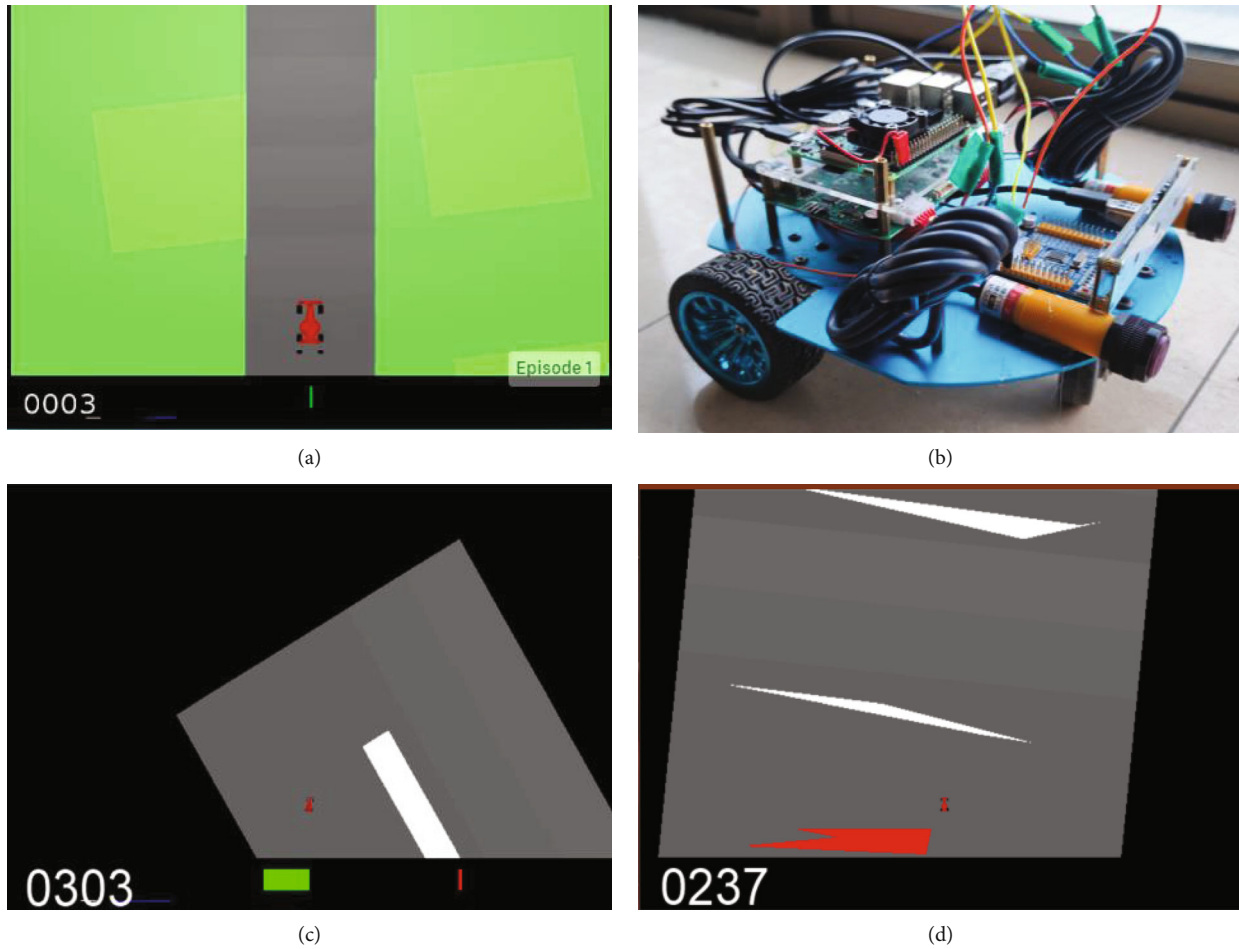


FIGURE 4: (a) OpenAI Gym CarRacing; (b) Our wheeled robots; and (c, d) Randomly generated obstacles on tracks.



FIGURE 5: An episode in training, with the obstacle in white, the end of the road in red, and the viewpoint following the Agent, where the car hit the obstacle, terminating this episode.

Car2D would present an RGB image generated according to the limited viewing angle in front of the Agent and add random noise before providing it to the controller.

2.4. Action Selector. As the core component of the controller, the action selector was implemented based on the DQN algorithm. Compared with the Q-Learning algorithm [22], this algorithm approximated the value function with a convolutional neural network and leveraged the experience replay mechanism to improve the efficiency of the neural network. In addition, a neural network called the Main Net was introduced to the DQN algorithm to generate the current Q value and a Target Net with the same structure as the Main Net was introduced to generate the

target Q value. The parameters of Main Net were copied to Target Net every certain number of iterations. By reducing the correlation between the current Q value and the target Q value, the stability of the algorithm has been improved. The pseudocode of the DQN algorithm is provided in Algorithm 1 [23]. For a detailed description of DQN, please refer to [8, 9, 24], as only the relevant details of this model are presented here.

In each running cycle, the action selector makes action decisions based on the environment observed by the Agent and the actions performed in the previous running cycle. The direct use of the RGB image in the simulation environment as the input of the selector would make it tough to apply the model to the real physical environment. For

Input: Pixels and reward
Output: Q action-value function
Initialization
Initialize replay memory space D
Initialize the Q network (action-value function) \mathcal{Q} with random weights θ
Initialize target network (action-value function) $\widehat{\mathcal{Q}}$ with weights $\theta^- = \theta$
1: Forepisode = 1 to M do
2: Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
3: For $t = 1$ to T do
4: Following ϵ -greedy policy, select
$a_t = \begin{cases} \text{a random action with probability } \epsilon \\ \arg \max_a \mathcal{Q}(\phi(s_t), a; \theta) \text{ otherwise} \end{cases}$
5: Run action a_t in an emulator and observe the reward r_t and image x_{t+1}
6: Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
7: Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
8: Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
9: Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \widehat{\mathcal{Q}}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
10: Calculate the loss (Perform a gradient descent step on)
$(y_j - \mathcal{Q}(\phi_j, a_j; \theta))^2$
11: Train and update weights θ of \mathcal{Q}
12: End
13: End

ALGORITHM 1: DQN.

TABLE 1: Symbols and rewards terms.

	Symbols
t	Running cycle
a_t	Set of actions at t th cycle
P	Position set of agent at t th cycle
x_t, y_t	Agent's central coordinates at t th cycle
r	Reward
O	Position set of the obstacle
T	Position set of the track
E	Position set of the end
K	Constant
\emptyset	Empty set
	Rewards terms
Distance reward	$r_d = K_d(y_t - y_{t-1})$
Obstacle collision penalty	$r_c = K_c \text{ if } P \cap O \neq \emptyset, \text{ otherwise } 0$
Off the track penalty	$r_o = K_o \text{ if } P \cap T \neq \emptyset, \text{ otherwise } 0$
Time penalty	$r_t = K_t t$
Obstacle avoidance reward	$r_a = K_a \text{ if } P \cap E \neq \emptyset, \text{ otherwise } 0$
Speed reward	$r_s = K_s \text{ if } a_t = \text{gas}, \text{ otherwise } 0$

the same input of the model in the real physical environment and the simulation environment, we processed the RGB image from the camera in front of the robot as a matrix describing the position of the obstacles, denoted as M . The subactions of Turn Left and Turn Right were mutually exclusive; thus, only 12 states were available for the action space of the action selector. We used a one-dimensional vector with a length of 3 to describe the actions run in the previous cycle, denoted as a . Therefore, the state space of the running cycle t was characterized as follows:

$$S_t = M_t a_{t-1}, \quad (1)$$

Table 1 presents the symbols and rewards terms. The reward function was denoted as R , indicating that the rewards obtained by $S_t \rightarrow S'_t$ through the action a_t , which was defined as follows:

$$R_t(S_t, a_t, S'_t) = r_d + r_c + r_o + r_t + r_a + r_s. \quad (2)$$

Distance reward encouraged the Agent to move forward as much as possible; the obstacle collision penalty was implemented for hitting an obstacle; the off-the-track penalty was implemented for crossing the boundary; the time penalty encouraged the Agent to avoid obstacles as quickly as possible to prevent them stopping in place; the obstacle avoidance reward was implemented for successful obstacle avoidance; and the speed reward was implemented for the throttle action under the premise of

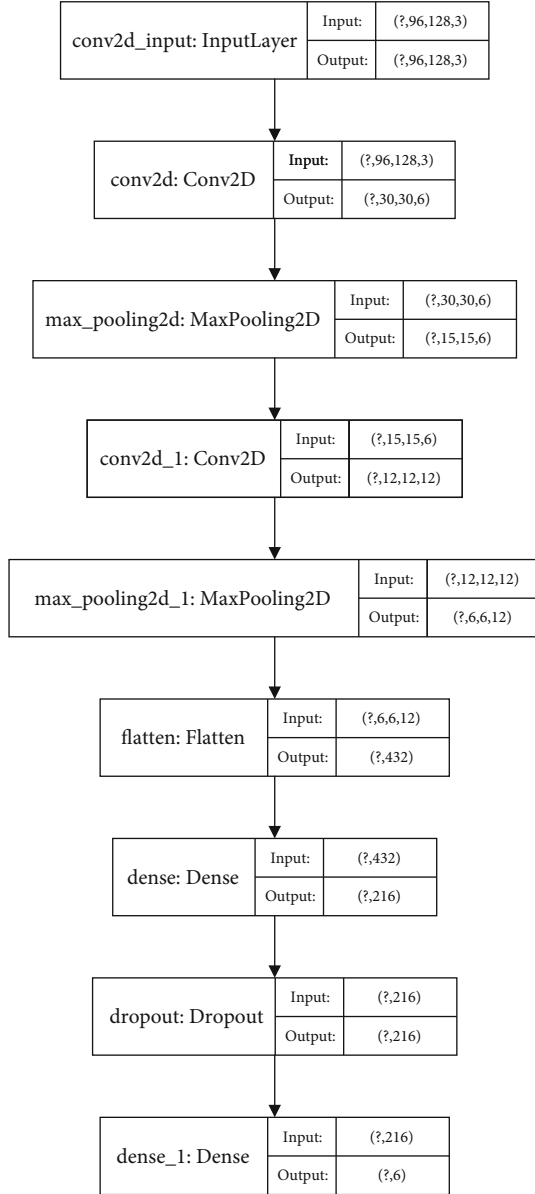


FIGURE 6: Network structure of selectors.

successful obstacle avoidance for another reward, encouraging the Agent to move as fast as possible. It is important to note that the multiple rewards and penalty mechanisms we designed were not superimposed on the Agent at the same time. We adopted the idea of curriculum learning (CL) [25], in which the training started with simple, single, and small obstacles, and the current episode was not stopped even after hitting an obstacle. After the Agent was capable of completing small task objects, the task difficulty would be increased, such as by increasing the area and number of obstacles or by randomizing the location of obstacles.

The score function of the current state was denoted as $Q(S, a)$, which evaluated the current action taking the future state into account, indicating the expectation of the sum of the rewards obtained by the Agent after running the action

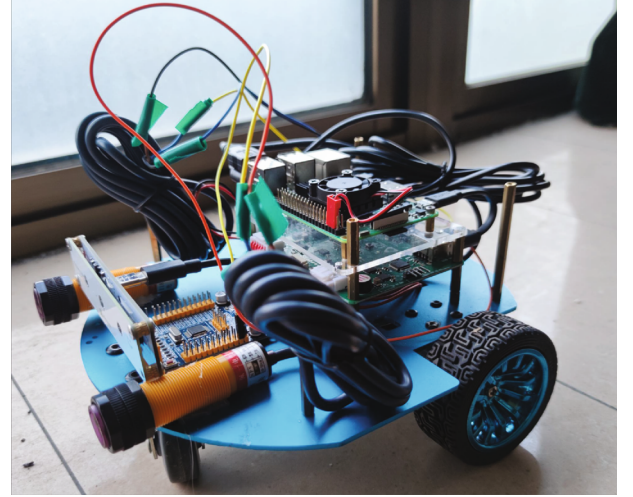


FIGURE 7: Self-built wheeled robot.

a in the state S until the end of the current episode. For the state S_t , and the action A_t run, there exists:

$$Q(s, a) = E[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots | A_t = a, S_t = s], \quad (3)$$

where E represents the expectation, γ , the discount factor ($\gamma \in (0, 1)$), which was used to measure the degree of influence of future rewards and current rewards on the Q value. The closer γ was to 1, the greater the influence of future rewards, and $\gamma = 0.85$ was taken for training. The optimal value action function of Q was denoted as Q^* , and there exists:

$$Q^*(s, a) = \max E[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots | A_t = a, S_t = s], \quad (4)$$

then the state transition equation can be obtained:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [TargetQ - Q(s, a)], \quad (5)$$

where α is the decay learning rate of γ . Taking Q as a label, the loss function of network training can be obtained:

$$L(\theta) = E[(TargetQ(\theta) - Q(s, a, \theta))^2], \quad (6)$$

where $TargetQ$ can be expressed as follows:

$$TargetQ(\theta) = R(s, a, s') + \gamma \max Q(s', a', \theta). \quad (7)$$

Figure 6 provides the structure of the deep convolutional neural network used in our Main Net, and the Target Net had an identical structure and different parameters. Different from the structure of general DQN algorithms, we inserted a Dropout layer in the last fully connected layer, which was only enabled during training and not when the model was in use.

2.5. Action Runner. In the action runner, only two actions, “Turn Right” and “Turn Left,” which provided the deflection angle of the robot, required training. The two were also based

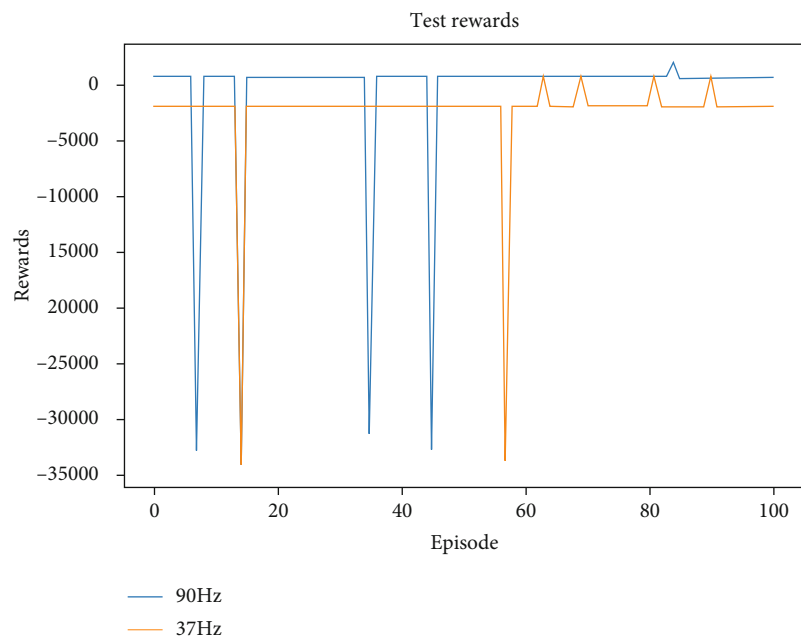


FIGURE 8: Rewards curve for a model trained at 37 Hz for 5,600 rounds and validated 100 times at 90 Hz and 37 Hz.

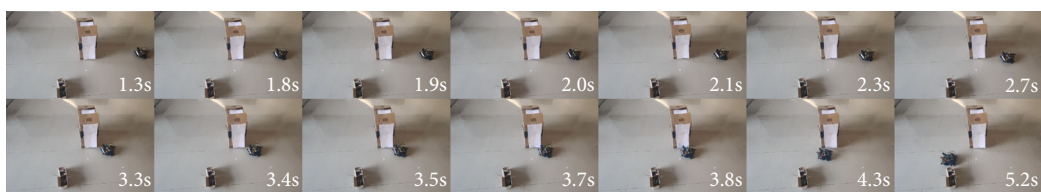


FIGURE 9: Wheeled robot avoiding irregular obstacles with the controller.

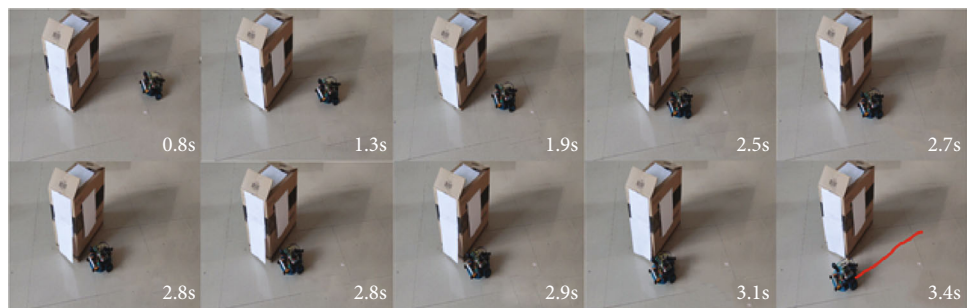


FIGURE 10: Wheeled robot avoiding multiple obstacles with the controller.

on the DQN algorithm, with the same characteristics of the space state of the input as the action selector, thus not elaborated here. However, the action space had a total of 36 states, which were categorized from 0° to 180° by a step length of 5° . Experimental experience showed that the deflection angle exceeding 180° would cause the robot to move in the opposite direction and eventually leave the runway.

3. Results and Discussion

This section presents the experimental results in the real environment and an analysis of the drawbacks of the model.

(a) Experiment Setting

We used a self-built wheeled robot, as shown in Figure 7, to validate the model. The model employed STM32F4 for the chip of the driver board, Raspberry Pi 4B for the upper computer, and ubuntu 18.04 for the system of the upper computer. An infrared camera and two independent infrared distance meters were applied to collect environmental characteristics. The matrix for the position description of the obstacle was constructed mainly through the identification and ranging of obstacles.

Tests showed that the configuration of the running frequency during training and verification in the simulation

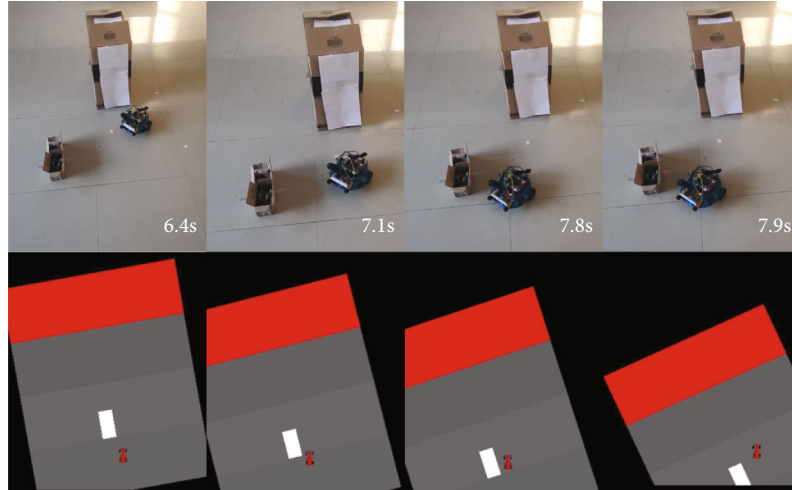


FIGURE 11: Large angular deviation for obstacle avoidance rather than crossing gaps. Agents run similar actions in the simulation environment.

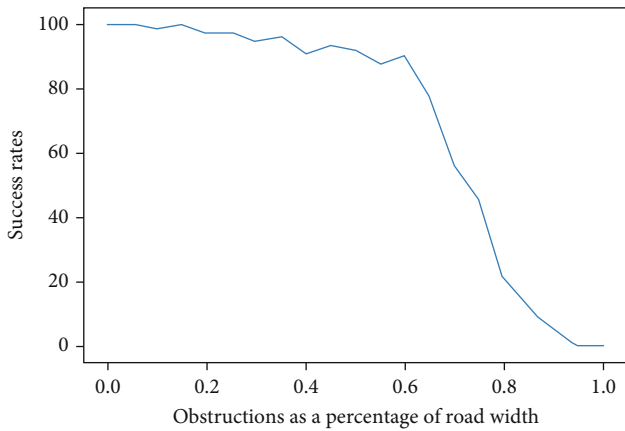


FIGURE 12: Poorer obstacle avoidance effect with bigger obstacle.

environment directly affected the success rate of obstacle avoidance. Figure 8 shows the rewards curve for a model validated 100 times at 90 Hz and 37 Hz, which was trained at 37 Hz for 5,600 rounds.

It can be seen from the curve that the same model accomplished the obstacle avoidance task in most cases when running at 90 Hz but failed at 37 Hz. This means that the model has to be run enough times in a second for the robot to have a coherent obstacle avoidance action. Finally, we set the training and testing frequency in the simulation environment to 90 Hz to ensure training efficiency while preventing the actions of the Agent from oscillating. When tested in a real environment, the controller would see its actions oscillate as it runs too fast. Therefore, we had the hard front-end hardware on the wheeled robot that provided environmental information run at a frequency of 55 Hz to monitor obstacles in real-time. The running frequency of the controller was set to 30 Hz to allow for a smoother running of the robot and avoid oscillations.

As the deployed model failed to play its full role due to the limitation of computing power, we quantified the

model to improve its running efficiency with reference to some methods introduced or mentioned in [26–28]. During the test, the robot was set with a fixed initial speed, and the obstacle avoidance controller was activated when an obstacle was identified to control the robot to avoid the obstacle.

Both the models used in the simulation environment and during the training were written in Python, but most of the programs for the robot were written in C++ and C languages to improve the efficiency of the model.

(b) Experimental Results

To validate the effectiveness of the controller, we conducted the test in three cases: a single obstacle, multiple obstacles (Figure 9), and irregular obstacles (Figure 10), with a total of 75 different obstacle positions. In more than 200 tests, successful obstacle avoidance was observed in most cases for multiple or irregular obstacles, with a success rate of up to 90%.

Unlike in our simulation environment, there existed no road limitations for some robot tasks in the real environment, such as in wild meadows and deserts. To test the generalization ability of the controller, we removed the road restrictions from the real environment and conducted the test 100 times, in which the controller also completed the obstacle avoidance task, with a success rate hitting 91%. However, there were several times when the obstacle avoidance effect failed to meet the expectations. In the case of a small gap between two obstacles, a farther path instead of passing through the gap would be the option (Figure 11 top half), similar to some performance in the simulation environment with small or few obstacles (Figure 11 bottom half). It seemed that when conditions permitted, the controller would leverage the drivable space furthest and stay away from obstacles as much as possible. We set 20 different ratios of obstacles to the width of the road in the simulation environment, each tested 100 times. Figure 12 shows the curve for the success rate of obstacle avoidance.

It can be seen that the success rate of obstacle avoidance became lower with the increase in the ratio of the obstacle to the road width. The obstacle avoidance failed when the Agent could not pass through the gap between obstacles, with a success rate of 90%. Our idea was validated.

4. Conclusion

This paper proposed a hierarchical controller for robot obstacle avoidance in progress, which enabled the robot to avoid obstacles without path planning flexibly. By decomposing the obstacle avoidance control task into multiple sub-actions of GAS, Turn Left, Turn Right, and Braking, an action strategy could be decomposed into multiple strategies for separate training. The idea of curriculum learning (CL) was used to improve training efficiency. With the limited information in front, our controller brought us a step closer to the real action of people dealing with unknown environments.

The controller was trained in a simulation environment before being deployed on a wheeled robot, producing satisfactory results with consistent performance compared with the simulation environment. Wheeled robots deployed with controllers yielded a success rate of up to 90% in more than 200 obstacle avoidance tests. It also applies to an environment similar to the wild without road restrictions, exhibiting good performance.

The controller was less effective in the case of dense obstacles with small gaps. More adjustments in running may be required for dynamic obstacles to yield better results. For these cases, traditional obstacle avoidance methods may outperform our controller when complete environmental information is available. We will make more efforts to solve problems and improve the controller.

Data Availability

All data, models, and code generated or used during the study appear in the submitted article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work described in this paper was fully supported by a grant from the Student's Platform for Innovation and Entrepreneurship Training Program of ANHUI UNIVERSITY OF TECHNOLOGY.

References

- [1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous Robot Vehicles*, pp. 396–404, Springer, New York, NY, 1986.
- [2] H. Zhijiu, W. Wenjiang, L. Xiaowei, Z. Dan, and L. Chunxin, "An improved artificial potential field method constrained by a dynamic model," *Journal of Shanghai University: Natural Science Edition*, vol. 6, pp. 879–887, 2019.
- [3] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [4] R. Yonetani, T. Taniai, M. Barekatin, M. Nishimura, and A. Kanazaki, "Path planning using neural a* search," in *International Conference on Machine Learning*, pp. 12029–12039, 2021.
- [5] H. Sang, Y. You, X. Sun, Y. Zhou, and F. Liu, "The hybrid path planning algorithm based on improved a* and artificial potential field for unmanned surface vehicle formations," *Ocean Engineering*, vol. 223, article 108709, 2021.
- [6] B. K. Patle, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: on path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019.
- [7] M. Waltz and K. S. Fu, "A heuristic approach to reinforcement learning control systems," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 390–398, 1965.
- [8] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," 2013, <https://arxiv.org/abs/1312.5602>.
- [9] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [10] J. Hwangbo, J. Lee, A. Dosovitskiy et al., "Learning agile and dynamic motor skills for legged robots," *Robotics*, vol. 4, no. 26, p. eaau5872, 2019.
- [11] J. Tan, T. Zhang, E. Coumans et al., "Sim-to-real: learning agile locomotion for quadruped robots," 2018, <https://arxiv.org/abs/1804.10332>.
- [12] M. Neunert, T. Boaventura, and J. Buchli, "Why off-the-shelf physics simulators fail in evaluating feedback controller performance—a case study for quadrupedal robots," *Advances in Cooperative Robotics*, pp. 464–472, 2017.
- [13] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Model identification via physics engines for improved policy search," 2017, <https://arxiv.org/abs/1710.08893>.
- [14] G. Brockman, V. Cheung, L. Pettersson et al., "Openai gym," 2016, <https://arxiv.org/abs/1606.01540>.
- [15] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [16] G. Berseth, C. Xie, P. Cernek, and M. Van de Panne, "Progressive reinforcement learning with distillation for multi-skilled motion control," 2018, <https://arxiv.org/abs/1802.04765>.
- [17] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," *AAAI*, vol. 90, pp. 796–802, 1990.
- [18] J. Merel, A. Ahuja, V. Pham et al., "Hierarchical visuomotor control of humanoids," 2018, <https://arxiv.org/abs/1811.09656>.
- [19] R. C. Arkin and R. C. Arkin, *Behavior-Based Robotics*, MIT press, 1998.
- [20] M. Hutter, C. Gehring, D. Jud et al., "Anymal—a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 38–44, Daejeon, Korea (South), 2016.
- [21] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.
- [22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3–4, pp. 279–292, 1992.

- [23] Y. Li, "Deep reinforcement learning: an overview," 2017, <https://arxiv.org/abs/1701.07274>.
- [24] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," 2018, <https://arxiv.org/abs/1811.12560>.
- [25] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, Montreal, Quebec, Canada, 2009.
- [26] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4820–4828, LasVegas,NV,USA, 2016.
- [27] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: towards lossless CNNs with low-precision weights," 2017, <https://arxiv.org/abs/1702.03044>.
- [28] W. Fei, W. Dai, C. Li, J. Zou, and H. Xiong, "General bitwidth assignment for efficient deep convolutional neural network quantization," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021.