Sensors and Materials, Vol. 36, No. 5 (2024) 1959–1982
MYU Tokyo

S & M 3647

Indoor Mobile Robot Path Planning and Navigation System Based on Deep Reinforcement Learning

Neng-Sheng Pai, Xiang-Yan Tsai, Pi-Yun Chen,* and Hsu-Yung Lin

Department of Electrical Engineering, National Chin-Yi University of Technology, Taichung 41170, Taiwan

(Received December 15 2023; accepted May 13, 2024)

Keywords: deep reinforcement learning, behavior cloning, YOLO-v7-tiny, A* algorithm, DWA algorithm

In this paper, we propose an autonomous navigation system architecture for indoor mobile robots that combines the advantages of end-to-end (E2E) autonomous driving and traditional navigation algorithms. The architecture aims to overcome the challenges of traditional navigation algorithms relying heavily on high-precision localization and E2E struggling to make good decisions when unable to detect target objects. A neural network is trained using deep reinforcement learning in a simulated environment, and the approach of behavior cloning is introduced to stabilize the training process. With this approach, the trained neural network can make action decisions based solely on 2D LiDAR data and images captured by cameras, eliminating the reliance on high-precision localization systems and overcoming the challenges of traditional navigation algorithms. In real-world environments, the YOLO-v7-tiny model is used for object detection in indoor settings. When the target object is far away, A* and DWA algorithms are employed for path planning to ensure safe and efficient navigation. These algorithms can find the globally optimal path and perform local obstacle avoidance, thus achieving autonomous navigation in indoor environments.

1. Introduction

In industrial automation, unmanned mobile vehicles play a crucial role. In the past, automated guided vehicles (AGVs) were used for product transportation, but they required guidance through methods such as wires, markers, or magnetic strips, resulting in increased costs.(1) Nowadays, autonomous mobile robots (AMRs) are employed, relying on sensors such as light detection and ranging (LiDAR) and cameras, utilizing simultaneous localization and mapping (SLAM) to construct maps, and incorporating path planning algorithms for navigation. Achieving precise localization remains a demanding requirement in AMRs, and obtaining a high-accuracy positioning system remains a challenge.(2)

On the other hand, end-to-end (E2E) autonomous driving aims to enable vehicles to make autonomous decisions on the basis of sensor inputs. Compared with traditional AMRs, E2E approaches exhibit greater robustness in autonomous navigation, requiring lower precision in localization. However, implementing E2E poses significant challenges.(3) Conventional

*Corresponding author: e-mail: chenby@ncut.edu.tw
https://doi.org/10.18494/SAM4827

ISSN 0914-4935

autonomous navigation systems are typically rule-based, incurring substantial labor costs and struggling to handle unforeseen situations.

In recent years, with the advancement of power electronics, artificial intelligence has found widespread applications in various fields. Deep learning (DL) and reinforcement learning (RL) stand as two of the most popular research directions in the AI domain. DL utilizes neural networks to approximate various functions, enabling automatic feature learning from vast amounts of data, and is applied to tasks such as image recognition, speech recognition, and natural language processing. RL is a machine learning approach where agents learn from interactions with the environment, balancing exploration and exploitation. Deep RL (DRL), a combination of DL and RL, has found applications in robotics, computer vision, finance, medicine, and more. Notably, DRL achieved remarkable breakthroughs in the field of Go, such as DeepMind's method based on Monte Carlo tree search,[4] which defeated several Go masters. OpenAI's RL from the human feedback (RLHF) approach[5] was also employed in training large-scale natural language processing networks, as seen in ChatGPT.

2. Related Work

The literature review of this study is divided into three main parts: mobile robot navigation, deep RL, and the current state of E2E autonomous driving development.

2.1 Mobile robot navigation

In the case of mobile robot navigation, we focus on robot localization, environment perception, map construction, and path planning in this research. For instance, Chan *et al.* introduced a method for fusing laser-based SLAM and visual SLAM, achieving a positioning error of less than 5% in actual distance.[6] Hao and Deng utilized the theory of phased array ultrasonic testing (PAUT) and developed a three-degree-of-freedom (DOF) scanning robot for automated weld seam inspection.[7]

2.2 DRL

In the domain of deep RL, the breakthrough achieved by DeepMind in 2013, where deep RL successfully surpassed human experts in Atari games using deep RL, sparked a fervor in the field.[8] Haarnoja *et al.*, on the other hand, argued against the approach of considering only the maximum action value in RL, as exemplified by Deep Q-Network (DQN). Instead, they proposed the use of a Boltzmann distribution for action values, while simultaneously seeking to maximize both rewards and entropy. This led to the development of soft actor-critic (SAC), wherein the policy is updated by minimizing the Kullback–Leibler Divergence (KLD) between action value distributions.[9]

2.3 E2E autonomous driving

In the past, numerous researchers have attempted to achieve E2E capabilities through machine learning approaches. One of the earliest attempts dates back to 1989, where the ALVINN system utilized camera images and laser rangefinders to update a fully connected neural network in a supervised learning manner, enabling the network to make decisions regarding vehicle navigation actions.[10] In 2016, Bojarski *et al.* employed convolutional neural networks (CNNs) to directly map raw camera images to steering actions, utilizing training data derived from human demonstrations for decision-making.[11] With the rise of deep RL, Yu *et al.* utilized DQNs in a simulated environment, taking images as input and deciding the optimal actions from nine discrete options.[12]

3. Methodology

In this paper, we propose an autonomous navigation system that combines traditional path planning with DL techniques, as illustrated in Fig. 1. In this architecture, the robot's end is responsible for map construction and gathers information from cameras, 2D LiDAR, and odometry, which is then transmitted to the computer end. At the computer end, the system analyzes the images received from the camera to determine if any target objects are observed and subsequently makes navigation decisions using either conventional algorithms or neural networks. The objective of this system is to provide smooth and efficient navigation.

3.1 DRL

In this section, we will elaborate on the training of neural networks in the system, utilizing the SAC algorithm for training. RL learns through interaction with the environment. However, the cost of interacting with the real-world environment can be prohibitively high, particularly when conducting experiments on physical robots, which require substantial time and financial

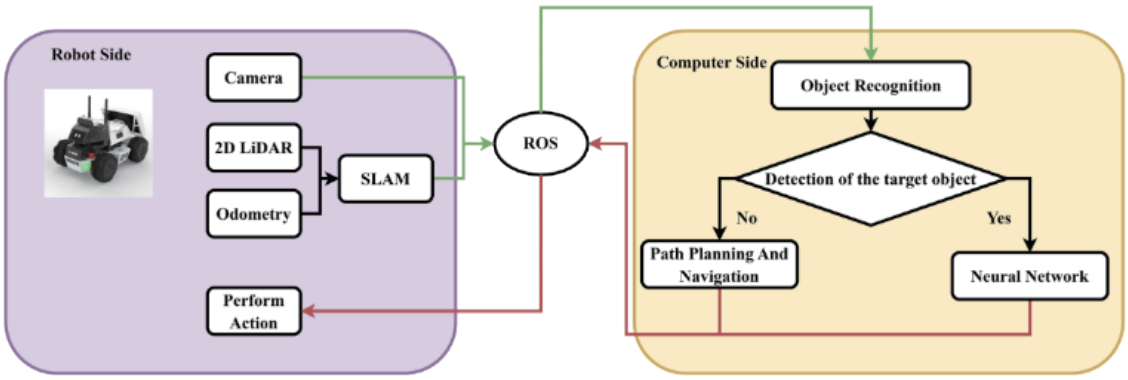


Fig. 1. (Color online) System architecture diagram.

resources. Simulated environments have been widely adopted in RL. Training in a simulated world allows for the faster acquisition of abundant experiences, facilitating further policy optimization, while also reducing the costs associated with real-world experimentation.

First, it is necessary to define the state space, action space, and reward function of the problem. The state space describes the system's states, the action space defines the actions the system can take, and the reward function evaluates the agent's performance after executing actions.

3.1.1 Action space

The action space is defined as a set of elements containing linear velocity (v) and angular velocity (ω), as shown in Eq. (1), subject to constraints imposed by the limitations of the robot.

$$Action\ Space = \begin{Bmatrix} v : [-0.22 \sim 0.22]\ \mathrm{m/s} \\ \omega : [-2.5 \sim 2.5]\ \mathrm{rad/s} \end{Bmatrix} \tag{1}$$

3.1.2 State space

Considering the disparity between the simulated world and the real world, relying solely on images for state definition is insufficient. Therefore, in this paper, the state space is defined as a collection of laser scan points and the center points of target objects in the binarized image. As illustrated in Fig. 2, the left image represents the camera image captured by the agent in the simulated world, which is then binarized as shown in the right image. In the binarized image, the leftmost and rightmost points (index) are identified and named as $Target_{right}$ and $Target_{left}$, respectively. Consequently, the center point of the target object is determined, as shown in Eq. (2).

$$Target = \left(Target_{right} - Target_{left}\right) / 2 + Target_{left} \tag{2}$$

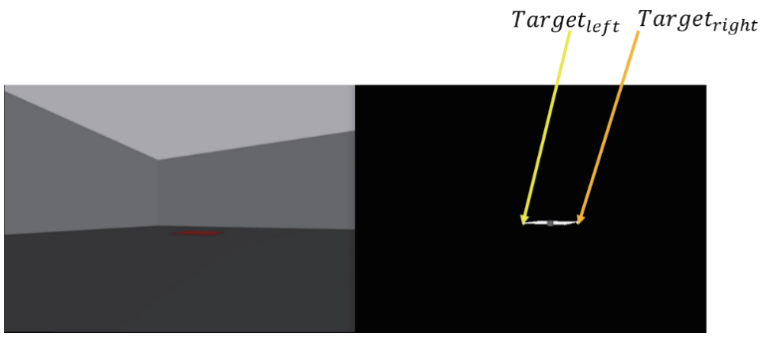


Fig. 2. (Color online) Diagram of target object state space.

Therefore, the state is defined as Eq. (3) in this paper, where scan represents the values sampled from the LiDAR point cloud.

$$State\,Space = \left[Target, scan_1, scan_2, \cdots, scan_k\right] \tag{3}$$

3.1.3 Reward

The design of rewards significantly affects the learning performance of the agent. In this study, multiple aspects of rewards were employed, including distance reward, angle reward, action reward, obstacle distance, and termination reward. Among them, the distance reward is represented by Eq. (4), where x and y represent the position vectors of the agent and target in space, and the denominator is the Euclidean distance between the agent and the target. Here, x_i and y_i denote the coordinates of the agent and target, respectively. The closer the agent is to the target, the higher the reward obtained, encouraging the agent to move towards the target. To avoid division by zero, λ is set to a very small value in this study, preventing infeasible or erroneous results.

$$r_{dis} = \frac{1}{\sqrt{\sum_{i=1}^{n}\left(x_i - y_i\right)^2 + \lambda}} \tag{4}$$

The angle reward is designed on the basis of the angle between the target position and the agent's heading direction. To guide the agent to learn how to reach the target more efficiently, the angle reward is defined as shown in Eq. (5), aiming to provide larger rewards when the agent's heading direction aligns more towards the target. Here, $\theta_{goal\ angle}$ represents the angle between the coordinates of the target and agent, and θ_{yaw} denotes the heading direction of the agent's vehicle in the world coordinate system.

$$r_{angle} = -\left|\theta_{goal\ angle} - \theta_{yaw}\right| \tag{5}$$

The design of angle reward is illustrated in Fig. 3, where the agent receives a higher reward as it aligns more towards the target object. This design aims to guide the agent to reach the target as rapidly as possible.

To encourage the agent to reach the target as rapidly as possible, i.e., aiming for the agent to arrive at the target with maximum speed and move as straight as possible, the action reward is defined as

$$r_{action} = -\left|\omega\right| + v. \tag{6}$$

In addition, to encourage the agent to avoid obstacles, the distance between the agent and the obstacles is considered as a penalty factor. The closer the agent is to the obstacles, the greater the penalty, motivating the agent to actively avoid collisions. If the minimum point from the 2D

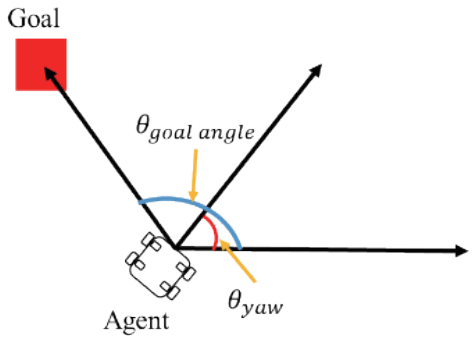


Fig. 3. (Color online) Diagram of angle reward.

LiDAR is below a certain threshold, it is deemed as being too close to an obstacle, resulting in a deduction in score. The magnitude of this penalty term, denoted as r_{scan}, is adjusted on the basis of specific circumstances. When the agent successfully reaches the target, a high reward is given, whereas collisions incur deductions and penalties.

3.1.4 Network architecture

In this paper, we utilize a neural network for the agent's action decision-making. The network architecture is depicted in Fig. 4. It consists of a state input network defined by Eq. (3), which undergoes processing through multiple layers of neurons. The output of the network is the mean (μ) and standard deviation (σ) of the actions. A ReLU activation function is applied to the layer preceding the standard deviation to ensure non-negativity. On the basis of the output mean and standard deviation, a random variable is sampled from a Gaussian distribution. The sampled variable is then compressed within the range [−1, 1] through the tanh function and multiplied by the action space boundaries defined in Eq. (1) to conform to the defined action space.

We design a network responsible for evaluating the quality of an action taken by the agent. The purpose of this network is to assess the quality of the action on the basis of the current state input and the action decided by the agent, and return an evaluation value to aid the agent in making the next action decision. The network architecture is depicted in Fig. 5, with the defined state and the action determined by the actor network as inputs. The outputs are the action values Q_v and Q_ω, which correspond to the evaluations of the desirability of the agent's chosen vectors v and ω, respectively.

3.1.5 Training process

The training of the agent is achieved through interaction with the environment. In this paper, the SAC algorithm is used to train the neural network. A batch of experiences is sampled from the experience replay buffer, and the training of the critic network and policy network is performed through offline updates.

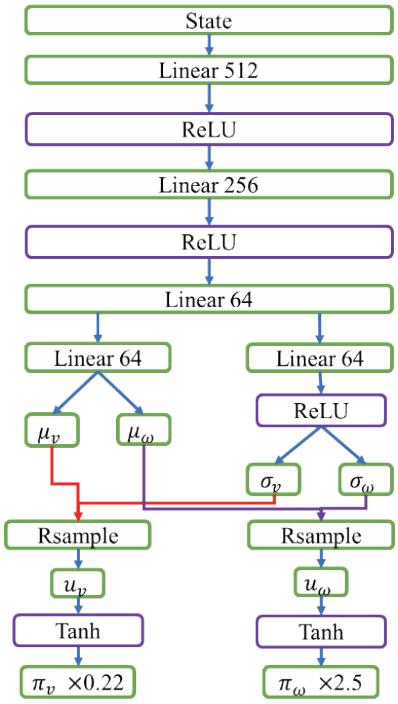


Fig. 4. (Color online) Actor network diagram.

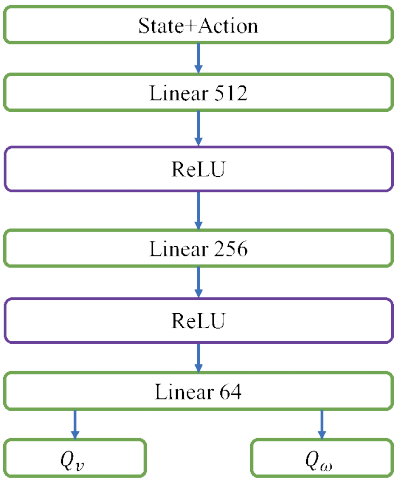


Fig. 5. (Color online) Critic network diagram.

The update process for the critic network is shown in Eq. (7),[9] where r represents the rewards obtained from sampling trajectories from the experience replay buffer, and γ and α are hyperparameters representing the discount factor and temperature coefficient, respectively. The network parameters are updated using the gradient descent algorithm.

$$Critic\,loss = \left\| Q(s,a) - r + \gamma\left(Q(s_{t+1},a_{t+1}) - \alpha\log\left(\pi\left(a_{t+1}\,|\,s_{t+1}\right)\right)\right)\right\|_2^2 \tag{7}$$

The parameter update process for the actor network is shown in Eq. (8), where the state input s_t from the sampled experience in the replay buffer is fed into the actor network to compute a_t^- and $\log(\pi(a_t|s_t))$. The state s_t and action a_t^- are then inputted into two critic networks to estimate the action values Q. The minimum of the estimated action values, $Q(s_t, a_t)$, is selected. The actor network parameters are updated using the gradient descent algorithm.

$$E\left[\log\pi(s_t) - \frac{1}{\alpha}Q(s_t,a_t)\right] \tag{8}$$

The temperature coefficient α is used to measure the relative importance between rewards and entropy. It is reasonable to have α that can vary, and its update process is shown in Eq. (9), where the target entropy $\hat{H}$ is a hyperparameter defined as the negative size of the action space, −2, in this paper.

$$E\left[-\alpha \log \pi\left(s_t\right)-\alpha \hat{H}\right] \tag{9}$$

In the simulated world, the agent relies on sensor readings to make action decisions and stores the interaction trajectories with the environment in an experience replay buffer. When the number of experiences in the buffer reaches the batch size, the neural networks, including the policy network, critic network, target critic network, and the temperature coefficient, are updated sequentially. If the number of experiences in the buffer is insufficient to reach the batch size, the agent continues to interact with the environment to collect more experiences. The training process ends when the agent reaches the predefined training iterations. This is the update process of the neural networks in the simulated world, as depicted in Fig. 6.

3.2 Behavior cloning

In RL, the learning process through interaction with the environment can be slow. Applying supervised learning methods to autonomous navigation involves directly mapping input states to output actions. This can be seen as a regression problem, as it attempts to predict a continuous output value (action). In this case, the approach of using many labeled training examples to train a model and learn the mapping relationship between input states and output actions is known as

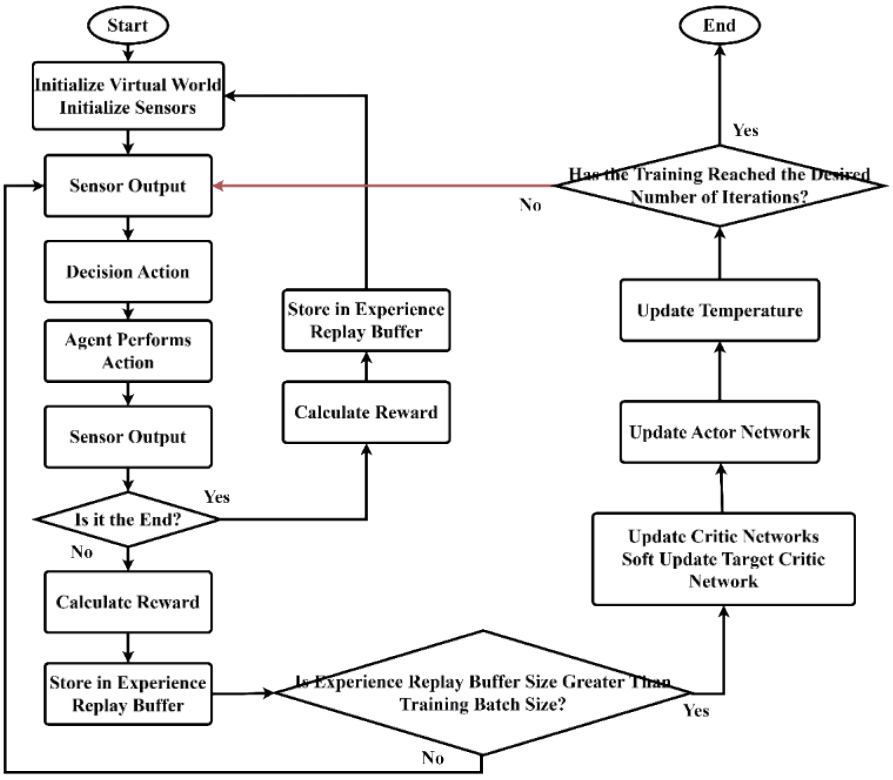


Fig. 6. (Color online) Training process of neural networks in simulated world.

behavior cloning. However, supervised learning often requires a large amount of labeled training data, which can be challenging for real-time applications such as autonomous navigation.

To overcome the limitation of requiring a large amount of labeled data in supervised learning, a combination of supervised learning and RL is employed. Supervised learning is used to train a set of initial parameters, and then RL is used to further optimize these initial parameters. Such a combination approach is commonly referred to as hybrid learning and enables the rapid training and optimization of agents without the need for a large amount of labeled data.

As shown in Fig. 7, the state is first extracted from the sensor readings in the simulated world. Then, human-defined decision actions are used as labels for supervised learning. The same state is inputted into the initial network of the actor network. The network is trained using the least squares method as the loss function to update the decision network of SAC. Here, θ_π represents the decision network of SAC, and ρ denotes the learning rate for supervised learning.

3.3 YOLO-v7-tiny network

Despite the model's excellent performance in the simulated world, real-world applications may encounter various challenges and difficulties. For instance, as mentioned in Sect. (3) regarding the design of the state space, finding the center point of the target object, Target, in the real world is not as straightforward as it is in the simulated world using binary thresholding methods.

As shown in Fig. 8, (a) and (b) depict the images captured by the camera in the simulated and real-world environments, respectively. The binary thresholding results of the images (a) and (b) are shown in (c) and (d), respectively. It can be observed that in the simulated world, the image (a) after binary thresholding in (c) clearly highlights the target object, making it easy to find the center point of the target. Therefore, in this paper, YOLO-v7-tiny[13] will be utilized to locate the center point, Target, of the target object in the state space.

YOLO-v7-tiny is divided into a backbone network and a head layer, and the architecture of the backbone network is illustrated in Fig. 9. In this architecture, CBL(−1, 32, 33, s = 2)

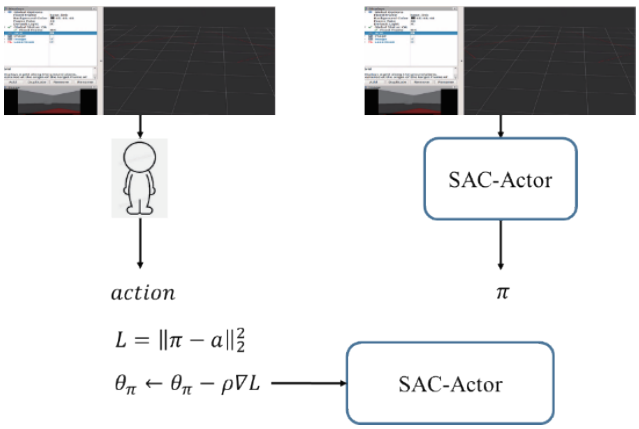


Fig. 7. (Color online) Diagram of supervised optimization for RL training.

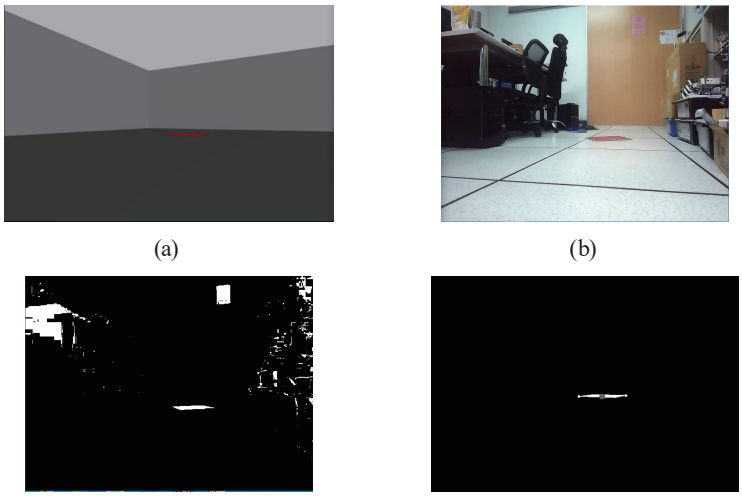


Fig. 8. (Color online) Comparison of state space in simulated and real-world environments. (a) Gazebo agent capturing images, (b) real-world agent capturing images, (c) binary thresholding of Gazebo images, and (d) binary thresholding of real-world images.

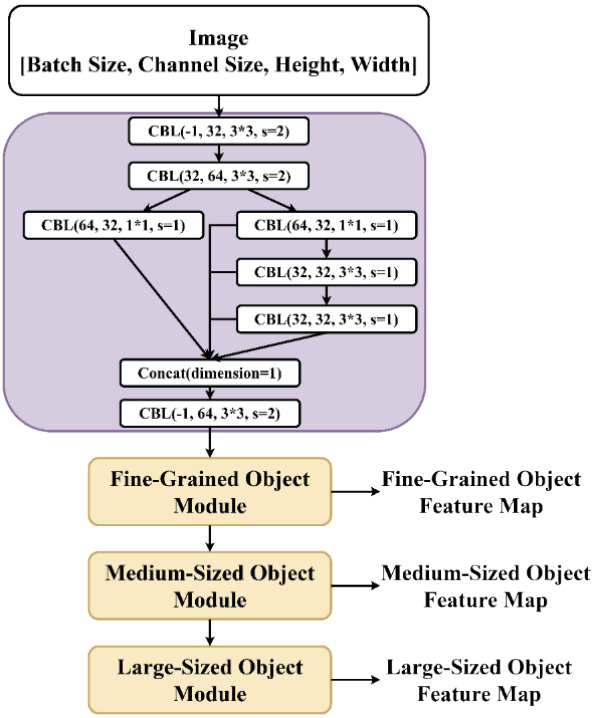


Fig. 9. (Color online) Diagram of the YOLO-v7-tiny backbone network.[13]

represents a convolutional layer followed by batch normalization[14] and then a LeakyReLU activation function.[15] The specific parameters for CBL(−1, 32, 33, s = 2) are as follows: assuming the input image size is [1, 3, 600, 480] (with a batch size of 1, an RGB image with 3 channels, and dimensions of 600 and 480), −1 denotes the depth of the previous layer, which is 3. CBL(−1, 32, 33, s = 2) performs a dot product with 32 convolutional kernels of size 33, where the

kernels move two pixels at a time. The resulting feature map has a size of [1, 32, 300, 240]. The operation Concat(dimension = 1) indicates that the feature map will be concatenated along the first dimension, which corresponds to the depth dimension.

In YOLO-v7-tiny, the backbone network extracts three different sizes of feature maps and passes them to the head layer, as illustrated in Fig. 10. These three feature maps are obtained from different layers of the backbone network. For instance, the feature map extracted from the last layer is specialized for detecting large objects and thus contains more distinctive features compared with the previous two feature maps. In the head layer, these three feature maps are upsampled to the same size and then subjected to feature extraction and fusion. Here, the Upsample(2, "nearest") layer indicates a 2× upsampling scale using the nearest-neighbor interpolation method. The final output of the head layer comprises information about three sizes of bounding boxes (bbox). Subsequently, non-maximum suppression (NMS) is applied to eliminate redundant boxes, retaining only the bbox with the highest probability as the final predicted box.

3.4 Path planning and navigation

When the agent is unable to perceive the target object, the neural network may struggle to make informed decisions. Therefore, in this study, we employed a path planning navigation algorithm to guide the agent along a predefined path until the target object becomes visible. The

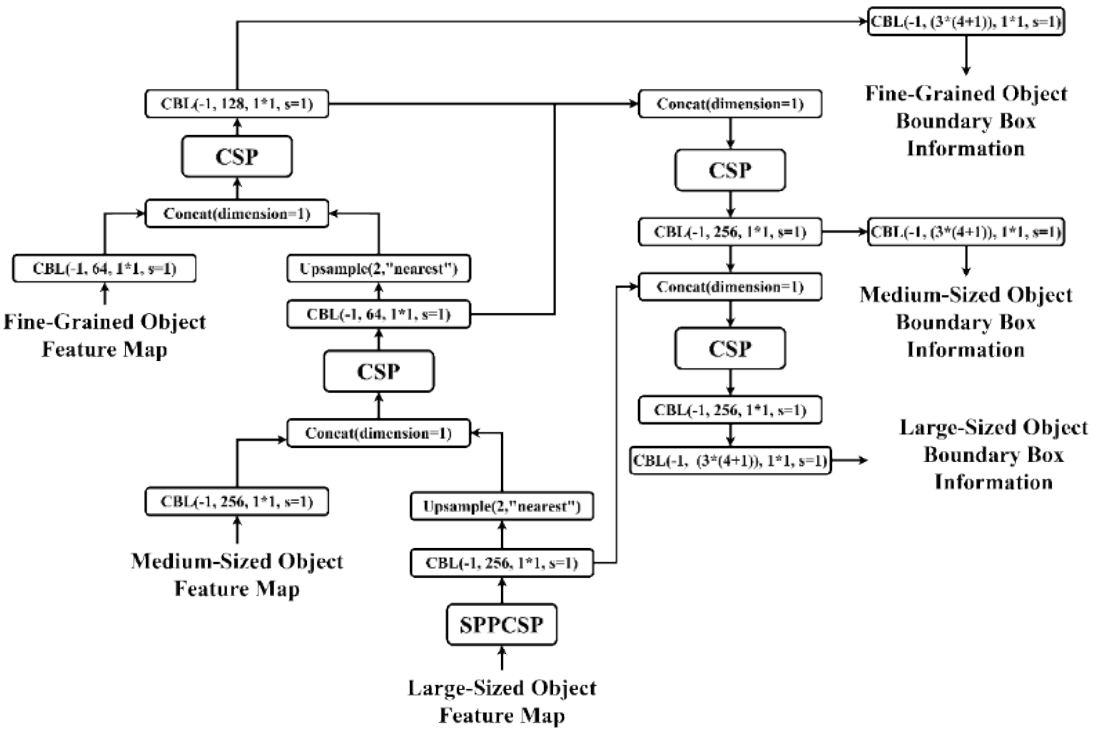


Fig. 10. Diagram of the YOLO-v7-tiny head layer.

A search algorithm (A*)[16] in combination with the dynamic window approach (DWA)[17] algorithm will be utilized to implement the path planning navigation when the agent cannot detect the target object. This approach is based on a pre-established map and localization system for navigation.

Initially, the environment undergoes map construction, and the agent's own position is estimated using angular and linear velocities. The A*+DWA algorithm is then employed to calculate the optimal path for the agent along the map. The implementation leverages the ROS Navigation package.[18]

3.5 Integrated E2E and AMR autonomous navigation system

In the neural network and traditional path planning and navigation algorithms, each has its own advantages in controlling mobile vehicles. In this study, the neural network relies on camera images and the feedback from 2D LiDAR to make path decisions for navigation without the need for precise localization. However, it may struggle to make optimal decisions when the target object is not visible. On the other hand, the traditional A*+DWA algorithm relies on a constructed map and precise localization to accomplish navigation.

The proposed system flowchart is depicted in Fig. 11. At the beginning of the system, the camera continuously captures the environment and employs the trained YOLO-v7-tiny object detection to detect the target object. If the target object is detected, the state is provided to the neural network for action decision-making. When the target object is not detected, the system utilizes the A* algorithm to plan a global path and employs the DWA algorithm for navigation. This process is repeated until the target object is reached.

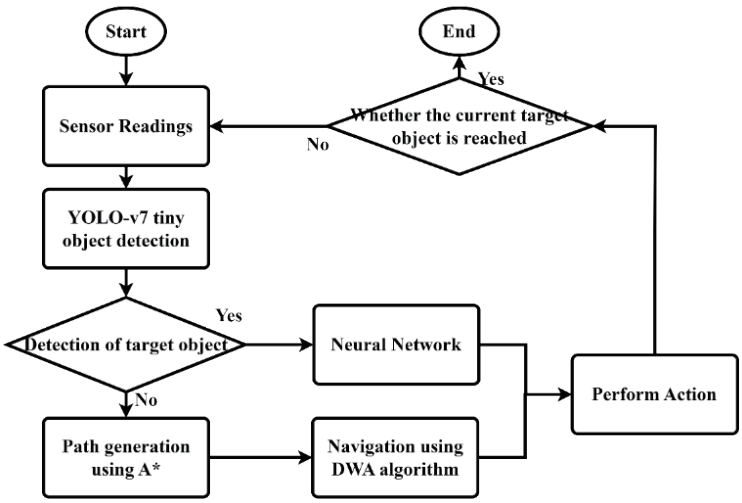


Fig. 11. System flowchart.

4. Experiments

4.1 Experimental procedure and objectives

We conducted a detailed experimental evaluation of the proposed autonomous navigation framework. First, the stability of the neural network training will be tested in the simulated world using Gazebo.[19] We will explore the enhancement of RL by incorporating behavior cloning as a training optimization method. Additionally, the performance characteristics of RL algorithms, path planning and navigation techniques, and the proposed combined approach will be compared. Subsequently, the trained network will be transferred to the real-world environment for evaluation, and experimental assessments will be conducted to validate the effectiveness and practicality of the proposed autonomous navigation framework presented in this paper.

4.2 Experiments in Gazebo

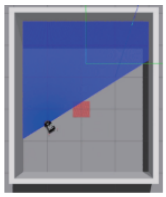
In Gazebo, three simulated world environments were created for training purposes, namely, the obstacle-free, static obstacle, and random moving obstacle maps. These three environments correspond to three different reward schemes. As shown in Fig. 12, when the agent is in an environment without obstacles, it should strive to reach the target and avoid colliding with the surrounding walls. Therefore, the reward includes a combination of localization distance, angle, and action rewards, designed to encourage the agent to achieve the goal of moving towards the target. In the map with static obstacles (b), the agent should be cautious in avoiding obstacles and reaching the target as rapidly as possible. Hence, it is necessary to provide a reward that guides the agent in avoiding obstacles, in addition to the rewards used in (a). In the case of the map with dynamic obstacles (c), the rewards remain the same. The agent should be attentive in avoiding dynamic obstacles while making progress towards the target, taking into account the movement of the obstacles as well.

4.2.1 Map 1

The SAC algorithm was employed for RL training in the obstacle-free map, using the cumulative reward per episode as the evaluation criterion. The sampling of the state space was based on the number of points in the LiDAR point cloud, and selecting different numbers of points could affect the stability of the training. Therefore, in the same obstacle-free map, we utilized 24, 50, and 100 2D LiDAR return values as inputs to the neural network and evaluated their performance.

As shown in Fig. 13, in terms of training performance, using 24 points yields slightly better results than using 50 and 100 points. In this experimental environment, utilizing 24 2D LiDAR return values outperforms using 50 and 100 2D LiDAR, indicating that achieving better training outcomes with fewer inputs is possible.

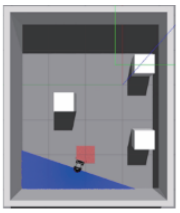
The method that combines RL and DL first involves manually controlling the agent to navigate through the obstacle-free map for 20 episodes, collecting 869 state-action pairs. Then,



4*5m Obstacle-free

$reward = r_{dis} + r_{angle} + r_{action}$

(a)



4*5m Static obstacles

$reward = r_{dis} + r_{angle} + r_{scan} + r_{action}$

(b)

4*5m Random moving obstacles

$reward = r_{dis} + r_{angle} + r_{scan} + r_{action}$

(c)

Fig. 12. RL maps and rewards. Maps (a) 1, (b) 2, and (c) 3.

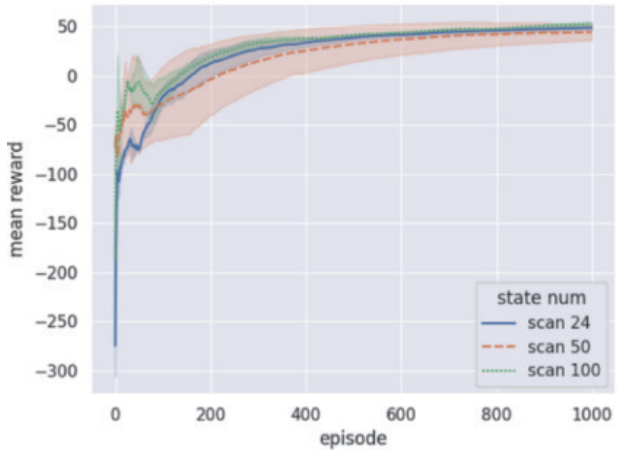


Fig. 13. (Color online) Testing with different state quantities.

DL is employed for pretraining, conducting 500 training iterations with a learning rate of 3×10^{-4}. Subsequently, the SAC algorithm is applied for further training. The results, as shown in Fig. 14, clearly demonstrate that the SAC algorithm with the integration of DL outperforms the standard SAC algorithm in terms of average reward.

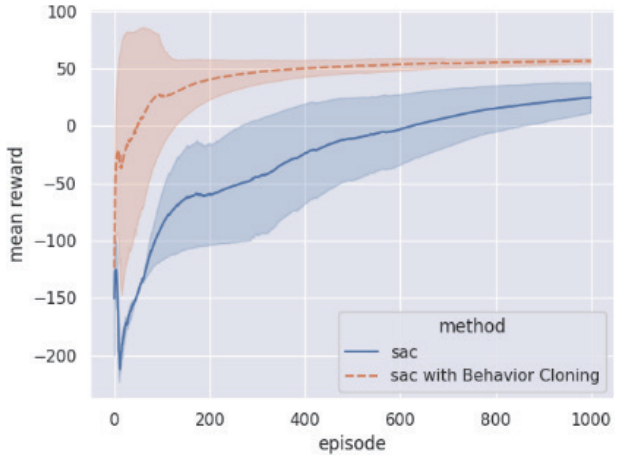


Fig. 14. (Color online) SAC combined with behavior cloning results chart.

4.2.2 Map 2

In the static obstacle map, the 2D LiDAR returns were also divided into 24, 50, and 100 points as inputs to the neural network. On the basis of the results from Fig. 15, it can be observed that in this map, the neural network using 24 2D LiDAR points as input slightly outperforms the networks with 50 and 100 points. Therefore, the choice of using 24 points as input for pretraining the neural network, as shown in Fig. 16, ensures a more stable convergence performance.

4.2.3 Map 3

In the dynamic obstacle map, the neural network's input state does not converge effectively with 50 LiDAR data points, while with 24 data points, it achieves convergence but exhibits significant fluctuations. However, with 100 data points, it demonstrates the most stable performance, as illustrated in Fig. 17.

Similarly, under this optimal condition, utilizing expert data-based DL pretraining leads to faster convergence but with relatively less stability, as shown in Fig. 18. This could be attributed to the interference or lack of adaptability of the model's pretraining with expert data under specific input states.

In comparing the performance characteristcs of various algorithms on different maps, the evaluation metric is based on predefined rewards. The compared algorithms include RL, traditional algorithms, RL with DL pretraining, standalone deep learning, and the neural network integrated with path planning and navigation proposed in this paper. In the previous experiments, three distinct maps were used, namely, the map without obstacles, the map with static obstacles, and the map with randomly moving obstacles (Maps 1 to 3, respectively).

Table 1 presents the comparison results of various algorithms on different maps. In the table, SAC represents the RL algorithm, navigation denotes the traditional algorithm, SAC with behavior cloning refers to the RL algorithm with DL pretraining, Behavior cloning represents

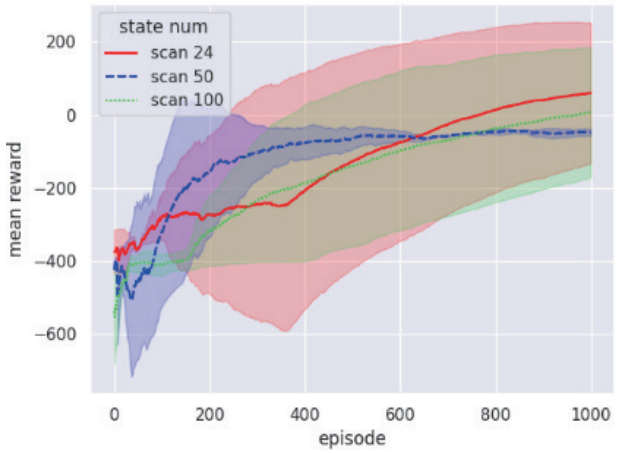


Fig. 15. (Color online) Testing of different state quantities in static obstacle maps.

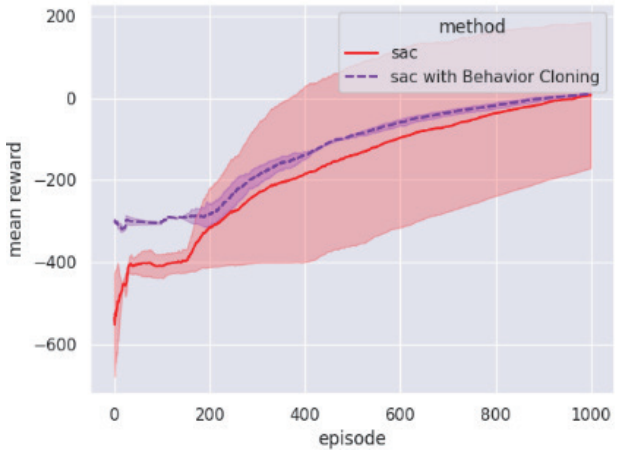


Fig. 16. (Color online) Testing of static obstacle map combined with behavior cloning.

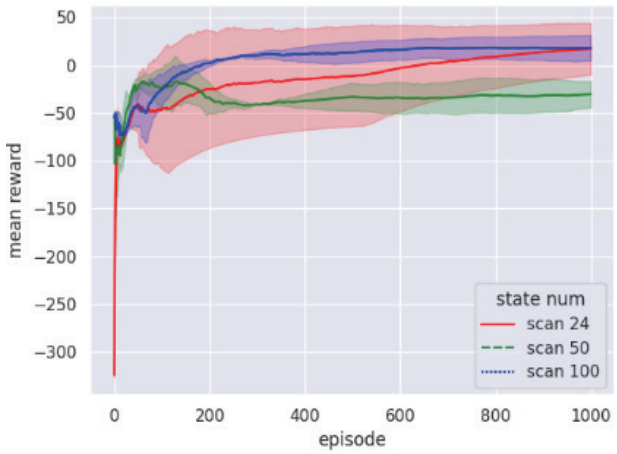


Fig. 17. (Color online) Testing of different state quantities in dynamic obstacle maps.

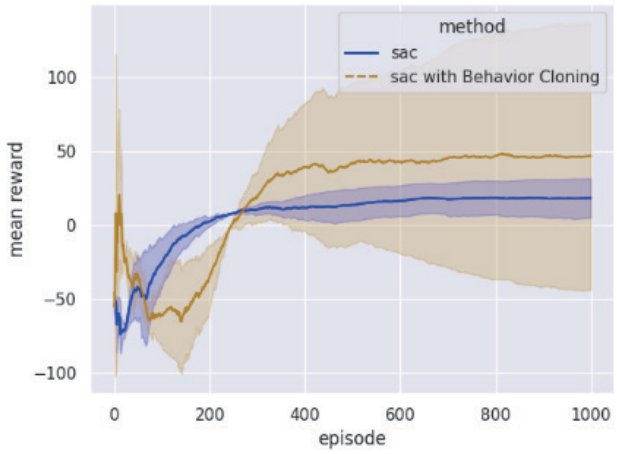


Fig. 18. (Color online) Testing of dynamic obstacle map combined with behavior cloning.

Table 1
Algorithm architecture comparison table.

Map	Method				
	SAC	Navigation	SAC with behavior cloning	Behavior cloning	Navigation with SAC (ours)
Map 1	105	126	110	−9	119
Map 2	40	105	42	−53	117
Map 3	114	94	13	−12	119

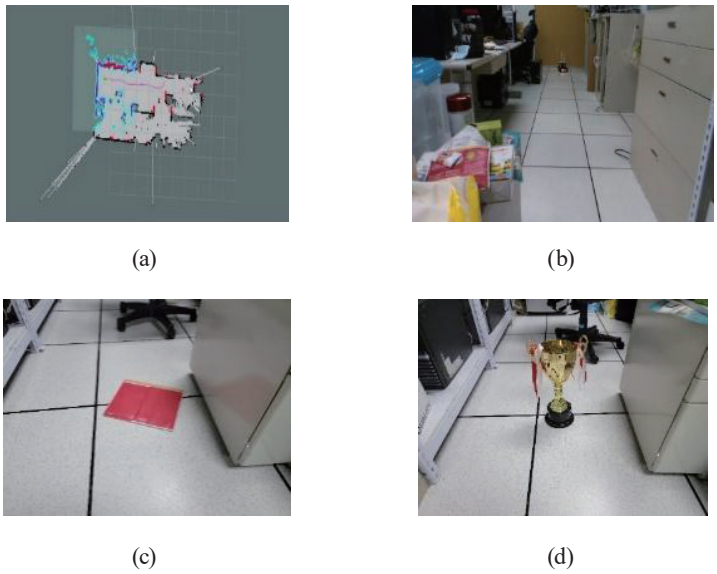
the DL algorithm, and Navigation with SAC represents the proposed approach of integrating neural network with path planning and navigation. To simulate real-world localization bias, the navigation algorithm and navigation with SAC in the aspect of localization are subjected to a certain degree of noise. The SAC algorithm utilizes the best-performing network from the previous section, and the DL network uses a pretrained network. In each algorithm, the agent should reach the target point 10 times, and the average reward per episode is used to evaluate the algorithm's performance. It can be observed that the proposed method in Maps 2 and 3 outperforms the other four algorithms, while navigation excels in the obstacle-free map compared with other algorithms.

4.3 Experiments in real world

The proposed framework was applied in a real-world environment using Limo[20] as the experimental platform.

4.3.1 Only navigation

In the navigation algorithm, the process involves map construction followed by path planning and navigation, as illustrated in Figs. 19(a) and 19(b). The agent aims to reach the vicinity of the target locations (c) and (d). This map-based navigation approach can only provide approximate



(a) (b)

(c) (d)

Fig. 19. (Color online) Navigation diagram. (a) Map construction. (b) Point of departure. (c) Plane target (red box target). (d) Trophy target.

coordinates of the target locations, which is why in this paper, we introduce a certain degree of noise in the localization aspect for the navigation algorithm and the proposed navigation with SAC method, as presented in Table 1.

As shown in Fig. 20, the navigation algorithm effectively guides the agent towards the target locations. However, since the actual coordinates of the target locations are estimated manually, the real-world agent does not reach the exact positions of the targets.

4.3.2 Only neural network

When the neural network, trained through RL, is applied to the real-world environment, it is integrated with YOLO-v7-tiny to achieve target perception and recognition. Figure 21 illustrates the third-person perspective of the neural network applied to the real-world Limo motion trajectory. The results demonstrate that when Limo detects a target object, it autonomously moves towards the target. However, in cases where Limo cannot visually perceive the target object, it initiates self-rotation to search for the target. Nevertheless, it may encounter difficulties in accurately locating the target when it is far away.

When the RL network is transferred to the real-world environment and detects the target object while moving, it encounters challenges due to the effect of time series. Assuming at time t, the neural network makes action decisions v and ω on the basis of the environmental state, Limo should be located at coordinates (x, y). According to the original design, the difference between time intervals t and $t + 1$ should approach zero. However, in the real world, Limo is actually positioned at coordinates $(\dot{x}, \dot{y})$, leading to a deviation between the observed environmental state at time $t + 1$ and the ideal scenario, which in turn results in significant oscillations and instability in Limo movements.

Fig. 20. (Color online) Navigation process illustration (from left to right, from top to bottom).

Fig. 21. (Color online) Navigation process illustration, starting from the first image in the first row and ending at the third image in the last row.

In this paper, to address the issue of significant oscillations, a modification was made to the tanh activation function by changing its slope through tanh compression. As depicted in Fig. 22, the tanh input was multiplied by a value between 0 and 1, resulting in a smooth compression of the action distribution. This modification effectively mitigated the problem of severe oscillations in the agent's movements.

Limo was instructed to search for the target object three times in a map without obstacles, and the trajectories are illustrated in Fig. 23. The black trajectory represents the movement without any modification to the tanh slope, while the red trajectory shows the result after changing the tanh slope. It can be observed that by modifying the tanh slope, the severe oscillations in Limo's movement were effectively resolved, resulting in smoother trajectories.

When using neural networks alone, achieving satisfactory static obstacle avoidance results in the real world can often be challenging. Therefore, in this paper, we introduce some constraints when the neural network outputs actions. Environmental perception information is obtained by reading 2D LiDAR returns. To ensure that the acquired 2D LiDAR returns have a consistent

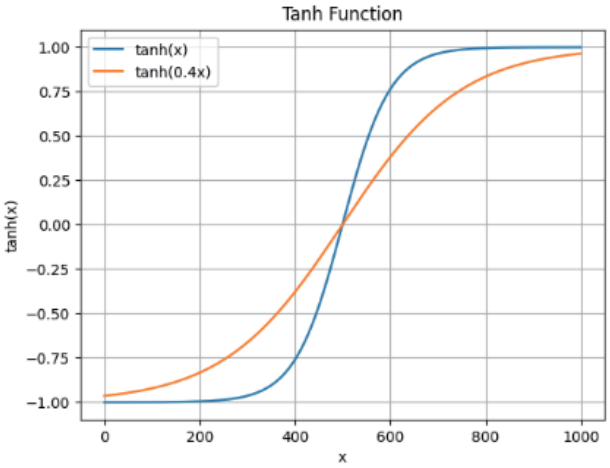


Fig. 22. (Color online) Diagram illustrating the change in tanh slope.

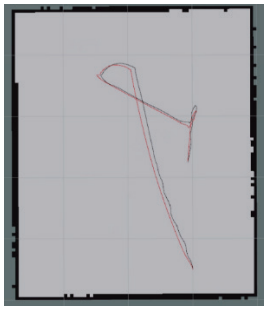


Fig. 23. (Color online) Limo trajectory diagram.

number of points, a fixed interval sampling of c points is adopted. To ensure that the measurements for left and right turns are equal, c is chosen as an even number, as shown in Eq. (10). The relationship between re_{scan} and the real 2D LiDAR is depicted in Fig. 24.

$$re_{scan} = \left[scan_1, scan_2, \cdots, scan_c\right] \tag{10}$$

Subsequently, the minimum value in re_{scan} is identified, and when this value is below a certain threshold, the action is switched from being output by the neural network to directly selecting the index of the minimum value in re_{scan}. The angular velocity for obstacle avoidance is then chosen as the corresponding value from the range of angular velocities ω^{-}, defined in Eq. (11), while the linear velocity is set to 0.22 m/s. The ω^{-} range is formed by concatenating two lists, one ranging from 0 to 2.5 divided into $c/2$ parts, and the other ranging from −2.5 to 0 divided into $c/2$ arts. The interval size $\phi\omega$ for each part can be calculated by dividing the total range of 2.5 by the number of parts, $c/2$. The neural network integrates the obstacle avoidance algorithm, as shown in Algorithm 1.

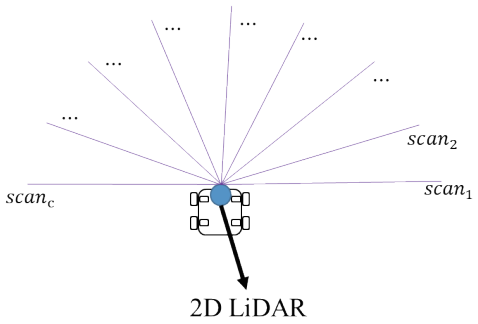


Fig. 24. (Color online) Diagram depicting the relationship between re_{scan} and physical 2D LiDAR.

Algorithm 1. Neural network combined with obstacle avoidance algorithm

Input: re_{scan}, c, *state*
Output: v, ω
function AVOIDANCE(re_{scan}, c, *state*)
  v ← 0.22 m/s
  $\omega \leftarrow [0, \phi\omega, 2\phi\omega, ..., (c/2-1)\phi\omega, 2.5, -(c/2-1)\phi\omega, ..., -\phi\omega, 0]$
  if min(re_{scan}) < z **then**
    $index \leftarrow$ argmin(re_{scan})
    $\omega \leftarrow \omega[index]$
    return v, ω
  else
    $v, \omega \leftarrow network(state)$
    return v, ω
  end if
end function

$$\omega^{-}: \left[0, \phi\omega, 2\phi\omega, \ldots, \left(\frac{c}{2}-1\right)\phi\omega, 2.5, -2.5, -\left(\frac{c}{2}-1\right)\phi\omega, \ldots, -\phi\omega, 0\right] \text{rad/s} \tag{11}$$

Figure 25 demonstrates the application of the neural network with the obstacle avoidance algorithm in a real-world environment containing numerous static obstacles. In this scenario, Limo successfully avoids obstacles and reaches the target point with high efficiency.

In Fig. 26, the system effectively navigates around dynamically appearing obstacles, demonstrating its ability to avoid obstacles and reach the target point in such dynamic environments.

4.3.3 Navigation with SAC

The proposed framework, which combines both approaches, is illustrated in Fig. 27. The results demonstrate that by integrating the two algorithms, the system effectively addresses the challenges of navigating towards distant target objects and dealing with cases where target localization is not possible.

Figure 28 illustrates the results of tracking the trophy target object. The proposed method performs well across different categories of target objects and only requires training a single YOLO-v7-tiny model to handle various types of target.

Fig. 25. (Color online) Results of static obstacle environment testing.

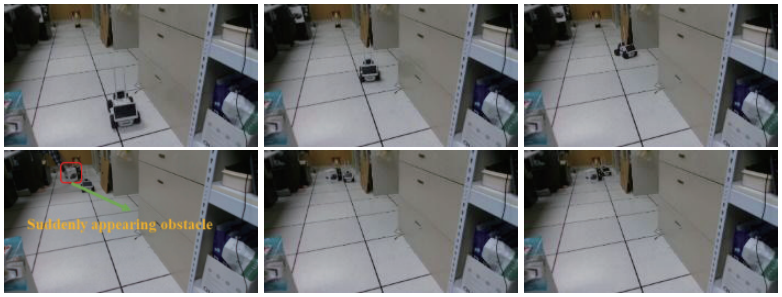


Fig. 26. (Color online) Results of dynamic obstacle environment testing.

Fig. 27. (Color online) Diagram illustrating the fusion of traditional navigation and RL navigation to track a red box target object.

Fig. 28. (Color online) Diagram illustrating the fusion of traditional navigation and RL navigation to track a trophy target object.

5. Conclusion

In this paper, we proposed the architecture of an autonomous navigation system for indoor mobile robots, which successfully combines the advantages of E2E autonomous driving and traditional navigation algorithms to overcome the challenge of traditional navigation algorithms relying too much on high-precision positioning. The E2E module is designed using the deep RL method, and the autonomous navigation system is realized by using 2D-LiDAR and the camera combined with the object detection technology YOLO-v7-tiny. Through behavioral replication and stability training, the deep RL network is successfully migrated to the real world, improving the adaptability and stability of the system in the real environment. At the same time, the A* algorithm is used for global optimal path planning, and the DWA algorithm is combined to complete local path planning, which effectively solves the problem that RL neural networks are difficult to make good decisions when making decisions.

Acknowledgments

This work was supported by the Ministry of Science and Technology, Taiwan, under contract number MOST 110-2221-E-167-034.

References

1 M. Shneier and R. Bostelman: NIST (2015) 1. https://doi.org/10.6028/NIST.IR.8022
2 C. Cadena, L. Carlone, H. Carrillo, Y. Latif, and D. Scaramuzza: IEEE Trans. Rob. **32** (2016) 1. https://doi.org/10.1109/TRO.2016.2624754

3 A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad: IEEE Trans. Neural Networks Learn. Syst. **33** (2022) 6. https://doi.org/10.1109/TNNLS.2020.3043505
4 D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis: Nature **529** (2016) 1. https://doi.org/10.1038/nature16961
5 Y. Bai, J. Wu, A. Krause, and P. Abbeel: arXiv. (2022). https://doi.org/10.48550/arXiv.2204.05862
6 S. H. Chan, P. T. Wu, and L. C. Fu: 2018 IEEE Int. Conf. Systems, Man, and Cybernetics (SMC) (IEEE, 2018) 1263.
7 P. Hao and Z. Q. Deng: 2006 6th World Congress on Intelligent Control and Automation (2006) 8178.
8 V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis: Mach. Learn. (2013) 1. https://doi.org/10.48550/arXiv.1312.5602
9 T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine: Int. Conf. Machine Learning (2018).
10 A. Pomerleau: Proc. Adv. Neural Inf. Process. Syst. (1989) 305–313.
11 M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba: (2016) 1. https://doi.org/10.48550/arXiv.1604.07316
12 A. Yu, R. Palefsky-Smith, and R. Bedi: arXiv (2016) 1. https://doi.org/10.48550/arXiv.1604.07316
13 A. Y. Wang, A. Bochkovskiy, and H. Y. M. Liao:arXiv (2022). https://doi.org/10.48550/arXiv.2207.02696
14 S. Ioffe and C. Szegedy: (2015) 1. https://doi.org/10.48550/arXiv.1502.03167
15 A. L. Maas, A. Y. Hannun, and A. Y. Ng: Proc. ICML **30** (2013) 1–3. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf
16 P. E. Hart, N. J. Nilsson, and B. Raphael: IEEE Trans. Syst. Man Cybern.: Syst. **4** (1968) 1. https://doi.org/10.1109/TSSC.1968.300136
17 D. Fox, W. Burgard, and S. Thrun: IEEE Rob. Autom. Mag. **4** (1997) 22. https://doi.org/10.1109/100.580977
18 ROS Wiki:https://wiki.ros.org/navigation#References (accessed September, 2021).
19 Gazebo: https://gazebosim.org/home (accessed March, 2023).
20 Agilex Robotics. (n.d.). Limo: A ROS-based Intelligent Mobile Robot. GitHub https://github.com/agilexrobotics/Limo-doc (accessed March, 2023).