Applied Intelligence (2022) 52:15600–15616
https://doi.org/10.1007/s10489-022-03191-2

Robot navigation in a crowd by integrating deep reinforcement learning and online planning

Zhiqian Zhou[1] · Pengming Zhu[1] · Zhiwen Zeng[1] · Junhao Xiao[1] · Huimin Lu[1] · Zongtan Zhou[1]

Accepted: 5 January 2022 / Published online: 17 March 2022

Abstract
Navigating mobile robots along time-efficient and collision-free paths in crowds is still an open and challenging problem. The key is to build a profound understanding of the crowd for mobile robots, which is the basis of a proactive and foresighted policy. However, since the interaction mechanisms among pedestrians are complex and sophisticated, it is difficult to describe and model them accurately. For the excellent approximation capability of deep neural networks, deep reinforcement learning is a promising solution to the problem. However, current model-free learning-based approaches in crowd navigation always neglect planning and still lead to reactive collision avoidance policies and shortsighted behaviors. Meanwhile, most model-based learning-based approaches are based on state values, imposing a substantial computational burden. To address these problems, we propose a graph-based deep reinforcement learning method, social graph-based double dueling deep Q-network (SG-D3QN), that (i) introduces a social attention mechanism to extract an efficient graph representation for the crowd-robot state, (ii) extends the previous state value approximator to a state-action value approximator, (iii) further optimizes the state-action value approximator with simulated experiences generated by the learned environment model, and (iv) then proposes a human-like decision-making process by integrating model-free reinforcement learning and online planning. Experimental results indicate that our approach helps the robot understand the crowd and achieves a high success rate of more than 0.99 in the crowd navigation task. Compared with previous state-of-the-art algorithms, our approach achieves better performance. Furthermore, with the human-like decision-making process, our approach incurs less than half of the computational cost.

Keywords Crowd navigation · Mobile robot · Deep reinforcement learning · Online planning

1 Introduction

In the last two decades, mobile service robots have been introduced into various crowded environments, such as campuses, canteens, hospitals, and exhibition halls. To perform their tasks (e.g., patrolling, carrying food, transporting pharmaceuticals, and guiding visitors), they continuously move through the crowd. In these scenarios, navigating mobile robots along collision-free and time-efficient paths is a fundamental and crucial problem. Despite the undeniable progress made in recent years, it remains an open and challenging problem.

The main challenge comes from pedestrians, who have their policies and intents and make their decisions independently. Various explicit or implicit interaction mechanisms facilitate collaboration in the crowd. These interaction mechanisms are inherently complex and difficult to describe and model accurately. However, without an explicit model of crowd interactions, mobile robots cannot develop an understanding of crowd scenarios, let alone construct a proactive and foresighted collision avoidance strategy.

Various approaches have been proposed to address the problem. Model-based approaches, such as social force models [1, 2] and velocity obstacles [3–5], aim to simplify the interaction mechanisms. They facilitate the finding of collision-free action in a minimal computation time. However, oversimplified and reactive interaction mechanisms limit the robot's ability to understand various

This work was supported by the National Natural Science Foundation of China [61773393, U1913202, U1813205]

✉ Huimin Lu
lhmnew@nudt.edu.cn

Extended author information available on the last page of the article.

crowd scenarios and result in shortsighted and occasionally unnatural behaviors.

Other approaches, categorized as trajectory-based approaches, separate the crowd navigation task into two disjoint subtasks: first predicting pedestrians' trajectories and then planning collision-free paths. Various complex models, such as deep neural networks, have been used to describe the interaction mechanisms explicitly. They have achieved amazing performance on pedestrian trajectory datasets [6–8]. However, only a small portion of them have been applied in crowd navigation [9, 10]. The main reason is that they always suffer from high computational costs with increasing crowd size and density. In addition, the *freezing robot problem* [11] is also a common problem in trajectory-based approaches, in which predicted trajectories cover such a large portion of the space that the robot cannot find a plausible path [12].

With the rapid development of machine learning algorithms, some researchers have turned their attention to reinforcement learning (RL), especially deep reinforcement learning (DRL). These learning-based approaches aim to find an optimal policy and gain the maximal long-term return. Depending on the form of state inputs of deep neural networks, they can be further divided into sensor-level and agent-level approaches. Sensor-level approaches take raw sensor readings as state inputs, such as laserscans [13] or images. They have the advantage that all perceptual information is directly fed into the networks. However, building a good understanding of crowd scenarios is difficult due to the lack of explicit high-level representations.

In contrast, agent-level approaches require agent-level state inputs [14–17], which are extracted with various clustering, tracking, and multi-sensor fusion algorithms. The most immediate and important benefit is that the agent-level representation enables explicit extraction of the structure information and analysis of the crowd interaction mechanism. In addition, agent-level representation is useful for incorporating trajectory predictions into a learning-based planning framework [10], which is the key to a proactive and foresighted navigation policy. However, these approaches still face two key challenges. The first is to deal with the changing crowd size. Typical deep neural networks require a predetermined input size. The second challenge is to aggregate the neighbors' information efficiently.

In this work, we focus on integrating model-free DRL with online planning to provide a proactive and foresighted crowd navigation method named social graph-based double dueling deep Q-network (SG-D3QN). Here, the D3QN is used to generate coarse state-action values and select candidate optimal actions intuitively, while online planning is built to further refine the coarse state-action values by performing rollouts. The illustration of SG-D3QN can be found in Fig. 1.

Compared with previous works, the proposed framework is improved in the following points. First, the agent-level crowd states are constructed as graph data and the social attention mechanism is introduced to aggregate neighbors' information selectively. Second, the previous state value estimator is extended to the D3QN, coarsely and intuitively estimating state-action values. To improve the accuracy of the state-action value estimator, a large number of simulated experiences generated by the learned environment model are introduced into the training process. Third, a human-like decision-making process is built by integrating the D3QN and online planning. Benefitting from the forward-moving action-clipping, the proposed method significantly reduces the computational cost of each decision. Then, the reward function is redesigned based on current navigation scenarios to enhance the training convergence. Later, a large number of simulation experiments are designed to evaluate the proposed approach. Experimental results show that the proposed method helps the robot understand the crowd scenario well, outperforms state-of-the-art methods, and can be extended to non-holonomic robots and visible scenarios. Finally, a hardware experiment on a Fetch robot proves that the proposed method be migrated to real robots. Open-source code and a hardware demonstration video are available at github.com/nubot-nudt/SG-D3QN.

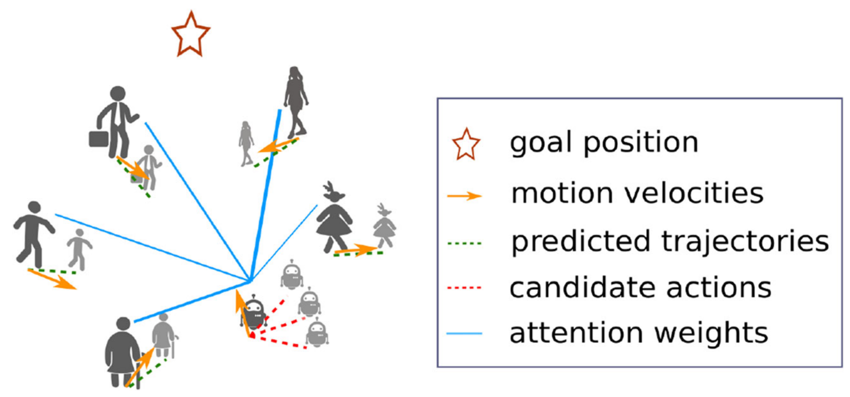


Fig. 1 Illustration of SG-D3QN. When moving in a crowd, the robot selectively aggregates pedestrians' information with social attention weights, generates the candidate optimal actions by ranking the coarse state-action values, and refines the coarse state-action values by performing rollouts on the current state

2 Related work

In this section, we summarize prior work on agent-level learning-based crowd navigation, and then, we briefly introduce related work on graph neural networks (GNNs). Finally, we review related work on integrating planning and learning.

2.1 Agent-level learning-based crowd navigation

Agent-level learning-based approaches originate from the multi-robot collision avoidance problem. In [16], Chen et al. pioneeringly developed a decentralized multi-agent collision avoidance algorithm based on DRL. They built a value network to estimate the expected time to the goal. Many researchers followed this work and proposed various improvements. To address varying crowd size, Everett et al. used long short-term memory (LSTM) to encode the crowd state into a fixed-length representation [18, 19]. Pedestrians' states were fed into the recurrent network according to their distances to the robot. The recurrent architecture simplified the interaction mechanism and did not fully exploit the structural information of the crowd. A further method was proposed in [14], in which neighborhood agents were encoded in an occupied map. However, since the occupied map was coarse-grained and local, some structural information was still lost. In more recent works, researchers realized the power of GNNs to extract structural information. In [15], two graph convolutional networks (GCNs) were trained: one for encoding the robot-robot state and one for predicting attention weights. Another temporally parallel work can be found in [20], where the crowd was modeled as a relational graph.
Inspired by [20] and [15], a similar graph-based representation of the crowd is built in this work. Our work differs from their approach in selectively aggregating neighbors' information. The attention weights in [15] are trained based on human gaze data, and the relational weights in [20] are inferred with a Gaussian similarity function. In contrast, in this work, the attention weights are generated by introducing the social attention mechanism, which models attention in the crowd better [21].

2.2 Graph neural network

GNNs are a family of neural networks that can deal with graph-structured data. GNNs explicitly extract high-level and flexible graph representations of the attributes and structure of the graph and are widely used in various classification or regression applications, such as action recognition, relational reasoning, traffic network forecasting, and pedestrian trajectory prediction. Among GNNs, GCNs attract much attention for their simplicity and efficiency. They use a binary adjacency matrix to represent edges between nodes and propagate neighbor information by graph convolution [22], which can take the weighted average of a node's neighborhood information. According to the notion of convolution, GCNs can be further divided into two categories: spectral-based and spatial-based. Spectral-based approaches develop graph convolutions by removing noise from graph signals based on spectral graph theory, while spatial-based approaches redefine graph convolutions by propagating node information along edges. Spatial-based approaches have been rapidly growing in recent years due to their attractive efficiency, flexibility, and generality.

Since a crowd typically produces non-Euclidean data, it is natural for us to build a GCN to extract a high-level representation of the crowd state. Furthermore, because the process of aggregating information depends on interactions among agents, we select the graph attention network (GAT), a typical spatial-based approach, to encode the robot-crowd state. In our implementation, the social attention mechanism [21] is introduced to compute attention weights, which will be described in detail in Section 3.2.

2.3 Integrating planning and learning

Learning and planning are two crucial ideas for solving sequential decision problems. The former aims to generate a state/state-action value estimator by end-to-end training, while the latter uses the prior environment model to perform rollouts and find the optimal policy, e.g., the A* algorithm.

DRL approaches can be divided into two principal categories: model-based RL and model-free RL [23]. Model-free RL approaches model the sequence planning problem similar to a supervised learning task and directly map states to probability distributions of actions. After extensive training, they learn a value estimator of long-term returns. However, because of the lack of a straightforward planning process, the generated strategies are inherently reactive [24]. Model-based RL approaches consider planning as their main component. They derive possible future trajectories based on an a priori environment model, estimate the state/state-action value with learned approximators, and then select the optimal action with search algorithms. As a result, they allow for flexible migration to new environments and avoid extreme situations. A non-negligible problem for them is that they often need an accurate predictive model. As DRL approaches continue to evolve, an increasing number of researchers are becoming aware of these issues and trying to improve them by combining learning with online planning, such as the value iteration network (VIN) [24], value prediction network (VPN) [25], TreeQN [26], and MuZero [27].

When it comes to crowd navigation, many learning-based approaches are model-free, such as most sensor-level learning-based approaches mentioned above [13]. They focus on learning, neglect planning, and still lead to reactive collision avoidance policies and shortsighted behaviors. Other model-based learning-based approaches integrate learning and planning. However, they are based on state values, rather than state-action values, imposing a huge computational burden [14–17, 20]. The main reason is that in the decision-making process, all child states are required to be traversed, although most of them are useless.

Inspired by [25, 26], we extend previous state value approximators to a state-action value estimator in this work. Based on the learned state-action value estimator, a human-like decision-making process is proposed to reduce meaningless expansions, in which the robot first evaluates the state-action values coarsely and intuitively, and then refines the coarse state-action values selectively.

3 Methodology

This section begins with modeling the crowd navigation problem, followed by a newly proposed framework named SG-D3QN, which is shown in Fig. 2. The proposed framework consists of three parts: a two-layer GAT to extract an efficient graph representation from the crowd state, a D3QN to estimate state-action values coarsely, and an online planner to refine coarse state-action values. Later, we further describe the one-step decision process, which constitutes the navigation policy. Finally, many implementation details are introduced at the end of this section.

3.1 Crowd navigation modeling

As a typical sequential decision-making problem, the crowd navigation task can be formulated as a Markov decision

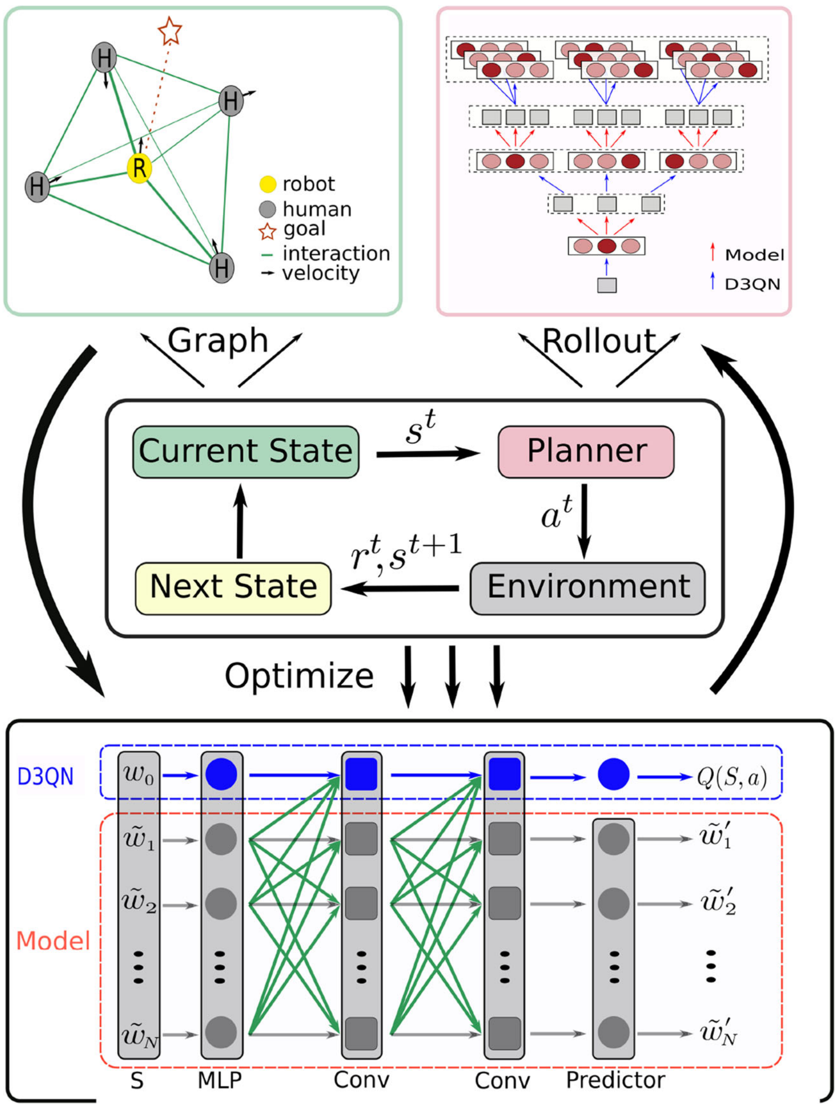


Fig. 2 Framework of SG-D3QN. The crowd-robot state is described in the form of graph data. Then, the D3QN utilizes a two-layer GAT to extract a high-level representation and directly evaluates the state-action values of the current state. The core of the online planner is the rollout performance based on the learned D3QN and environment model, both of which are optimized with simulated trajectories

process (MDP), described by a tuple $M = \langle S, A, P, R, \gamma \rangle$ [16]. S is the state space, A is the action space, P is the state transition model, R is the reward function, and γ is the discount factor. Following the convention, a lowercase with superscript t indicates the variable's value at time t. For example, s^t, a^t and r^t represent the state, the action and the immediate reward at time t, respectively. All of them will be described in detail as follows.

3.1.1 State space

Assumption 1 *The environment is deterministic, and all agents can obtain the position and velocity information of pedestrians accurately.*

Suppose a mobile robot navigates in a crowd of N pedestrians over discrete time steps, and there are $N + 1$ independent agents in the planar workspace. We number agents with subscript i, 0 for the robot and $i(i > 0)$ for the ith pedestrian. For every agent, its configuration can be divided into two parts: the observable state and the hidden state. The observable state consists of the agent's position $\mathbf{p} = [p_x, p_y]$, velocity $\mathbf{v} = [v_x, v_y]$ and radius ρ, while the hidden state includes the agent's intended goal position $\mathbf{g} = [g_x, g_y]$, preferred velocity v_p and heading angle ϕ. The robot can only observe the observable state of pedestrians. Therefore, the robot's state/observation s can be represented as:

$$\begin{aligned} s &= [w_0, \tilde{w}_1, ..., \tilde{w}_N] \\ w_i &= [p_x, p_y, v_x, v_y, \rho, g_x, g_y, v_p, \phi] \in \mathbb{R}^9, i = 0, 1, ..., N \\ \tilde{w}_i &= [p_x, p_y, v_x, v_y, \rho] \in \mathbb{R}^5, i = 1, 2, ..., N \end{aligned} \tag{1}$$

where w_i denotes the full configuration of agent i, and $\tilde{w}_i$ means the observable configuration of pedestrian i. Both of w_i and $\tilde{w}_i$ are described in the robot-centric coordinate, which has an origin at the center of the robot and an x-axis pointing towards the robot's goal.

3.1.2 Action space

Assumption 2 *The robot is modeled as a unicycle, of which the action space consists of 81 discrete actions.*

In this work, we follow the setting of discrete action space from previous works [14, 16, 20]. Specifically, the robot's action at time t can be described by:

$$a^t = (v_0^t, \delta_0^t), \tag{2}$$

where v_0^t and δ_0^t are the target speed and steering angle of the robot. Specifically, the action space consists of the stop action ($a^t = (0.0, 0.0)$) and 80 other discrete actions with 5 values of v_0^t evenly spaced in $(0, v_p]$ and 16 values of δ_0^t evenly spaced in $[0, 2\pi)$. The setting can be easily extended to car-like robots by limiting the steering angle.

3.1.3 State transition model

Assumption 3 *There is no communication among agents, and they make their decisions independently.*

The state transition model $P(s^{t+1}, r^t | s^t, a^t)$ is determined by the agents' kinematics. To simplify the problem, all simulated pedestrians are set as holonomic agents, and their kinodynamic constraints are omitted. Their actions have the same form as a_0^t, but their action space is continuous. Besides, it should be noted that due to the lack of a communication network, agents are never informed of others' actions. Therefore, pedestrians' actions are unknown for the robot and the crowd navigation problem is always modeled further as a partially observable markov decision problem (POMDP) [15, 19, 20]. Then, the state transition model of agent i can be described as:

$$\begin{aligned} \phi_i^{t+1} &= \phi_i^t + \delta_i^t \\ \mathbf{v}_i^{t+1} &= \left[v_i^t cos(\phi_i^{t+1}), v_i^t sin(\phi_i^{t+1}) \right] \\ \mathbf{p}_i^{t+1} &= \mathbf{p}_i^t + \mathbf{v}_i^{t+1} \triangle t. \end{aligned} \tag{3}$$

Here, $\triangle t$ is the time step, which is set to 0.25 s. As mentioned above, the robot has never been informed of pedestrians' actions. Therefore, it only knows its own state transition model and has to build a trajectory predictor for pedestrians to replace their state transition model, such as a linear motion model [14, 16], and a recurrent neural network [20].

3.1.4 Reward function

The reward function is a highly essential point in DRL. However, previous studies [14, 15, 20] directly apply the reward function from [16], which was originally designed to resolve the noncommunicating two-agent collision avoidance problem. As the scene continues to expand, the mismatched reward makes the training process challenging and results in poor training convergence [20]. Therefore, the reward function is redesigned in this work, which includes three parts: r_g, r_c and r_s. Among them, r_g is designed to navigate the robot towards its goal, r_c is built to penalize collision cases, and r_s is designed to reward the robot for maintaining a safe distance from all pedestrians. Formally, the reward function at time t can be given as:

$$r^t = r_g^t + r_c^t + r_s^t, \tag{4}$$

where r_g^t, r_c^t and r_s^t are given by:

$$r_g^t = \begin{cases} 0.25 & \text{if} \quad ||\mathbf{p}_0^t - \mathbf{g}_0|| < 0.2 \\ 0.2(||\mathbf{p}_0^{t-1} - \mathbf{g}_0|| - ||\mathbf{p}_0^t - \mathbf{g}_0||) & \text{otherwise} \end{cases} \tag{5}$$

$$r_c^t = \begin{cases} -0.25 & \text{if} \quad ||\mathbf{p}_0^t - \mathbf{p}_i^t|| < \rho_0 + \rho_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$\begin{aligned} r_s^t &= \sum_{i=1}^{N} f(\mu_i^t, \mu_s) \\ f(\mu_i^t, \mu_s) &= \begin{cases} \mu_i^t - \mu_s & \text{if} \quad \mu_i^t < 0.2 \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{7}$$

where $\mu_s = 0.2$ denotes the threshold distance that the robot needs to maintain from pedestrians at all times, and μ_i^t is the actual minimum separation distance between the robot and the ith pedestrian at time t. Note that these parameters in the reward function are determined experimentally.

3.1.5 Deep Q-learning

After formulating the crowd navigation problem as an MDP, we can utilize deep Q-learning, a type of model-free RL, to find the optimal policy, which maps the agent-level state s^t to the optimal action a^t. The optimal policy is defined as $\pi^* : s^t \rightarrow a^t$. The corresponding optimal state-action function is denoted by $Q^*(s^t, a^t)$, which indicates the expected discounted return of (s^t, a^t) in reaching its goal. Therefore, the optimal policy $\pi^* : s^t \rightarrow a^t$ can be determined by the optimal state-action value function $Q^*(s^t, a^t)$, which is described by:

$$\pi^*(s^t) = \arg\max_{a^t} Q^*(s^t, a^t). \tag{8}$$

Then, finding the optimal policy can be transformed into the problem of finding the optimal state-action value function. Because the optimal state-action value function satisfies the Bellman optimal equation, described in (9), the Bellman update equation is built to optimize the state-action value function recursively.

$$\begin{aligned} Q^*(s^t, a^t) = &\sum_{s^{t+1}, r^t} P(s^{t+1}, r^t | s^t, a^t) \\ &\times [r^t + \gamma \max_{a^{t+1}} Q^*(s^{t+1}, a^{t+1})], \end{aligned} \tag{9}$$

where $\gamma \in (0, 1)$ is the discount factor that balances the immediate and future rewards.

3.2 Graph representation with social attention

In this work, a social attention mechanism is introduced into the GAT to learn a good graph representation of the crowd-robot state. Different from [14], attention weights are computed for all agents, regardless of robot or pedestrian. Both robot-human interactions and human-human interactions can be modeled via the same graph convolution operation.

In the graph representation, the nodes and edges represent agents and direct interactions among agents. Since agents often have different types and dimensions, (e.g., $w_0 \in \mathbb{R}^9$ and $w_i \in \mathbb{R}^5, i > 0$), multilayer perceptrons (MLPs) are utilized to extract the fixed-length latent states. Immediately afterward, the latent states are fed into subsequent graph convolutional layers. Considering that neural networks are often composed of a stack of multiple layers, the superscript l is used to distinguish latent states at different layers. Here, the MLPs are marked as layer 0. Therefore, latent states extracted with MLPs are given as:

$$\begin{aligned} H^0 &= \left[h_0^0, h_1^0, ..., h_N^0\right] \\ h_0^0 &= \Psi_r(w_0; W_r) \\ h_i^0 &= \Psi_p(w_i; W_p), \quad i = 1, 2, ..., N, \end{aligned} \tag{10}$$

where Ψ_r and Ψ_p are MLPs with rectified linear unit (ReLU) activations, and their network weights are represented by W_r and W_p, respectively.

Then, a social attention mechanism is introduced into the spatial graph convolution operation to model interactions in a crowd. First, a query matrix Q and a key matrix K are built to transform the input features into higher-level features. Then, these features are concatenated to compute attention coefficients e_{ij} via a fully connected network (FCN). The FCN is followed by a LeakyReLU nonlinearity with a negative input slope of 0.2. Finally, the attention coefficients are normalized via a softmax function. A sketch of the layerwise graph convolution operation on the ith node and its neighborhood is shown in Fig. 3, and the corresponding layerwise graph convolution rule is given as:

$$\begin{aligned} e_{ij} &= \text{LeakyRuLU}(a(q_i \,||\, k_j)) \\ \alpha_{ij} &= \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathbb{N}(i)} \exp(e_{ik})} \\ h_i^{l+1} &= \sigma\left(\sum_{j \in \mathbb{N}(i)} \alpha_{ij}^l h_j^l\right), \end{aligned} \tag{11}$$

where $q_i = \Psi_q(h_i; Q)$, $k_j = \Psi_k(h_j, K)$, and $||$ is the concatenation operation. A layerwise FCN is utilized as the attention function $a(\cdot) = \Psi_a(\cdot; 1)$, mapping concatenated states to attention weights. α_{ij} is the normalized attention weight, indicating the importance of the jth node to the ith node. $\mathbb{N}(i)$ is the neighborhood of the ith node in the graph. Similar to previous work [14–17, 20], all agents can obtain accurate observable states for other pedestrians. Therefore, the neighborhood of the ith agent includes all pedestrians.

Considering that indirect interactions in the crowd are also quite important, we equipped the GAT with two graph

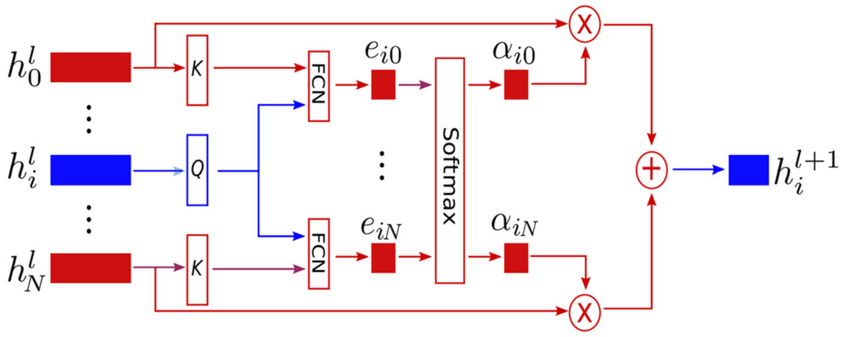


Fig. 3 Layerwise graph convolution operation on the ith node and its neighborhood. The superscript l denotes the layer number, Q is the query matrix and K is the key matrix

convolutional layers. In addition, to avoid the vanishing gradient problem and speed up the training process, the outputs from all layers are added together to form the final output of the GAT. The skip connection operation is as follows:

$$H = H^0 + H^1 + H^2, \tag{12}$$

where H is the final output of the GAT and H^0, H^1, and H^2 are the hidden states from different layers.

3.3 Graph-Based Deep Q-learning

Encouraged by the great success of DQNs [28, 29] in RL, a D3QN is built to estimate state-action values in this work. The significant difference from the previous state value network is that the D3QN requires only the current state as input. It is a model-free RL method and does not need to traverse subsequent states.

As shown at the bottom of Fig. 2, our D3QN includes three separate modules: MLPs that extract fixed-length latent states, a two-layer GAT to obtain graph representations H, and a dueling architecture to estimate state-action values. The whole network is trained in an end-to-end manner; namely, all parameters are trained jointly rather than step-by-step.

For the specific implementation, the D3QN has a dueling architecture, which consists of two streams, one for the state value and one for state-dependent advantage functions [30]. In this work, a two-layer MLP, $\Psi_c(H; \alpha)$, is built as a common convolutional feature learning module, and two fully connected layers, $\Psi_v(\cdot; \beta)$ and $\Psi_d(\cdot; \eta)$, are used to obtain a scalar $V(H; \alpha, \beta)$ and an $|A|$-dimensional vector $D(H, a; \alpha, \eta)$. Here, H is the final graph representation mentioned in (12), and α, β and η are the parameters of the dueling architecture. Finally, the state-action function $Q(H, a; \cdot)$ can be described by

$$Q(H, a; \alpha, \beta, \eta) = V(H; \alpha, \beta) + D(H, a; \alpha, \eta). \tag{13}$$

Besides, in traditional deep Q-learning, the training process only focus on the optimal state-action value. However, when performing rollouts the robot needs to explore more than one actions to improve its policy. Therefore, it is necessary to optimize both of the optimal state-action value and these sub-optimal state-action values. Inspired by Dyna [31], the learned environment model is applied to develop simulated experiences. In developing simulated experiences, the robot will only expand on the top-k actions, rather than on all actions as [31]. Finally, an optimization based on simulated experiences is added into each update.

Algorithm 1 Deep Q-learning.

```
   Output: Q-value network: $Q(\cdot\,;\theta)$
1  Initialize empty experience replay memory $E$
2  Initialize target D3QN with random weights $\theta$
3  Duplicate D3QN $Q'(\cdot\,;\theta') \leftarrow Q(\cdot\,;\theta)$
4  for $episode = 1, \ldots, N_{eps}$ do
5      Initialize random crowd-robot state $s^0$
6      while no Arrival, Collision or Timeout do
7          $p \leftarrow rand(0, 1)$
8          if $p < \epsilon$ then
9              $a^t \leftarrow random\ action$
10         else
11             $a^t \leftarrow \arg\max_{a^t} Q(s^t, a^t; \theta)$
12         Execute $a^t$ and obtain $r^t$ and $s^{t+1}$
13         Assimilate tuple $(s^t, a^t, r^t, s^{t+1})$ into $E$
14         $s^t \leftarrow s^{t+1}$
15     Randomly sample subset $M$ from $E$
16     Optimize $Q(\cdot\,;\theta)$ with $M$ and $Q'(\cdot\,;\theta')$
17     Generate $M'$ based on $M$
18     Optimize $Q(\cdot\,;\theta)$ with $M'$ and $Q'(\cdot\,;\theta')$
19     for every $C$ episodes do
20         evaluate value network $Q(\cdot\,;\theta)$
21         $Q'(\cdot\,;\theta') \leftarrow Q(\cdot\,;\theta)$
```

The learning algorithm is shown in Algorithm 1, where θ denotes all parameters of the D3QN. At the beginning of each episode, the crowd-robot state s^0 is randomly initialized (line 5). Afterward, the robot is simulated to move in the crowd until terminal states, which consist of *Arrival*, *Collision* and *Timeout*. In this work, the maximum navigation time is set to 30 s. The robot is equipped with an

ϵ-greedy policy (line 7-11), in which ϵ gets progressively smaller. The experience replay technique [29, 32] is also used to increase data efficiency and reduce the correlation among samples. At every step, the newly generated state value pairs are assimilated into an experience replay memory E (line 12-14) with a size of 10^5. In every training, minibatch experiences M are sampled from E to update the network (line 15-16). And then, a simulated experience subset M' is developed by M and the learned environment model (Line 17). Later, M' is used to re-optimize the network (Line 18). To promote convergence, the network $Q(\cdot\,;\theta)$ is cloned to build a target network $Q'(\cdot\,;\theta')$ (line 19-21), which generates the Q-learning target values for the next C episodes. Here, the mean-square error of Q-value estimation on minibatch M or M' is defined as:

$$L(\theta) = \frac{1}{|M|}\sum_{s\in M}(r + \gamma \max_{a^{t+1}} Q'(s^{t+1}, a^{t+1}; \theta') - Q(s^t, a^t; \theta))^2, \tag{14}$$

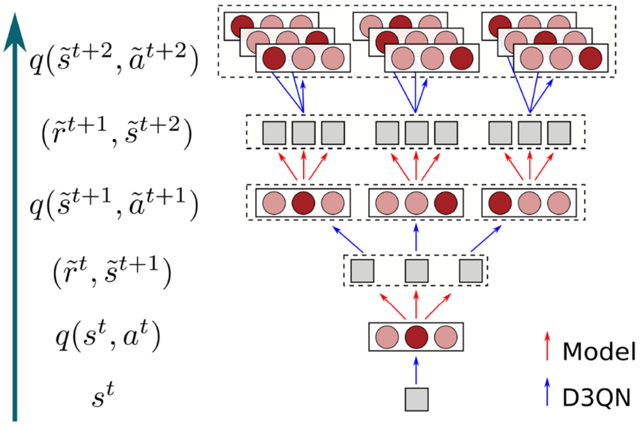
where M can be replaced by M' and $|\cdot|$ means the number of elements in the set.

3.4 Online planning based on rollout performance

Previous works have proven that a D3QN has the potential to partly address the shortsightedness problem in crowd navigation. However, with unknown human behavior policies, it is difficult to generate a perfect network. Inspired by [25, 33, 34], the learned D3QN is combined with online planning to refine coarse state-action values by performing rollouts on the current state and recursively building up a look-ahead tree. The look-ahead tree has two hyperparameters: depth d and width k, where d determines the number of steps in the forward search and k determines the number of child nodes at each expansion.

Compared with previous works [14, 16, 20], the biggest difference in rollouts is that the proposed method builds up a human-like decision-making process, first roughly estimated and then carefully refined. Based on the coarse state-action values generated by the D3QN, action clipping can be moved forward before the traversal of all child states in rollouts. Therefore, at each node expansion, only child states with top-k action values must be traversed, which avoids a lot of useless rollouts and results in a much lower computational cost.

A two-step rollout in the look-ahead tree search diagram is shown in Fig. 4 and the decision process is shown in Algorithm 2. At each expansion, the D3QN is first used to coarsely evaluate state-action values and generate the best candidate actions A' (line 7). And then, the environment model is utilized to predict the corresponding rewards and subsequent states based on the current states



$q(\tilde{s}^{t+2}, \tilde{a}^{t+2})$
$(\tilde{r}^{t+1}, \tilde{s}^{t+2})$
$q(\tilde{s}^{t+1}, \tilde{a}^{t+1})$
$(\tilde{r}^{t}, \tilde{s}^{t+1})$
$q(s^t, a^t)$
s^t
Model
D3QN

Fig. 4 Two-step rollout on state s^t. Here, gray squares denote states, while colored discs denote actions. Besides, a blue arrow and a red arrow are used to denote the learned D3QN and the environment model, respectively. To distinguish the simulated trajectories from real trajectories, the imaged state, reward, and action are marked with $\tilde{}$. Therefore, simulated trajectories begin at state s, and action a can be represented as $(s^t, a^t, \tilde{r}^t, \tilde{s}^{t+1}, \tilde{a}^{t+1}, \tilde{r}^{t+1}, \tilde{s}^{t+2}, \tilde{a}^{t+2}, \cdots)$

and candidate actions (line 15). Continuously repeating the process of node expansion, a lot of simulated trajectories are constructed, and coarse state-action values generated by the D3QN can be refined on these simulated trajectories according to (15) (line 12-17). Finally, the refined state-action values of candidate best actions, denoted by $Q^d(s^t, a^t, \theta)$, are utilized to determine the truly best action (line 4).

$$Q^d(s^t, a^t; \theta) = \begin{cases} Q(s^t, a^t; \theta) & \text{if } d = 0 \\ \frac{d}{d+1}Q(s^t, a^t; \theta) + \frac{1}{d+1}(\tilde{r}^t \\ + \gamma \max_{\tilde{a}^{t+1}} Q^{d-1}(\tilde{s}^{t+1}, \tilde{a}^{t+1}; \theta)) & \text{otherwise.} \end{cases} \tag{15}$$

3.5 Implementation details

In this work, the hidden units of $\Psi_r(\cdot)$, $\Psi_h(\cdot)$, Ψ_r, Ψ_v and Ψ_d have dimensions of (64,32), (64,32), (128), (128) and (128), respectively. In every layer, the output dimensions of $\Psi_q(\cdot)$ and $\Psi_k(\cdot)$ are 32. In the dueling architecture, Ψ_r, Ψ_v and Ψ_d have output dimensions of 128, 1, and 81, respectively. All the parameters are trained via RL and updated by adaptive moment estimation (Adam) [35]. The learning rate and discount factor are 0.0005 and 0.9, respectively. In addition, the exploration rate of the ϵ-greedy policy decays linearly from 0.5 to 0.1 in the first 5000 episodes and remains at 0.1 in the remaining 5000 episodes. For the rollout performance, the planning depth and planning width are set to 1 and 10, respectively.

Algorithm 2 Online planning with D3QN.

```
    Input: Q-value network $Q(\cdot;\theta)$, prior environment
           model $P$, planning depth $d$, planning width $k$
    Output: trajectory $s^{0:t_f}$
1   while not reach goal do
2       update $t$, receive new observation $s^t$
3       $A', V \leftarrow$ DPlan($s^t, d, k, Q, P$)
4       $a^t \leftarrow \arg\max_{a^t \in A'} V(a^t)$
5   return $s^{0:t_f}$
6   Function DPlan($s^t, d, k, Q, P$):
7       $A' \leftarrow$ actions with top-$k$ of $Q(s^t, a^t; \theta)$
8       for $a^t \in A'$ do
9           $V(a^t) \leftarrow$ QRefine($s^t, a^t, d, k, Q, P$)
10      return $A', V$
11  Function QRefine($s^t, a^t, d, k, Q, P$):
12      if $d$=0 then
13          return $Q(s^t, a^t; \theta)$
14      else
15          $\tilde{r}^t, \tilde{s}^{t+1} \leftarrow P(\tilde{r}^t, \tilde{s}^{t+1} | s^t, a^t)$
16          $A', V \leftarrow$ DPlan($\tilde{s}^{t+1}, d-1, k, Q, P$)
17          return
            $\frac{d}{d+1} Q(s^t, a^t; \theta) + \frac{1}{d+1}(\tilde{r}^t + \gamma \max_{\tilde{a}^{t+1} \in A} V(\tilde{a}^{t+1}))$
```

4 Experiments

4.1 Simulation setup

SG-D3QN has been evaluated in both simple and complex scenarios, which are built based on the simulation environment named CrowdNav [14]. Simple scenarios include five pedestrians, which are controlled by optimal reciprocal collision avoidance (ORCA) [4]. The pedestrians' initial positions and goal positions are set on the circle, symmetric about the circle's center. Therefore, all agents will interact near the center of the circle at approximately 4.0 s. In addition to the five circle-crossing pedestrians, complex scenarios introduce another five square-crossing pedestrians, whose initial positions and goal positions are sampled randomly in a square with a side length of 10 m. The square shares the same center as the circle. In either simple or complex scenarios, once the pedestrian arrives at his or her goal position, a new goal position will be randomly reset within the square. The former goal position is recorded as a turning position. The initial position, the turning position, and the final goal position of pedestrian 0 are marked in Fig. 6a. Note that the robot is invisible in both scenarios, and simulated pedestrians will never give way to the robot. The main reason for the setting is that we hope the robot takes full responsibility for collision avoidance and proactively avoids pedestrians. Such an active collision avoidance policy is the final goal of this work.

4.2 Qualitative evaluation

4.2.1 Training process

513

SG-D3QN is trained in both simple and complex scenarios. The resulting training curves are shown in Fig. 5. At the beginning of training, completing the crowd navigation task with a randomly initialized model is difficult, and most termination states are timeout. As the training continues, the robot quickly learns to keep a safe distance from the pedestrians and slowly understands the crowd. In the last period of training, SG-D3QN achieves relatively stable performance. The difference between different scenarios is predictable. The main reason is that there are more interactions in complex scenarios, making the environment more challenging than in simple scenarios. The detailed quantitative results are described in Section 4.3.

4.2.2 Collision avoidance behaviors

With the learned policy, the robot can safely and quickly reach its goal position in both simple and complex scenarios. The resulting trajectory diagrams are shown in Fig. 6. In complex scenarios, the robot has to pay much attention to avoiding pedestrians than in simple scenarios, resulting in a rougher trajectory and a longer navigation time. In both simple and complex scenarios, the robot performs proactive and foresighted collision avoidance behaviors. The robot can always recognize and avoid the approaching interaction center of the crowd. For example, in the simple scenario, the robot turns right at approximately 1.5 s to avoid potential encirclement at 5.0 s. In addition, as shown in Fig. 6b, the robot is also able to avoid areas with intensive interaction very well.

4.2.3 Attention modeling

Figure 7 shows the attention weights in two test crowd scenes. In both simple and complex scenarios, the robot pays much attention to pedestrians who are close it, e.g., Pedestrian 2 and 4 in Fig. 7a and Pedestrian 5 in Fig. 7b. It also pays attention to pedestrians who will interact with it, such as Pedestrian 3 and 5 in Fig. 7a and Pedestrian 8 in Fig. 7b. Additionally, robot's attention weights are also affected by pedestrians' velocities. Taking Pedestrian 3 and 6 in Fig. 7b for example, their positions are similar, but their velocities are in nearly opposite directions, which results in an obvious difference in their attention weights. Another interesting point is the self-attention weight of the robot.

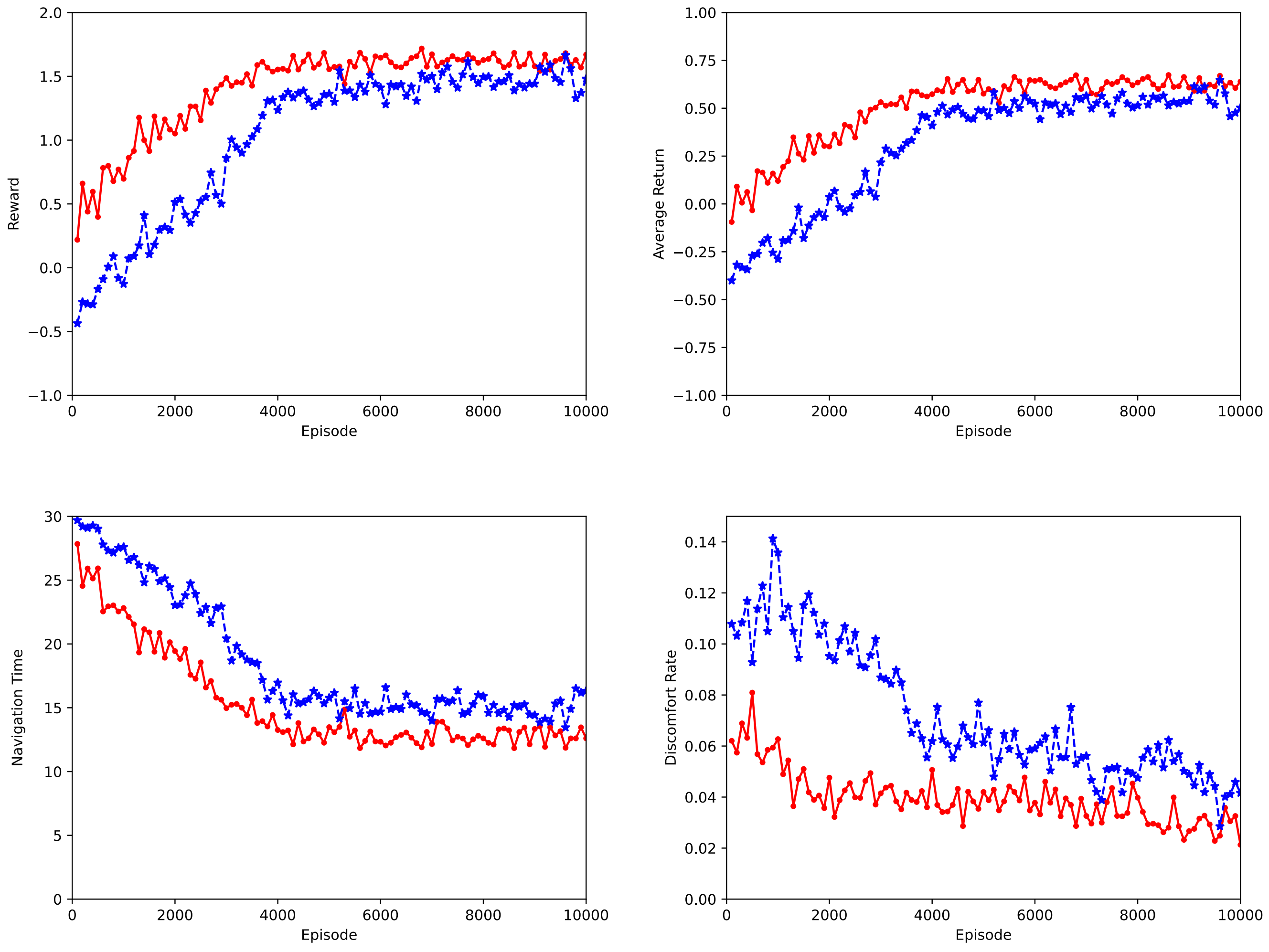


Fig. 5 Training curves in both simple scenarios (red and solid) and complex scenarios and complex scenarios (blue and dashed). From top to bottom, left to right, the y-axis corresponds to the total reward, average return, navigation time and discomfort rate per episode, averaged over the last 100 episodes

The fewer pedestrians there are around the robot, the greater the self-attention weight. This means that the robot can balance its navigation task and collision avoidance behavior. We highly recommend readers to watch our demo video.

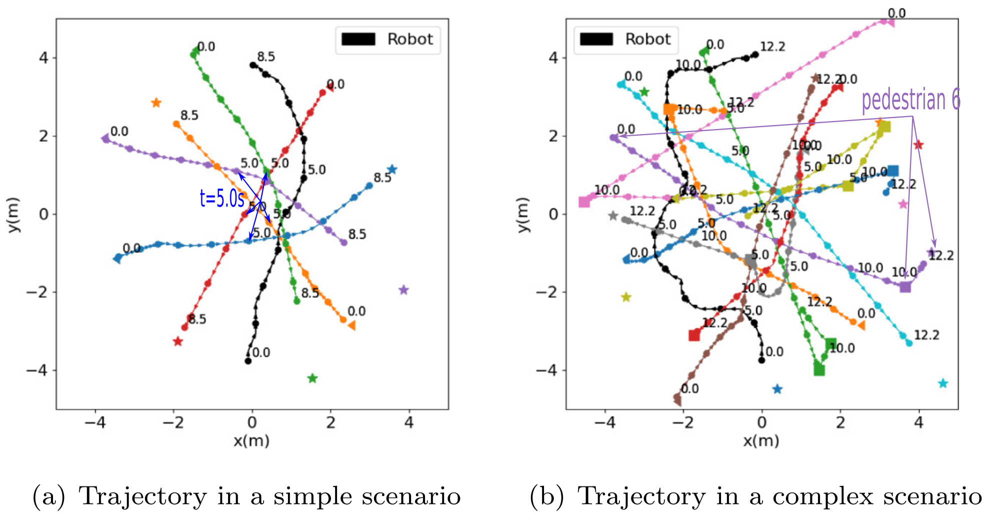


Fig. 6 Trajectory diagrams for a simple scenario (**a**) and a complex scenario (**b**). Here, the discs represent agents, black for the robot and other colors for pedestrians. The numbers near the discs indicate the time. The time interval between two consecutive discs is 1.0 s. Small arrows on the line indicates the forward direction. Here, the initial positions, the turning positions and the final goal positions are marked with triangles, squares and five-pointed stars, respectively

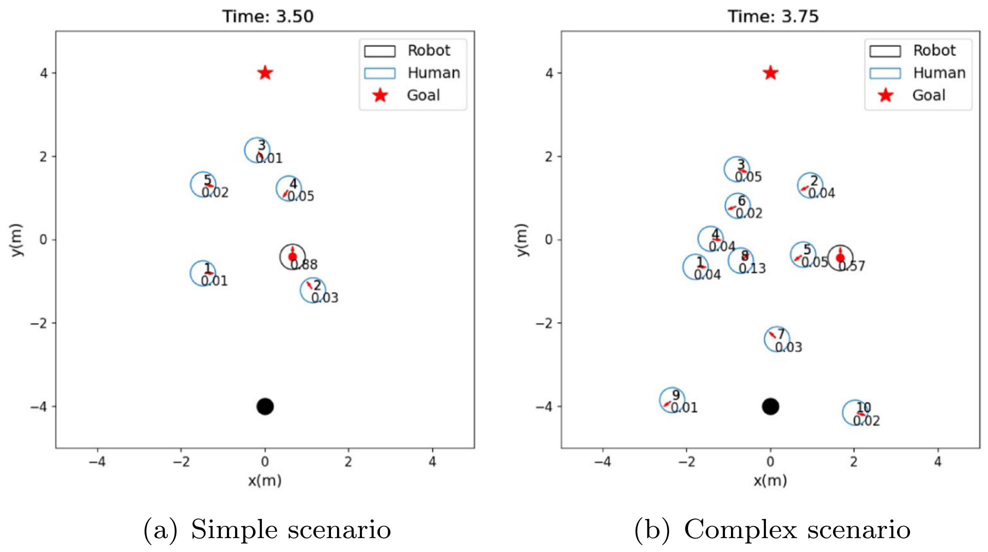


Fig. 7 Robot attention weights for agents in a simple scenario (**a**) and a complex scenario (**b**). The natural numbers are the serial numbers, and the two-digit decimals are the attention weights the robot gives to agents. The red arrows indicate the velocity attitudes of agents. The initial position and goal position of the robot are marked by a black disc and a red five-pointed star, respectively. Besides, a red dot at the center of the circle is used to distinguish the robot from pedestrians

4.3 Quantitative evaluation

4.3.1 Performance comparison

Three existing state-of-the-art methods, ORCA [4], local-map-self-attention reinforcement learning (LM-SARL) [14] and model predictive relational graph learning (MP-RGL) [20], are implemented as baseline methods. In the implementation of ORCA, the target pedestrian' radii increase by $\mu_s = 0.2$ to maintain a safe distance from other pedestrians. Both one-step and two-step implementation of MP-RGL are tested in our experiments, represented by MP-RGL-Onestep and MP-RGL-Twostep, respectively. Following [20], the planning width of MP-RGL-Twostep is set to 2. In addition, to verify the effect of online-planning, a D3QN version of SG-D3QN is also developed as a contrast algorithm by setting the planning depth to 0. Besides, three versions of SG-D3QN, with their planning depths of 1, 2, and 3, are tested in our experiments. Here, we use SG-D3QN-Onestep, SG-D3QN-Twostep, and SG-D3QN-ThreeStep to distinguish them. For a fair comparison, all RL methods are trained with the reshaped reward function, which makes our MP-RGLs achieve much better performance than the original version proposed in [14, 20]. In addition, the training process is repeated five times, and the resulting models are evaluated with 1000 random test cases. The random seeds in the five training processes are 0,1,2,3,4. Finally, the statistical results are shown in Table 1 (for simple scenarios) and Table 2 (for complex scenarios).

In the quantitative evaluation, the metrics include *"Success"*, the success rate of the robot reaching its goal safely; *"Collision"*, the rate of the robot colliding with pedestrians; *"Nav. Time"*, the navigation time to reach the goal in seconds; *"Disc. Num"*, the number of times the robot violates the safety distance of a pedestrian (if the robot violates more than one pedestrian at the same time, the number of violated pedestrians will be added to *Disc. Num*); *"Return"*, the discounted cumulative return averaged over steps, which considers a combination of earned rewards and time efficiency; and *"Run Time"*, the run time per

Table 1 Quantitative results in simple scenarios

Methods	Success	Collision	Nav. Time (s)	Disc. Num	Avg. Return	Run time (ms)
ORCA [4]*	0.8240±0.0000	0.1760±0.0000	12.07±0.00	2700.0±0.0	0.555±0.000	**0.55±0.00**
D3QN [14]	0.9728±0.0087	0.0274±0.0087	11.25±0.11	959.0±431.6	0.697±0.023	1.37±0.01
LM-SARL [14]	0.9952±0.0027	0.0048±0.0027	10.41±0.07	1320.6±255.4	0.737±0.008	36.89±0.95
MP-RGL-Onestep [20]	0.9946±0.0046	0.0032±0.0016	10.63±0.17	1099.6±369.2	0.724±0.013	32.11±0.66
MP-RGL-Twostep [20]	0.9956±0.0036	0.0036±0.0024	10.41±0.14	918.6±276.2	0.735±0.013	93.48±1.29
SG-D3QN-Onestep	0.9970±0.0012	0.0030±0.0012	10.07±0.14	881.0±197.2	0.744±0.009	6.90±0.21
SG-D3QN-Twostep	**0.9984±0.0021**	**0.0016±0.0021**	9.96±0.12	**676.0.0±139.5**	0.750±0.006	28.17±0.76
SG-D3QN-Threestep	0.9982±0.0021	0.0018±0.0021	**9.90±0.13**	677.8±117.9	**0.752±0.006**	127.61±2.55

(*) Note that ORCA is not affected by the random seed, so its standard deviation is always 0
Bold entries mean the best performance in six quantitative metrics

Table 2 Quantitative results in complex scenarios

Methods	Success	Collision	Nav. Time (s)	Disc. Num	Avg. Return	Run Time (ms)
ORCA [4]*	0.7690±0.0000	0.2220±0.0000	13.88±0.00	5884.0±0.0	0.432±0.000	**0.57±0.00**
D3QN [14]	0.8890±0.0071	0.1110±0.0071	13.80±1.01	2003.4±14.9	0.587±0.028	1.66±0.01
LM-SARL [14]	0.9844±0.0020	0.0158±0.0021	11.87±0.07	2281.4±273.7	0.686±0.012	44.17±0.34
MP-RGL-Onestep [20]	0.9898±0.0038	0.0090±0.0024	11.87±0.17	1366.0±193.7	0.699±0.010	43.18±0.42
MP-RGL-Twostep [20]	0.9936±0.0021	0.0064±0.0021	11.63±0.18	**1226.0±130.5**	0.715±0.003	126.92±1.23
SG-D3QN-Onestep	0.9924±0.0021	0.0076±0.0021	11.18±0.18	1450.6±412.2	0.711±0.015	8.79±0.27
SG-D3QN-Twostep	0.9940±0.0010	0.0040±0.0010	11.05±0.16	1364.4±123.0	0.717±0.005	37.19±0.33
SG-D3QN-Threestep	**0.9964±0.0021**	**0.0036±0.0021**	**10.92±0.17**	1347.8±34.0	**0.724±0.001**	169.29±2.44

(*) Note that ORCA is not affected by the random seed, so its standard deviation is always 0
Bold entries mean the best performance in six quantitative metrics

decision in milliseconds on a computer with an i9-10900K CPU. Here, all metrics are averaged over steps across all episodes and test cases, and ± is used to indicate the standard deviation measured using five independently seeded training runs.

As a reactive policy, the ORCA method has the highest collision rate, the longest navigation time, and the best real-time performance in both simple and complex scenarios. It can easily induce the robot to fall into the *inevitable collision state* (ICS) [36]. D3QN achieves a better performance than ORCA, average success rates of 0.9728 and 0.889 in simple and complex scenarios. The results can be attributed to the reinforcement learning algorithm. After being trained with sufficient experience, the agent can always generate a foresighted policy. Besides, as the scenario is expanded from a simple one to a complex one, the collision rate of D3QN has increased dramatically to 0.1110, nearly four times that of simple scenarios. The complex scenario makes it difficult to directly learn an optimal control policy mapping the raw state to the best action in complex scenarios, and a purely model-free reinforcement learning method seems inadequate.

As expected, all LM-SARL, MP-RGLs, and SG-D3QNs achieve performances far superior to D3QN. In particular, LM-SARL experiences a similarly dramatic decline in performance when the scenario is extended from simple to complex. Its coarse local map does not deal with complex scenarios very well. When comparing SG-D3QNs with MP-RGLs, it is easy to see that at the same planning depth, SG-D3QNs always achieve better performance than MP-RGLs, especially in terms of the *Nav. Time*. Besides, in terms of the *Run Time*, SG-D3QN-Onestep achieves a better run-time performance than LM-SARL, MP-RGLs, other SG-D3QNs, averaging 6.90 ms and 8.79 ms per decision in simple and complex scenarios. With the same planning depth, SG-D3QNs require approximately one-quarter of MP-RGLs' run time. These experimental results show that SG-D3QN does reduce the computational cost significantly, which can be attributed to its human-like decision-making process.

Finally, it is also worth mentioning that integrating state/state-action value estimators can always improve the robot's navigation policy. The multi-step variants of either MP-RGL or SG-D3QN always achieve a better performance than the one-step version. Besides, when we further compare the three variants of SG-D3QN, we can find that as the planning depth increases, its performance, especially in terms of the *Nav. Time*, is gradually improving. SG-D3QN-Threestep achieves a high success rate of 0.9984 in simple scenarios, which remains 0.9963 in complex scenarios. At the same time, it has the best performance in time efficiency, with a navigation time of 9.90 s and 10.92 s in simple and complex scenarios, respectively. Considering that the robot has to give way to other pedestrians in

Table 3 Statistical results in crowds with various size

Crowd Size	Success	Collision	Nav. Time (s)	Disc. Num	Avg. Return	Run. Time
6	0.9982±0.0012	0.0018±0.0012	10.30±0.74	600.8±164.0	0.741±0.009	29.98±0.59
8	0.9954±0.0023	0.0046±0.0023	10.67±0.18	871.4±131.4	0.731±0.008	34.39±0.33
10	0.9940±0.0010	0.0040±0.0010	11.05±0.16	1364.4±123.0	0.717±0.005	37.19±0.33
12	0.9918±0.0036	0.0080±0.0041	11.36±0.20	1666.0±96.4	0.705±0.016	41.55±0.41
14	0.9826±0.0051	0.0174±0.0051	11.99±0.22	2351.4±216.6	0.674±0.009	44.908±0.46
16	0.9760±0.0113	0.0240±0.0113	12.38±0.20	2938.8±309.9	0.656±0.016	48.50±0.40

Table 4 Quantitative results with a rotational constraint of $\pi/6$

Scenario	Method	Success	Collision	Nav. Time (s)	Disc. Num	Avg. Return
Simple	SG-D3QN-Onestep	0.9878±0.0070 ↓	0.0122±0.0070 ↑	10.25±0.21 ↑	1327.0±396.8 ↑	0.711±0.011 ↓
	SG-D3QN-Twostep	0.9918±0.0036 ↓	0.0080±0.0041 ↑	9.90±0.12 ↓	1101.8±311.9 ↑	0.734±0.006 ↓
	SG-D3QN-Threestep	0.9936±0.0027 ↓	0.0064±0.0027 ↑	9.81±0.17 ↓	1134.8±364.8 ↑	0.739±0.007 ↓
Complex	SG-D3QN-Onestep	0.9566±0.0065 ↓	0.0434±0.0065 ↑	11.38±0.21 ↑	2284.4±362.6 ↑	0.646±0.098 ↓
	SG-D3QN-Twostep	0.9690±0.0037 ↓	0.0310±0.0037 ↑	10.91±0.11 ↓	1922.2±264.3 ↑	0.694±0.006 ↓
	SG-D3QN-Threestep	0.9740±0.0058 ↓	0.0260±0.0058 ↑	10.78±0.76 ↓	1830.6±189.2 ↑	0.702±0.006 ↓

The upward/downward arrow indicates that the item is larger/smaller than the corresponding one in the Table 1 or 2

navigation actively, its performance has far exceeded our expectations.

4.3.2 Scalability in crowd size

Thanks to the graph-based representation of the crowd-robot state, SG-D3QN trained in fixed-size crowds can handle crowds with various size naturally. Here, SG-D3QN-Twostep trained in complex scenarios with 10 pedestrians is used to navigate the robot in the crowd with size of 6, 8, 12, 14, and 16. Statistical results are shown in Table 3.

The results, as expected, show a slight decrease in success rate and average return as the crowd size increases. However, even though the crowd size has increased by 60%, SG-D3QN-Twostep can still achieve a high success rate of about 0.976. What is more, when the crowd size has decreased, the performance of SG-D3QN-Twostep can be further improved. Besides, the increase in crowd size has a near-linear effect on the computational cost. Therefore, SG-D3QN-Twostep can still be effective even in a large-sized crowd. These experimental results fully illustrate that the proposed method has good scalability in crowd size.

4.3.3 Incorporating kinematic constraints

As mentioned in Section 3.1.2, the proposed method can be extended to car-like robots by limiting the headings component of action. In this work, the rotational constraint of the robot is set to $\pi/6$, which can be adjusted according to the actual robot model. Three variants of SG-D3QN are tested in both of simple and complex scenarios, and their statistical results are shown in Table 4.

Obviously, the proposed method can still keep a great performance under a rotation-constrained setting. In both simple and complex scenarios, three variants of SG-D3QN achieve a success rate of at least 0.95. Especially, SG-D3QN-Threestep achieves a success rate of 0.9936 with a navigation time of 9.81 s in simple scenarios and a success rate of 0.9740 with a navigation time of 10.78 s in simple scenarios. However, looking back at statistical results in Tables 1 and 2, it is easy to find that the introduction of rotational constraint causes a worse performance. With the rotational constraint of $\pi/6$, SG-D3QNs always achieve a lower success rate and average return than the holonomic one. Besides, it should be noticed that with SG-D3QN-Twostep and SG-D3QN-Threestep, the robot with a rotational constraint can reach its goal a little faster than the holonomic one. In contrast, the constrained robot violates the safety space of pedestrians more frequently, reflected in the rapid increase in *Disc. Num*. It is because that the constrained robot has a limited action space and results in more aggressive behavior than the holonomic one. Finally, when comparing the performance of three variants of SG-D3QN, it can be seen that the increase in planning depth is still effective in improving the rotation-constrained robot's navigation policy, just like the experimental results of the previous holonomic robot.

Table 5 Quantitative results in visible scenarios

Scenario	Robot model	Success	Collision	Nav. Time (s)	Disc. Num	Avg. Return
Simple	Holonomic	0.9994±0.0009 ↑	0.0006±0.0009 ↓	9.83±0.14 ↓	797.6±148.1 ↑	0.752±0.008 ↑
	Rotation-Constrained	0.9968±0.0026 ↑	0.0032±0.0026 ↓	9.81± 0.08↓	1184.8±340.0 ↑	0.734±0.005 ↑
Complex	Holonomic	0.9986±0.0010 ↑	0.0014±0.0010 ↓	10.85±0.21 ↓	1262.2±287.7 ↓	0.723±0.005 ↑
	Rotation-Constrained	0.9906±0.0034 ↑	0.0094±0.0034 ↓	10.86±0.14 ↓	1937.6±244.4 ↑	0.702±0.008 ↑

The upward/downward arrow indicates that the item is larger/smaller than the corresponding one in the Tables 1, 2, or 4

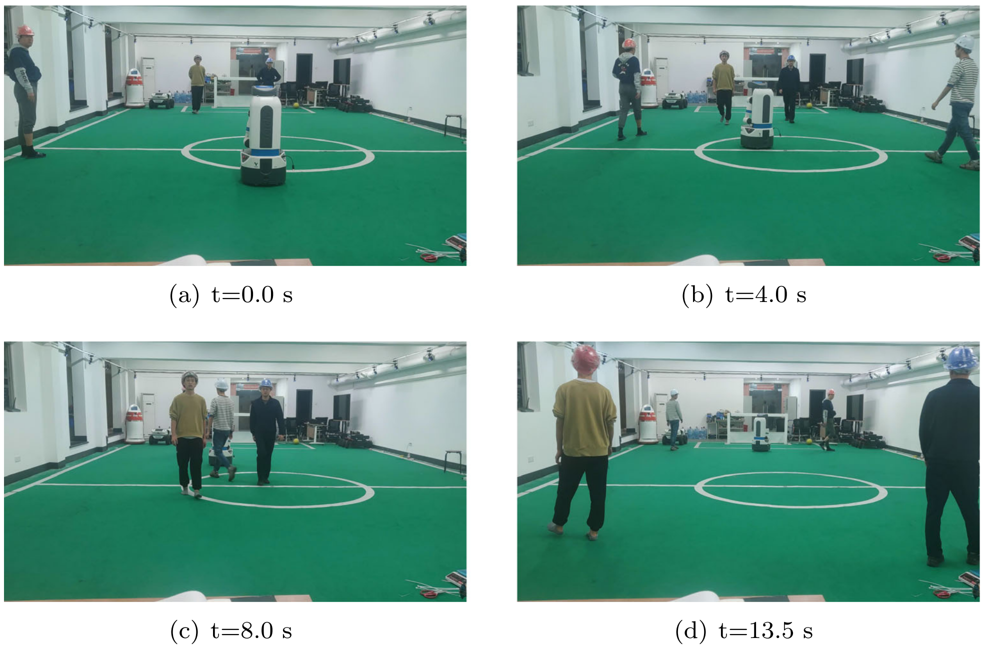


Fig. 8 Screenshots from hardware experiment. Note that the maximum speed of the robot is set to 0.7 m/s

(a) t=0.0 s (b) t=4.0 s (c) t=8.0 s (d) t=13.5 s

4.3.4 Navigating around interactive agents

Consider that in real applications, pedestrians are often able to see the robot and will react to its movements. We further test the trained model in the visible setting. Unlike experiments in Sections 4.3.1 and 4.3.3, pedestrians in this section can see the simulated robot and avoid it. However, due to the lack of communication network among the robot and pedestrians, they still do not know the coming action of the robot. Therefore, they could still collide with each other. Besides, considering the run-time performance, only SG-D3QN-Twostep is used here, which can run at a frequency of approximately 30 Hz. Besides, both holonomic and rotation-constrained robots are tested in visible scenarios. The statistical results are shown in Table 5.

As expected, SG-D3QN-Twostep achieves a better performance in all scenarios than in the invisible setting. Especially for the rotation-constrained robot in the complex scenarios, the collision rate is reduced by 69.7%, from 0.0310 to 0.0094. The improved performance can be attributed to pedestrians' avoidance behaviors. Although pedestrians and the robot do not communicate with each other, they can still cooperate to avoid the vast majority of collisions. The results also show that the invisible setting is more challenging than the visible setting. Besides, it must be pointed out that the visible setting also results in the rise in *Disc. Num*. Its main reason is that the environment model is trained in the invisible scenarios, where pedestrians never react to the robot's action.

4.4 Hardware experiment

In addition to simulation experiments above, we also deploy the trained policy with a rotational constraint of $\pi/6$ on a Fetch robot with an i5 CPU (see Fig. 8. Same as simulation experiments, the robot starts at (0, -4) and its goal locates at (0, 4). Four volunteers participate in the hardware experiment. Finally, the video demo can be found at github.com/nubot-nudt/SG-D3QN.

5 Conclusion

In this paper, we propose SG-D3QN, a graph-based RL method for social robot navigation. It achieves a high success rate of more than 0.99 in the two types of scenarios. Compared to previous state-of-the-art methods, SG-D3QN achieves better performance and requires less computational cost. The excellent performance can be attributed to the three innovations that have been proposed in this work. The first innovation is the introduction of a social attention mechanism into the spatial graph convolution operation. The improved two-layer GAT is available to extract an efficient graph representation for the crowd-robot state. The second innovation is applying D3QN to directly evaluate coarse state-action values of current states and quickly generate the best candidate actions. It greatly reduces the computational cost of making a decision. The third innovation is the integration of learning and online planning, which constructs a human-like decision-making process. By performing rollouts on the current state, the coarse state-action values are refined with a tree search, thus providing a proactive and foresighted policy. These innovations are also useful in other similar applications.

There is still much room for improvement, particularly related to: a) continuous action space, b) inaccurate sensing, and c) stochasticity of pedestrian behavior. Therefore, in future work, we will further introduce policy-based RL approaches into our framework to extend the current discrete action space to continuous action space. Meanwhile, we will further verify the proposed method in more realistic scenarios, such as simulation environments with sensory noise and pedestrian trajectory datasets. Besides, we plan to introduce historical trajectories of pedestrians to further enhance the robot's understanding of the crowd scenario, as in trajectory prediction.

References

1. Ferrer G, Garrell A, Sanfeliu A (2013) Robot companion: A social-force based approach with human awareness-navigation in crowded environments. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 1688–1694
2. Helbing D, Molnár P (1995) Social force model for pedestrian dynamics. Phys Rev E 51:4282–4286
3. Fiorini P, Shiller Z (1998) Motion planning in dynamic environments using velocity obstacles. Int J Robot Res 17(7):760–772. https://doi.org/10.1177/027836499801700706
4. Van den Berg J, Guy SJ, Lin M, Manocha D (2011) Reciprocal n-body collision avoidance. In: Robotics Research, pp 3–19
5. Van den Berg J, Lin M, Manocha D (2008) Reciprocal velocity obstacles for real-time multi-agent navigation. In: 2008 IEEE International Conference on Robotics and Automation (ICRA), pp 1928–1935
6. Alahi A, Goel K, Ramanathan V, Robicquet A, Fei-Fei L, Savarese S (2016) Social lstm: Human trajectory prediction in crowded spaces. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 961–971
7. Lisotto M, Coscia P, Ballan L (2019) Social and scene-aware trajectory prediction in crowded spaces. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp 2567–2574
8. Sun J, Jiang Q, Lu C (2020) Recursive social behavior graph for trajectory prediction. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp 657–666
9. Chen Y, Liu M, Wang L (2018) Rrt* combined with gvo for real-time nonholonomic robot navigation in dynamic environment. In: 2018 IEEE International Conference on Real-time Computing and Robotics (RCAR), pp 479–484
10. Rudenko A, Kucner TP, Swaminathan CS, Chadalavada RT, Arras KO, Lilienthal AJ (2020) ThÖr: Human-robot navigation data collection and accurate motion trajectories dataset. IEEE Robot Autom Lett 5(2):676–682
11. Trautman P, Krause A (2010) Unfreezing the robot: Navigation in dense, interacting crowds. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 797–803
12. Aoude GS, Luders BD, Joseph JM, Roy N, How JP (2013) Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns. Auton Robot 35(1):51–76
13. Long P, Fan T, Liao X, Liu W, Zhang H, Pan J (2018) Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp 6252–6259
14. Chen C, Liu Y, Kreiss S, Alahi A (2019) Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In: 2019 International Conference on Robotics and Automation (ICRA), pp 6015–6022
15. Chen Y, Liu C, Shi BE, Liu M (2020) Robot navigation in crowds by graph convolutional networks with attention learned from human gaze. IEEE Robot Autom Lett 5(2):2754–2761
16. Chen YF, Liu M, Everett M, How JP (2017) Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp 285–292
17. Chen YF, Liu M, Everett M, How JP (2017) Socially aware motion planning with deep reinforcement learning. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 1343–1350
18. Everett M, Chen YF, How JP (2018) Motion planning among dynamic, decision-making agents with deep reinforcement learning. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 3052–3059
19. Everett M, Chen YF, How JP (2021) Collision avoidance in pedestrian-rich environments with deep reinforcement learning. IEEE Access 9:10357–10377
20. Chen C, Hu S, Nikdel P, Mori G, Savva M (2020) Relational graph learning for crowd navigation. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 10007–10013
21. Vemula A, Muelling K, Oh J (2018) Social attention: Modeling attention in human crowds. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp 4601–4607
22. Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2021) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32(1):4–24
23. Sutton RS, Barto AG (2018) Reinforcement learning: An introduction, 2 edn. The MIT Press
24. Tamar A, Wu Y, Thomas G, Levine S, Abbeel P (2017) Value iteration networks
25. Oh J, Singh S, Lee H (2017) Value prediction network. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, pp 6118–6128
26. Farquhar G, Rocktäschel T, Igl M, Whiteson S (2018) Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning
27. Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, Guez A, Lockhart E, Hassabis D, Graepel T, Lillicrap T, Silver D (2020) Mastering atari, go, chess and shogi by planning with a learned model. Nature 588:604–609
28. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. In: NIPS Deep Learning Workshop
29. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533

30. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N (2016) Dueling network architectures for deep reinforcement learning. In: Proceedings of The 33rd International Conference on Machine Learning, vol 48, pp 1995–2003
31. Sutton RS, Szepesvari C, Geramifard A, Bowling MP (2012) Dyna-style planning with linear function approximation and prioritized sweeping
32. Lin LJ (1993) Reinforcement learning for robots using neural networks. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh
33. Oh J, Guo X, Lee H, Lewis R, Singh S (2015) Action-conditional video prediction using deep networks in atari games. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol 2, pp 2863–2871
34. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, Van den Driessche G, Graepel T, Hassabis D (2017) Mastering the game of go without human knowledge. Nature 550:354–359
35. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations
36. Fraichard T, Asama H (2004) Inevitable collision states – a step towards safer robots? Adv Robot 18:1001–1024

Zhiqian Zhou received his Bachelors of Engineering (2017) from the National University of Defense Technology (NUDT), Changsha, China, where he is working towards the Ph.D. The focus of his research lies on mobile robotics, especially on navigation, path/trajectory planning, human-robot interaction and multi-robot coordination.

Pengming Zhu received his B.S. (2018) from the Beijing Institute Of Technology (BIT), China, and his M.E. (2020) from the National University of Defense Technology (NUDT), China, respectively, where working towards the Ph.D. The focus of his research lies on multi-agent reinforcement learning algorithms, multi-robot cooperation, shared control.

Zhiwen Zeng received his M.E. (2011) and Ph.D.(2016) degrees in control science and engineering from the National University of Defense Technology (NUDT), and his B.S. degree from the University of Electronic Science and Technology of China in 2009. From 2019, he serves as an associate professor on Robotics and Cybernetics. His research interests include path planning, human-robot interaction, multi-robot coordination and the application to intelligent and robotic system.

Junhao Xiao received his Ph.D. (2013) from the Institute of Technical Aspects of Multimodal Systems (TAMS), Department of Informatics, University of Hamburg, Hamburg, Germany, and his B.E. degree (2007) from the National University of Defense Technology, Changsha, China. From 2017, he serves as an associate professor on Robotics and Cybernetics. The focus of his research lies on mobile robotics, especially on localization, mapping, path planning, human-robot interaction and multi-robot coordination.

Huimin Lu received the B.E. degree in automation, M.E. and Ph.D. degrees in control science and engineering from the National University of Defense Technology (NUDT), Changsha, China, in 2003, 2005, and 2010, respectively. He joined the Department of Automation, NUDT, in 2010, where he is now a Professor. His research interests which include mobile robotics, mainly on robot vision, multi-robot coordination, human-robot interaction.

Zongtan Zhou received his Bachelors of Engineering (1990), Masters of Engineering (1994) and and Ph.D. (1998) from the National University of Defense Technology (NUDT), China. He was promoted to Professor in 2007. His research interests include brain–computer interface, cognitive neuroscience, image/signal processing, computer/biological vision, and neural networks.

Affiliations

Zhiqian Zhou[1] · **Pengming Zhu**[1] · **Zhiwen Zeng**[1] · **Junhao Xiao**[1] · **Huimin Lu**[1] · **Zongtan Zhou**[1]

Zhiqian Zhou
zhiqian.zhou13@hotmail.com

Pengming Zhu
zhupengming@nudt.edu.cn

Zhiwen Zeng
zengzhiwen@nudt.edu.cn

Junhao Xiao
junhao.xiao@ieee.org

Zongtan Zhou
narcz@163.com

[1] Robotics Research Center, College of Intelligence Science and Technology, National University of Defense Technology, Changsha 410073, China