DOI: 10.1111/tgis.12620

RESEARCH ARTICLE

Transactions in GIS WILEY

# Local motion simulation using deep reinforcement learning

Dong Xu[1,2] | Xiao Huang[2] | Zhenlong Li[2] | Xiang Li[1]

[1]Key Laboratory of Geographical Information Science (Ministry of Education), School of Geographic Sciences, East China Normal University, Shanghai, China

[2]Geoinformation and Big Data Research Laboratory, Department of Geography, University of South Carolina, Columbia, SC, USA

**Correspondence**
Zhenlong Li, Geoinformation and Big Data Research Laboratory, Department of Geography, University of South Carolina, Columbia, SC, USA.
Email: zhenlong@sc.edu

Xiang Li, Key Laboratory of Geographical Information Science (Ministry of Education), School of Geographic Sciences, East China Normal University, Shanghai, China .
Email: xli@geo.ecnu.edu.cn

## Abstract

Traditional local motion simulation focuses largely on avoiding collisions in the next frame. However, due to its lack of forward looking, the global trajectory of agents usually seems unreasonable. As a method of optimizing the overall reward, deep reinforcement learning (DRL) can better correct the problems that exist in the traditional local motion simulation method. In this article, we propose a local motion simulation method integrating optimal reciprocal collision avoidance (ORCA) and DRL, referred to as ORCA-DRL. The main idea of ORCA-DRL is to perform local collision avoidance detection via ORCA and smooth the trajectory at the same time via DRL. We use a deep neural network (DNN) as the state-to-action mapping function, where the state information is detected by virtual visual sensors and the action space includes two continuous spaces: speed and direction. To improve data utilization and speed up the training process, we use the proximal policy optimization based on the actor–critic (AC) framework to update the DNN parameters. Three scenes (circle, hallway, and crossing) are designed to evaluate the performance of ORCA-DRL. The results reveal that, compared with the ORCA, our proposed ORCA-DRL method can: (a) reduce the total number of frames, leading to less time for agents to reach their destination; and (b) effectively avoid local optima, evidenced by smoothed global trajectories.

  

## 1 | INTRODUCTION

Multi-agent navigation, as one of the hot topics in GIScience, has attracted widespread attention in the past few decades due to its important role in robotics (McLurkin & Demaine, 2009), virtual reality (Pettré et al., 2006), and evacuation simulation (Pidd, De Silva, & Eglese, 1996). At the micro level, the goal of local motion simulation is to find a smooth and collision-free shortest path for agents to move from one position to another. Moreover, it aims to ensure that agents do not collide with each other during movement. Based on the existing literature, multi-agent navigation is generally composed of two parts: global path planning, committed to smoothing the shortest path (Kallmann, 2014; Van Toll, Cook, & Geraerts, 2012), and local motion simulation, committed to avoiding static and dynamic obstacles (Fiorini & Shiller, 1998; Helbing & Molnar, 1995). Until now, numerous methods have been focused on local motion simulation. Some of the early efforts in the 1980s included the agent-based model developed by Reynolds (1987) to simulate flock/herd behaviors, where agents can conduct a series of behaviors such as seeking, arriving, pursuing, and following a path. Helbing, in contrast, proposed a social force model (SFM) to formulate local dynamics for agents (Helbing & Molnar, 1995). SFM-based research has been used widely for evacuation (Helbing, Farkas, & Vicsek, 2000), group behavior (Moussaïd, Perozo, Garnier, Helbing, & Theraulaz, 2010), and self-organizing behavior (Helbing, Molnár, Farkas, & Bolay, 2001). However, the parameters between various forces in the aforementioned models need to be carefully designed to balance the movements of the agent towards reality (Curtis & Manocha, 2014).

Unlike SFM, where forces are applied to describe various behaviors, velocity obstacle (VO) methods can successfully avoid collisions with a promising performance by using computational geometry (Fiorini & Shiller, 1998; Snape, Van den Berg, Guy, & Manocha, 2011; Van den Berg, Lin, & Manocha, 2008). Van den Berg, Guy, Lin, and Manocha (2011) proposed an optimal reciprocal collision avoidance (ORCA) method, which is widely used in many studies as one of the best collision-free motion methods, such as crowd simulation (Narain, Golas, Curtis, & Lin, 2009), group behaviors (Karamouzas & Overmars, 2011), and trajectory planning (Guy, Lin, & Manocha, 2010). Similar to SFM, however, ORCA only guarantees collision avoidance in the next frame and tends to produce unreasonable behavior, which can be observed from the global trajectories of simulated agents (Godoy, Karamouzas, Guy, & Gini, 2015).

With the advancements in hardware and software over the past decade, machine learning has achieved remarkable results in image classification (Krizhevsky, Sutskever, & Hinton, 2012), object tracking (Bertinetto, Valmadre, Henriques, Vedaldi, & Torr, 2016), and natural language processing (Sutskever, Vinyals, & Le, 2014). Researchers began to study the movement rules of agents using the emerging data-driven methods (Bera, Kim, & Manocha, 2015; Kim, Bera, Best, Chabra, & Manocha, 2016; Lee, Choi, Hong, & Lee, 2007; Zhong, Cai, Luo, & Yin, 2015). Different from traditional methods, data-driven methods allow movement rules to be learned from life, thus better reconstructing the movements of agents in various scenarios. Despite the success of those machine learning methods, the learned movement rules usually have many limitations. For example, the current learned behavior in one scene is difficult to apply to other scenes, and the method precision depends largely on the camera coverage (Kim et al., 2016) and attributes of the video stream (Bera et al., 2015).

With the AlphaGo (Silver et al., 2017) defeating human Go players, deep reinforcement learning (DRL) has gradually attracted interest. Unlike supervised learning and unsupervised learning, DRL, designed to optimize long-term rewards rather than single-step optimization, is a strategic learning method based on the Markov process. By setting the relevant reward function, DRL encourages agents to find the optimal action in order to get the most rewards at a particular state (Sutton & Barto, 2018). There are many popular DRL methods, such as the double deep Q-network (Double DQN) (Van Hasselt, Guez, & Silver, 2016), twin delayed deep deterministic policy gradient (TD3) (Fujimoto, van Hoof, & Meger, 2018), soft actor–critic (SAC) (Haarnoja, Zhou, Abbeel, & Levine, 2018), and proximal policy optimization (PPO-Clip) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). With a similar concept to DRL, local motion simulation also has Markov nature with the necessity to choose the appropriate action in a particular state to avoid collisions. The difference, however, is that traditional motion simulation

only considers collision avoidance in the current state, ignoring global trajectory optimization. Therefore, to optimize the global trajectory, some studies started to incorporate DRL in agent motion simulation (Lee, Won, & Lee, 2018; Long et al., 2018). Experiments showed that DRL-based motion simulation leads to more reasonable and smoother trajectories than traditional models. However, the integration of DRL and motion simulation is still in a rather nascent stage and deserves further exploration.

In this article, we propose a hierarchical judgment mechanism for local motion simulation where DRL is applied to calculate the desired velocity for each agent, and ORCA is applied to avoid local collision. Our goal is to guarantee collision avoidance during motion simulation and make the global trajectory as smooth as possible. Our contributions are threefold. (a) We applied the DRL to smooth the global motion trajectory. Through the application of DRL, agents are granted certain foresight; therefore, they can adjust their motion and avoid possible conflicts in advance, leading to a smooth overall trajectory. (b) We selected ORCA to avoid collision. In DRL-based motion planning, the movement of the agent is related to the reward function. The use of ORCA can reduce the dilemma of parameter adjustment between goal reward and collision reward in the reward function. (c) We applied the PPO-Clip algorithm to train the deep neural network (DNN). Under the premise of achieving good performance, PPO-Clip can improve sampling efficiency and is relatively simple to implement.

The remaining sections are organized as follows. Section 2 summarizes the existing literature concerning local motion planning of agents, DRL, and agent motion planning based on DRL. Section 3 introduces the ORCA and DRL method, defines the state space, action space, and reward function, and describes the network architecture. Section 4 describes the settings of the relevant parameters in the environment and the computer configuration and introduces three designed scenes to test our proposed method (ORCA-DRL) by comparing it with the traditional ORCA method. Section 5 presents and discusses the results of frame analysis and trajectory analysis, followed by a discussion of the limitations and future directions in Section 6 and conclusions in Section 7.

## 2 | RELATED WORK

Based on the research scale, multi-agent navigation can generally be divided into three categories: macroscopic scale, mesoscopic scale, and microscopic scale. Macroscopic motion simulation usually extracts the space to a node-arc network model, where agents are viewed as the flow on an arc (Cova & Johnson, 2003; Friesz, Luque, Tobin, & Wie, 1989; Merchant & Nemhauser, 1978). Macroscopic motion simulation is often applied in outdoor congestion analysis and optimization to assist decision-making (Li, Li, Xu, Xu, & Zhang, 2018). At mesoscale, motion simulation studies usually focus on indoor situations, which have attracted increasing attention with the development of indoor GIS in the past two decades (Li, Claramunt, & Ray, 2010; Xu, Hijazi, Mebarki, Meouche, & Abune'Meh, 2016). Mesoscopic motion simulation first abstracts the research area into a series of regular grids, then simulates agent movement by updating the occupancy state of the grid at each frame (Kirchner, Nishinari, & Schadschneider, 2003; Varas et al., 2007; Yuan & Tan, 2007; Zheng, Zhong, & Liu, 2009). It has developed many successful cases in indoor emergency evacuation, especially in a rather dynamic environment (Zheng, Jia, Li, & Zhu, 2011). However, the macroscopic model and the mesoscopic model both have their own limitations. The former fails to consider the individual behavior of the agent, while the latter discretizes the state and action space, thus making it difficult to accurately simulate individual behavior. In comparison, the microscopic local motion simulation generally abstracts the agent into a disc with a certain radius and divides time into discrete frames. Each agent, as a completely autonomous entity with no implicit communication, calculates the optimal velocity at each frame according to the environment, including the velocity and position of the surrounding agents.

This article explores motion simulation at the microscopic scale, where each agent is defined as an independent entity with a specific shape and can move in any configuration space (Lozano-Perez, 1990). In the remainder of this section, we briefly review the related research on local motion simulation and the DRL, as well as the progress of local motion simulation based on DRL.

## 2.1 | Local motion simulation

In local motion simulation, each agent is regarded as an automatic self-judgment robot whose goal is to move to the destination while avoiding obstacles and other agents. The SFM quantifies the motion decision via the influence of the interaction of attraction and repulsive force (Helbing & Molnar, 1995; Helbing et al., 2000; Karamouzas, Sohre, Narain, & Guy, 2017). The force-based model can be extended to describe the agent's complex dynamic behavior, such as emergency evacuation (Han & Liu, 2017), social group behaviors (Mehran, Oyama, & Shah, 2009), and self-organizing behavior (Helbing et al., 2001). Given the difficulty in balancing the parameter settings among various forces, the SFM often shows an unreasonable trajectory (Curtis & Manocha, 2014). Different from the SFM, the VO model focuses only on local collision avoidance and has a promising performance (Fiorini & Shiller, 1998). Given its ability to guarantee collision avoidance, the velocity-based method has been widely used, for example the reciprocal velocity obstacle (RVO) model is an extension of the VO model applied to reduce oscillation by considering the reciprocal effect (Van den Berg et al., 2008). To further improve the performance of the RVO model, Van den Berg et al. (2011) proposed a better approach, ORCA, which constructs the collision avoidance region by allowing each agent to take half the responsibility of avoiding pairwise collisions (i.e. the reciprocal property) and calculates the optimal velocity by simplifying the problem to a low-dimensional linear program. ORCA demonstrates powerful capabilities in both efficiency and robustness to *n*-body collision avoidance. Besides, it could easily be extended to three-dimensional-space collision avoidance. Due to its excellent performance in collision avoidance, ORCA has successfully replaced RVO and is widely used in numerous studies as one of the best collision-free motion methods. For example, Guy et al. (2009) proposed a highly parallel collision avoidance method based on ORCA, which simulates thousands of agents in a real-time manner. Snape et al. (2011) proposed a hybrid reciprocal velocity obstacle to solve the oscillation problem. Alonso-Mora, Breitenmoser, Rufli, Beardsley, and Siegwart (2013) proposed a non-holonomic ORCA method that guarantees a visually appealing trajectory. Despite the fact that all of these models can ensure the agent does not collide at the next frame, they failed to provide a mechanism to ensure the smoothness of the global trajectory (Godoy et al., 2015).

To smooth the entire motion trajectory, researchers have proposed a collision avoidance method based on simulated visual information (Dutra, Marques, Cavalcante-Neto, Vidal, & Pettré, 2017; Ondřej, Pettré, Olivier, & Donikian, 2010). Through the incorporation of virtual visual judgment, a more realistic and smooth motion trajectory can be obtained. Unlike the aforementioned traditional methods where the collision is only detected when the distance is immensely close, in visual collision avoidance, agents usually check the possibility of collision in advance. The vision-based method, however, is often computationally complex and sensitive to parameter design. Furthermore, many studies extract pedestrian movement patterns from surveillance videos, allowing the authenticity and reliability of the simulation to be achieved by collecting and analyzing the action choices of pedestrians at different states via machine learning-based data-driven approaches (Bera et al., 2015; Lee et al., 2007; Zhong et al., 2015). However, the accuracy of such studies largely depends on the pedestrian trajectories extracted from the video stream. In addition, the purpose of those studies is to study the behavior of pedestrians, rather than to propose an algorithm to avoid collisions. Besides, given that the trajectories are obtained in a specific scenario, those methods lack the ability to be generalized to other scenarios.

## 2.2 | Deep reinforcement learning

As one branch of machine learning, DRL often serves as a means in agent-based modeling to optimize decision problems with Markov properties (Schmidhuber, 2015; Sutton & Barto, 2018). Given the current environment state, a reward is calculated after a certain agent takes its action, leading to the next state. While the state is being updated, agents take further action. This process continues until the interaction ends. In DRL, all the agents are expected to take optimal action for a certain state to maximize the expected value of the total reward.

DRL can be roughly divided into a value-based deep Q-network (DQN) method (Mnih et al., 2015) and a strategy-based policy gradient (PG) method (Williams, 1992). DQN is an on-policy method that does not require complete interactive trajectory information, allowing a feasible solution to be given in a rather short time. There are many mature DQN methods at present. For example, Double DQN utilizes a separate target network to estimate the target value, leading to a reduced parameter correlation between the estimated value and the target value by delaying the update of the target network (Van Hasselt et al., 2016). By separating the state-action value within the network into state value and advantage value, Duel DQN can effectively avoid unsatisfactory states and speed up the learning performance (Wang et al., 2015). However, the value-based method cannot easily be extended to a high-dimensional continuous space and can easily fall into local optima, given the fact that it does not rely on global trajectory data (Sutton, McAllester, Singh, & Mansour, 2000). In comparison, the PG method is able to output the action directly based on the state, achieving a globally optimal solution from calculations in high-dimensional space (Williams, 1992). However, this method generally requires a complete interaction trajectory, usually leading to long training time. The actor–critic (AC) framework is the learning framework that combines the aforementioned two methods (Konda & Tsitsiklis, 2000; Peters and Schaal, 2008). The AC framework is able to speed up the training process and extend into continuous action space.

The aforementioned methods are on-policy methods, where the training data need to be acquired from the current policy, while the previously collected data cannot be used. To improve data utilization, off-policy methods have been developed. As an off-policy method, actor–critic with experience replay increases the sample efficiency through experience replay and obtains satisfactory results under the premise of interacting with the environment as little as possible (Wang et al., 2016). Lillicrap et al. (2015) proposed a deep deterministic policy gradient (DDPG), allowing the Q-function and policy to be learned in a continuous space. Haarnoja et al. (2018) proposed the SAC, aiming to maximize the entropy in actor learning. Although the sampling efficiency is improved in off-policy methods, it requires a large amount of memory space to store historical trajectory data. Besides, most of the time, the actor network and the critic network cannot be shared, therefore high-performance computing is always required. Given those limitations, Schulman et al. (2017) proposed a PPO-Clip algorithm, which improves the sample efficiency via important sampling, at the same time constraining policy update via clipped probability ratios (a suitable replacement for the Kullback–Leibler [KL] penalty). The PPO-Clip algorithm provides a shared network where the actor and the critic are able to share the majority of the network. Coupled with the multi-process technique, PPO-Clip can be regarded as a hardware-friendly algorithm, compared with SAC and DDPG. Although PPO-Clip is intrinsically an on-policy algorithm, it can take advantage of historical data with acceptable memory space. Given its numerous advantages, PPO-Clip received wide attention and became the default DRL algorithm at OpenAI, a famous artificial intelligence (AI) company. It has been widely adopted in many applications, such as quadrotor control (Lopes, Ferreira, da Silva Simões, & Colombini, 2018), multi-joint arm control (Tieck et al., 2018), and locomotion behavior learning (Heess et al., 2017). In this article, we adopted the PPO-Clip algorithm to train our DNN model (described in Section 3.2).

## 2.3 | Motion simulation using DRL

Many studies have applied DRL in motion simulation. Based on different foci, DRL-supported motion simulation can generally be divided into single- and multi-agent navigation. For single-agent navigation, Prescott and Mayhew (1992) proposed a DRL method to dictate obstacle avoidance behavior through a virtual visual sensor. In their method, they minimized the negative reward arising from collisions by exploring the surrounding environment. Xia and El Kamel (2015) proposed a DQN method to automatically navigate mobile vehicles in an unknown environment. Ramezani Dooraki and Lee (2018) proposed an end-to-end memory-based DRL method that allows agents to automatically explore unknown environments while avoiding obstacles. Wu, Esfahani, Yuan, and Wang (2019) proposed a double deep Q-learning network method to directly map the raw depth images to control

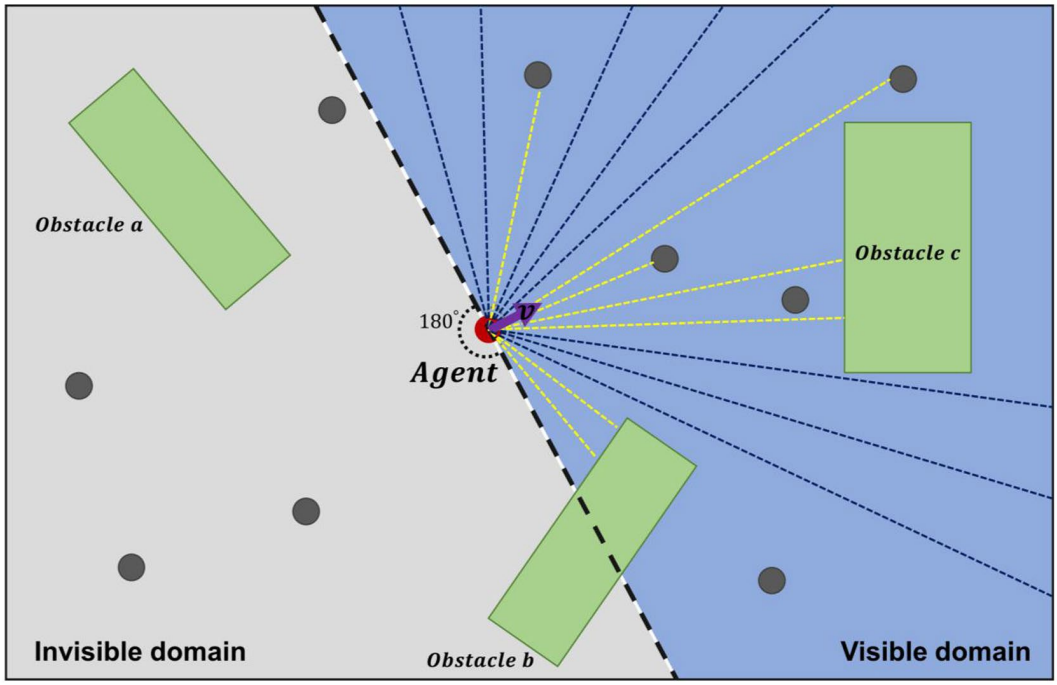


**FIGURE 1** Environment checking through visual sensors

commands. However, single-agent navigation is usually used for environmental detection for real robots, where the purpose is to avoid obstacles.

Different from single-agent simulation, multi-agent navigation requires agents to avoid moving obstacles, including other agents. Torrey (2010), for instance, applied reinforcement learning in crowd simulations and demonstrated its high feasibility. Torrey (2010) first established a state-action tabular and then obtained the optimal solution via the Q-learning method. Godoy et al. (2015) proposed an adaptive learning approach based on multi-armed bandits to compensate for the inefficient global behavior generated in existing physical models. Their experimental simulations showed that their methods could effectively reduce local collisions and smooth the simulation process, leading to more realistic agent behavior. Lee et al. (2018) proposed a DRL-based AC framework in crowd simulation with great generalization ability, in which the reward function is divided into goal reward, collision reward, and smooth reward. Their experiment used visual sensors as the state input, and magnitude along with velocity direction as outputs. Ravichandran, Yang, Peters, Lansner, and Herman (2018) regarded agent simulation as a multi-objective problem by applying a modular RL method. Their model shows good performance without further parameter tuning. Long et al. (2018) presented a decentralized collision avoidance policy for multi-robot systems by directly detecting the environment via raw sensors, further serving as primary inputs to the DRL network. Multi-agent navigation needs each agent to detect the dynamic environment and guides the agent to get to the destination without collision by setting multiple reward functions. However, it usually requires careful adjustment of the parameters to achieve the dual requirements of smoothing the trajectory and avoiding collisions.

In summary, popular local motion simulation methods, such as ORCA, can effectively avoid the collision of agents but are prone to unreasonable trajectories, especially in complicated situations due to the lack of predictability. The application of the DRL method aims to optimize the global motion trajectory by defining suitable reward functions. However, local motion simulation based on the DRL alone is likely to cause collisions, given its unstable nature (Irpan, 2018; Stadelmann et al., 2018). Therefore, we propose a simulation algorithm

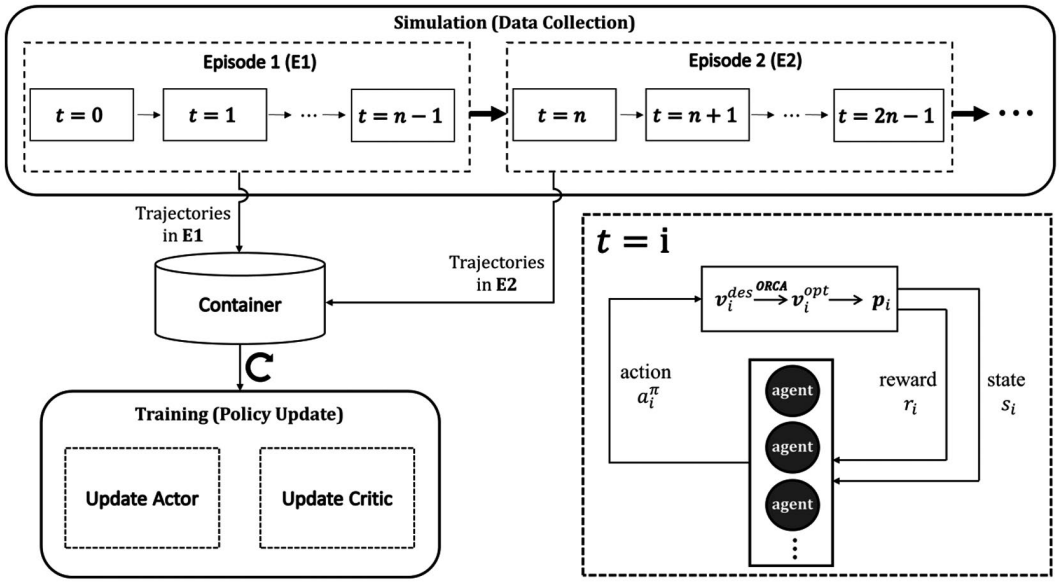


**FIGURE 2** General workflow

that combines ORCA and DRL (termed ORCA-DRL) to guarantee collision avoidance via ORCA while achieving a smooth global trajectory via DRL. This study does not focus on the cross-comparison of various local simulation methods, but aims to illustrate the improvement when the DRL algorithm is introduced and integrated into the widely used ORCA model.

## 3 | METHODOLOGY

### 3.1 | Overview

We assume $N$ agents in the local environment and each agent has its predefined destination (Figure 1). The goal of each agent is to reach the destination with the least frames and avoid collision with obstacles and other agents during movement. For simplicity, we set the environment to be a two-dimensional space $\mathbb{R}^2$, where each agent is represented by a disc with radius $r$ and the obstacles, shown as green rectangles in Figure 1, are represented by a combination of line segments. For each agent $i$, we define its position at frame $t$ as $\boldsymbol{p}_i^t$(2d-vector), the destination position as $\boldsymbol{g}_i$(2d-vector), and the velocity at time $t$ as $\boldsymbol{v}_i^t$(2d-vector). $\boldsymbol{v}_i^t$ further includes speed $v_i^t$(magnitude) and direction $a_i^t$(magnitude), the desired velocity as $\boldsymbol{v}_i^{\text{pref}}$(2d-vector), the optimal velocity as $\boldsymbol{v}_i^{\text{opt}}$(2d-vector), and the maximum speed limit as $v_i^{\max}$(magnitude). The position and velocity of the surrounding obstacles are observed by agents as they consider other agents as moving obstacles. We define the interval between two adjacent frames as $\varsigma$. At each frame, each agent updates its position based on its current velocity $\boldsymbol{v}^t$ calculated by the motion planning method. The simulation ends when all agents arrive at their destination position.

Agents check the surrounding environment through visual sensors at each frame. Specifically, as shown in Figure 1, the current agent is marked as a red circle with the current velocity $\boldsymbol{v}$ and the obstacles are marked as green rectangles. The visual sensor of each agent consists of a 180° viewing angle and $m$ emitted rays ($m = 256$ in our experiment) at equal intervals from the agent's center position. If a ray detects an obstacle, it will be truncated as shown by the yellow dashed line, meaning that the agent's exploration in this direction is obscured by the

obstacle (agents do not see through obstacles). A total of $m$ segments with different lengths are derived, representing the environmental information observed by the agent at the current frame.

The whole process in our research can be divided into two parts (Figure 2): the simulation process and the training process. The simulation process consists of multiple episodes. At each episode ($n$ frames), the trajectories of agents obtained from each frame $t$ are saved into a container, in which the trajectories are further utilized to update the actor and critic, two parameters in our proposed DNN model. After training of an episode, the trajectories in that episode are released from the container and the trajectories from the next episode are fed to the container. The detailed simulation process in each frame $t$ is composed of the following five steps: (a) extracting the state $s_t$ from the environment; (b) inputting $s_t$ to the agent policy, i.e. the DNN, and outputting the action $a_t^{\pi}$ based on policy $\pi$; (c) transferring $a_t^{\pi}$ to a desired velocity $\boldsymbol{v}_i^{\text{pref}}$ and feeding $\boldsymbol{v}_i^{\text{pref}}$ to the ORCA to produce an optimal collision-free velocity $\boldsymbol{v}_i^{\text{opt}}$; (d) further adjusting the agent position based on $\boldsymbol{v}_i^{\text{opt}}$; and (e) proceeding to the next frame $s_{t+1}$ and providing a reward $r_t$ to the next frame. In the following sections, we present more details about our simulation architecture.

## 3.2 | Optimal reciprocal collision avoidance

We applied ORCA to select the optimal velocity while avoiding collisions. ORCA first calculates the VO region for the current agent and ORCA regions (half-plane non-collision velocity set) for all surrounding agents, then further selects the optimal velocity from the intersection of ORCA regions.

Located at $\boldsymbol{P}_A$ and $\boldsymbol{P}_B$, two agents $A$ and $B$ are moving with velocity $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$, respectively (Figure 3a). We define a ray function, starting at $\boldsymbol{p}$ and heading to $\boldsymbol{v}$:

$$\Gamma(\boldsymbol{p},\boldsymbol{v}) = \{\boldsymbol{p} + \lambda\boldsymbol{v} | \lambda \geq 0\} \tag{1}$$

we further define $\boldsymbol{v}_{A-B} = \boldsymbol{v}_A - \boldsymbol{v}_B$. The $\text{VO}_B^A$ region can be calculated using the formula proposed by Fiorini and Shiller (1998):

$$\text{VO}_B^A = \{\boldsymbol{v} | \Gamma(\boldsymbol{P}_A, \boldsymbol{v}_{A-B}) \cap (B - A) \neq \emptyset\} \tag{2}$$

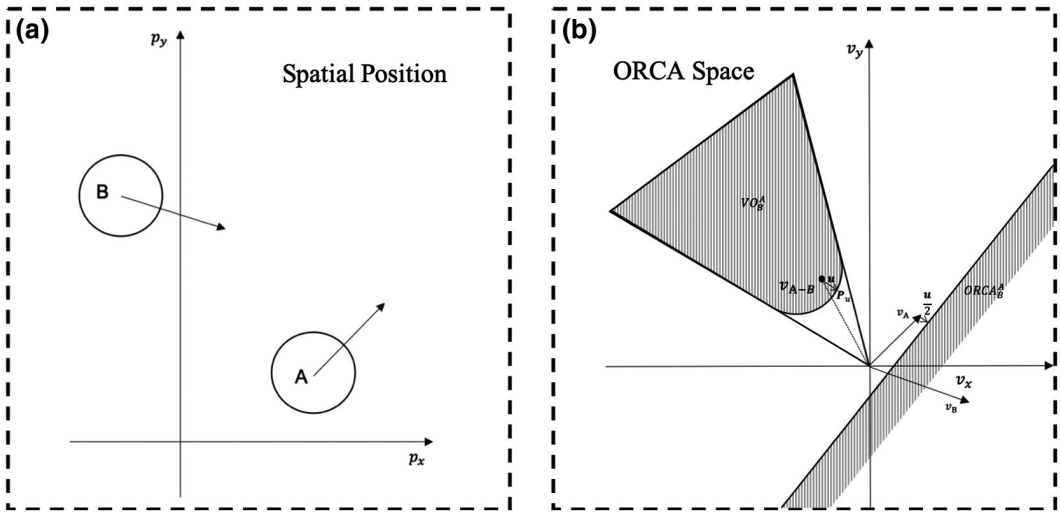


**FIGURE 3** ORCA method: (a) spatial relationship between A and B; and (b) velocity relationship and the ORCA space

where $B \oplus -A$ represents the configuration space of agent $B$ in relation to agent $A$. Further, whether a collision might happen can be checked through the relationship between $\boldsymbol{v}_{A-B}$ and the $\mathrm{VO}_B^A$ region (Figure 3b). If $\boldsymbol{v}_{A-B}$ falls within the $\mathrm{VO}_B^A$ region, then collision will occur within $\tau$. If $\boldsymbol{v}_{A-B}$ falls outside the $\mathrm{VO}_B^A$ region, then collision is avoided. If a collision is determined to occur, ORCA finds the nearest point $\boldsymbol{P}_u$ from $\boldsymbol{v}_{A-B}$ to the $\mathrm{VO}_B^A$ edge (the normal at $\boldsymbol{P}_u$ is termed $\boldsymbol{n}_u$). We further define $\boldsymbol{u} = \boldsymbol{P}_u - \boldsymbol{v}_{A-B}$.

For agent $A$, we assume that its desired velocity in the next time is $\boldsymbol{v}_A^{\mathrm{pref}}$. Based on the reciprocal principle between the two agents, $\boldsymbol{v}_A^{\mathrm{pref}}$ needs to fall into the range $\left[0, \frac{1}{2}\boldsymbol{u}\right]$ to avoid collision. The ORCA region for agent $B$ to agent $A$ is defined as follows:

$$\mathrm{ORCA}_B^A = \left\{ \boldsymbol{v} \middle| \left( \boldsymbol{v} - \left( \boldsymbol{v}_A^{\mathrm{pref}} + \frac{1}{2}\boldsymbol{u} \right) \right) \boldsymbol{n}_u \geq 0 \right\} \tag{3}$$

with $n$ agents ($B$, $C$, $D$, ...) surrounding agent $A$, a total of $n$ half-plane solution sets are derived, respectively, as $\mathrm{ORCA}_B^A$, $\mathrm{ORCA}_C^A$, $\mathrm{ORCA}_D^A$, ... The optimal velocity for agent A is derived from the intersection of all $n$ half-plane solution sets (closest to the $\boldsymbol{v}_A^{\mathrm{pref}}$ and minimally changed compared with $\boldsymbol{v}_A^{t-1}$). If the intersection is empty or the intersection violates kinematic principles, agent A stops moving at the next frame to avoid collision. Given the reciprocal property, the simulation results will not be affected by the order of the agents being simulated.

## 3.3 | Deep reinforcement learning

### 3.3.1 | AC framework

Given the fact that the current position (state) of the agent is only related to the previous position (last state), the trajectory smoothing can be solved via DRL as it belongs to the Markov decision process.

We assume that each agent optimizes its own trajectory independently, allowing it to reach the destination with the fewest frames. A single agent-based DRL method is applied, since we assume there is no explicit cooperation or competition among agents but they can sense the positions and velocities of their neighbors, thus affecting their own movement decisions. The DRL method applied in this study generally consists of three parts: environment, agent, and reward function. At each frame $t$, we define the agent action as $a_t^\pi$, the current environment state as $s_t$, and the policy as $\pi$. Based on $s_t$ and $a_t^\pi$, feedback, including the next state $s_{t+1}$ and the corresponding reward value $r_{t+1}$, will be determined by the environment. When the interaction is over, a trajectory $\tau$ is formed containing all the states and actions, that is $\tau = \left(s_1, a_1^\pi, s_2, a_2^\pi, \ldots, s_n, a_n^\pi\right)$. The total reward, defined as $R_\pi(\tau)$, equals $\sum_{t=1}^{n} \gamma^{t-1} r_t$, where $\gamma$ is a discount factor. The aim of DRL is to derive an optimal trajectory $\tau^{\mathrm{opt}}$ so that $R\left(\tau^{\mathrm{opt}}\right)$ is maximized. Since $R_\pi(\tau)$ is a random variable, we maximize the expectation of $\overline{R_\pi}$ ($E_{\tau\sim\pi}\left[R_\pi(\tau)\right]$). $\overline{R_\pi}$ can then be approximated via multi-sampling:

$$E_{\tau\sim\pi}\left[R_\pi(\tau)\right] = 1/N \sum_{n=1}^{N} R_\pi\left(\tau^n\right) p_\pi\left(\tau^n\right) \tag{4}$$

where $p_\pi(\tau)$ represents the union probability for the given trajectory under the policy $\pi$, that is:

$$p_\pi(\tau) = p\left(s_1\right) p_\pi\left(a_1^\pi | s_1\right) p\left(s_2 | s_1, a_1^\pi\right) p_\pi\left(a_2^\pi | s_2\right) \cdots p_\pi\left(a_n^\pi | s_n\right) \tag{5}$$

Further, $R\left(\tau^{opt}\right)$ can be calculated using a gradient ascent algorithm, where the total gradient under the current strategy can be described as:

$$\nabla\overline{R_\pi} = 1/N\sum_{n=1}^{N}\sum_{t=1}^{T} R_\pi\left(\tau^n\right)\nabla\log p_\pi\left(a_t^n|s_t^n\right) \tag{6}$$

In practice, we can replace $R_\pi\left(\tau^n\right)$ with the advantage function $\mathcal{A}$, defined as follows:

$$\mathcal{A} = r_{t+1} + V\left(s_{t+1}\right) - V\left(s_t\right) \tag{7}$$

where $V\left(s_t\right)$ serves as a baseline to reduce the variance and $r_{t+1}+V\left(s_{t+1}\right)$ represents the Q-value, that is $R_\pi(\tau)$. Schulman, Moritz, Levine, Jordan, and Abbeel (2015) proposed a generalized advantage estimation (GAE) method to further reduce the variance of policy gradient estimates. GAE introduced a new advantage function as $\mathcal{A}^{GAE} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\mathcal{A}_{t+l}$, where $\lambda$ is a hyperparameter between [0, 1]. In this article, we use GAE as our advantage function.

### 3.3.2 | Proximal policy optimization

To improve the sampling efficiency, Schulman et al. (2017) proposed a PPO-Clip algorithm that uses importance sampling to combine old policy $\pi'$ and current policy $\pi$. In our study, we applied this algorithm to utilize data from policy $\pi'$ to multi-update our DNN. The ratio between the two policies is defined as:

$$\text{ratio} = \pi(a|s)/\pi'(a|s) \tag{8}$$

where $\pi(a|s)$ and $\pi'(a|s)$ represent the probability of choice action $a$ in state $s$ under policy $\pi$ and $\pi'$, respectively.

To avoid the situation where the ratio changes too much in one update, PPO-Clip further applies a clipping parameter to restrict the update. We define the policy objective function $L^{ppo\_clip}$ as below:

$$L^{\text{ppo\_clip}} = E\left[\min\left(\text{ratio}\times\mathcal{A}^{GAE},\ \text{clip}\left(\text{ratio}, 1-\in, 1+\in\right)\times\mathcal{A}^{GAE}\right)\right] \tag{9}$$

where $\in$ is the clip parameter.

To maximize $L^{\text{ppo\_clip}}$, the ratio is trained by detaching the gradient of $\mathcal{A}^{GAE}$. If $\mathcal{A}^{GAE}>0$, we increase the ratio, but limit it within $1+\in$; while if $\mathcal{A}^{GAE}<0$, we reduce the ratio, but limit it within $1-\in$. Assuming the batch size is $n$, we define the mean square error (MSE) loss function to optimize the critic (i.e. to make $V^{estimate}$ and $V^{true}$ as close as possible). The MSE loss function is defined as follows:

$$L = \frac{\sum_i^n\left(V_i^{\text{estimate}} - V_i^{\text{true}}\right)^2}{n} \tag{10}$$

where

$$V^{\text{estimate}} = V_{\pi'} \tag{11}$$

$$V^{\text{true}} = \mathcal{A}^{GAE} + V_\pi \tag{12}$$

In the remainder of this section, we describe the state spaces, action spaces, and reward functions involved in DRL, and further elaborate on our network structure.

*State space*

$S^t = \left[\mathcal{O}^t_{\text{self}}, \mathcal{O}^t_{\text{env}}\right]^T$. $\mathcal{O}^t_{\text{self}}$ is composed of two velocities: $\boldsymbol{v}^t$ and $\boldsymbol{v}^t_{g-p}$, where $\boldsymbol{v}^t_{g-p}$ is a unit vector with direction towards the destination position at frame $t$. $\mathcal{O}^t_{\text{env}}$ represents the dynamic environment an agent detected. From the description above, the environmental information at frame $t$ observed by an agent can be described as an $m$-dimensional vector $\mathcal{O}^t_m$. Given the dynamic characteristics of the environment, we select the last four frames as the environment input, so $\mathcal{O}^t_{\text{env}} = \left[\mathcal{O}^{t-3}_m, \mathcal{O}^{t-2}_m, \mathcal{O}^{t-1}_m, \mathcal{O}^t_m\right]^T$.

*Action space*

In traditional motion planning, $\boldsymbol{v}^{\text{pref}}$ is always towards the direction of the destination position, potentially leading to a local optimum. The DNN applied in our study aims to smooth the trajectory by generating an adaptive $\boldsymbol{v}^{\text{pref}}$ based on different states. The action space $\mathcal{A}^t$ has two components, $\theta$ and $\mu$. They respectively represent the direction and magnitude of the velocity. $\theta$ is the direction of the agent's desired velocity $\boldsymbol{v}^{\text{pref}}$, ranging from $\theta_{\min}$ to $\theta_{\max}$, while $\mu$ is restricted to follow the kinetic principle, that is, the maximum speed of an agent $i$ cannot surpass $v_i^{\max}$ $\left(\in \left[0, v_i^{\max}\right]\right)$. The pair $\theta$and$\mu$, representing the desired velocity $\boldsymbol{v}^{\text{pref}}$, will be further fed into ORCA for collision judging, resulting in an optimal velocity $\boldsymbol{v}^{\text{opt}}$.

*Rewards function*

The rewards function $\mathcal{R}$ consists of two parts—the goal reward ($\mathcal{R}^{\text{goal}}$) and the collision reward ($\mathcal{R}^{\text{collide}}$):

$$\mathcal{R} = \mathcal{R}^{\text{goal}} + \mathcal{R}^{\text{collide}} \tag{13}$$

where $\mathcal{R}^{\text{goal}}$ is further defined as

$$\mathcal{R}^{\text{goal}} = \begin{cases} w_f, & \text{if } \| \boldsymbol{g}_i - \boldsymbol{p}_i^t \| < r \\ w_g * \left( \| \boldsymbol{g}_i - \boldsymbol{p}_i^{t-1} \| - \| \boldsymbol{g}_i - \boldsymbol{p}_i^t \| \right), & \text{otherwise} \end{cases} \tag{14}$$

where $w_g$ is the weight of $\mathcal{R}^{\text{goal}}$, $\| \boldsymbol{g}_i - \boldsymbol{p}_i^{t-1} \|$ and $\| \boldsymbol{g}_i - \boldsymbol{p}_i^t \|$ respectively represent the distance from agent $i$ to its destination in the last frame $t-1$ and in the current frame $t$. If the current position $\boldsymbol{p}_i^t$ is closer to the goal $\boldsymbol{g}_i$ than the previous position $\boldsymbol{p}_i^{t-1}$, then agent $i$ receives a positive goal reward and vice versa.

We further define $\mathcal{R}^{\text{collide}}$ to push agents away from each other if they stay too close, resulting in congestion:

$$\mathcal{R}^{\text{collide}} = \begin{cases} w_c, & \text{if } \| \boldsymbol{P}_o - \boldsymbol{P}_i \| < w_d \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

where $\| \boldsymbol{P}_o - \boldsymbol{P}_i \|$ represents the distance between agent $i$ and another agent $o$. The greater the penalty $w_c$, the more sensitive agents are to congestion.

### 3.3.3 | DNN architecture

The neural network architecture (Figure 4) is fed with $\mathcal{O}^t_{\text{env}}$ and $\mathcal{O}^t_{\text{self}}$ at each frame. The outputs of the architecture are the actor and the critic. Three one-dimensional convolution layers (out-channel 32, kernel size 3, and stride 1)

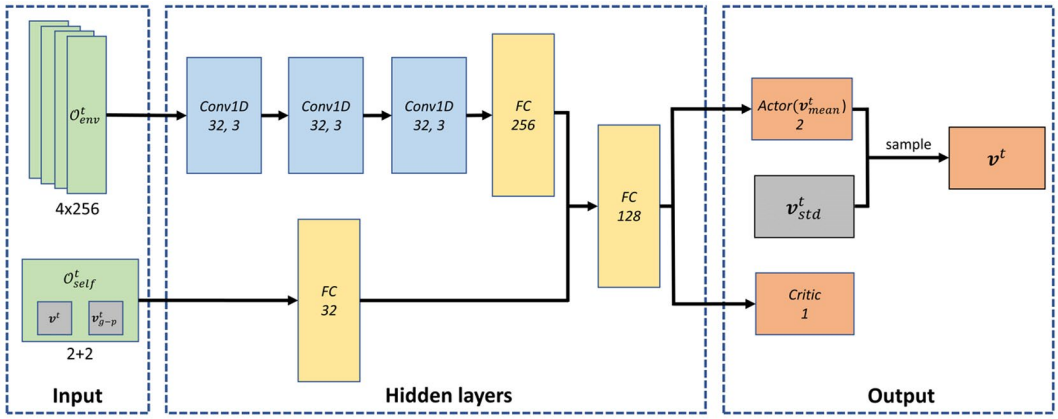


**FIGURE 4** Deep neural network architecture

are applied for feature extraction purposes. MaxPooling operations with kernel = 2 and ReLU nonlinearities (Nair & Hinton, 2010) are applied after each convolution layer. The resulting $\mathcal{O}^t_{\text{env}}$, after passing through three stacked one-dimensional convolution layers, serves as input to a fully connected layer with 256 rectifier units. As for $\mathcal{O}^t_{\text{self}}$ (a four-dimensional vector), we first feed it into a fully connected layer with 32 rectifier units and then combine it with the feature extracted from $\mathcal{O}^t_{\text{env}}$. The resulting combination is passed to a fully connected layer with 128 rectifier units. The actor layer consists of two values: (1) mean of the speed ($v^t_{\text{mean}}$); and (2) direction ($a^t_{\text{mean}}$). A sigmoid and a hyperbolic tangent activate function (tanh) are used to limit $v^t_{\text{mean}}$ to a range between 0 and 1 and $a^t_{\text{mean}}$ to a range between −1 and 1.

We further define $\boldsymbol{v}^t_{\text{mean}} = [v^t_{\text{mean}}, a^t_{\text{mean}}]$ and derive the final $\boldsymbol{v}^t$ from a Gaussian distribution $\mathcal{N}(v^t_{\text{mean}}, v^t_{\text{std}})$, where $v^t_{\text{std}}$ is the standard deviation, a constant hyperparameter. After that, we transform the $\boldsymbol{v}^t$ to $\boldsymbol{v}^{\text{pref}}$ by multiplying the corresponding coefficients.

## 4 | EXPERIMENT AND SCENE

### 4.1 | Coding environment

The implementation is based on a Python coding environment, where the PyTorch package (Paszke et al., 2019) is used to build a DNN for training the state-to-action relational mappings and the Gym package (Brockman et al., 2016) is used to visualize the simulation. The simulation runs on a computer with Ubuntu 18.04 in an environment that consists of i5 CPU, 8G RAM, NVIDIA GTX1060 3G. A multi-core acceleration technique is used to speed up the data collection process, where each core as a worker tackles a simulation simultaneously. The collected data are then passed to the GPU to accelerate the neural network training. The hyperparameter setting used in this study is presented in Table 1.

### 4.2 | Scenes

We set up three scenes to verify our method: circle, hallway, and crossing (Figure 5). Those scenes were selected because they are regarded as benchmarks for testing motion simulation performance in many studies (Alonso-Mora et al., 2013; Dutra et al., 2017; Godoy et al., 2015). In all scenes, a local coordinate system (centered at all three scenes) is used, where each agent is abstracted as a disc with radius 2.0 (similar settings can be found at http://gamma.cs.unc.edu/RVO2/). We define the time interval between two adjacent frames to be 0.5 s.

**TABLE 1** PPO-Clip hyperparameters used in local motion simulation

| Hyperparameter | Value |
|---|---|
| Learning rate | 3e-4 |
| Clipping parameter ($\in$) | 0.2 |
| Discount ($\gamma$) | 0.99 |
| GAE parameter ($\lambda$) | 0.95 |
| Frames in an episode | 20 |
| Batch size | 10 |
| Number of workers | 6 |
| Speed | (0.0, 2.0) |
| Angle | $(-\pi/2, \pi/2)$ |
| $w_f$ | 5.0 |
| $w_g$ | 3.0 |
| $w_c$ | −10.0 |
| $w_d$ | 6.0 |

Details of the three scenes are described as follows.

- **Circle scene (**Figure 5a**).** The circle scene describes a scenario in which agents are initially equidistantly distributed on a circumference (Figure 5a1). The goal of each agent is to arrive at the center-symmetric side of the circle (Figure 5a2). Figure 5a presents an 8-agent case, however, we also test a 16-agent case as a comparison. The circle scene is a typical scene to simulate the motion of agents towards a destination position in a circle-shaped environment (Alonso-Mora et al., 2013; Long et al., 2018; Van den Berg et al., 2008). We set the radius of the circle scene as 60.0.
- **Crossing scene** (Figure 5b**).** Aiming to simulate the behavior of agents at a crossroads, the crossing scene has been widely applied in crowd simulation (Godoy et al., 2015; Ondřej et al., 2010). In our study, we divided our agents into two groups, respectively located below and at the left of the intersection (Figure 5b1), with 10.0 as the interval among agents. The goal of the group of agents located below is to cross the intersection and arrive at the top side of the intersection. Similarly, agents located at the left side of the intersection aim to arrive at the opposite side of the intersection (Figure 5b2). The extent of the scene is set to be 100.0, while the road width is set to be 40.0. In addition to testing the 18-agent case (with 9 agents in each group) presented in Figure 5b, we also test a 32-agent case with 16 agents in each group.
- **Hallway scene (**Figure 5c**).** The hallway scene aims to investigate a scenario where two groups of agents are moving towards each other (Dutra et al., 2017; Helbing, Farkas, Molnar, & Vicsek, 2002). In our study, we divide our agents into two groups, located respectively at the opposite sides of a hallway (Figure 5c1), with 10.0 as the interval among agents. The goal of each agent is to move across the hallway and eventually arrive at the corresponding location of the agent in the other group (Figure 5c2). In this study, we set the road width as 40.0, the length and width of the wall as 200.0 and 10.0, respectively. Besides testing the 18-agent case presented in Figure 5c, we also test a 32-agent case as a comparison.

## 5 | RESULTS AND DISCUSSION

In this study, we conduct analysis from two different perspectives: frame and trajectory. In frame analysis, we compare model performance in different frames, while in trajectory analysis, we focus on the smoothness of their

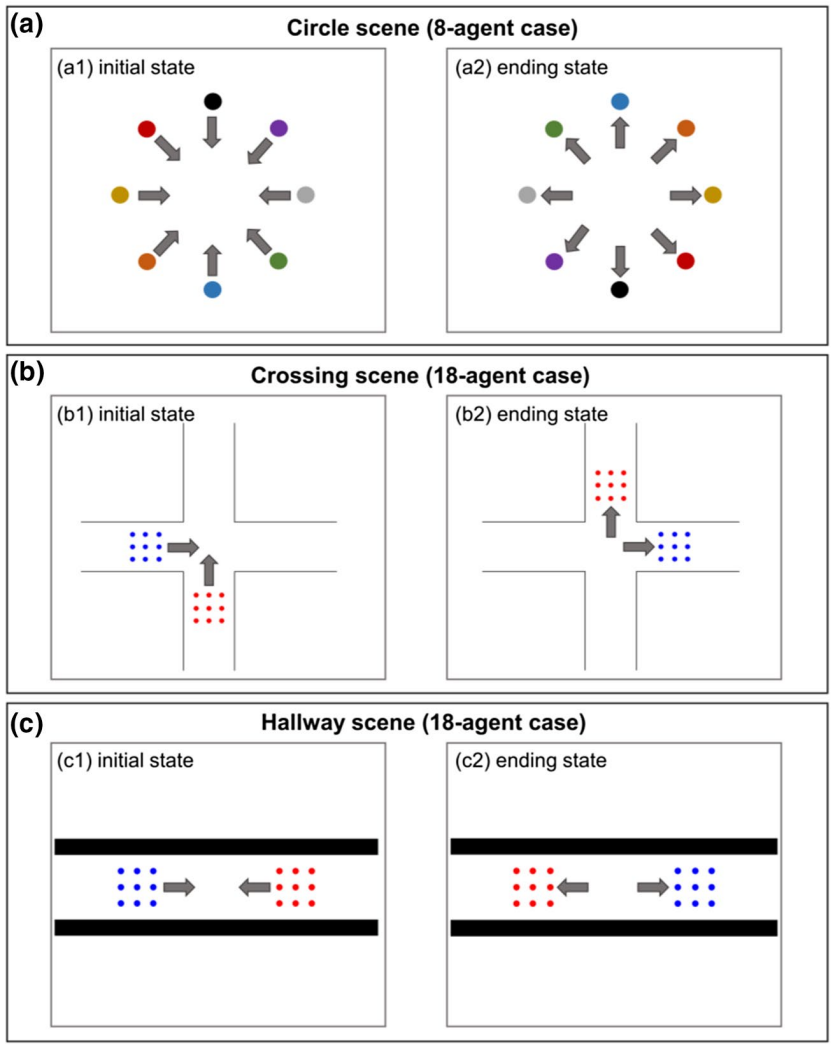


**FIGURE 5** (a) Circle scene (8-agent case); (b) crossing scene (18-agent case); and (c) hallway scene (18-agent case)

global trajectory. For the circle scene, an interval of 15 frames is used to present the performance of different models. An interval of 20 frames is used for the hallway scene and the crossing scene, because the simulating processes for these two scenes are slightly longer. The two models to be compared are the traditional ORCA method (termed ORCA) and our modified ORCA method supported by DRL (termed ORCA-DRL).

## 5.1 | Frame analysis

### 5.1.1 | Circle scene

To analyze agents' behavior in the circle scene, we present the model performance every 15 frames, from 0- to 90-frames. We also test two cases with different numbers of agents: 8- and 16-agent cases (Figure 6). For the traditional ORCA method, in the beginning, agents are heading straight towards their destinations (center-symmetric

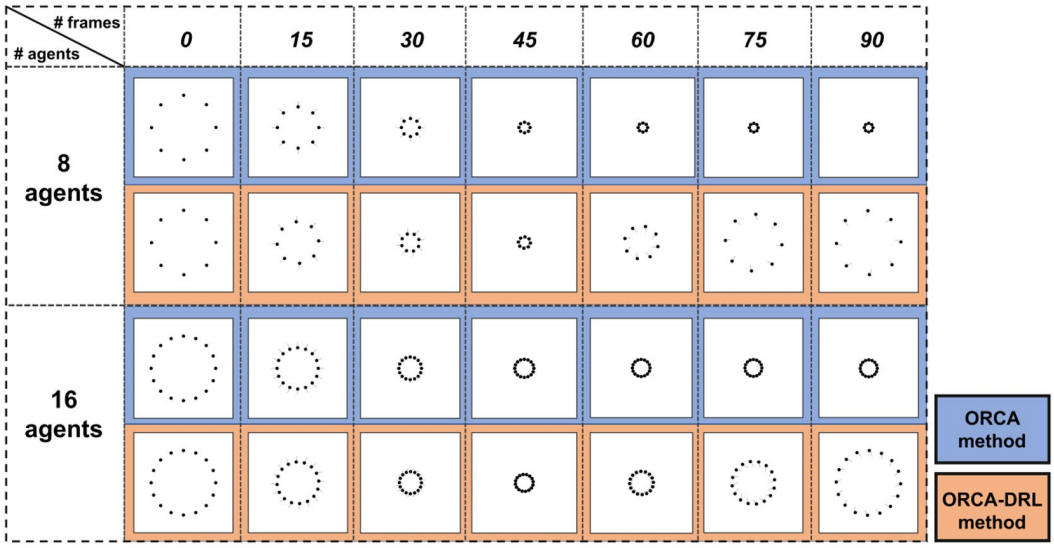


**FIGURE 6** Model performance in different frames in the circle scene

locations from their initial locations). However, at around the 45-frame, agents are trapped in the center position and stop moving, in both the 8- and 16-agent cases. This is presumably due to the fact that the OCRA method only optimizes the current state, potentially leading to a local optimum.

In comparison, starting from the 30-frame, agents in the OCRA-DRL method no longer head directly towards their destinations. After a short encounter at the 45-frame, agents separate smoothly from each other without the permanent entanglement observed in the traditional ORCA method. Later on, agents in ORCA-DRL gradually arrive at their target locations.

The ability of the ORCA-DRL method to solve the entanglement at the center location results from the nature of DRL. ORCA-DRL agents consider the best action to maximize the reward in each state, which gives them the capability to avoid potential collisions in the future. In comparison, ORCA agents keep their original velocity at the beginning, given the local optimum, resulting in missing the best window to avoid collision in the circle scene. ORCA agents stop moving at about the 45-frame because the intersection of all ORCA half-plane solution sets of those agents is zero. The good performance in both the 8- and 16-agent cases demonstrates that our ORCA-DRL method successfully avoids local optima and has great generalization potential in other crowded cases.

### 5.1.2 | Hallway scene

In this scene, we showcase the performance of two models every 20 frames, starting from 0- to 120-frames. Two cases, the 18- and 32-agent cases, are presented in Figure 7. Although the agents in both ORCA and ORCA-DRL arrived at their destination before 120-frame in both cases, agents' movement patterns from the two methods differ slightly at the beginning and differ greatly when two groups of agents meet. At the 20-frame, we notice that two groups of ORCA agents still head straight towards each other, given that this direction is determined as an optimal solution by ORCA at the current state. In comparison, two groups of ORCA-DRL agents shifted their position slightly vertically, making it easier for them to cut through the other group when the meeting of the two groups happened.

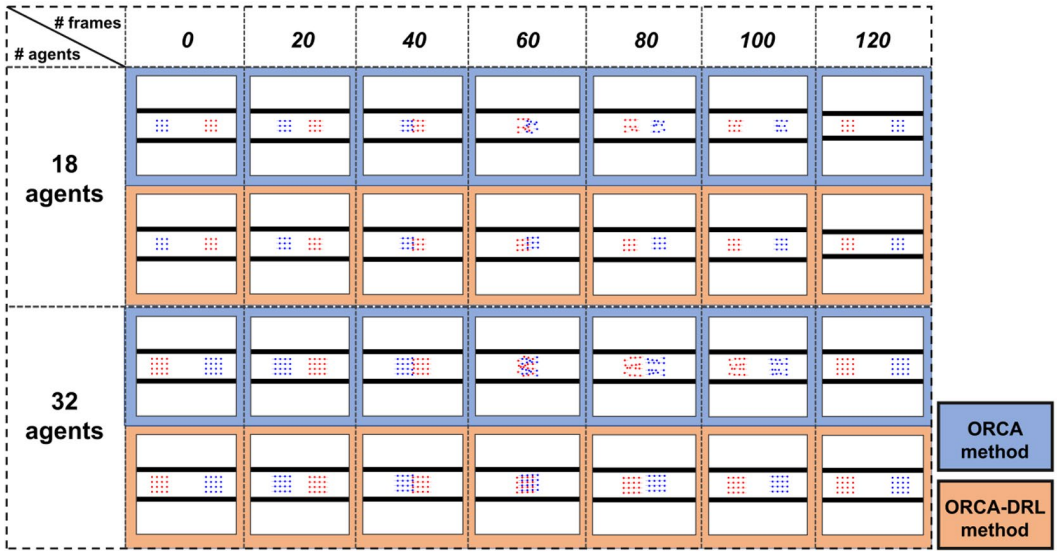


**FIGURE 7** Model performance in different frames in the hallway scene

At the 60-frame, when two groups of agents meet at the center of the hallway, ORCA agents tend to exhibit chaotic patterns, while ORCA-DRL agents are able to maintain a relatively stable position. At the 80-frame, after the meeting of the two groups, ORCA-DRL agents manage to keep the initial formation, while ORCA agents attempt to correct their position to return to the previous formation.

Comparing the total number of frames the two methods take for their agents to arrive at their destinations, in the 16-agent case, ORCA takes a total of 106 frames while ORCA-DRL takes 99 frames. In the 32-agent case, ORCA takes 116 frames, while ORCA-DRL takes 109 frames.

As the number of agents increases, we observe a greater chaotic pattern for ORCA agents when two groups of agents meet each other. ORCA-DRL agents are able to maintain their initial formation during the group meeting phase, leading to fewer frames for those agents to arrive at their destinations in both cases. The similar performance of our ORCA-DRL method in both the 18- and 32-agent cases again demonstrates that our model has great generalization capability in the hallway scene.

### 5.1.3 | Crossing scene

Similar to the hallway scene, the crossing scene investigates the agents' behavior when two groups of agents meet. In this scene, we present the model performance at an interval of 20 frames, from 0- to 120-frames (Figure 8). It is obvious that ORCA-DRL agents in both cases (18- and 32-agent cases) finish the simulation before the 120-frame, while several ORCA agents still need to adjust their positions after the 120-frame. In the 18-agent case, ORCA agents take 128 frames to arrive at their destination, while ORCA-DRL agents take 102 frames. In the 32-agent case, ORCA agents take 139 frames, while ORCA-DRL agents take 114 frames. The results suggest that agents in ORCA-DRL are more efficient as they take less time to achieve the destination compared with ORCA agents. The major difference in model performance can be observed at the 60-frame (group meeting phase) and the 80-frame (recovery phase). ORCA-DRL agents exhibit less chaos during the group meeting phase by keeping a relatively stable formation compared with ORCA agents. Given the chaotic behavior during the group meeting phase, ORCA agents need to return to the initial formation, leading to a significantly longer time (more frames) for agents to reach their destinations.

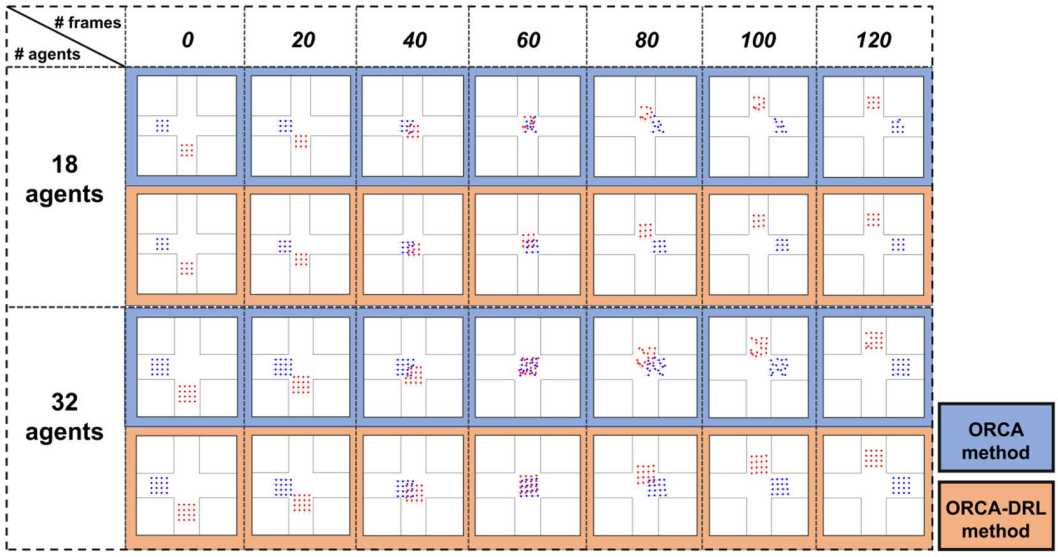


**FIGURE 8** Model performance in different frames in the crossing scene

## 5.2 | Trajectory analysis

In this section, we further analyze the smoothness of the trajectories of ORCA agents and ORCA-DRL agents in the three designed scenes. We also investigate the model generalization performance by testing the models with different numbers of agents.

### 5.2.1 | Circle scene

Figure 9 presents the trajectories of ORCA agents and ORCA-DRL agents in the circle scene in both the 8- and 16-agent cases. We observe that ORCA agents fail to reach their destinations as they stop moving in the center position (Figure 9a). As mentioned in Section 3.1, the size of $VO_B^A$ (VO region of A in relation to B) is inversely proportional to the distance between agent A and B. The further from A to B, the smaller $VO_B^A$, and the smaller the probability that A falls into $VO_B^A$, and vice versa. The trajectories of ORCA agents indicate that ORCA agents head straight to the center at the beginning because agents' velocities are outside the VO region of other agents, and heading straight to the destination is clearly the optimal solution at the current stage. However, by the time ORCA agents start to consider collisions, the intersection of all ORCA half-plane solution sets for those agents becomes zero, resulting in the ceasing of movement for all ORCA agents.

Different from the behaviors of ORCA agents, ORCA-DRL agents tend to shift their headings at the beginning of the simulation and eventually reach their final destination without colliding with other agents, leaving a center-symmetric overall trajectory (Figure 9b). ORCA-DRL agents, trained by the DNN via the PPO-Clip, gradually learned the optimal actions that need to be taken in the current state to maximize the reward. Therefore, after realizing that heading straight potentially results in an unsolvable dilemma, ORCA-DRL agents start to take alternative actions to increase the probability of positive rewards. The highly center-symmetric trajectories of ORCA-DRL agents are due to the similar state parameters passed to the DNN. Again, the success of both the 8- and 16-agent cases demonstrates that our ORCA-DRL method can be generalized to a more crowded case.

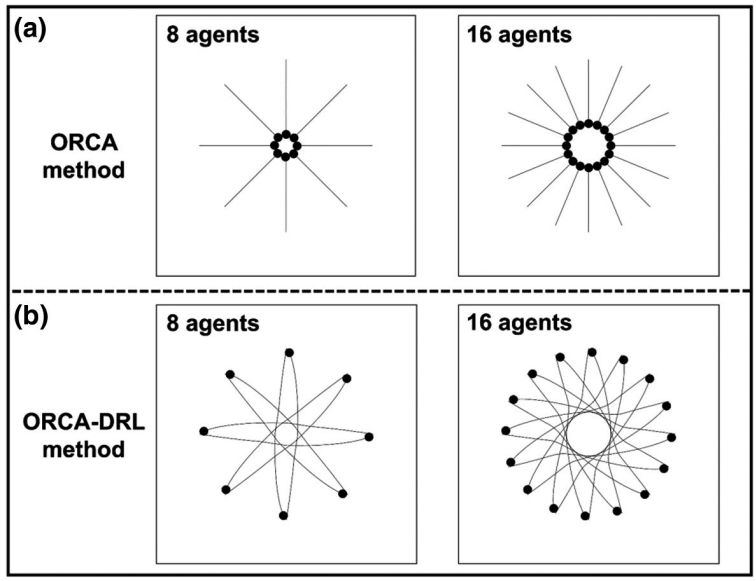


**FIGURE 9** Agents' trajectories in the circle scene

### 5.2.2 | Hallway scene

The hallway scene aims to investigate the scenario where two groups of agents move in opposition, avoid collision during the meeting of groups, and eventually reach the final destinations. The performance of the models can be determined via the trajectory pattern. The more chaotic the trajectory pattern, the poorer the model performance. Figure 10 presents the trajectories of ORCA agents (Figure 10a) and ORCA-DRL agents (Figure 10b) in both the 18- and 32-agent cases. In general, ORCA agents exhibit more chaotic movements, especially during the group meeting phase, evidenced by their intertwined trajectories. The chaotic trajectories also lead to significantly more time for ORCA agents to reach their destinations. ORCA-DRL agents, however, are able to maintain their relative formation during the group meeting phase, evidenced by their smooth and untwined trajectories. The smooth trajectories allow ORCA-DRL agents to easily regroup after the meeting and reach their destinations in a shorter time (with fewer frames). As we increase the number of agents from 18 to 32, ORCA-DRL agents present similar trajectories, suggesting a good generalization capability of the ORCA-DRL method. In comparison, we observe more chaotic behaviors of ORCA agents in the 32-agent case, indicating that the limitation of the ORCA method is exaggerated in a more crowded case. The trajectories of ORCA-DRL agents demonstrate that they are able to consider potential collisions earlier by conducting directional changes in advance, in contrast to the behavior of ORCA agents who consider potential collisions only when an agent's velocity falls into the VO region.

### 5.2.3 | Crossing scene

Different from the hallway scene where two groups move towards each other, the crossing scene investigates agents' behaviors when two groups move, merge, and disengage with their velocity perpendicular. Figure 11 presents the trajectories of ORCA agents (Figure 11a) and ORCA-DRL agents (Figure 11b) in both the 18- and 32-agent cases. It can be observed that both ORCA and ORCA-DRL agents adjust their positions to avoid collisions

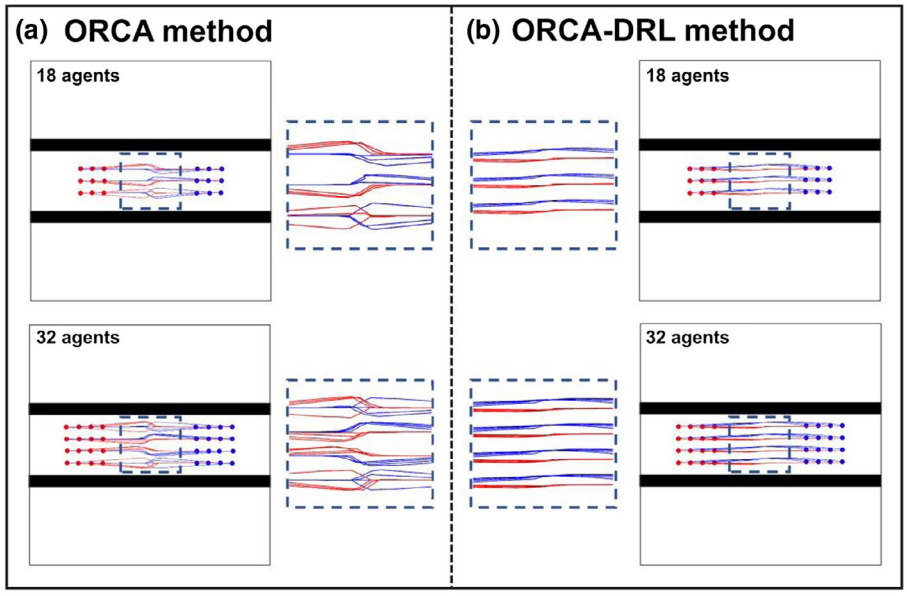


**FIGURE 10** Agents' trajectories in the hallway scene

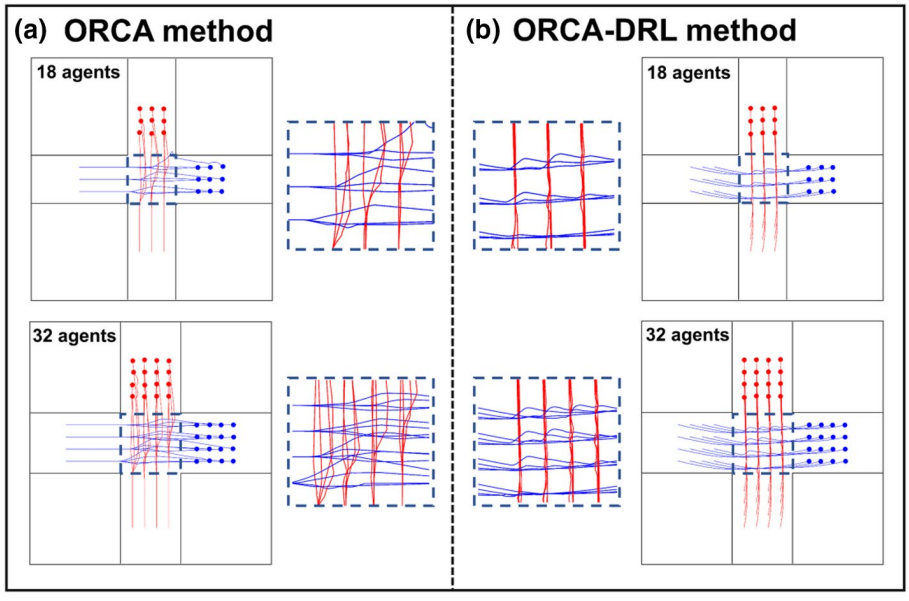


**FIGURE 11** Agents' trajectories in the crossing scene

with other surrounding agents. Similar to the results from the hallway scene, however, the trajectories from ORCA agents are more disorganized and intertwined (Figure 11a), suggesting more chaotic movements from ORCA agents during the group meeting phase compared with those from ORCA-DRL agents. As we increase the number of agents from 18 to 32, the movements from ORCA agents become more chaotic, evidenced by their messier trajectories. In comparison, ORCA-DRL agents achieve a consistent performance with smooth trajectories regardless of the increasing number of agents, demonstrating the great applicable potential of the ORCA-DRL method in more crowded cases.

## 6 | LIMITATIONS AND FUTURE DIRECTIONS

Although the ORCA-DRL method can generate a smoother overall trajectory than the ORCA method in local motion simulation, its limitations are worth mentioning.

Firstly, the ORCA-DRL method collects the external state using visual sensors. Therefore, the more the number of rays, the more detailed the external environment an agent can sense. However, as the number of rays increases, the computational complexity increases in geometric progression, leading to longer training time. To solve this problem, some DRL-based robot motion simulation methods utilize images collected by the robot in real time as input. Those images serve as the direct input of the external state to DNN for learning purposes (Wu et al., 2019). Other studies apply simultaneous localization and mapping, a commonly used space exploration method, to obtain the space information of the robot (Huang & Gupta, 2008; Temeltas & Kayak, 2008). The potential of using directly captured images as model input deserves further investigation.

Secondly, dynamically adapting to different scenes is rather difficult for ORCA-DRL, given its offline nature (Godoy et al., 2015). For large-scale motion planning studies, tasks are usually divided into global path planning and local collision avoidance (Guy et al., 2009). In path planning, scholars have explored the potential of subdividing the global motion planning into multiple local motion simulations. That is, the ultimate goal of each agent is segmented into many optimal target locations (Kallmann, 2014; Van Toll et al., 2012). Further research is needed to subdivide the overall environment into several specific scenes, allowing DNN parameters to be automatically switched, depending on where each agent is located.

Thirdly, although we assume that the environment is fully known, the state information is still restricted as only four continuous frames are fed to the DNN model. Recently, the rapid development of the recurrent neural network (Dung, Komeda, & Takagi, 2008; Heess, Hunt, Lillicrap, & Silver, 2015) and 3D convolution allow the spatiotemporal features of agents to be simulated (Hara, Kataoka, & Satoh, 2017; Xu, Das, & Saenko, 2017), providing a great opportunity to pursue fully observable Markov decision-making. In future studies, we plan to incorporate the aforementioned methods into our DRL-supported local motion simulation model.

Lastly, in this study, the DNN model is trained via the PPO-Clip, which has demonstrated great efficiency even with limited computational power. Other off-policy methods, including TD3, SAC, and DDPG, have become more popular recently. The potential of those methods in the proposed framework deserves further exploration.

## 7 | CONCLUSIONS

As one of the hot topics in multi-agent navigation, local motion simulation has attracted widespread attention in the past few decades due to its important role in many fields, including robotics, evacuation simulation, and virtual reality.

The traditional local simulation method, ORCA, can ensure collision avoidance of agents in the next frame but usually fails to generate reasonable behavior, evidenced by the chaotic global trajectories of simulated agents. In this article, we proposed a local motion simulation method integrating ORCA and DRL, referred to as the ORCA-DRL method, where local collision avoidance detection is achieved via ORCA and trajectory smoothing is achieved via DRL. We designed a DNN framework to facilitate state-to-action mapping. In our designed framework, a certain state is composed of both internal and external information, which can be detected by virtual visual sensors. The action space further includes two continuous spaces: speed and direction. PPO-Clip, based on an AC framework, is also applied to improve data utilization and speed up the training process.

We compared the performance of traditional ORCA and our proposed ORCA-DRL in three individual scenes: circle, hallway, and crossing. Compared with the traditional ORCA, the proposed ORCA-DRL reduces the total number of frames for agents to achieve their predefined destination. From a trajectory perspective, ORCA-DRL effectively avoids the local optimum problem by generating a smoother global trajectory compared with the

traditional ORCA method. Furthermore, the model revealed a great generalization performance, indicating that our proposed method could be used in other scenarios or in more crowded cases. Nevertheless, ORCA-DRL has several limitations such as computational complexity and temporal restrictions (four frames as input). Further research is needed to tackle those limitations and investigate model performances in larger and more complicated scenes.

## ORCID

*Xiao Huang* https://orcid.org/0000-0002-4323-382X
*Zhenlong Li* https://orcid.org/0000-0002-8938-5466

## REFERENCES

Alonso-Mora, J., Breitenmoser, A., Rufli, M., Beardsley, P., & Siegwart, R. (2013). Optimal reciprocal collision avoidance for multiple non-holonomic robots. In A. Martinoli (Ed.), *Distributed autonomous robotic systems* (Springer Tracts in Advanced Robotics, Vol. *83*, pp. 203–216). Berlin, Germany: Springer.

Bera, A., Kim, S., & Manocha, D. (2015). Efficient trajectory extraction and parameter learning for data-driven crowd simulation. In *Proceedings of the 41st Graphics Interface Conference*, Halifax, Nova Scotia, Canada (pp. 65–72). Toronto, Ontario, Canada: Canadian Information Processing Society.

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., & Torr, P. H. (2016). Fully-convolutional Siamese networks for object tracking. In G. Hua, & H. Jégou (Eds.), *Computer vision: ECCV 2016 Workshops*, Amsterdam, The Netherlands, October 8–10 and 15–16, 2016, Proceedings, Part II (pp. 850–865). Berlin, Germany: Springer.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *Openai gym*. arXiv preprint arXiv:1606.01540.

Cova, T. J., & Johnson, J. P. (2003). A network flow model for lane-based evacuation routing. *Transportation Research Part A: Policy & Practice*, *37*, 579–604.

Curtis, S., & Manocha, D. (2014). Pedestrian simulation using geometric reasoning in velocity space. In U. Weidmann, U. Kirsch, & M. Schreckenberg (Eds.), *Pedestrian and evacuation dynamics 2012* (pp. 875–890). Cham, Switzerland: Springer.

Dung, L. T., Komeda, T., & Takagi, M. (2008). Reinforcement learning for POMDP using state classification. *Applied Artificial Intelligence*, *22*, 761–779.

Dutra, T. B., Marques, R., Cavalcante-Neto, J. B., Vidal, C. A., & Pettré, J. (2017). Gradient-based steering for vision-based crowd simulation algorithms. *Computer Graphics Forum*, *36*(2), 337–348.

Fiorini, P., & Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, *17*, 760–772.

Friesz, T. L., Luque, J., Tobin, R. L., & Wie, B.-W. (1989). Dynamic network traffic assignment considered as a continuous time optimal control problem. *Operations Research*, *37*, 893–901.

Fujimoto, S., van Hoof, H., & Meger, D. (2018). *Addressing function approximation error in actorcritic methods*. arXiv preprint arXiv:1802.09477.

Godoy, J. E., Karamouzas, I., Guy, S. J., & Gini, M. (2015). Adaptive learning for multi-agent navigation. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, Istanbul, Turkey (pp. 1577–1585). Liverpool, UK: IFAAMAS.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., & Dubey, P. (2009). Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, New York, NY (pp. 177–187). New York, NY: ACM.

Guy, S. J., Lin, M. C., & Manocha, D. (2010). Modeling collision avoidance behavior for virtual humans. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, Toronto, Ontario, Canada (Vol. *2*, pp. 575–582). Liverpool, UK: IFAAMAS.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. arXiv preprint arXiv:1801.01290.

Han, Y., & Liu, H. (2017). Modified social force model based on information transmission toward crowd evacuation simulation. *Physica A: Statistical Mechanics & Its Applications*, *469*, 499–509.

Hara, K., Kataoka, H., & Satoh, Y. (2017). Learning spatio-temporal features with 3D residual networks for action recognition. In *Proceedings of the 2017 IEEE International Conference on Computer Vision*, Venice, Italy (pp. 3154–3160). Piscataway, NJ: IEEE.

Heess, N., Hunt, J. J., Lillicrap, T. P., & Silver, D. (2015). *Memory-based control with recurrent neural networks*. arXiv preprint arXiv:1512.04455.

Heess, N., Dhruva, T. B., Sriram, S., Lemmon, J., Merel, J., Wayne, G., …Riedmiller, M. (2017). *Emergence of locomotion behaviours in rich environments*. arXiv preprint arXiv:1707.02286.

Helbing, D., Farkas, I., & Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature*, *407*, 487–490.

Helbing, D., Farkas, I. J., Molnar, P., & Vicsek, T. (2002). Simulation of pedestrian crowds in normal and evacuation situations. *Pedestrian & Evacuation Dynamics*, *21*, 21–58.

Helbing, D., & Molnar, P. (1995). Social force model for pedestrian dynamics. *Physical Review E*, *51*, 4282–4286.

Helbing, D., Molnár, P., Farkas, I. J., & Bolay, K. (2001). Self-organizing pedestrian movement. *Environment & Planning B*, *28*, 361–383.

Huang, Y., & Gupta, K. (2008). RRT-SLAM for motion planning with motion and map uncertainty for robot exploration. In *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France (pp. 1077–1082). Piscataway, NJ: IEEE

Irpan, A. (2018). *Deep reinforcement learning doesn't work yet*. Retrieved from https://www.Alexirpan.com/2018/02/14/rl-hard.html

Kallmann, M. (2014). Dynamic and robust local clearance triangulations. *ACM Transactions on Graphics*, *33*, 161.

Karamouzas, I., & Overmars, M. (2011). Simulating and evaluating the local behavior of small pedestrian groups. *IEEE Transactions on Visualization & Computer Graphics*, *18*, 394–406.

Karamouzas, I., Sohre, N., Narain, R., & Guy, S. J. (2017). Implicit crowds: Optimization integrator for robust crowd simulation. *ACM Transactions on Graphics*, *36*, 1–13.

Kim, S., Bera, A., Best, A., Chabra, R., & Manocha, D. (2016). Interactive and adaptive data-driven crowd simulation. In *Proceedings of the 2016 IEEE Conference on Virtual Reality*, Greenville, SC (pp. 29–38). Piscataway, NJ: IEEE

Kirchner, A., Nishinari, K., & Schadschneider, A. (2003). Friction effects and clogging in a cellular automaton model for pedestrian dynamics. *Physical Review E*, *67*, 056122.

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In S. A. Solla, T. K. Leen, & T. K. Müller (Eds.), *Advances in neural information processing systems* (Vol. *12*, pp. 1008–1014). Cambridge, MA: MIT Press.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, Lake Tahoe, NV (Vol. *1*, pp. 1097–1105).

Lee, J., Won, J., & Lee, J. (2018). Crowd simulation by deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, Limassol, Cyprus (p. 2). New York, NY: ACM.

Lee, K. H., Choi, M. G., Hong, Q., & Lee, J. (2007). Group behavior from video: A data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Goslar, Germany (pp. 109–118). New York, NY: ACM.

Li, X., Claramunt, C., & Ray, C. (2010). A grid graph-based model for the analysis of 2D indoor spaces. *Computers Environment & Urban Systems*, *34*, 532–540.

Li, X., Li, Q., Xu, X., Xu, D., & Zhang, X. (2018). A novel approach to developing organized multi-speed evacuation plans. *Transactions in GIS*, *22*, 1205–1220.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., …Wierstra, D. (2015). *Continuous control with deep reinforcement learning*. arXiv preprint arXiv:1509.02971.

Long, P., Fanl, T., Liao, X., Liu, W., Zhang, H., & Pan, J. (2018). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation*, *Brisbane*, *Australia* (pp. 6252–6259). Piscataway, NJ: IEEE.

Lopes, G. C., Ferreira, M., da Silva Simões, A., & Colombini, E. L. (2018). Intelligent control of a quadrotor with proximal policy optimization reinforcement learning. In *Proceedings of the 2018 Latin American Robotic Symposium*, Joao Pessoa, Brazil (pp. 503–508). Piscataway, NJ: IEEE

Lozano-Perez, T. (1990). Spatial planning: A configuration space approach. In I. J. Cox, & G. T. Wilfong (Eds.), *Autonomous robot vehicles* (pp. 259–271). Berlin, Germany: Springer.

McLurkin, J., & Demaine, E. D. (2009). A distributed boundary detection algorithm for multi-robot systems. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO (pp. 4791–4798). Piscataway, NJ: IEEE.

Mehran, R., Oyama, A., & Shah, M. (2009). Abnormal crowd behavior detection using social force model. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami Beach, FL (pp. 935–942). Piscataway, NJ: IEEE

Merchant, D. K., & Nemhauser, G. L. (1978). A model and an algorithm for the dynamic traffic assignment problems. *Transportation Science*, *12*, 183–199.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., …, Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

Moussaïd, M., Perozo, N., Garnier, S., Helbing, D., & Theraulaz, G. (2010). The walking behavior of pedestrian social groups and its impact on crowd dynamics. *PLoS ONE*, *5*, e10047.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel (pp. 807–814). New York, NY: ACM.

Narain, R., Golas, A., Curtis, S., & Lin, M. C. (2009). Aggregate dynamics for dense crowd simulation. In *Proceedings of ACM SIGGRAPH Asia 2009*, Yokohama, Japan. New York, NY: ACM.

Ondřej, J., Pettré, J., Olivier, A.-H., & Donikian, S. (2010). A synthetic-vision based steering approach for crowd simulation. *ACM Transactions on Graphics*, *29*, 1–9.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., …Desmaison, A. (2019). PyTorch: An imperative style, high performance deep learning library. In *Proceedings of the 2019 Advances in Neural Information Processing Systems Conference*, Vancouver, BC, Canada (pp. 8024–8035). San Diego, CA: NIPS.

Peters, J., & Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, *71*, 1180–1190.

Pettré, J., Ciechomski, P. D. H., Maïm, J., Yersin, B., Laumond, J. P., & Thalmann, D. (2006). Real-time navigating crowds: Scalable simulation and rendering. *Computer Animation & Virtual Worlds*, *17*, 445–455.

Pidd, M., De Silva, F., & Eglese, R. (1996). A simulation model for emergency evacuation. *European Journal of Operational Research*, *90*, 413–419.

Prescott, T. J., & Mayhew, J. E. (1992). Obstacle avoidance through reinforcement learning. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Advances in neural information processing systems* (Vol. *4*, pp. 523–540). New York, NY: Morgan Kaufmann.

Ramezani Dooraki, A., & Lee, D.-J. (2018). An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments. *Sensors*, *18*, 3575.

Ravichandran, N. B., Yang, F., Peters, C., Lansner, A., & Herman, P. (2018). Pedestrian simulation as multi-objective reinforcement learning. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents*, Sydney, Australia (pp. 307–312). New York, NY: ACM.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, Anaheim, CA (pp. 25–34). New York, NY: ACM.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). *High-dimensional continuous control using generalized advantage estimation*. arXiv preprint arXiv:1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., … Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354–359.

Snape, J., Van den Berg, J., Guy, S. J., & Manocha, D. (2011). The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, *27*, 696–706.

Stadelmann, T., Amirian, M., Arabaci, I., Arnold, M., Duivesteijn, G. F., Elezi, I., … Tuggener, L. (2018). Deep learning in the wild. In *Proceedings of the Eighth IAPR TC3 Workshop on Artificial Neural Networks in Pattern Recognition*, Siena, Italy.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014).Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Proceedings of the 27th International Conference on Neural Information Processing Systems*, Cambridge, MA (Vol. *2*, pp. 3104–3112). Cambridge, MA: MIT Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000).Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, & K. Müller (Eds.), *Proceedings of the 12th International Conference on Neural Information Processing Systems*, Cambridge, MA (pp. 1057–1063). Cambridge, MA: MIT Press.

Temeltas, H., & Kayak, D. (2008). SLAM for robot navigation. *IEEE Aerospace & Electronic Systems Magazine*, *23*, 16–19.

Tieck, J. C. V., Pogančić, M. V., Kaiser, J., Roennau, A., Gewaltig, M. O., & Dillmann, R. (2018). Learning continuous muscle control for a multi-joint arm by extending proximal policy optimization with a liquid state machine. In *Proceedings of the 27th International Conference on Artificial Neural Networks*, Rhodes, Greece (pp. 211–221). Lausanne, Switzerland: ENNS.

Torrey, L. (2010). Crowd simulation via multi-agent reinforcement learning. In *Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*, Palo Alto, CA. Menlo Park, CA: AAAI.

Van den Berg, J., Guy, S. J., Lin, M., & Manocha, D. (2011). Reciprocal *n*-body collision avoidance. In C. Pradalier, R. Siegwart, & G. Hirzinger (Eds.), *Robotics research* (Springer tracts in advanced robotics, Vol. *70*, pp. 3–19). Berlin, Germany: Springer.

Van den Berg, J., Lin, M., & Manocha, D. (2008). Reciprocal velocity obstacles for real-time multiagent navigation. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA (pp. 1928–1935). Piscataway, NJ: IEEE.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, Phoenix, AZ. Menlo Park, CA: AAAI.

Van Toll, W. G., Cook, A. F. IV, & Geraerts, R. (2012). A navigation mesh for dynamic environments. *Computer Animation & Virtual Worlds*, *23*, 535–546.

Varas, A., Cornejo, M., Mainemer, D., Toledo, B., Rogan, J., Munoz, V., & Valdivia, J. (2007). Cellular automaton model for evacuation process with obstacles. *Physica A: Statistical Mechanics & Its Applications*, *382*, 631–642.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016). *Sample efficient actor-critic with experience replay*. arXiv preprint arXiv:1611.01224.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). *Dueling network architectures for deep reinforcement learning*. arXiv preprint arXiv:1511.06581.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Wu, K., Esfahani, M. A., Yuan, S., & Wang, H. (2019). Depth-based obstacle avoidance through deep reinforcement learning. In *Proceedings of the 5th International Conference on Mechatronics and Robotics Engineering*, Rome, Italy (pp. 102–106). New York, NY: ACM.

Xia, C., & El Kamel, A. (2015). A reinforcement learning method of obstacle avoidance for industrial mobile vehicles in unknown environments using neural network.In E. Qi, J. Shen, & R. Dou (Eds.), *Proceedings of the 21st International Conference on Industrial Engineering and Engineering Management*, Paris, France (pp. 671–675). Paris, France: Atlantis Press.

Xu, H., Das, A., & Saenko, K. (2017). ). R-c3d: Region convolutional 3D network for temporal activity detection. In *Proceedings of the IEEE International Conference on Computer Vision*, Venice, Italy (pp. 5783–5792). Piscataway, NJ: IEEE.

Xu, M., Hijazi, I., Mebarki, A., Meouche, R. E., & Abune'Meh, M. (2016). Indoor guided evacuation: Tin for graph generation and crowd evacuation. *Geomatics, Natural Hazards & Risk*, *7*, 1–10.

Yuan, W., & Tan, K. H. (2007). An evacuation model using cellular automata. *Physica A: Statistical Mechanics & Its Applications*, *384*, 549–566.

Zheng, X., Zhong, T., & Liu, M. (2009). Modeling crowd evacuation of a building based on seven methodological approaches. *Building & Environment*, *44*, 437–445.

Zheng, Y., Jia, B., Li, X.-G., & Zhu, N. (2011). Evacuation dynamics with fire spreading based on cellular automaton. *Physica A: Statistical Mechanics & Its Applications*, *390*, 3147–3156.

Zhong, J., Cai, W., Luo, L., & Yin, H. (2015). Learning behavior patterns from video: A data-driven framework for agent-based crowd modeling. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, Istanbul, Turkey (pp. 801–809). Liverpool, UK: IFAAMAS.

**How to cite this article:** Xu D, Huang X, Li Z, Li X. Local motion simulation using deep reinforcement learning. *Transactions in GIS*. 2020;00:1–24. https://doi.org/10.1111/tgis.12620