SCIENCE CHINA
Information Sciences

August 2024, Vol. 67, Iss. 8, 180203:1–180203:14
https://doi.org/10.1007/s11432-023-4029-9

• RESEARCH PAPER •
Special Topic: UAV Swarm Autonomous Control

Autonomous multi-drone racing method based on deep reinforcement learning

Yu KANG[1,2,3], Jian DI[2*], Ming LI[1], Yunbo ZHAO[1] & Yuhui WANG[4]

[1]*Department of Automation, University of Science and Technology of China, Hefei* 230026*, China;*
[2]*Institute of Advanced Technology, University of Science and Technology of China, Hefei* 230088*, China;*
[3]*Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei* 230088*, China;*
[4]*College of Automation Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016*, China*

Received 10 September 2023/Revised 14 January 2024/Accepted 18 March 2024/Published online 25 July 2024

Abstract Racing drones have attracted increasing attention due to their remarkable high speed and excellent maneuverability. However, autonomous multi-drone racing is quite difficult since it requires quick and agile flight in intricate surroundings and rich drone interaction. To address these issues, we propose a novel autonomous multi-drone racing method based on deep reinforcement learning. A new set of reward functions is proposed to make racing drones learn the racing skills of human experts. Unlike previous methods that required global information about tracks and track boundary constraints, the proposed method requires only limited localized track information within the range of its own onboard sensors. Further, the dynamic response characteristics of racing drones are incorporated into the training environment, so that the proposed method is more in line with the requirements of real drone racing scenarios. In addition, our method has a low computational cost and can meet the requirements of real-time racing. Finally, the effectiveness and superiority of the proposed method are verified by extensive comparison with the state-of-the-art methods in a series of simulations and real-world experiments.

Keywords racing drone, autonomous multi-drone racing, sim-to-real, deep reinforcement learning, Markov game

1 Introduction

The racing drone is a rapidly developing multi-rotor unmanned aerial vehicle with exceptional velocity and maneuvering capabilities [1–3]. With its flexibility to cope with complex situations, it has extensive application prospects not only in the civilian fields but also gives rise to a novel sport called drone racing. During drone racing, a skilled human pilot usually operates the drone to achieve agile maneuvering based on onboard first-view images. The sport has attracted the attention of both the academic and business communities. Since studying the autonomous flight of racing drones could solve the pain of low-speed maneuvering in existing mission scenarios, scholars began to study how to remove human players from the racing of drones and automate racing tasks by designing flight control algorithms [4–6]. In existing research on autonomous racing of the single drone, the focus is usually on the perception of the environment, as well as the generation and tracking of the optimal path. Ref. [7] provided a framework that combines a learning-based perception module with a traditional control method to achieve autonomous racing flight. Furthermore, Ref. [8] investigated how to utilize the reinforcement learning method to maximize the performance of the single drone during the racing competition.

Research on autonomous drone racing has enormous hurdles due to the requirements of drone racing for quick and agile flight in complex environments and the rich interactions between drones [9,10]. Human pilots likewise need years of training to develop these skills. As a result, much of the existing research has focused on how to enable racing drones to meet or exceed professional human pilots in single drone time trials. However, in actual drone racing competitions, where multiple racing drones operated by human pilots are involved in the racing at the same time, frequent interactions between pilots can lead to rapid

* Corresponding author (email: dijian@mail.ustc.edu.cn)

changes in the situation on the field. Pilots need to frequently adjust their strategies according to the situation on the field, such as blocking, overtaking, and sometimes even cooperating with other drones. Interactions between drones have a critical impact on the outcome of the racing [11]. Therefore, it is not enough for existing studies to consider only single drone racing.

Until now, research on multi-drone racing is still very limited [12–14]. Nevertheless, these methods require the preacquisition of global information on the track and many human constraints, which are not compatible with actual multi-drone racing. First, these methods need to obtain the position information of all the gates on the whole track in advance, and then fit the centerline of the track and the track boundary according to the prior information. However, in real-world racing, drones can only obtain limited localized track information within the range of their onboard sensors. Second, these methods have the additional restriction of requiring the drone not to exceed the track boundary. Nevertheless, this restriction does not exist in actual drone racing and could lead to more conservative behavior from the drones. Crucially, these methods require the drone to solve the optimization problem online, and the computational time consumed grows dramatically with the number of projected steps and participants. This high demand for computational power leads to the need for drones to carry heavy high-performance computing equipment or to slow down before the algorithm solves for a result, but these run counter to the goal of drone racing.

The key to resolving these issues is providing drones with autonomous capabilities against other opponents in dynamic environments and reducing their computational burden. Because machines utilizing reinforcement learning (RL) can perpetually improve their strategies by interacting with the environment, they are capable of learning [15–17]. Meanwhile, the reinforcement learning algorithm's strategy network can quickly generate action commands, thereby shortening the decision-making time of the drone and improving its maneuverability. Consequently, a deep reinforcement learning approach can be considered for determining the optimal strategy in multi-drone racing by learning continuously throughout the race. Several studies [18–20] have already demonstrated the potential of applying deep reinforcement learning to drone maneuvers. Rodriguez-Ramos et al. [18] integrated a deep deterministic policy gradient (DDPG) algorithm to enable drones to perform continuous landing tasks on mobile platforms. Song et al. [20] obtained the time optimal trajectory of a single drone timing race through a deep reinforcement learning algorithm.

However, to the best of our knowledge, no research work has yet applied deep reinforcement learning methods to autonomous multi-drone racing research. For multi-drone racing, there are still some urgent issues that need to be addressed in existing research: (1) Existing methods mostly study how a single drone performs certain tasks, ignoring the rich interactions between drones in racing, and thus are not directly transferable to non-smooth multi-drone racing environments. (2) Existing methods do not take into account the characteristics of real racing environments and the movement of drones but are limited to specific simulated game environments, resulting in training results that cannot be migrated to real racing environments.

To address the above issues, we model the multi-drone racing problem as a Markov game and propose an autonomous multi-drone racing method based on deep reinforcement learning. For drones to learn racing skills commonly used by human experts, we model multiple goals during drone racing into an entire set of reward functions, achieving rewards for actions such as reaching the finish line, crossing gates, and overtaking drones, as well as penalties for actions such as rear-ending and crossing boundaries. In order to facilitate the migration of the proposed method from the simulation environment to the real world, we incorporate the dynamic response characteristics of the drone during real flight into the training process and introduce the independent proximal policy optimization (IPPO) algorithm [21] for training.

The main contributions of this paper are as follows:

(1) We propose a novel autonomous multi-drone racing method based on deep reinforcement learning. Unlike existing methods, the proposed method fully considers the rich interactions among multiple racing drones and has a stronger generalization ability.

(2) The proposed method does not rely on the global information of the track and track boundary constraints and takes into account the dynamic characteristics of racing drones. Compared with existing methods, the proposed method is more in line with the requirements of real drone racing scenarios.

(3) Unlike existing methods that need to solve the optimization problem online, the proposed method has a lower computational cost, can meet the requirements of real-time racing, and is more adaptable to strongly dynamic environments.

This paper is structured as follows. Section 2 gives the modeling of the racing scene. Section 3 describes

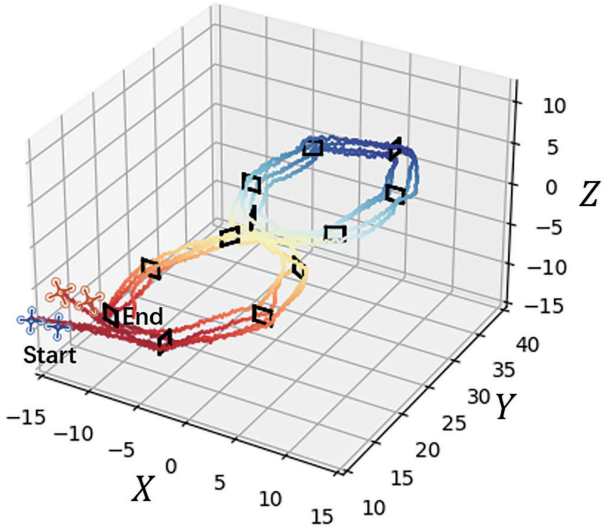


Figure 1 (Color online) Example of the three-dimensional track.

the proposed method in detail. Section 4 performs the related simulation experiments and result analysis. And finally, Section 5 summarizes the work of this paper.

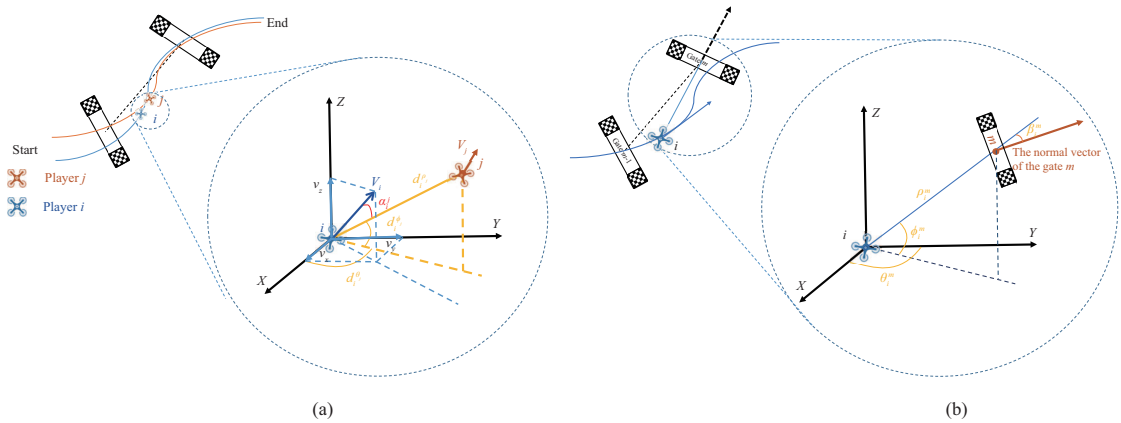
2 Racing scene modeling

As shown in Figure 1, the entire track consists of multiple gates in three-dimensional space with no specific physical boundaries. In a multi-drone racing competition, the first drone that sequentially traverses all the gates to reach the finish line wins. Let $p_i(t)$ denote the position of drone i at the moment of t. During the race, drones need to keep a safe distance from each other to avoid collision.

Observation spaces. The observation space, as described in Figure 2, consists mainly of two parts: the drone's observations of the relative positions of the other drones and the relative position of the current gate. For the former, we define the drone i's observation of the other drones as $o_i^{\text{others}} = [o_i^1, \alpha_i^1, \ldots, o_i^{i-1}, \alpha_i^{i-1}, o_i^{i+1}, \alpha_i^{i+1}, \ldots, o_i^n, \alpha_i^n]$, where n denotes the number of all drones and o_i^j denotes the position of drone j in the spherical coordinate system centered on the drone i: $o_i^j = (d_i^{\rho_j}, d_i^{\theta_j}, d_i^{\phi_j})$. Here α_i^j is used to denote the angle between the vector pointing from the drone i to the drone j and the flight direction vector of the drone i. Intuitively, when the two drones are relatively close together, the change in α gives a direct indication of whether the two drones are about to collide or move away from each other. For the latter, it is worth noting that we assume that the drone does not have global information about all the gates, so that at each moment the drone can only observe the current state of the gate $o_i^{\text{grid}_m} = [\rho_i^m, \theta_i^m, \phi_i^m, \beta_i^m]$, where $(\rho_i^m, \theta_i^m, \phi_i^m)$ denotes the position of the center of the current gate m in the drone's body-central spherical coordinate system. β_i^m denotes the angle between the normal vector of the gate m and the vector that points from the drone i to the center of the gate m, which in combination with the distance ρ_i^m in the spherical coordinate system can represent the degree of deviation of the drone's position relative to the center of the gate.

In the above observations, for the current drone, not every gate's information can be obtained, and the setting is obviously more reasonable. Intuitively, this setting is more in line with the actual scene of drones observing the gate by the onboard camera. At the same time, this setting allows the drone to train in scenarios with different numbers of gates, thereby improving the generalization ability of the strategy. As a result, the proposed method can be more conveniently transferred to real-world flight. It is worth noting that we have used a spherical coordinate system for all position observations, since in spherical coordinates the drone can directly obtain the distance relative to other drones or gates.

Action spaces. Following the foregoing observations, the drones output and execute action commands. The action space is specified as $V = [v_x, v_y, v_z]$, where v_x, v_y, v_z represent the velocity components of the drone in three directions in the drone-centered coordinate system. By putting the tanh activation function in the final layer and then scaling, we can regulate the final action command within a preset range.



(a) (b)

Figure 2 (Color online) Drone i's coordinate system. It is important to note that the above observations are based on drone i's own coordinate system as a reference. (a) The observation of drone i on drone j; (b) the observation of drone i on the current gate m.

3 Proposed method

In contrast to the approach proposed in [20], which formulates the problem as a Markov decision process (MDP), our approach involves modeling it as a Markov game with partial observability. This is necessary due to the inclusion of several drones that need to be taken into account [22]. A Markov game is characterized by N players and consists of a set of possible states and a set of all possible combinations of actions. The state transition function, denoted by $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_k \to \mathcal{S}$, allows for the determination of the state at the next moment, where $\mathcal{A}_i$ means the agent i's action space and $\mathcal{S}$ means the state space. Each agent i chooses an action according to the policy π_{θ_i} and then obtains the private reward from the function $r_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$. After the initial state is determined, the goal of each agent i is to maximize its excepted return:

$$R_i = \sum_{t=0}^{T} \gamma^t r_i^t, \quad r_i^t : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_k \to \mathbb{R}, \tag{1}$$

where γ represents the discount factor which is employed to strike a balance between long-term and short-term benefits. And T denotes the time horizon.

It is evident that the more suitable approach for representing the problem of multi-drone racing is to employ a Markov game framework. If we consider the problem as an MDP, where each agent's state observation does not include any information about the other agents, the state transition and reward functions of each agent are influenced by the actions of the other agents. This results in a non-stationary environment that poses challenges for traditional reinforcement learning algorithms.

3.1 Reward shaping

Intuitively, the total time taken by the drone to fly the entire track directly corresponds to the main goal of the task. Obviously, the rewards obtained for both achieving a successful gate crossing and completing the final goal are sparse, which greatly increases the difficulty for agents to perform policy searches in high-dimensional space. A popular solution to the long-term credit assignment problem in the current scenario is to design a reward that maximizes the approximation to the true goal while providing feedback at each time step of the agent. In car racing, a technique based on process projection on the center line of the track has shown good performance.

By linking neighboring gates with straight lines as the track centerline, we extend the projection-based path process reward in the drone racing problem. A track can be specified by a set of gates positioned in three-dimensional space, and no additional calculation is needed to create a smooth reference trajectory.

Position reward. The drone can be correlated with a particular line segment at any given time, contingent upon the subsequent gate it is poised to traverse. To calculate the path progress for one time step, we project the current position of the drone onto the above line segment and give two positions of the drone at adjacent time steps. We define the position reward function as follows:

$$r_{\text{pos}}(t) = s(p_i(t)) - s(p_i(t-1)), \tag{2}$$

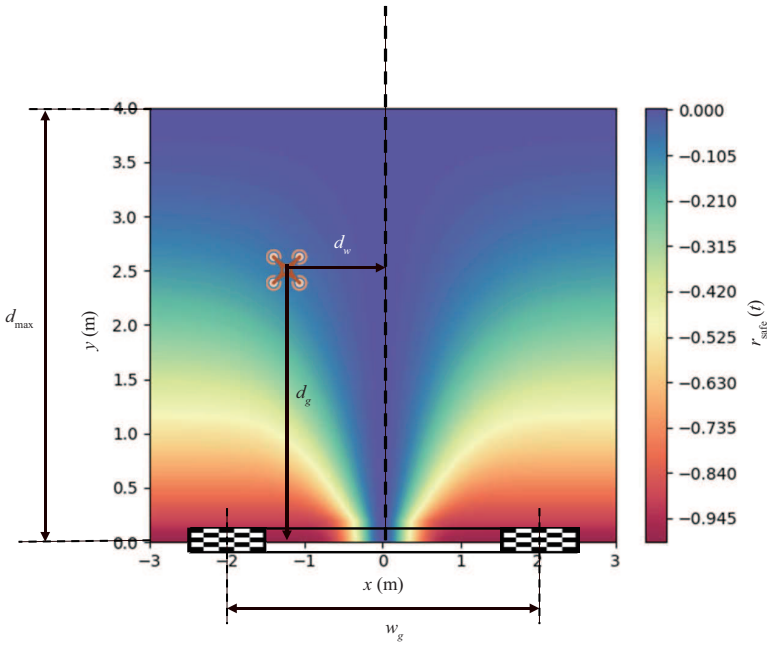


Figure 3 (Color online) Visualization of safety margin rewards.

where $s(p_i) = (p_i - p_m) \cdot (p_m - p_{m-1})/||(p_m - p_{m-1})||$ defines the distance between the projection point of the drone on the centerline of the adjacent gates and the center of the current gate. In the aforementioned equation, the variable p_m denotes the position of gate m and the variable p_i denotes the position of drone i. Therefore, when the value of r_{pos} is positive, it indicates that the drone is in the process of approaching the current gate. Conversely, when the value of r_{pos} is negative, it signifies that the drone is traveling away from the current gate.

Safety margin reward. To keep the drone from hitting the edge when crossing the gate, we define a safety margin reward to penalize the drones close to the gate edge to induce the drone to cross from closer to the center of the gate [20]. The safety margin reward is defined as follows:

$$\begin{aligned} r_{\text{safe}}(t) &= -f^2 \cdot \left(1 - \mathrm{e}^{(-\frac{0.3 \cdot d_w^2}{g})}\right) + \delta_r, \\ \delta_r &= \begin{cases} r_c, & \text{if gate collision}, \\ 0, & \text{else}, \end{cases} \end{aligned} \tag{3}$$

where $f = \max[1 - (d_g/d_{\max}), 0.0]$ and $g = \max[(1 - f) \cdot (w_g/d_{\max}), 0.05]$. d_w represents the distance of the drone to the normal vector of the gate and d_g defines the distance of the drone to the plane of the gate. The distance to the gate normal vector is normalized using the side length of the square gate w_g and $d_{\max}$ denotes the threshold of the vertical distance from the drone to the gate that triggers the safety margin reward, which is illustrated as shown in Figure 3. It is worth noting that the safety reward consists of two components, the former is a soft reward used to reduce the risk of impact. And r_c is the hard reward used to penalize the drone colliding with the gate, defined as follows:

$$r_c = -\min\left[\left(\frac{d_c}{w_g}\right)^2, 10.0\right], \tag{4}$$

where d_c defines the Euclidean distance from the collision point to the center of the gate.

Collision penalty. It is crucial to establish who is responsible for drone collisions when considering the multi-drone racing scenario, which involves a lot of drone contact. We employ collision avoidance mechanisms to steer the drones away from dangerous collisions in realistic scenarios, but we want the underlying safety mechanism to be activated less frequently during the race, allowing for smoother drone maneuvers and guiding the drones away from tailgating. So we introduce a tailgating penalty:

$$r_{\text{collision}} = -\mathrm{e}^{(-\frac{|d \cdot (|\beta| + \alpha) \cdot b|}{c})} - r_{\text{crash}}, \tag{5}$$

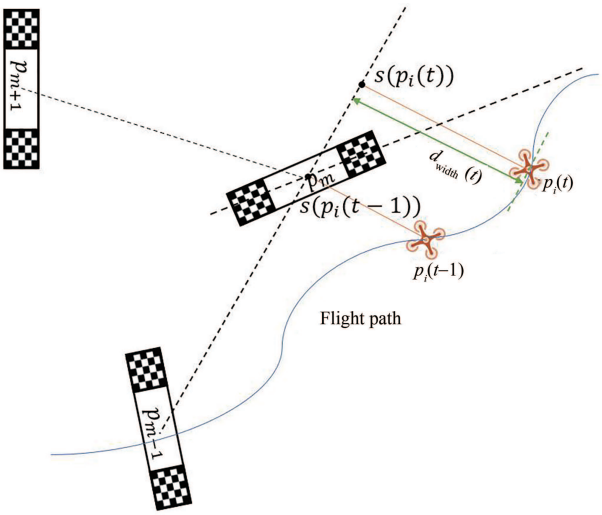


Figure 4 (Color online) Segment of the flight.

where c is the coefficient that determines the magnitude of the reward change, d represents the distance between the front and rear drones, and β denotes the angle between the line between the front and rear drones and the forward direction of the latter drone. When both d and β are less than the specified threshold, the collision reward is activated and the drone is guided to avoid tailgating. At the same time, to avoid malicious collisions with drones ahead, there is a collision penalty of a basic constant r_{crash}, with both sides suffering the same penalty.

Out of boundaries penalty. The experiment reveals that when the turning portion of our track, which lacks a physical barrier, exhibits sufficient curvature, the drones tend to converge towards a sub-optimal course during the training process. As shown in Figure 4, the path progress is positive on any two adjacent moments when the drone is flying along the current flight path, which means that the drone's r_{pos} is positive, but at this point the drone is gradually moving away from the track. So we introduce a out of boundaries penalty, which occurs when the drone is too far from the distance between the two gates' center lines $d_{\text{width}} > d_{\text{bound}}$, where α_d is the coefficient and d_{bound} is a constant that we set to control the range of the drone's exploration:

$$r_{\text{out}} = -\mathrm{e}^{\frac{d_{\text{width}}-d_{\text{bound}}}{\alpha_d}} + 1. \tag{6}$$

Overtaking reward. In order to incentivize drones to acquire the skill of overtaking, we provide a positive constant overtaking reward denoted by r_o. Additionally, to discourage drones from engaging in strategic behavior where they overtake and afterward decelerate intentionally to fall behind before overtaking again, we assign a comparable opposite incentive to being overtaken.

$$r_{\text{over}} = \begin{cases} +r_o, & \text{if overtake,} \\ -r_o, & \text{if be overtaken,} \\ 0, & \text{else.} \end{cases} \tag{7}$$

Total reward. Ultimately the final reward at each time step can be defined as

$$\begin{aligned} r(t) = {} & r_{\text{pos}}(t) + \lambda_1 \cdot r_{\text{safe}}(t) - \lambda_2 \cdot \|\omega_t\|^2 \\ & + \lambda_3 \cdot r_{\text{collision}}(t) + \lambda_4 \cdot r_{\text{out}} + \lambda_5 \cdot r_{\text{over}} + r_{\text{cross}}, \end{aligned} \tag{8}$$

where a penalty on the quadratic term of the angular velocity ω_t is added to avoid larger oscillations of the drone. r_{cross} is the reward for the drone crossing the gate, and there will be a positive reward for successfully crossing the gate and a corresponding penalty for missing it. The multiple components of the reward are analyzed in detail in the subsequent experimental section.

$$r_{\text{cross}} = \begin{cases} r_0, & \text{if through the gate,} \\ -r_1, & \text{if miss,} \\ 0, & \text{else.} \end{cases} \tag{9}$$

3.2 Policy training

For the Markov game scenario, the multi-agent deep deterministic policy gradient (MADDPG) algorithm proposed in [23] adopts the framework of centralized training and decentralized execution. However, perhaps due to the inherent limitations of the DDPG algorithm, the above method performs poorly in the current problem. We choose to train the drone using the IPPO algorithm [24], which is a straightforward extension of the PPO algorithm [25] in a multi-agent environment. Compared to other reinforcement learning algorithms, the PPO algorithm is popular in robotics because it is insensitive to reward shaping, has easily tunable hyperparameters, and performs very well.

In a multi-drone racing scenario, the RL algorithms face a great challenge: the high-dimensional policy search space, the action complexity, and the non-stationary environment caused by the change of agents' policy. Also, deep RL algorithms are prone to overfitting for simulated environments, which is a huge challenge for sim-to-real transfer. Here, we highlight several tricks that allow us to obtain rapid and stable training results that transfer well to realistic scenarios.

3.2.1 *PPO's training*

The PPO is an approach for solving trust region optimization problems, as shown in the following equations:

$$\max_{\theta} \ \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t{}^{\pi_{\text{old}}} \right] \tag{10}$$

$$\text{subject to} \ \ \mathbb{E}_t[\text{KL}(\pi_\theta(a_t|s_t) \| \pi_{\theta_{\text{old}}}(a_t|s_t))] \leqslant \delta, \tag{11}$$

where π_θ is a stochastic policy, θ denotes the vector of policy parameters, θ_{old} represents the vector of policy parameters before the update, a_t is the action, and s_t denotes the state. $A_t{}^{\pi_{\text{old}}}$ represents an estimator of the advantage function at time step t. $\mathbb{E}_t[\cdots]$ is the empirical average over a finite batch of samples. $\text{KL}(\cdot)$ is the Kullback-Leibler divergence.

The primary objective is to maintain reasonable proximity between the updated policy and its predecessor while incorporating the principles of monotonic improvement theory to prevent any degradation in policy performance with each iteration of the algorithm. Compared to other trust region optimization algorithms, the latter does not require the calculation of Kullback-Leibler divergence which greatly improves computational efficiency and works better in the original paper, so we use the latter for training and optimize the objective function as follows:

$$\mathbb{J}^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta) A_t{}^{\pi_{\text{old}}}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t{}^{\pi_{\text{old}}})], \tag{12}$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$. During the training process, we use the trick from [14] to perform a normalization operation on the advantage function. We use z-score to apply the generalized advantage estimation (GAE) normalization on a batch of data, which can greatly improve the performance of the algorithm.

3.2.2 *Parallel sampling scheme*

The sampling efficiency of reinforcement learning algorithms is still a very critical issue, and very large amounts of data are required for the training of the algorithms. DeepMind uses hundreds of GPUs for months to train AlphaGo [26] and trains thousands of agents simultaneously in Starcraft to get AlphaStar [27], where the amount of data required for each agent to learn is equivalent to hundreds of years of continuous human play. In general, the amount of data needed to train stronger agents is sufficient, so how to improve the sampling efficiency of agents in a simulation environment is a critical issue. We enable agents to perform policy searches in 50 parallel environments simultaneously, significantly speeding up data collection. In addition, parallelization can be used to increase the diversity of the data collected. When there is a large randomness in the generation of tracks, agents can be allowed to explore multiple different tracks at the same time. When the tracks are long, parallelization allows agents to obtain data for a segment of a track from multiple environments at each iteration of the policy, rather than only for a specific track. This avoids overfitting for a particular track and greatly improves the performance of agents.

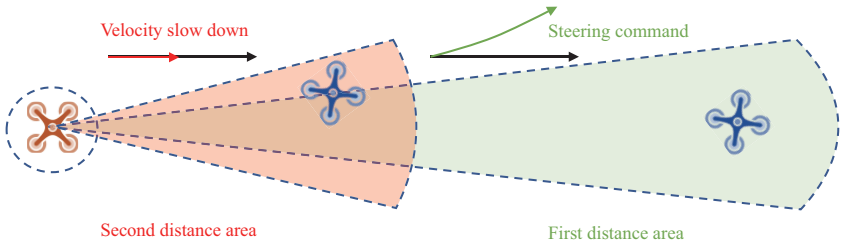


Figure 5 (Color online) Two-stage collision avoidance mechanism.

3.2.3 *Random initialization*

In order to avoid overfitting the reinforcement learning training drones to the simulated environment and to increase the diversity of the training data, we design multiple shapes of tracks consisting of different numbers of gates, with each episode initialized with a randomly chosen track. At the same time, we increase the uncertainty of the gate positions. After each initialization of the environment, the gates in the track change to some extent, and this change is reflected in the position and direction $[\Delta x, \Delta y, \Delta z, \Delta\alpha]$, where $\Delta x, \Delta y, \Delta z$ represent the change in the position of the gate, and $\Delta\alpha$ represents the change in the direction of the gate. The initial position of each drone is randomly generated within a spherical region of the fixed size of the assigned point, avoiding that a particular drone is always disadvantaged at the time of departure.

If the opponent starts winning early on, the drone will fail to learn the policy due to not receiving a positive reward, and at the same time the drone's opponent will end up with a sub-optimal strategy due to the drone's weakness. The multiple random initializations described above are performed to avoid overfitting for fixed tracks and initial positions, and to increase the randomness of policy exploration, thereby improving the generalization and stability of the model.

3.2.4 *Sim-to-real*

When using reinforcement learning algorithms, if we directly let agents explore in a realistic environment, many problems arise, such as very long training time and dangerous actions that occur during exploration. Although these problems can be avoided by using a simulated environment to avoid the high costs associated with dangerous actions, unfortunately, due to the reality gap, in most cases, agents trained in a simulated environment are not adapted to the real world. Therefore, before we migrate and deploy the strategy from simulation to reality, we need to address this issue in some way. Here, we simulate the observation noise of real-world drones by adding Gaussian noise to the drone's observations in the simulated environment, the magnitude of which can be adjusted according to the flight data in the real world. In realistic scenarios, drones have response time lags and do not respond to small or high-frequency oscillating commands. In order to make the dynamic response characteristics of the drone in the simulated environment closer to the realistic scenario, we analyze the real-world drone flight data and build its dynamic response model. During the training process, the drones' policy network outputs action commands, which are passed into this model to obtain the final execution action.

3.2.5 *Collision avoidance*

Although a large penalty is imposed when the drone collides with other drones during the training process, this soft mechanism cannot completely avoid collisions between drones. There is a need to introduce a low-level collision avoidance mechanism that can allow drones to make more aggressive decisions with safety guarantees, such as learning to take full advantage of each other's collision avoidance behavior to block an opponent's drone and keep themselves ahead of the game. In this paper, a two-stage collision avoidance mechanism is used. When another drone appears within the first distance area of the drone, the current drone's velocity is first deflected, and when another drone appears within the second distance area of the drone, the current drone is controlled to slow down, thus achieving collision avoidance. As shown in Figure 5, the first and second distance areas are represented by green and red, respectively.

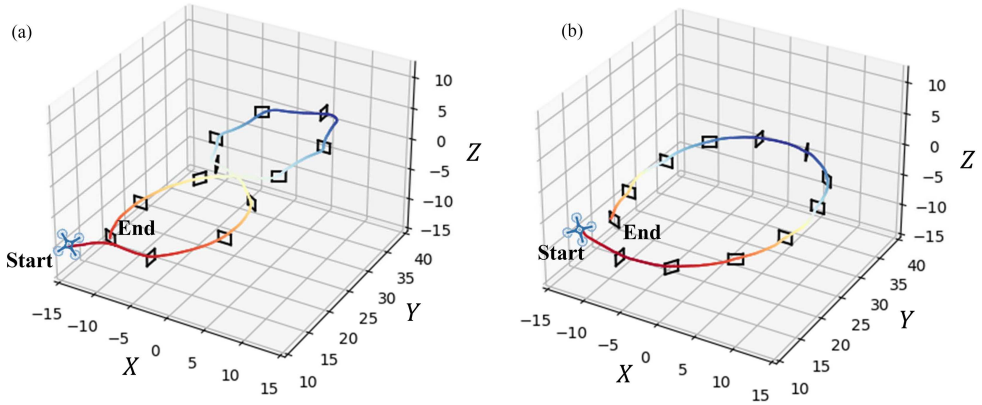


Figure 6 (Color online) Different tracks used in the simulation. (a) Track 1; (b) track 2.

4 Experiments and results

To verify the effectiveness and superiority of the proposed method, we conduct simulations and practical experiments, respectively. First, a simulator is constructed and a vectorized environment is employed in order to enhance sampling efficiency during training. This enables drones to collect data in parallel across multiple environments. We conduct autonomous racing flights of multi-drone in a simulated track, with tracks from the Microsoft Airsim [28]. Furthermore, a physical flight system is constructed to facilitate outdoor testing.

The following experimental findings will be used to explain the following research questions: (i) the reasons for our final choice of the IPPO algorithm; (ii) the role of multiple components in the complex reward function; (iii) the advantages of our algorithm over existing algorithms; (iv) the performance of our algorithm in the real world.

4.1 Simulation study

4.1.1 *Simulation setup*

The performance of our proposed method is evaluated on two tracks (as shown in Figure 6) that have several sharp turns, creating a complex game scenario for multi-drone racing. These scenarios allow for various opportunities for drones to engage in overtaking maneuvers and blocking strategies. During our training, the track for each episode is generated by combining several of these tracks' segments. This approach is adopted to enhance the algorithm's generalization capabilities and mitigate the risk of overfitting.

During the experiments, the minimum safe distance between drones with a radius of 0.2 m is 0.5 m. The single-step decision time is 0.05 s, and the maximum flight speed of the drones is 2 m/s. The network parameters in the algorithms of our work are updated based on Adam with a learning rate of $2.4e^{-4}$. It needs to collect data for 2048 steps from the environment before each policy iteration, which uses 8 epochs to optimize the surrogate loss function, and the minibatch size is 256. The entropy coefficient for the loss calculation is 0.014 and the discount factor $\gamma = 0.99$.

4.1.2 *Performance comparison of reinforcement learning algorithms*

We compare the IPPO algorithm with the multi-agent proximal policy optimization (MAPPO) algorithm [24], where both algorithms set the same hyperparameters. In contrast to the MAPPO method, the approach described here does not rely on the utilization of agent-specific global states. Instead, it involves the concatenation of the local observable information from all agents. The MAPPO algorithm, based on the PPO algorithm, uses centralized training and decentralized execution to overcome the issue of a non-stationary environment during the training of multi-drone racing games. It is a very traditional and sophisticated approach in the field of multi-agent reinforcement learning. Drones from the same group will naturally collaborate better in conflicts when there is group competition since they have the advantage of being able to incorporate information from all teammates throughout training. In this paper, we compare the two algorithms' performance in a racing scenario of individual warfare, i.e., without

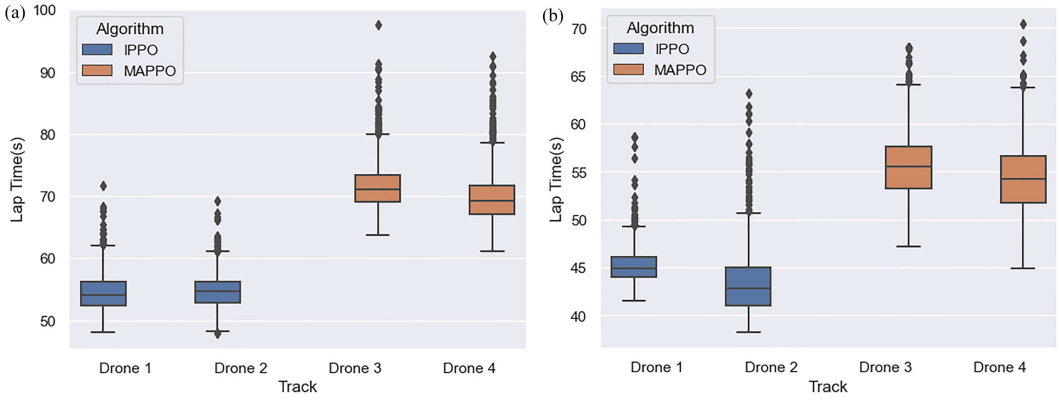


Figure 7 (Color online) Results of 1000 individual races with 4 drones on each of the two tracks. (a) 1000 races on Track 1; (b) 1000 races on track 2.

teams.

To evaluate the two algorithms, we race the IPPO-based drones against the MAPPO-based drones 1000 times on each of the two tracks to obtain the box plot of lap time as shown in Figure 7. There are four drones participating in the individual race and there is no group between the drones, two of which are based on the MAPPO algorithm and the other two based on the IPPO algorithm. From the results of the competition, we can clearly see that IPPO-based drones have a more significant advantage. The reason for this result in our analysis is that the MAPPO algorithm, when trained, needs to input information about the state-action pairs of all the agents into the value function network, which leads to an increasing input dimension of the network when the number of agents increases, making the learning of the value function more difficult. In summary, we choose the IPPO algorithm for this problem.

4.1.3 *Ablation experiments*

To examine the effects of the various parts of our proposed approach, such as the incentive components, we conduct ablation experiments. This subsection provides a detailed introduction to the learning curves and lap time box plots of all ablation experiments. Half of the mean evaluation's standard deviation under 10 separate random seed trials is shown by the graph's shaded area. For convenience of viewing, the learning curves are uniformly smoothed using a sliding average with a sliding window width of 13 time steps. It should be noted that the average lap time in the training results differs from the average lap time in the evaluation because the track for each episode is chosen randomly during training.

Safety reward. In order to prevent too many collisions with the gate edge during training, we design the safety reward to encourage the drone to travel closer to the center of the gate. Obviously, the drone can potentially venture to overtake or maintain its lead position by traversing from closer to the gate edge, especially when the gate is on the track corners. Therefore, the weight and size setting of the safety reward will influence the decision of whether the drone adopts the risk. We design comparison tests to allow us to clearly compare the training performance of drones in the setting with and without the safety reward.

The safety margins of the drone from the gate edge during each traversal are depicted in Figure 8. It is clear that having a safety incentive raises the drone's safety margins when navigating the gate and lessens the likelihood that it will strike the gate edge. For the sake of comparison, we do not terminate the competition after the collision occurred in this statistic. As a result, we can see from the lap time graph in Figure 9, that the drone becomes overly cautious as a result of the safety reward, which results in a certain amount of performance loss. However, in order to ensure the safety of drones, we actually need to minimize or even completely avoid collisions, so setting up a safety award is indispensable.

Overtaking reward. To enable the drone to learn the skills of overtaking and blocking, we design the overtaking reward to encourage the drone to overtake by giving a positive reward while the negative reward is set to encourage the drone to learn to block the rear drone. As shown in Figure 10, it can be seen that in the absence of an overtaking reward the drone performs better during training, having a lower average lap time. However, after placing the two drones on each of the two tracks for 1000 races, it is clear that the drone lacking the overtaking reward performs weakly during the race. Our analysis of the above results is that the drones are less aggressive during training when there is no overtaking

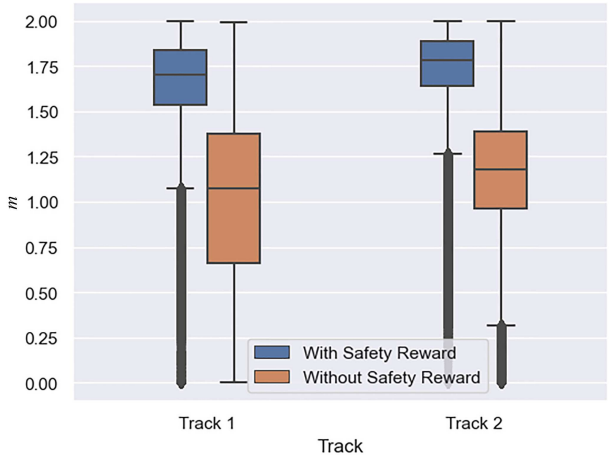


Figure 8 (Color online) Safety margins to the gate edge in 1000 races.

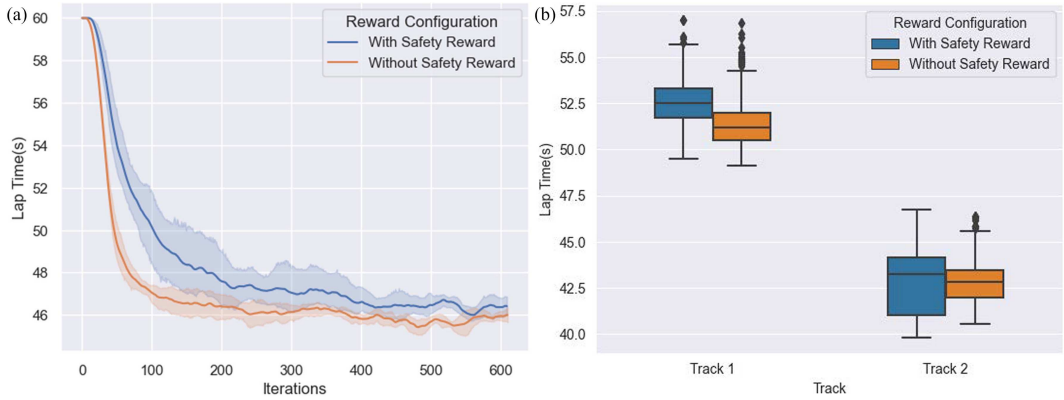


Figure 9 (Color online) Safety margins reward ablation experiments. (a) Training curves; (b) lap time of 1000 races.

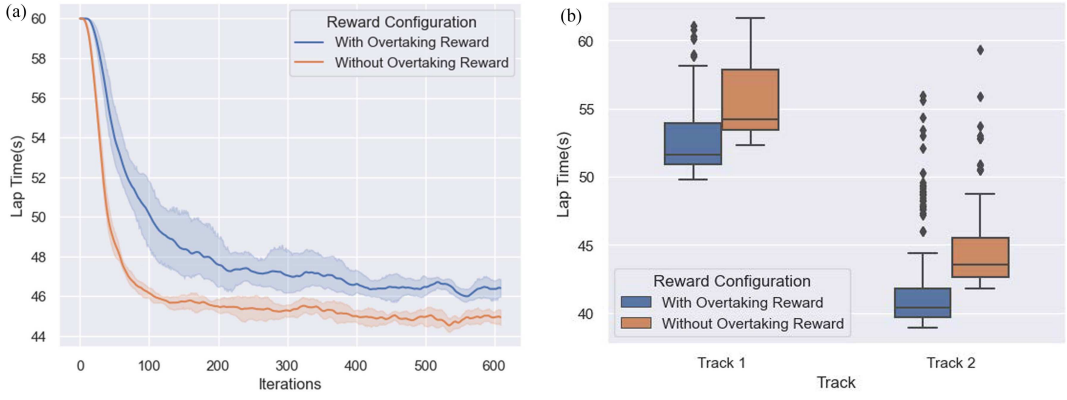


Figure 10 (Color online) Overtaking reward ablation experiments. (a) Training curves; (b) lap time of 1000 races.

reward and eventually the drones find a peaceful and orderly approach which results in shorter average lap times for everyone, but this is contrary to our original intention of multi-drone racing.

Out of boundaries penalty. A track consisting of multiple gates is without a specific boundary, leading to an excessive range of exploration, so we design a virtual boundary and introduce an out of boundaries penalty to avoid some ineffective exploration. We design controlled trials to evaluate the effect of this penalty on the training performance of the drone. From the lap time graph and race results in Figure 11, it is easy to see that after discarding the out of boundaries penalty, the variance of the drone's learning curve becomes larger, and it is at a significant disadvantage on lap time.

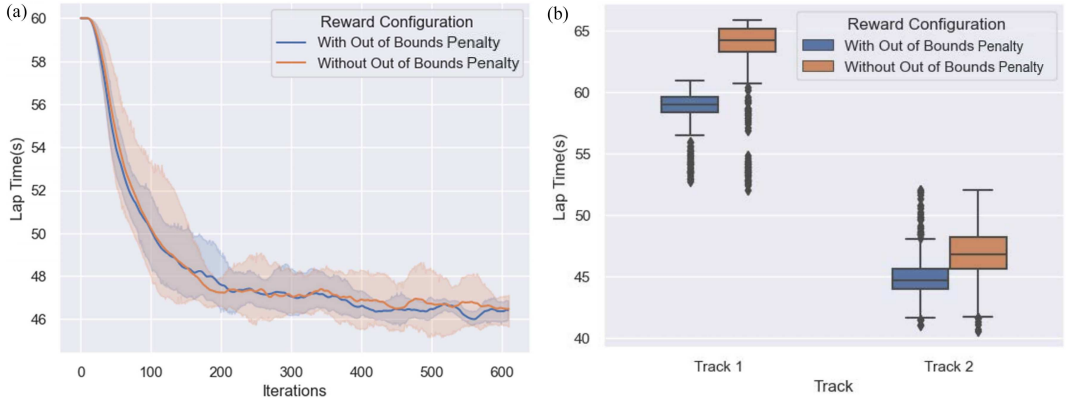


Figure 11 (Color online) Out of boundaries penalty ablation experiments. (a) Training curves; (b) lap time of 1000 races.

Table 1 1 vs. 1 individual racing results

Algorithm	Single step calculation time (s)	Track 1 lap time (s)	Track 2 lap time (s)
Our method	0.002230 ± 0.000416	52.378 ± 0.565	42.133 ± 0.785
GTP	1.211 ± 0.137	59.016 ± 2.988	46.376 ± 5.136

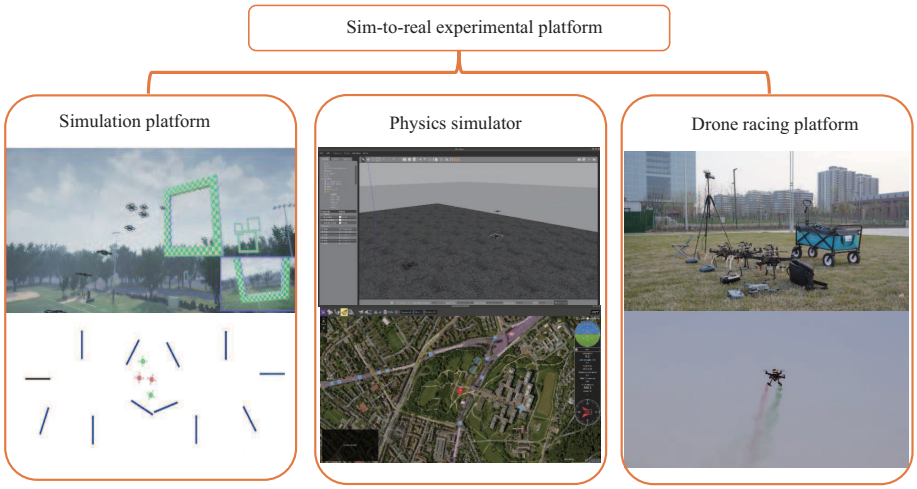


Figure 12 (Color online) Sim-to-real experimental multi-drone racing platform diagram.

4.1.4 *Performance comparison with other methods*

In order to further validate the performance of the proposed method, we conduct a comparative analysis between the proposed method and the game theory planner (GTP) algorithm [13] under the premise of track determination. The GTP algorithm collects data pertaining to all the gates and iteratively calculates an approximate Nash equilibrium solution within the track space. Due to the large arithmetic power required by this algorithm, it needs to use multiple central processing units in parallel.

The comparison results of the two algorithms are shown in Table 1. It can be found that the GTP algorithm not only requires the global information of multiple gates to be known but also the computation time of a single step is very large. Compared to the GTP algorithm, the single step computation time of our algorithm is in the order of milliseconds, while that of the GTP algorithm is in the order of seconds. Further, the performance of the algorithm proposed in this paper is also superior in terms of average lap time.

4.2 Experiments in the real world

4.2.1 *Sim-to-real experimental setup*

To verify the performance of the proposed method after migrating from the simulation environment to the real world, we build the sim-to-real drone racing platform as shown in Figure 12. The simulation platform

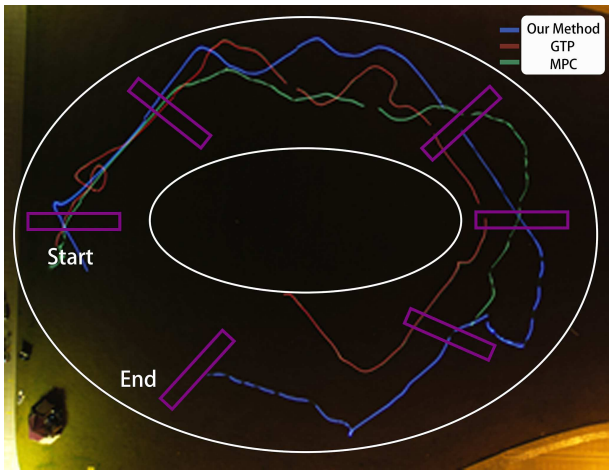


Figure 13 (Color online) Outdoor multi-drone racing experiment.

Table 2 Time of completing the track in an outdoor racing competition

Algorithm	Our method	GTP	MPC
Time (s)	57.771	61.874	67.236

is used for the training of our proposed method. Considering the differences between the simulated environment and the real world, we introduced the sim-to-real method designed in Subsection 3.2.4 in the training stage. And we also build a physics simulator. Compared to the "simulation platform", the physics simulator considers more physical system characteristics and can simulate more underlying electromechanical characteristics of the drone. At the same time, they can simulate possible factors such as wind and collisions in the environment. Therefore, the physics simulator can pre-test the performance of our proposed method, and at the same time test the accuracy of the information subscribed by the drone, providing an additional round of detection for the subsequent algorithm test in the real world, improving the test efficiency and safety. The racing drone platform is composed of racing drones, a ground terminal, and a ZGET Homer graph data transmission communication system. Among them, the racing drone is assembled by CUAV V5+ autopilot, data transmission module, GPS, remote controller, onboard computer, and power module. The ground terminal uses Dell Video Cassette G15, i7-12700H processor, and RTX3060 graphics card.

4.2.2 *Outdoor racing*

Based on the built sim-to-real drone racing platform, we conduct outdoor multi-drone racing experiments to verify the effectiveness and generalization of the proposed method. The experimental site is an outdoor lawn of 30 m $\times$ 20 m. Due to the site, the gates used in the racing experiment are virtual gates. The three drones participating in the racing experiment are respectively driven by the GTP method, model predictive control (MPC) method, and the proposed method. The experimental results are shown in Figure 13. It is obvious from the results that the proposed method still has good performance in the real world. For the sake of comparison, we have calculated the time from the starting point to the destination for each drone. As shown in Table 2, it is evident that the proposed method controls the drone with the shortest time consumption, which intuitively corresponds to the fact that it first reaches the finish line. Compared with GTP and MPC methods that require global track information, the proposed method that only requires local information has better racing performance.

5 Conclusion

In this paper, we propose a novel autonomous multi-drone racing method based on deep reinforcement learning. We model multiple goals during drone racing into an entire set of reward functions, achieving rewards for actions such as reaching the finish line, crossing gates, and overtaking drones, as well as penalties for actions such as rear-ending and crossing boundaries, in order for drones to learn racing skills

commonly used by human experts. We incorporate the dynamic response characteristics of the drone during real flight into the training process and introduce the IPPO algorithm for training to facilitate the migration of the proposed method from the simulation environment to the real world. Different from the state-of-the-art method, our method not only does not depend on global information about track and track boundary constraints but also has a lower computational burden and stronger environmental adaptability. Finally, a series of simulations and real-world experiments are carried out to demonstrate the superiority of our method.

Acknowledgements This work was supported in part by National Key Research and Development Program of China (Grant No. 2018AAA0100801) and National Natural Science Foundation of China (Grant No. 62033012).

References

1 Foehn P, Brescianini D, Kaufmann E, et al. AlphaPilot: autonomous drone racing. Auton Robot, 2022, 46: 307–320
2 Song Q H, Zeng Y, Xu J, et al. A survey of prototype and experiment for UAV communications. Sci China Inf Sci, 2021, 64: 140301
3 Hu J W, Wang M, Zhao C H, et al. Formation control and collision avoidance for multi-UAV systems based on Voronoi partition. Sci China Tech Sci, 2020, 63: 65–72
4 Rojas-Perez L O, Martinez-Carranza J. On-board processing for autonomous drone racing: an overview. Integration, 2021, 80: 46–59
5 de Wagter C, Paredes-Vallés F, Sheth N, et al. Learning fast in autonomous drone racing. Nat Mach Intell, 2021, 3: 923
6 Pfeiffer C, Wengeler S, Loquercio A, et al. Visual attention prediction improves performance of autonomous drone racing agents. Plos one, 2022, 17: e0264471
7 Loquercio A, Kaufmann E, Ranftl R, et al. Deep drone racing: from simulation to reality with domain randomization. IEEE Trans Robot, 2019, 36: 1–14
8 Song Y, Romero A, Müller M, et al. Reaching the limit in autonomous racing: optimal control versus reinforcement learning. Sci Robot, 2023, 8: eadg1462
9 Han Z, Wang Z, Pan N, et al. Fast-racing: an open-source strong baseline for SE(3) planning in autonomous drone racing. IEEE Robot Autom Lett, 2021, 6: 8631–8638
10 Hanover D, Loquercio A, Bauersfeld L, et al. Autonomous drone racing: a survey. IEEE Trans Robot, 2024, 40: 3044–3067
11 Di J, Chen S, Li P, et al. A cooperative-competitive strategy for autonomous multidrone racing. IEEE Trans Ind Electron, 2024, 71: 7488–7497
12 Spica R, Cristofalo E, Wang Z, et al. A real-time game theoretic planner for autonomous two-player drone racing. IEEE Trans Robot, 2020, 36: 1389–1403
13 Wang Z, Taubner T, Schwager M. Multi-agent sensitivity enhanced iterative best response: a real-time game theoretic planner for drone racing in 3D environments. Robotics Autonomous Syst, 2020, 125: 103410
14 Wang Z, Spica R, Schwager M. Game theoretic motion planning for multi-robot racing. In: Proceedings of the 14th International Symposium on Distributed Autonomous Robotic Systems, 2019. 225–238
15 Xu Y, Wu Z-G, Che W-W, et al. Reinforcement learning-based unknown reference tracking control of HMASs with nonidentical communication delays. Sci China Inf Sci, 2023, 66: 170203
16 Zhu J, Wei Y T, Kang Y, et al. Adaptive deep reinforcement learning for non-stationary environments. Sci China Inf Sci, 2022, 65: 202204
17 Tang W X, Li B, Li W X, et al. Reinforcement learning of non-additive joint steganographic embedding costs with attention mechanism. Sci China Inf Sci, 2023, 66: 132305
18 Rodriguez-Ramos A, Sampedro C, Bavle H, et al. A deep reinforcement learning strategy for UAV autonomous landing on a moving platform. J Intell Robot Syst, 2019, 93: 351–366
19 Ates U. Long-term planning with deep reinforcement learning on autonomous drones. In: Proceedings of Innovations in Intelligent Systems and Applications Conference (ASYU), 2020. 1–6
20 Song Y, Steinweg M, Kaufmann E, et al. Autonomous drone racing with deep reinforcement learning. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021. 1205–1212
21 de Witt C S, Gupta T, Makoviichuk D, et al. Is independent learning all you need in the starcraft multi-agent challenge? 2020. ArXiv:2011.09533
22 Littman M L. Markov games as a framework for multi-agent reinforcement learning. In: Proceedings of the 11th International Conference on International Conference on Machine Learning, 1994. 157–163
23 Lowe R, Wu Y I, Tamar A, et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In: Proceedings of Advances in Neural Information Processing Systems, 2017
24 Yu C, Velu A, Vinitsky E, et al. The surprising effectiveness of PPO in cooperative multi-agent games. In: Proceedings of Advances in Neural Information Processing Systems, 2022. 35: 24611–24624
25 Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. 2017. ArXiv:1707.06347
26 Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529: 484–489
27 Vinyals O, Babuschkin I, Czarnecki W M, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 2019, 575: 350–354
28 Madaan R, Gyde N, Vemprala S, et al. Airsim drone racing lab. In: Proceedings of NeurIPS 2019 Competition and Demonstration Track, 2020. 123: 177–191