Article

Reinforcement Learning-Based Formation Pinning and Shape Transformation for Swarms

Zhaoqi Dong [1,†], Qizhen Wu [2,†] and Lei Chen [1,3,*]

[1] Advanced Research Institute of Multidisciplinary Sciences, Beijing Institute of Technology, Beijing 100081, China; dongzhaoqi@bit.edu.cn
[2] School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China; wuqzh7@buaa.edu.cn
[3] Yangtze Delta Region Academy of Beijing Institute of Technology, Jiaxing 314000, China
* Correspondence: bit_chen@bit.edu.cn
† These authors contributed equally to this work.

Abstract: Swarm models hold significant importance as they provide the collective behavior of self-organized systems. Boids model is a fundamental framework for studying emergent behavior in swarms systems. It addresses problems related to simulating the emergent behavior of autonomous agents, such as alignment, cohesion, and repulsion, to imitate natural flocking movements. However, traditional models of Boids often lack pinning and the adaptability to quickly adapt to the dynamic environment. To address this limitation, we introduce reinforcement learning into the framework of Boids to solve the problem of disorder and the lack of pinning. The aim of this approach is to enable drone swarms to quickly and effectively adapt to dynamic external environments. We propose a method based on the Q-learning network to improve the cohesion and repulsion parameters in the Boids model to achieve continuous obstacle avoidance and maximize spatial coverage in the simulation scenario. Additionally, we introduce a virtual leader to provide pinning and coordination stability, reflecting the leadership and coordination seen in drone swarms. To validate the effectiveness of this method, we demonstrate the model's capabilities through empirical experiments with drone swarms, and show the practicality of the RL-Boids framework.

Keywords: reinforcement learning; Boids model; virtual leader; obstacle avoidance; drone swarms

check for updates

Citation: Dong, Z.; Wu, Q.; Chen, L. Reinforcement Learning-Based Formation Pinning and Shape Transformation for Swarms. *Drones* **2023**, *7*, 673. https://doi.org/10.3390/drones7110673

Academic Editor: Oleg Yakimenko

Received: 9 October 2023
Revised: 6 November 2023
Accepted: 8 November 2023
Published: 13 November 2023

1. Introduction

In recent years, swarm intelligence emerges as a promising approach to solve complex problems by harnessing the collective behavior of multiple autonomous agents [1–5]. Swarms consist of many relatively simple individuals interacting with each other and their environment to achieve self-organized behavior and accomplish tasks collectively [6,7]. Formation adjustment in response to dynamic task requirements or environmental conditions is crucial for maintaining efficiency and adaptability. Based on the Boids model [8], Vicsek investigates the conditions for reaching an agreement on the direction of motion of individuals in a swarm from a statistical mechanics perspective. He also proposes a model that simulates and explains the swarming, transport, and phase transition in a non-equilibrium system (Vicsek model) and finally implements a swarm of 30 UAVs in an outdoor environment flight [9]. Reference [10] proposes a predictive model that describes swarming based on the Boids model, which merges the local principles of the potential field model into an objective function and optimizes these principles with knowledge of the dynamics and environment of the agents.

One of the critical challenges in swarm systems is to optimize their size dynamically, based on the environmental conditions and task requirements. Traditional approaches to swarm transformation often rely on predefined rules or centralized control, limiting their flexibility and adaptability. Moreover, reinforcement learning demonstrates its power as an approach for training agents to make intelligent decisions in complex and dynamic environments.

Reinforcement learning offers a promising avenue for addressing the challenge by enabling swarm agents to learn and adapt their behaviors autonomously. Through interactions with the environment, agents can acquire knowledge and refine their decision-making processes based on feedback received as rewards or penalties. This iterative learning process empowers the agents to navigate intricate scenarios, optimize their swarm size, and efficiently fulfill given tasks.

There are three types in the academic community of reinforcement learning algorithms based on dynamic programming: (1) Value iteration. Ref. [11] adopts Q-learning networks to implement behavioral decision making in robots to improve the analysis and prediction capabilities of agents. However, Q-learning networks can not solve the problem of discrete actions in continuous states. Therefore, Ref. [12] proposes a framework for adaptive formation control of multiple robots based on Deep Q-learning networks; (2) Policy iteration. Policy iteration can solve the problem of continuous states and continuous actions. Ref. [13] uses a robust policy gradient algorithm to optimize a fully decentralized sensor-level collision avoidance policy; (3) The Actor–Critic algorithm. Due to the use of the Monte Carlo method, the gradient estimation variance of the algorithm is large. Therefore, the following reinforcement learning algorithm emerged. Ref. [14] develops a MAPPO-based distributed formation and obstacle avoidance approach in which agents use only their local and relative information to make motion decisions. Moreover, Ref. [15] uses a two-stage training scheme of imitation learning and reinforcement learning to propose a fused reward function to guide the agents. One fascinating application of RL is of simulating collective behavior, where groups of autonomous agents interact to achieve coordinated objectives [16–18].

In addition to reinforcement learning, we also introduce a virtual leader. The multi-robot control often uses the virtual leader approach due to its high robustness [19–21]. Olfati-Saber [22] has significantly contributed to advancing scalable flocking algorithms via a comprehensive theoretical and computational framework. Within this framework, there are three distinct flocking algorithms. The first algorithm adheres to the three fundamental rules outlined by Reynolds, while the third algorithm incorporates obstacle avoidance capabilities. Central to this framework is the second algorithm, designed as the primary flocking mechanism for agents navigating open space. In this second algorithm, the objective is to ensure that the agent group seamlessly follows the trajectory of a virtual leader. Therefore, a sophisticated navigational feedback mechanism is integrated into each agent. Crucially, it is presupposed that every agent within the group possesses information about the virtual leader, thereby qualifying as an informed agent. This assumption is pivotal, ensuring the entire group maintains cohesiveness and converges to the same velocity. The agent responds to the virtual leader like its actual neighbors. Its purpose is to introduce the task of directing, gathering, or manipulating the behavior of agents.

Therefore, this paper proposes a method to select the cohesive and repulsive parameters in the Boids model based on the Q-learning network. Our prescribed states, actions, and the development of special environmental interaction rules can filter the optimal combination of cohesion and repulsion parameters. By changing the parameters of cohesion and repulsion in the algorithm, we can achieve a simulation scenario with continuous obstacle avoidance and maximum coverage of space. At the same time, we introduce virtual leaders to facilitate stable coordination among intelligent agents and trigger the expansion and contraction of formations [23,24].

To tackle the problem, we also conduct experiments using meta-heuristics approaches such as genetic algorithms [25], ant colony optimization [26], and particle swarm optimization [27]. They achieve similar results with RL in the experimental results. However, they cannot handle the issue of the dynamic environment, and its time complexity is high due to its need for a large number of iterations. In some real-time planning scenarios, the environment could change rapidly. Although RL is computationally expensive in training (known as learning offline), once the policy is well trained, the decisions can be made quickly (known as planning online).

Experimental results demonstrate that the RL-Boids method exhibits good performance. The utilization of reinforcement learning offers a promising avenue for allowing Boids to adapt autonomously and make intelligent decisions. By allowing Boids to learn and optimize their behaviors via interactions with the environment, we achieve more robust and flexible collective behaviors.

The rest of this paper is organized as follows. In Section 2, we describe the Boids model and formulate the problem. In Section 3, we introduce the basic principles of the Q-learning network. In Section 4, we outline simulation scenarios, train parameters with Q-learning, and present the simulation results. In Section 5, we apply the algorithm in real robot experiments and confirm its feasibility with three drones. Finally, in Section 6, we draw some conclusions and prospects.

2. Model Description and Problem Formulation

2.1. Creation of Boids Model

To facilitate easy referencing, Table 1 provides the definitions for the essential symbols.

Table 1. Key Symbol Definitions.

Symbol	Definitions
$i, j \in \{1, \cdots, n\}$	The indexes of drones
$\boldsymbol{p_i}$	The position of drone i
$\boldsymbol{v_i}$	The velocity of drone i
$\boldsymbol{u_i}$	The control input of drone i
d_{ij}	The distance between drones i and j
$G = (V, E)$	The undirected perceptual graph
$V = \{1, \cdots, n\}$	The set of vertices
$E \subseteq V \times V$	The set of edges
N_i	The neighboring drones of drone i
k	The proportional coefficient
d_{com}	The communication distance among the drones
d_0	The boundary distance where repulsion and cohesion
k_{coh}	The cohesive parameter
k_{rep}	The repulsive parameter
w_{ij}	The weight associated with the communication link between drone i and drone j
D_{vl}	The desired direction towards the virtual leader
R_n	The repulsion from a neighboring drone
s, s'	The state
a	The action
r	The reward
S	The state space
γ	The discount factor
$\pi(a\|s)$	The probability of taking action a in state s under a probabilistic policy π
$P(s'\|s, a)$	The probability of taking action a from state s and transitioning to state s'

In this paper, we consider a swarm of n agents labelled as $i, j \in \{1, \cdots, n\}$. The position, velocity, and control inputs of the i-th agent are denoted by $\boldsymbol{p_i}, \boldsymbol{v_i}, \boldsymbol{u_i} \in \mathbb{R}^3$, respectively. Let $d_{ij} = \|\boldsymbol{p_j} - \boldsymbol{p_i}\|$ denote the central distance between agent i and agent j, where $\|\cdot\|$ denotes the Euclidean norm. We model the swarm with an undirected perceptual graph $G = (V, E)$, where the set of vertices $V = \{1, \cdots, n\}$ represents the agents and the set of edges $E \subseteq V \times V$ indicates that the agents $(i, j) \in E$ can communicate with each other. $N_i = \{j \in V \mid (i, j) \in E\}$ denotes the neighbor set G of the agents i, and $|\cdot|$ denotes the cardinality of the set.

We define the set of neighbors using the topological range, that is, the set N_i contains the neighbors $|N_i|$ in the vicinity of the agent i. This choice conveniently keeps the cardinality of the set of neighbors constant, and the above definition holds for biological swarms [28].

In the simulation, the dynamics of the agents are reproduced in discrete time. Let $\boldsymbol{p_i(k)}, \boldsymbol{v_i(k)}, \boldsymbol{u_i(k)} \in \mathbb{R}^3$ be the position, velocity, and control inputs of the *i-th* agent at the moment $k = \lfloor t/T \rfloor$, respectively. We can simplify the model appropriately based on the rules of repulsion and cohesion. At this point, the agents in the swarm can be forced to converge to different points and produce the expansion and contraction of the formation by altering the cohesion and repulsion algorithmic parameters. Set the communication distance among the agents as d_{com} and the boundary distance where repulsion and cohesion occur as $d_0 (d_{com} > d_0)$. Set the cohesive parameter as k_{coh} and the repulsive parameter as k_{rep}. When the distance $d_0(k) < d_{ij}(k) < d_{com}(k)$, cohesion occurs between agent i and neighbor j. The effect of neighbor j on the velocity of agent i can be expressed as

$$\boldsymbol{v_{ij}(k)} = \frac{k_{coh}\left(d_0(k) - d_{ij}(k)\right)}{d_{ij}(k)} \boldsymbol{p_j(k)} - \boldsymbol{p_i(k)} \tag{1}$$

Repulsion occurs between agent i and neighbor j when the distance $d_{ij}(k) < d_0(k)$. The effect of neighbor j on the velocity of agent i can be expressed as

$$\boldsymbol{v_{ij}(k)} = \frac{k_{rep}\left(d_{ij}(k) - d_0(k)\right)}{d_{ij}(k)} \boldsymbol{p_i(k)} - \boldsymbol{p_j(k)} \tag{2}$$

When the distance $d_{ij}(k) > d_{com}(k)$, there is no communication between agent i and neighbor j. The control input $\boldsymbol{u_i(k)}$ for any agent i can be denoted as

$$\boldsymbol{u_i(k)} = \sum_{j \in N_i} \boldsymbol{v_{ij}(k)} \tag{3}$$

The position $p_i(k+1)$ of any agent i at the next moment can be denoted as

$$\boldsymbol{p_i(k+1)} = \boldsymbol{p_i(k)} + \boldsymbol{u(k)} \tag{4}$$

2.2. Virtual Leader-Based Pinning Algorithm

At the same time, we employ a virtual leader-based clustering pinning algorithm, which plays a crucial role in achieving coordinated behavior among multiple drones. The virtual leader is positioned within the space and migrates along a predetermined trajectory line. The virtual leader serves as a reference point for the cluster, enabling seamless coordination and synchronization among individual units. This migration of the virtual leader serves a dual purpose: it provides both forward pinning and restriction for the formation of the swarm simultaneously (as depicted in Figure 1). This dynamic interaction with the virtual leader facilitates on-the-fly decision making for drones, enhancing their ability to navigate and perform tasks effectively.

Moreover, the pinning algorithm not only calculates the optimal position of the virtual leader, but it also dynamically adapts the positions of the entire agents. This algorithm considers multiple factors, including environmental conditions, obstacles, and desired objectives, to ensure the efficient operation and adaptability of the cluster in response to changing circumstances. By continuously computing and adjusting these positions, the algorithm optimizes the overall performance of the cluster, thereby enhancing its ability to accomplish complex tasks. This dynamic approach enables the swarm to navigate challenging environments and effectively respond to evolving situations, ultimately improving its overall effectiveness and efficiency.

- Initialization: Set the initial positions and velocities of all drones in the swarm, ensuring a suitable distribution across the desired workspace and define the desired trajectory or path for the swarm to follow, considering any specific objectives or constraints. We also need to determine the parameters for communication and coordination among the drones, such as the range and frequency of wireless communication and the method for exchanging information between drones. These parameters facilitate practical cooperation and synchronization within the swarm.
- Virtual Leader Update: Assuming that there are N drones forming a swarm, where the position of each drone is represented by (x_i, y_i) and the position of the virtual leader is represented by (x_v, y_v). The traction algorithm can calculate the position of the virtual leader using the following formula [29]:

$$x_v = (1/N)\sum(x_i) \tag{5}$$

$$y_v = (1/N)\sum(y_i) \tag{6}$$

  The above formula calculates the position of the virtual leader by taking a weighted average of the coordinates of all drones. By continuously updating the position of the virtual leader, other drones can adjust their behavior based on the motion of the virtual leader, enabling coordinated movement within the drones.
- Communication and Coordination: We need an information exchange that allows drones to adjust their trajectory and align with virtual leaders. One commonly used algorithm is the Distributed Average Consensus algorithm. This algorithm enables the drones to converge towards a typical trajectory by iteratively updating their own trajectory based on the information received from neighbors.
  Each drone maintains a local estimate of the desired trajectory, denoted as $x_i(k)$, where i represents the index of the drone and k denotes the iteration step. The update equation for each drone's local estimate can be expressed as:

$$x_i(k+1) = \sum_{j \in N_i} \boldsymbol{w_{ij}(x_j(k)) - (x_i(k))} \tag{7}$$

  Here, N_i represents the set of neighboring drones of drone i, w_{ij} represents the weight associated with the communication link between drone i and drone j, and the term $(x_j(k) - x_i(k))$ represents the difference between the trajectory estimates of drone j and drone i.
  The weights w_{ij} can be determined based on different criteria, such as distance, connectivity strength, or predefined weights. Common weight assignment strategies include uniform weights, distance-based weights, or dynamically adjusted weights.
  The consensus algorithm iteratively updates each drone's local estimate by considering the differences between its estimate and the estimates of its neighbors. This iterative process allows the drones to converge towards a common trajectory.
- Trajectory Adjustment: Let P_d be the position of the drone and P_{vl} be the position of the virtual leader. The desired direction towards the virtual leader is given by:

$$D_{vl} = \frac{P_{vl} - P_d}{\|P_{vl} - P_d\|} \tag{8}$$

  Considering the influence of neighboring drones, where each drone tries to avoid collisions while coordinating its movement, let P_n be the position of a neighboring drone. The repulsion from a neighboring drone is:

$$R_n = \frac{P_d - P_n}{\|P_d - P_n\|^2} \tag{9}$$

  and the total repulsion from all neighboring drones is:

$$R = \sum R_n \tag{10}$$

The overall direction can be calculated by combining the desired direction towards the virtual leader and the repulsion from neighboring drones. This can be a weighted sum depending on the importance of aligning with the virtual leader versus avoiding collisions.

$$D = w_1 \times D_{vl} + w_2 \times R \tag{11}$$

where w_1 and w_2 are weights that can be adjusted based on requirements.

By following this algorithm, the traction drones can move in a coordinated manner, effectively pulling or moving objects as a team. The virtual leader acts as a central point of reference, guiding the group towards the desired trajectory.

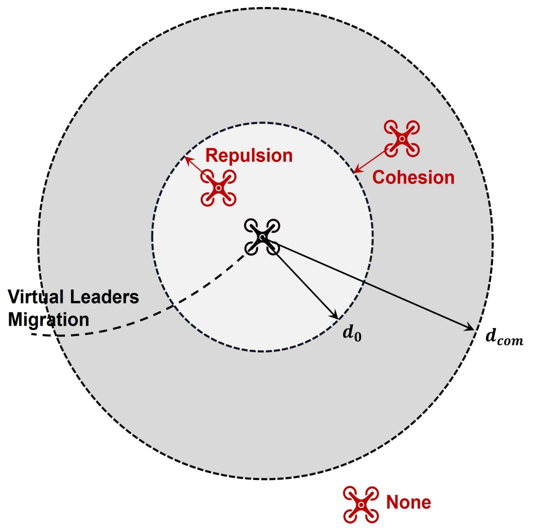


Figure 1. Diagram of the cohesion and repulsion control model.

3. Value-Based Reinforcement Learning Methods

In general, the process of reinforcement learning involves two objects: the environment and the agents. It also contains four basic elements, which are states, actions, rewards, and state transfer functions. To build a model for the swarm system and train it using Q-learning, we need to define the state space, action space, and reward function:

- State Space: The state space represents the current state of the swarm system, which includes relevant information about the environment and the swarm's configuration. It can include parameters such as the positions and velocities of individual drones, the position and velocity of the virtual leader, the distances between drones, and any other relevant variables.
- Action Space: The action space represents the available actions that the swarm system can take at each state. In this case, the action space consists of different combinations of the cohesion parameter k_{coh} and repulsion parameter k_{rep}. Each combination represents a potential configuration for the swarm system.
- Reward Function: The reward function quantifies the desirability or quality of a particular state–action pair. It provides feedback to the swarm system on the goodness or badness of its actions.

Using Q-learning, the swarm system can learn to select the optimal cohesion and repulsion parameters based on the current state and learn from the feedback via reward signals. Q-learning is a reinforcement learning algorithm that uses a Q-table to store and update the expected rewards for each state–action pair. Through an iterative process of exploration and exploitation, the swarm system learns the optimal policy that maximizes the cumulative rewards over time.

To evaluate the performance of the proposed method, extensive simulation experiments can be conducted. The swarm system can be trained using Q-learning with various combinations of parameters and reward functions. The performance metrics, such as the coverage efficiency and collision avoidance rate, can be measured and compared across different parameter settings. By analyzing the results, the optimal value of k can be determined, which maximizes the performance of the swarm system in achieving its objectives.

In reinforcement learning, the interaction among the agents and the environment can be summarized as follows: (1) The environment generates the state $s \in S$ of the environment at the current moment and passes it to the agents, of which S represents the state space. The state s describes the features of the current environment and contains the relevant information to support the decision making by the agents; (2) the agents generate corresponding actions $a \in A$ based on the state s and their own decision-making strategy $\pi(a|s)$, where A denotes the action space of the agents. Based on the states and actions, the environment generates the state of the environment at the next moment s' according to their own state transfer function $P(s'|s,a)$. At the same time, the agents will receive a reward signal r back from the environment, reflecting the gain obtained from the agents' decisions. This complete interaction process is known as a Markov Decision Process (MDP), and its characteristic is reflected in the fact that the state s' is determined by the state and action of the previous moment only, independent of the earlier state action. The goal of reinforcement learning is to maximize the cumulative rewards of the interaction between the agents and the environment [30,31]

$$\max \sum_{t=0}^{\infty} \gamma^t r_t \tag{12}$$

where γ is the discount factor. To achieve the goal, it is necessary to define the state-value function $V(s)$ to evaluate the goodness of the current state, which is expressed as the future expected cumulative reward from the current state s under strategy π

$$V(s) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s\right] \tag{13}$$

The larger the expectation value, the more advantageous the current state is for completing the task. Similarly, considering the action a performed in the current state according to policy π, the above state-value function can be written in the form of a state-action-value function as follows

$$Q(s,a) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s, a\right] \tag{14}$$

The value function-based reinforcement learning approach estimates values (12) and (13) based on interaction experience. Decisions are made based on the value function estimates. For example, based on obtaining an action-value function estimate $Q(s,a)$, the ϵ-greedy algorithm can be adapted for obtaining an implicit strategy for selecting the action with the largest action-value function

$$\pi(a|s) = arg\max_a Q(s,a) \tag{15}$$

To obtain an estimate of the value function, the state-value function can be written in the following form

$$V(s) = E_\pi[r + \gamma V(s')|s] \tag{16}$$

Correspondingly, the state-action-value function can be written as

$$Q(s,a) = E_\pi[r + \gamma Q(s',a')|s,a] \tag{17}$$

The above equation is known as the Bellman equation and expresses the relationship between the value of the state (state-action) and the subsequent state (state-action). Assuming that there exists an optimal estimate of the value function $V^*(s)$, then the Bellman equation in the form of the state-value function can be written as the Bellman optimal equation

$$V(s) = \max_{a \in A}\{E_\pi[r + \gamma V(s')|s]\} \tag{18}$$

Similarly, the expression for the Bellman optimality equation in the form of a state-action-value function is

$$Q(s,a) = E_\pi\left[r + \gamma \max_{a \in A} Q^*(s',a')|s,a\right] \tag{19}$$

The optimal strategy can be solved based on (15), which is expressed as follows

$$\pi^*(a|s) = arg\max_a Q^*(s,a) \tag{20}$$

To obtain the optimal value function estimate, the idea of dynamic programming is usually used. The value function of the previous step is estimated from the value function of the current step, thus transforming the Bellman optimality equation into an iterative update in the form of

$$V_{k+1}(s) = \max_{a \in A}\{r + \gamma \textstyle\sum_{s',r} [P(s'|s,a)V_k(s')]\} \tag{21}$$

Correspondingly, for the state-action-value function, the updated expression is

$$Q_{k+1}(s,a) = r + \gamma \textstyle\sum_{s',r} \left[P(s'|s,a)\max\limits_{a \in A} Q_k(s',a')\right] \tag{22}$$

Theoretically, it can be shown that $Q(s,a)$ eventually converges to the optimal form after the above value function iterations [32]. Researchers have proposed a class of Temporal Difference (TD) methods using Markov properties that allow for the update of the value function estimate to be performed at each new moment of state entry. The iterative update strategy for this class of methods is shown in the following equation

$$V(s_t) + \alpha r[r_t + \gamma V(s_{t+1}) - V(s_t)] \rightarrow V(s_t) \tag{23}$$

where $r_t + \gamma V(s_{t+1}) - V(s_t)$ is referred to the TD error for a single-step decision. The Q-learning algorithm takes the TD objective $r_t + \gamma \max\limits_{a \in A} Q(s_{t+1},a)$ as an estimate of the optimal state-action function, so the flow chart of the Q-learning algorithm can be shown in Algorithm 1, where the strategy used for evaluation and updating is the ϵ-greedy algorithm.

Algorithm 1 Q-learning algorithm

```
Input: Environment($E$); Action space($A$); Initial state($s_0$);
    Discount factor($\gamma$); Learning rate($\alpha$).
Output: Policy($\pi$)
1: $Q(x,a) = 0, \pi(x,a) = \frac{1}{|A(x)|}$;
2: $x = x_0$;
3: for all $t = 1, 2, \cdots$ do
4:     $a = \pi^\epsilon(x) \rightarrow r, x'$;
5:     $a' = \pi(x')$;
6:     $Q(x,a) = Q(x,a)+$
       $\alpha(r + \gamma \max[Q(x',a')] - Q(x,a))$;
7:     $\pi(x) = arg\max_{a'} Q(x,a'')$;
8:     $x = x'$;
9: end for
```

The provided pseudocode outlines the Q-learning algorithm, which is a model-free reinforcement learning algorithm used to learn the value of an action in a particular state. Below is a detailed explanation of the pseudocode.

Inputs:

- Environment (E): The setting in which the agent operates.
- Action space (A): All possible actions the agent can take.
- Initial state s_0: The starting point of the agent in the environment.
- Discount factor γ: The degree to which future rewards are diminished compared to immediate rewards.
- Learning rate α: How much new information overrides old information.

Output:

- Policy π: A strategy that the agent follows, mapping states to the best action to perform in that state.

Algorithm Steps:

1. Initialization: $Q(x,a) = 0$: The Q-value for all state–action pairs is initialized to zero. $\pi(x,a) = \frac{1}{|A(x)|}$: The policy is initialized to be a uniform distribution, where each action is equally probable if there are $|A(x)|$ actions possible in state x.
2. Set initial state: $x = x_0$: The agent starts at the initial state x_0.
3. Learning Loop: This loop continues indefinitely, iterating over each time step t.
    - Action Selection:
      $a = \pi^{\epsilon}(x)$: An action a is chosen using the ϵ-greedy policy from the current policy π. This selects the best action most of the time, but occasionally a random action to explore the environment.
      The agent performs action a, receives a reward r, and transitions to a new state x'.
    - Action Value Update:
      The Q-value of the current state–action pair (x,a) is updated using the Bellman equation incorporating the learning rate α, the received reward r, the discount factor γ, and the maximum Q-value of the subsequent state x'.
    - Policy Update:
      $\pi(x) = \arg\max_{a'} Q(x,a'')$: The policy is updated to choose the action with the highest Q-value for state x.
    - State Transition:
      $x = x'$: Update the state to the new state x'.

The loop represents the process of the agent interacting with the environment, receiving feedback in the form of rewards and updating its policy to maximize those rewards over time. As the algorithm proceeds, the Q-values converge towards the optimal values and the policy π converges towards the optimal policy, balancing exploration and exploitation.

For some simple tasks where the state and action spaces are low-dimensional discrete spaces, $Q(s,a)$ can be represented as a two-dimensional table. The rows of the table represent certain states and actions, and each cell of the table stores the value under the corresponding state and action. Based on the idea of dynamic programming, the values in the table can converge to an optimal value in the above-mentioned value iteration. In each decision, the action with the highest value in the current state is selected. However, for more complex tasks with a higher dimensional state and action spaces, the table structure is no longer applicable but requires a parametric model such as a neural network to characterize the value function and then adjust the model parameters using parametric optimization methods such as gradient descent with the objective of minimizing the TD error, so that it gradually approximates the optimal value function estimate.

In this paper, we can describe the simulation scenario and the different combinations of parameters as the state space S and the action space A, respectively.

4. Simulation Scenarios and Results

In this paper, our primary focus is on studying a swarm system consisting of 32 agents. Our investigation centers around the dynamic manipulation of cohesion and repulsion algorithm parameters. By dynamically adjusting these parameters, we create a versatile simulation scenario that enables continuous obstacle avoidance and maximizes spatial coverage. The effectiveness of this approach is visually depicted in Figure 2, where the swarm adeptly navigates through obstacles while efficiently covering the available space.

To facilitate the learning process and decision making of the agents, we employ a Q-table as a critical component of our methodology. The Q-table, as illustrated in Figure 3a, serves as a tabular representation of the state–action pairs and their corresponding Q-values. It enables the agents to make informed decisions based on the accumulated knowledge and experiences gained during the learning process.

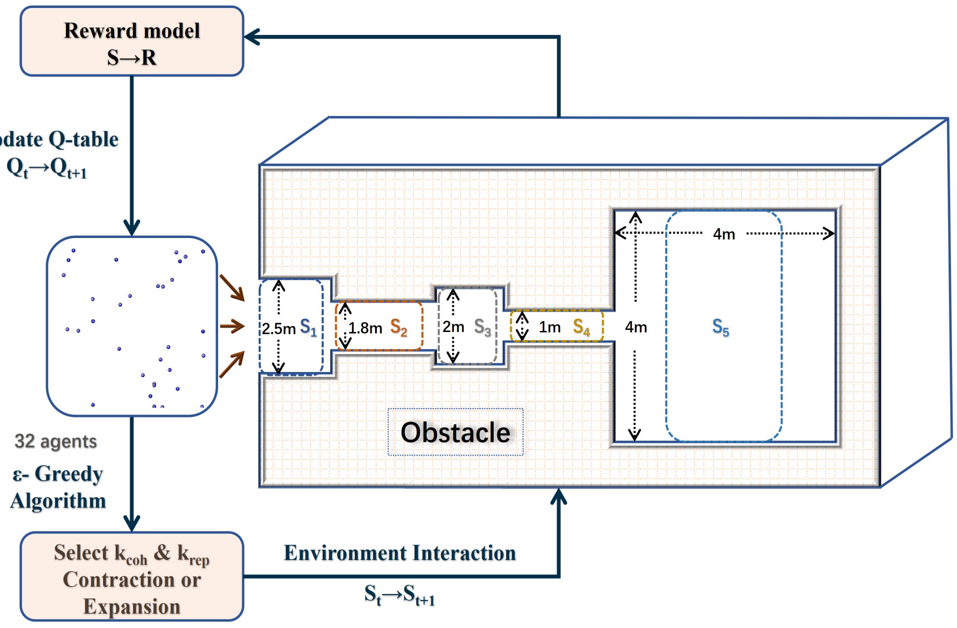
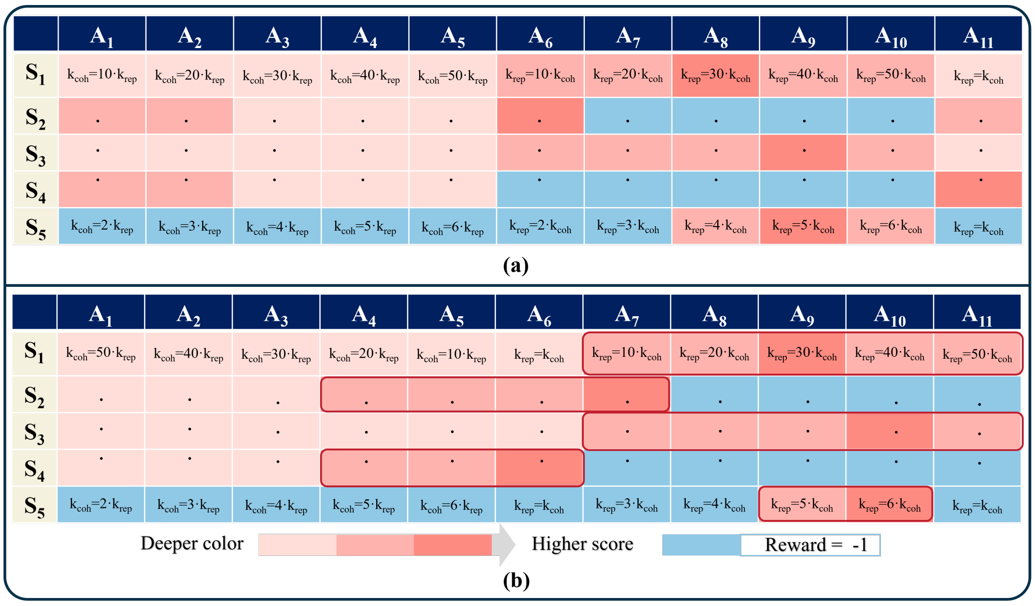


Figure 2. Improved Boids model based on Q-learning network.



	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
S_1	$k_{coh}=10 \cdot k_{rep}$	$k_{coh}=20 \cdot k_{rep}$	$k_{coh}=30 \cdot k_{rep}$	$k_{coh}=40 \cdot k_{rep}$	$k_{coh}=50 \cdot k_{rep}$	$k_{rep}=10 \cdot k_{coh}$	$k_{rep}=20 \cdot k_{coh}$	$k_{rep}=30 \cdot k_{coh}$	$k_{rep}=40 \cdot k_{coh}$	$k_{rep}=50 \cdot k_{coh}$	$k_{rep}=k_{coh}$
S_2	.	.	.	.	.	.	.	.	.	.	.
S_3	.	.	.	.	.	.	.	.	.	.	.
S_4	.	.	.	.	.	.	.	.	.	.	.
S_5	$k_{coh}=2 \cdot k_{rep}$	$k_{coh}=3 \cdot k_{rep}$	$k_{coh}=4 \cdot k_{rep}$	$k_{coh}=5 \cdot k_{rep}$	$k_{coh}=6 \cdot k_{rep}$	$k_{rep}=2 \cdot k_{coh}$	$k_{rep}=3 \cdot k_{coh}$	$k_{rep}=4 \cdot k_{coh}$	$k_{rep}=5 \cdot k_{coh}$	$k_{rep}=6 \cdot k_{coh}$	$k_{rep}=k_{coh}$

(a)

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
S_1	$k_{coh}=50 \cdot k_{rep}$	$k_{coh}=40 \cdot k_{rep}$	$k_{coh}=30 \cdot k_{rep}$	$k_{coh}=20 \cdot k_{rep}$	$k_{coh}=10 \cdot k_{rep}$	$k_{rep}=k_{coh}$	$k_{rep}=10 \cdot k_{coh}$	$k_{rep}=20 \cdot k_{coh}$	$k_{rep}=30 \cdot k_{coh}$	$k_{rep}=40 \cdot k_{coh}$	$k_{rep}=50 \cdot k_{coh}$
S_2	.	.	.	.	.	.	.	.	.	.	.
S_3	.	.	.	.	.	.	.	.	.	.	.
S_4	.	.	.	.	.	.	.	.	.	.	.
S_5	$k_{coh}=2 \cdot k_{rep}$	$k_{coh}=3 \cdot k_{rep}$	$k_{coh}=4 \cdot k_{rep}$	$k_{coh}=5 \cdot k_{rep}$	$k_{coh}=6 \cdot k_{rep}$	$k_{rep}=k_{coh}$	$k_{rep}=3 \cdot k_{coh}$	$k_{rep}=4 \cdot k_{coh}$	$k_{rep}=5 \cdot k_{coh}$	$k_{rep}=6 \cdot k_{coh}$	$k_{rep}=k_{coh}$

Deeper color 🟥 Higher score 🟦 Reward = -1

(b)

Figure 3. Q-table (**a**) State and Action (**b**) Value.

The reward function in our approach is designed to incentivize desirable behaviors and penalize undesirable ones. Specifically, we have defined two components of the reward function:

- Obstacle avoidance reward: This component rewards agents for successfully navigating past obstacles. The reward is computed as the sum of the distances among agents when they successfully avoid obstacles. By encouraging agents to maintain a safe distance from obstacles, this component promotes effective obstacle avoidance behavior. Conversely, if agents fail to surmount obstacles, a penalty of -1 is incurred, discouraging collision or unsuccessful navigation attempts.
- Diffusion reward: This component focuses on maintaining a safe distance from walls and adhering to the expansion criteria. When agents maintain a safe distance from walls and meet the expansion criteria, the reward is calculated as the cumulative inter-agent distance. This encourages agents to spread out evenly within the swarm and avoid clustering near walls. On the other hand, if agents make contact with walls or do not meet the expansion criteria, a penalty of -1 is imposed, discouraging undesired behavior such as wall collisions or failure to achieve the desired expansion.

Our strategy for implementing the ϵ-Greedy algorithm is carefully designed to optimize learning efficiency throughout 100 training sessions, with the current training session denoted as num. We employ the following approach:

- Exploration in Early Stages: During the initial training phases when num is small, we prioritize exploration by assigning a relatively high value to ϵ. This exploration-centric approach allows the algorithm to thoroughly explore and understand the environment, facilitating the discovery of potentially optimal solutions.
- Exploitation in Later Stages: As the training progresses and the rewards associated with different combinations of cohesion and repulsion parameters become well estimated, excessive exploration becomes unnecessary. Therefore, we gradually reduce the value of ϵ as the number of training sessions increases. To achieve this, we utilize the floor function denoted as $\lfloor \cdot \rfloor$, which ensures a smooth reduction in ϵ. By decreasing ϵ, we shift the focus towards exploiting the learned knowledge, enabling more informed and optimal decision making.

This strategic adjustment of ϵ strikes a balance between exploration and exploitation throughout the training process. It allows the algorithm to effectively explore the solution space in the early stages, while gradually transitioning towards exploiting the acquired knowledge in later stages. This approach maximizes the learning efficiency of the ϵ-Greedy algorithm, leading to an improved performance and convergence towards optimal solutions,

$$\epsilon = 1 - 0.1\lfloor \tfrac{num}{10} \rfloor \tag{24}$$

where $\lfloor \cdot \rfloor$ denotes the floor function. It is defined by $\lfloor x \rfloor = \max\{m \in \mathbb{Z} | m \leq x\}$, where x is a real number and $\mathbb{Z}$ denotes a set of integers.

By using the Q-table values depicted in Figure 3b and by training the Q-learning network, we can determine the optimal action for each state corresponding to the optimal parameters of cohesion and repulsion under different obstacle scenarios. These parameter (The real training parameters are in Appendix A) sets are subsequently incorporated into the Boids model for simulation. The performance of the Q-learning algorithm can be highly sensitive to its hyperparameters, such as the learning rate, discount factor, and policy exploration parameters. Tuning these can be non-trivial and may require extensive experimentation.

The simulation results, as shown in Figure 4, serve as compelling evidence of the effectiveness of our approach. The agents adeptly navigate through obstacles while simultaneously maintaining an optimal inter-agent distance and strategically maximizing their spread within the given spatial constraints. This demonstrates the successful mastery of our agents in tackling complex navigation tasks.

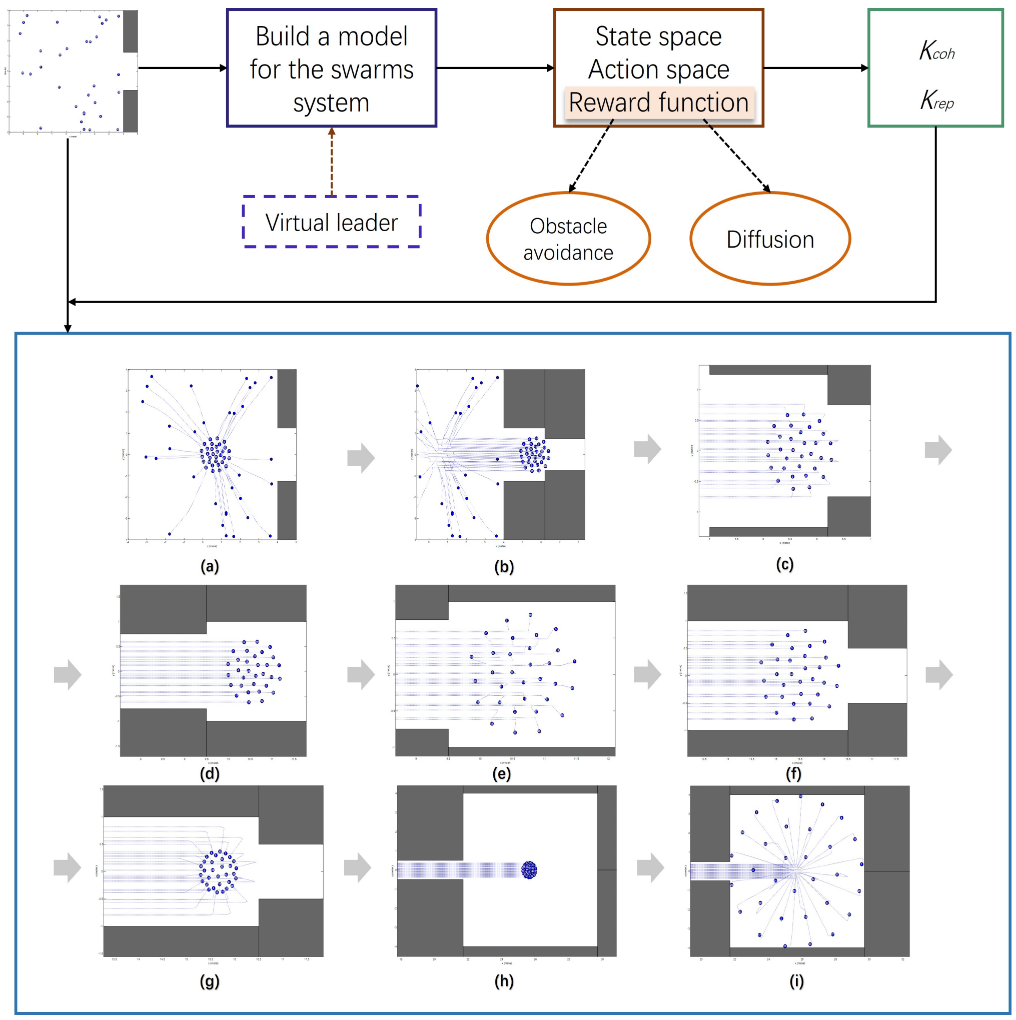


Figure 4. Framework of RL-Boids. In the above part of the algorithm described the workflow, the following part of the simulation results. The dots in (**a**–**i**) represent drones. (**b**,**d**,**f**,**h**) show the virtual leader moving the drone forward through traction. (**a**,**c**,**g**) show the contraction of the drone swarms. (**e**,**i**) show the expansion of drone swarms.

In simulation experiments, Q-learning typically requires discretizing the state and action spaces. This discretization can lead to a loss of fidelity, as continuous nuances in agent behavior and environment states may not be captured. We also compare the use of the Deep Q-learning method [33,34]. Deep reinforcement learning can provide adaptive and intelligent decision-making capabilities for optimizing the network performance. But, it may not directly address swarm behavior or coordination among drones, which is different from the RL-Boids approach.

5. Real Robot Experiments

The experimental section of this research aims to validate the efficacy of the proposed reinforcement learning-based approach for drone swarms. Therefore, a real-world experiment is conducted utilizing three drones in a controlled environment. The overarching objective of this experiment is to convincingly demonstrate the capability of drone swarms to navigate a predefined space adeptly, concurrently avoiding obstacles and optimizing coverage.

In this experiment, three identical drones are used as the robotic agents. These drones have various sensors, including wireless communication modules and inertial measurement units (IMUs). These sensors are pivotal in enabling the drones to perceive their surroundings accurately. They are of a quadcopter design, ideal for stability and maneuverability in various directions.

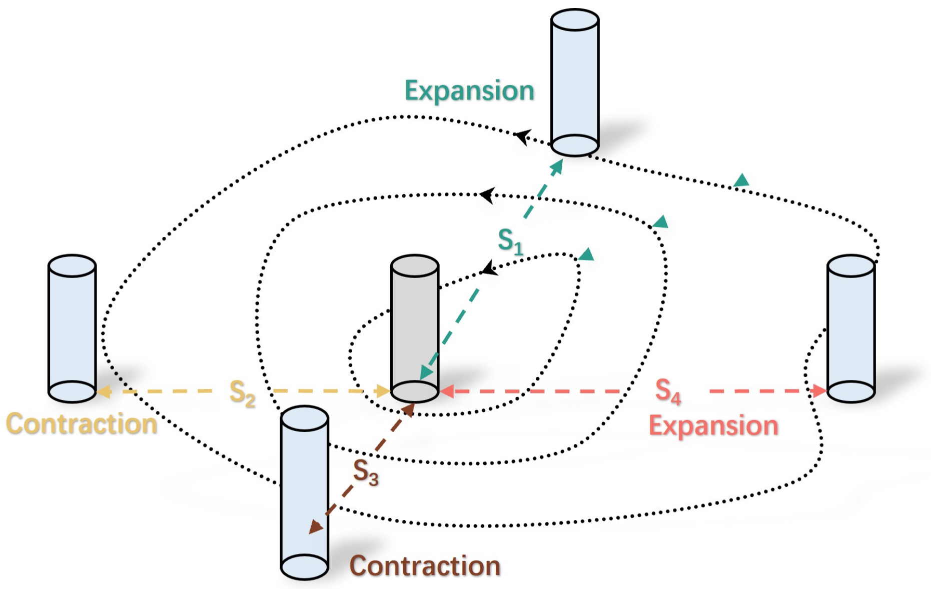


Figure 5. Schematic diagram of indoor real machine experiment.

A central computer system monitors the drones' movements and handles heavy computation if the drones' onboard computers are not capable of real-time processing. The RL algorithm is pre-trained in simulations with similar obstacle configurations to bootstrap the learning process. Hyperparameters are adjusted based on preliminary tests to balance exploration and exploitation.

The experimental environment consisted of an open indoor space with five strategically placed obstacles composed of non-reflective material to avoid sensor distortion. These obstacles are strategically placed to emulate real-world scenarios where the drones must deftly navigate around static objects, mirroring challenges encountered in practical settings such as urban environments or complex terrains. The experiment area is rigged with a motion capture system to provide external validation of the drones' movements and the system's internal measurements. The primary mission for the drones is to seamlessly arrange themselves in a circular formation while adeptly circumventing potential collisions with the obstacles. A schematic diagram depicting the experimental setup is visually represented in Figure 5.

To achieve the desired circular formation, the drones utilize a virtual leader–follower approach. The clustering pinning algorithm calculates the optimal position, which serves as the virtual leader. Moreover, the others follow its movements while maintaining the desired distance and angles. This formation control is crucial for successfully executing expansion and contraction maneuvers.

The core of the experiment revolves around the integration of reinforcement learning (RL) techniques and the body model of the drones. RL algorithms enable the drones to learn and adapt their behaviors based on environment interactions. They learn to apply the right direction to avoid obstacles and maintain the circular formation.

Simultaneously, the drones aim to maximize the coverage of the available space. They expand and contract their formation dynamically, adjusting their positions relative to the virtual leader and obstacles. This dynamic adaptation allows them to explore and cover the entire space efficiently.

The experimental results demonstrate the effectiveness of the proposed approach. As shown in Figure 6, the drones successfully form a circular swarm, avoid obstacles, and maximize coverage of the experimental space. The RL-based control system exhibits adaptability and responsiveness in real-time, ensuring smooth navigation.

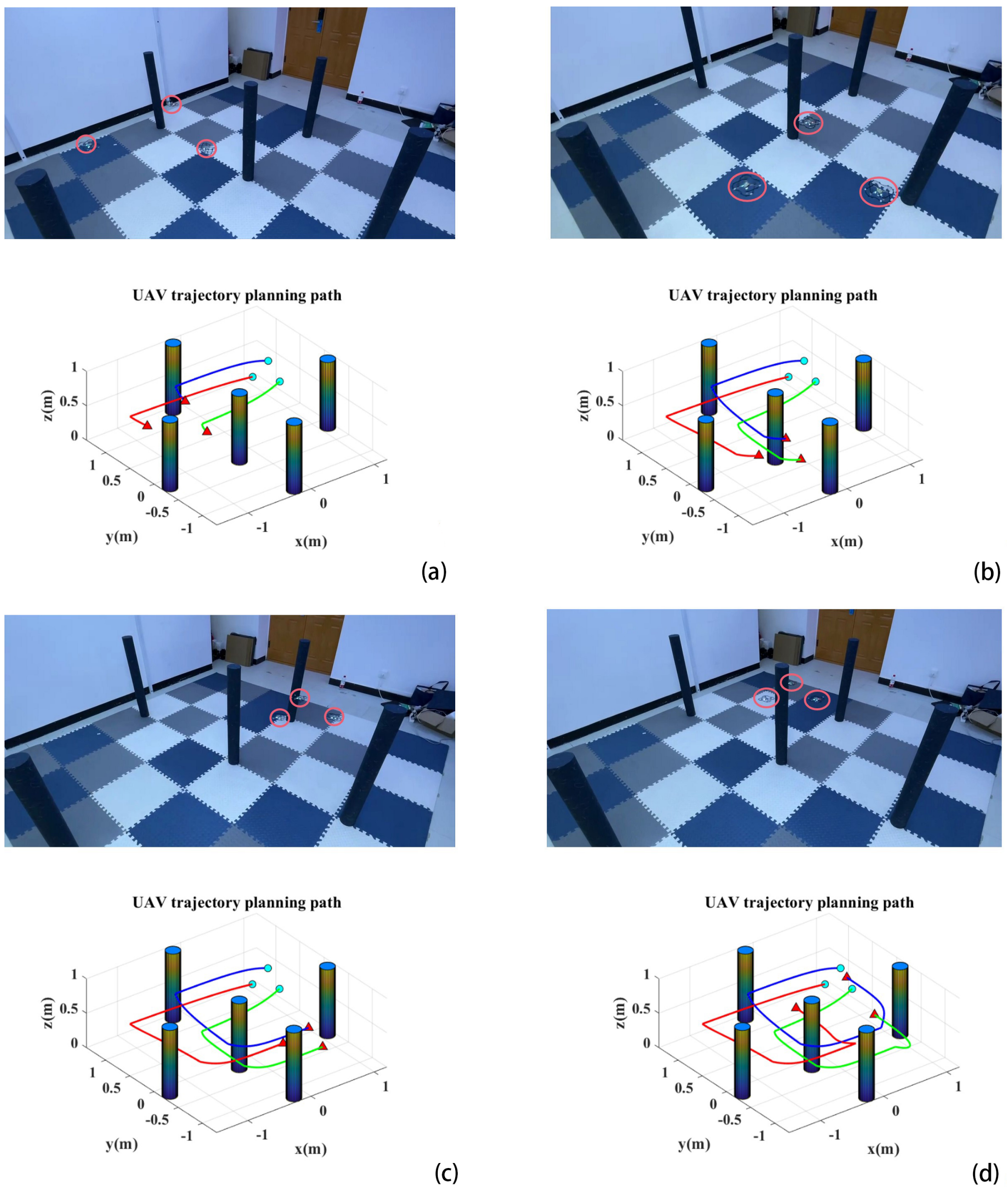


Figure 6. Simulation and indoor real-world experiments for expansion and contraction scalability. (**a**,**d**) are composed of real-world experiments and simulations. The part circled in red in the top half is the drone, and the following is the simulation experiment of its current state. (**a**,**d**) show the contraction of the drone swarm. (**b**,**c**) show the expansion of drone swarms. The experimental video link is as follows: https://www.bilibili.com/video/BV1YN411H7K6 (accessed on 20 September 2023).

Throughout the experiment, the drones consistently avoid collisions with the obstacles. The reinforcement learning algorithms enable them to learn from their interactions, improving their obstacle-avoidance capabilities.

The swarm of drones effectively covers the entire experimental space, demonstrating the potential for such intelligent expansion and contraction strategies in various applications, such as surveillance, search and rescue, and environmental monitoring.

6. Conclusions

This paper has proposed a Q-learning network training method to optimize cohesion and repulsion parameters for the simplified Boids model. This optimization has aimed to achieve continuous obstacle avoidance and maximum space coverage of drones in the simulation environment. Based on this method, we train the theoretical optimal value of the given parameters in the set scenario. The RL-Boids method proposed in this paper has provided analogies and references for related research in the drone swarms field. In future research, our group will explore better reinforcement learning methods based on the Boids model and carry out experimental validation work with drones.

Author Contributions: Conceptualization, Z.D. and Q.W.; methodology, Q.W.; software, Q.W.; validation, Z.D., Q.W. and L.C.; formal analysis, Z.D.; investigation, Q.W.; resources, Z.D.; data curation, Z.D.; writing—original draft preparation, Z.D.; writing—review and editing, Q.W. and L.C.; visualization, Z.D. and Q.W.; supervision, L.C.; project administration, L.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science Foundation of China under Grants 62088101 and 62003015.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Table A1. The Q-table is a matrix where each row corresponds to a state ($S1$ through $S5$) and each column corresponds to an action ($A1$ through $A11$). Each cell in the table contains a numerical value, which represents the learned value of taking a given action in a given state, with the objective of maximizing the cumulative reward.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
S_1	81	97	113	124	133	265	173	256	867	647	345
S_2	159	130	106	88	84	217	1014	−1	−1	−1	−1
S_3	32	58	65	72	102	124	156	221	268	391	215
S_4	85	100	127	164	245	630	−1	−1	−1	−1	−1
S_5	−1	−1	−1	−1	−1	−1	−1	−1	79	397	−1

References

1. Han, C.; Yin, J.; Ye, L.; Yang, Y. NCAnt: A network coding-based multipath data transmission scheme for multi-UAV formation flying networks. *IEEE Commun. Lett.* **2020**, *25*, 1041–1044. [CrossRef]
2. Li, S.; Fang, X. A modified adaptive formation of UAV swarm by pigeon flock behavior within local visual field. *Aerosp. Sci. Technol.* **2021**, *114*, 106736. [CrossRef]
3. Zhang, B.; Sun, X.; Liu, S.; Deng, X. Adaptive differential evolution-based distributed model predictive control for multi-UAV formation flight. *Int. J. Aeronaut. Space Sci.* **2020**, *21*, 538–548. [CrossRef]
4. Nakagawa, E.Y.; Antonino, P.O.; Schnicke, F.; Capilla, R.; Kuhn, T.; Liggesmeyer, P. Industry 4.0 reference architectures: State of the art and future trends. *Comput. Ind. Eng.* **2021**, *156*, 107241. [CrossRef]
5. Hu, B.; Sun, Z.; Hong, H.; Liu, J. UAV-aided networks with optimization allocation via artificial bee colony with intellective search. *EURASIP J. Wirel. Commun. Netw.* **2020**, *2020*, 40. [CrossRef]
6. Kim, J.; Oh, H.; Yu, B.; Kim, S. Optimal task assignment for UAV swarm operations in hostile environments. *Int. J. Aeronaut. Space Sci.* **2021**, *22*, 456–467. [CrossRef]
7. Pham, Q.V.; Huynh-The, T.; Alazab, M.; Zhao, J.; Hwang, W.J. Sum-rate maximization for UAV-assisted visible light communications using NOMA: Swarm intelligence meets machine learning. *IEEE Internet Things J.* **2020**, *7*, 10375–10387. [CrossRef]
8. Reynolds, C.W. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Comput. Graph.* **1987**, *21*, 25–34. [CrossRef]
9. Vásárhelyi, G.; Virágh, C.; Somorjai, G.; Nepusz, T.; Eiben, A.E.; Vicsek, T. Optimized flocking of autonomous drones in confined environments. *Sci. Robot.* **2018**, *3*, eaat3536. [CrossRef]
10. Soria, E.; Schiano, F.; Floreano, D. Predictive control of aerial swarms in cluttered environments. *Nat. Mach. Intell.* **2021**, *3*, 545–554. [CrossRef]
11. Wang, M.; Zeng, B.; Wang, Q. Research on motion planning based on flocking control and reinforcement learning for multi-robot systems. *Machines* **2021**, *9*, 77. [CrossRef]
12. Bai, C.; Yan, P.; Pan, W.; Guo, J. Learning-based multi-robot formation control with obstacle avoidance. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 11811–11822. [CrossRef]

13. Long, P.; Fan, T.; Liao, X.; Liu, W.; Zhang, H.; Pan, J. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 6252–6259.
14. Yan, Y.; Li, X.; Qiu, X.; Qiu, J.; Wang, J.; Wang, Y.; Shen, Y. Relative distributed formation and obstacle avoidance with multi-agent reinforcement learning. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 23–27 May 2022; pp. 1661–1667.
15. Sui, Z.; Pu, Z.; Yi, J.; Wu, S. Formation control with collision avoidance through deep reinforcement learning using model-guided demonstration. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 2358–2372. [CrossRef] [PubMed]
16. Buşoniu, L.; Babuška, R.; De Schutter, B. Multi-agent reinforcement learning: An overview. In *Innovations in Multi-Agent Systems and Applications-1*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 183–221.
17. Nguyen, T.T.; Nguyen, N.D.; Nahavandi, S. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Trans. Cybern.* **2020**, *50*, 3826–3839. [CrossRef] [PubMed]
18. Canese, L.; Cardarilli, G.C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Re, M.; Spanò, S. Multi-agent reinforcement learning: A review of challenges and applications. *Appl. Sci.* **2021**, *11*, 4948. [CrossRef]
19. Leonard, N.E.; Fiorelli, E. Virtual leaders, artificial potentials and coordinated control of groups. In Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228), Orlando, FL, USA, 4–7 December 2001; Volume 3, pp. 2968–2973.
20. Droge, G. Distributed virtual leader moving formation control using behavior-based MPC. In Proceedings of the 2015 American Control Conference (ACC), Chicago, IL, USA, 1–3 July 2015; pp. 2323–2328.
21. Saska, M.; Baca, T.; Hert, D. Formations of unmanned micro aerial vehicles led by migrating virtual leader. In Proceedings of the 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand, 13–15 November 2016; pp. 1–6.
22. Olfati-Saber, R. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Trans. Autom. Control.* **2006**, *51*, 401–420. [CrossRef]
23. Rooban, S.; Javaraiu, M.; Sagar, P.P. A detailed review of swarm robotics and its significance. In Proceedings of the 2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), Erode, India, 7–9 April 2022; pp. 797–802.
24. Kyzyrkanov, A.; Atanov, S.; Aljawarneh, S.; Tursynova, N.; Kassymkhanov, S. Algorithm of Coordination of Swarm of Autonomous Robots. In Proceedings of the 2023 IEEE International Conference on Smart Information Systems and Technologies (SIST), Astana, Kazakhstan, 4–6 May 2023; pp. 539–544.
25. Holland, J.H. Genetic algorithms. *Sci. Am.* **1992**, *267*, 66–73. [CrossRef]
26. Dorigo, M.; Birattari, M.; Stutzle, T. Ant colony optimization. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–39. [CrossRef]
27. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
28. Ballerini, M.; Cabibbo, N.; Candelier, R.; Cavagna, A.; Cisbani, E.; Giardina, I.; Lecomte, V.; Orlandi, A.; Parisi, G.; Procaccini, A.; et al. Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. *Proc. Natl. Acad. Sci. USA* **2008**, *105*, 1232–1237. [CrossRef]
29. Din, A.; Jabeen, M.; Zia, K.; Khalid, A.; Saini, D.K. Behavior-based swarm robotic search and rescue using fuzzy controller. *Comput. Electr. Eng.* **2018**, *70*, 53–65. [CrossRef]
30. Greenwald, A.; Hall, K.; Serrano, R. Correlated Q-learning. In Proceedings of the ICML, Washington, DC, USA, 21–24 August 2003; Volume 3, pp. 242–249.
31. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
32. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
33. Guo, J.; Huo, Y.; Shi, X.; Wu, J.; Yu, P.; Feng, L.; Li, W. 3D aerial vehicle base station (UAV-BS) position planning based on deep Q-learning for capacity enhancement of users with different QoS requirements. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 1508–1512.
34. Arani, A.H.; Hu, P.; Zhu, Y. HAPS-UAV-Enabled Heterogeneous Networks: A Deep Reinforcement Learning Approach. *arXiv* **2023**, arXiv:2303.12883.