Robotics and Autonomous Systems 167 (2023) 104489

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

A deep multi-agent reinforcement learning framework for autonomous aerial navigation to grasping points on loads

Jingyu Chen [a,*], Ruidong Ma [a], John Oyekan [b]

[a] Department of Automatic Control and Systems Engineering, The University of Sheffield, Sheffield, United Kingdom
[b] Department of Computer Science, University of York, York, United Kingdom

ARTICLE INFO

Article history:
Received 22 June 2022
Received in revised form 28 April 2023
Accepted 23 June 2023
Available online 10 July 2023

Keywords:
Cooperative navigation
Multi-agent reinforcement learning
Learning from demonstration
Curriculum learning

ABSTRACT

Deep reinforcement learning, by taking advantage of neural networks, has made great strides in the continuous control of robots. However, in scenarios where multiple robots are required to collaborate with each other to accomplish a task, it is still challenging to build an efficient and scalable multi-agent control system due to increasing complexity. In this paper, we regard each unmanned aerial vehicle (UAV) with its manipulator as one agent, and leverage the power of multi-agent deep deterministic policy gradient (MADDPG) for the cooperative navigation and manipulation of a load. We propose solutions for addressing navigation to grasping point problem in targeted and flexible scenarios, and mainly focus on how to develop model-free policies for the UAVs without relying on a trajectory planner. To overcome the challenges of learning in scenarios with an increasing number of grasping points, we incorporate the demonstrations from an Optimal Reciprocal Collision Avoidance (ORCA) algorithm into our framework to guide the policy training and adapt two novel techniques into the architecture of MADDPG. Furthermore, curriculum learning with the attention mechanism is utilized by reusing knowledge from fewer grasping points to facilitate the training of a load with more points. Our experiments were validated by a load with three, four and six grasping points respectively in *Coppeliasim* simulator and then transferred into the real world with *Crazyflie* quadrotors. Our results show that the average tracking deviations from the desirable grasping point to the final position of the UAV can be less than 10 cm in some real-world experiments. Compared with state-of-the-art model-free reinforcement learning and swarm optimization algorithms, results show that our proposed methods outperform other baselines with a reasonable success rate especially in the scenarios with more grasping points. Furthermore, the learned optimal policies enable UAVs to reach and hover over all the grasping points before manipulation without any collision. We conducted a comprehensive analysis of both targeted and flexible navigation, highlighting their respective advantages and disadvantages.

1. Introduction

The growing demand for aerial delivery of loads by UAVs has motivated engineers to search for autonomous solutions that are efficient, safe, and flexible for deployment in various industries. Recent research has extensively studied the challenging cooperative transport tasks for aerial delivery. Most of these studies have focused on applying analytical equations to the modelling of a dynamic slung-load model while taking into account the coupling effects of an arbitrary number of drones [1,2]. Additionally, learning-based methods in both model-based [3] and model-free [4] fashions have been explored to improve load tracking performance. Before the transportation of a load, navigation to the grasping points is also a significant procedure that often requires complex coordination, especially for a large number of drones. For example, in Fig. 1, we consider a practical scenario where UAVs are required to grasp the component of a rocket. in order to approach all grasping locations, the controller needs to consider how to generate collision-free trajectories for multiple robots while guaranteeing time and computation efficiency. The path planning problem can be solved by simultaneous localization and mapping (SLAM) and a navigation stack which gives us prior knowledge of the environment.

In contrast to these aforementioned methods, our work focuses on a learning-based cooperative navigation that interacts with the environment and does not rely on any predefined map. In [5], the authors developed an end-to-end decentralized algorithm based on proximal policy optimization (PPO) to realize collision-free navigation tasks with up to 100 robots using the

* Corresponding author.
E-mail addresses: jchen118@sheffield.ac.uk (J. Chen), rma17@sheffield.ac.uk (R. Ma), john.oyekan@york.ac.uk (J. Oyekan).

https://doi.org/10.1016/j.robot.2023.104489

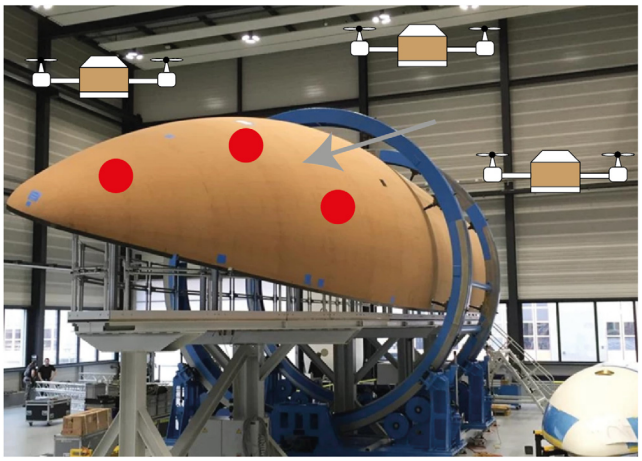


Fig. 1. A practical scenario of cooperative aerial navigation to a component of a rocket in a factory. Red dots represent the corresponding grasping locations.

onboard sensor observation. Moreover, Deep Q-learning (DQN) was utilized to shepherd a flock of drones to a goal position in the environment with obstacles [6]. In these cases, the agents are only required to optimize their independent goals, which means that the goal positions are already assigned when optimizing the policy.

On the contrary, when the goal assignments for each agent are not pre-defined, cooperative navigation can be viewed as a coverage task in swarm robotics. In this case, the overall coverage rate of the environment is more important than individual goals. In [7], a behavioural controller consisting of several swarm behaviours is proposed to navigate multiple drones to uniformly cover the payload by the onboard perception without the goal position. However, rule-based coordination typically requires a significant amount of time to observe and adjust to emergent behaviour, making it inefficient for deployment in the industry. Additionally, multi-agent Q-learning can be utilized to distribute a team of UAVs to the locations of forest fires based on local information, as demonstrated in [8].

In this paper, we consider the aforementioned two approaches in the context of cooperative navigation to grasping points. Some cases of cooperative **flexible navigation** require a quick and efficient way to distribute UAVs with the same roles to all the grasping points. In other cases of cooperative **targeted navigation**, a UAV with a specific role is required to go to a particular location. As a result, we contribute to learning-based solutions for two kinds of aerial navigation (targeted and flexible navigation) and manipulation where drones are trained to form predefined grasping formations above a load. As in [9], we designed our control system with a hierarchical structure to tackle both navigation and manipulation tasks respectively. To realize this, we develop our algorithm based on multi-agent deep deterministic policy gradient (MADDPG), a kind of multi-agent reinforcement learning (MARL) method where the joint states and actions are considered to maximize the team reward. The learned policies are deployed to plan safe routes for drones to reach all grasping points without any collision in a bounded environment.

However, as the number of grasping points and robots increases, the difficulty of navigation tasks also increases significantly [10]. This can cause the algorithms to fail to converge. Therefore, since the reciprocal collision avoidance (ORCA) method has the same action space and observation space as that of Reinforcement learning (RL), we use an expert policy and expert trajectories from ORCA to guide the training of MADDPG towards achieving the same reward level as the expert. In this work, we do not expect our proposed control system to replace traditional path planning methods like ORCA. Instead, we contribute by introducing further steps towards learning-based control strategies for engineering cases. Finally, we apply an attention-based actor–critic algorithm to enable curriculum learning. This allows the transfer of previous experience to a new scenario. This is important because deep reinforcement learning (RL) algorithms are known to perform well when the task and state distributions remain the same. However, when there is a change in the transition dynamics or state dimensions of the environment, the controller may need to be re-trained. This is not practical for industrial applications that require flexibility. To address this issue, we propose a curriculum learning approach with an attention-based framework. We start by training the controller on a load with fewer grasping points and gradually increase the complexity of the task to include more points. The model is reloaded without re-training from scratch, allowing for quick adaptation to new scenarios.

Our contributions are summarized as follows:

- Considering flexible and targeted navigation, we present the first hierarchical learning-based solution for the aerial cooperative manipulation task by aerial robots equipped with suction cups.
- To address the challenges of training MADDPG in more complex scenarios, we introduce demonstrations from the ORCA method using two strategies: pretraining with prioritized sampling and behaviour cloning.
- We constructed a real-world control pipeline for our algorithms, integrating equipment such as Aruco markers and cameras. To confirm the feasibility of our approach, we validated it using micro UAVs, specifically the *Crazyflie*.
- A curriculum learning with attention framework is proposed for targeted navigation to facilitate the adaption to new payloads.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the background of MARL, learning from demonstration and curriculum learning. Section 3 puts forward the problem definition while Section 4 comprehensively illustrates our proposed framework based on MADDPG. The experimental results and discussions are presented in Section 5 followed by the conclusion in Section 6.

2. Related work

2.1. Multi-agent reinforcement learning

Reinforcement learning (RL) is an algorithm that takes inspiration from animal learning in psychology. It is a trial-and-error (TE) learning process where an agent learns an action policy to maximize long-term rewards through interactions with an environment. When more than one agent is involved, the problem is formulated as a multi-agent reinforcement learning (MARL) problem. Since the strategies of other agents are always changing, the environment becomes non-stationary. As a result, an individual's optimal policy is influenced by the actions of other agents. [11] discussed how nonstationarity results in the incompatible use of a replay buffer in multi-agent Q-learning. They proposed adding a low-dimensional *fingerprint* into the Q function to condition on the policies of other agents. Based on the actor–critic framework, [10] designed a centralized critic which considers the state–action pairs of other agents to solve the non-stationary problem. Besides, the attention structure can be added to the centralized critic to better model the policies of other agents [12]. The multi-task learning could also be achieved by an actor–critic algorithm with a single set of parameters [13].

Another challenge of deep MARL is the *curse of dimensionality*. This means that the dimension of action and observation space grows exponentially as the number of agents increases. The idea of gathering all states of a system into the input of a single neural network is not feasible. As a result, centralized training and decentralized execution are adopted as the mainstream architecture for MARL. In this case, agents use their local perception to make decisions while only requiring global states for training. However, the critic neural network still faces the issue of large input dimensions during the training. In [14], the author applied mean feature embeddings (MFE) to represent states of the adjacent agents in a local sample-based view. The efficiency of this encoding method has been proved in the large dimensionality of MARL. Our work has a similar algorithm in [10], but makes contributions to the optimization method and neural network (NN) architecture to allow it to adapt to scenarios with higher dimensions of states.

2.2. Learning from demonstration

Imitation learning (IL) trains an agent to mimic the behaviour of a human or another agent by demonstration data, without having to explicitly specify a reward function. Behaviour cloning is a simple method of imitation learning. It predicts the expert's actions, given a certain input state, by the supervised learning of the state action pairs from the demonstration. One of the early applications of behaviour cloning was validated in the field of autonomous navigation by mapping sensor data into the steering direction of a vehicle [15]. However, behaviour cloning suffers from the issue of covariate shift, which occurs when the distribution of states or observations in the test environment differs from the distribution in the training data. This issue can be solved by using a generative model to generate the actions which are close to the distribution of the expert actions in an adversarial fashion [16]. Imitation learning can be used to speed up the learning process in reinforcement learning by allowing the agent to learn from a demonstration of an expert's behaviour. As for model-free reinforcement learning algorithms, large amounts of self-generated data from interactions with the environment are often required. However, this is not realistic when the cost of collecting this data can be prohibitively expensive, and in some cases, the trial-and-error process can pose significant risks, particularly for robotics tasks where mistakes can result in catastrophic outcomes. Therefore, pretraining an agent with a small amount of demonstration data can help the agent learn a good policy faster and more efficiently. The idea is to use the demonstration data to provide the agent with a good initial policy that it can refine through further exploration [17]. Several techniques have been proposed to improve the exploration in reinforcement learning, such as demonstration buffer, newly designed behaviour cloning loss, and Q-filter [18]. Moreover, recent progress on offline reinforcement learning showed that it is possible to achieve good performance on collected data without the need for exploration [19]. Inverse reinforcement learning (IRL) is another approach of imitation learning where the reward function is learned from the demonstration data, and then used for later policy learning [20]. However, it can be computationally expensive and may require a large amount of demonstration data to learn an accurate reward function which can be generalized to new situations. Imitation learning with RL has been widely applied in robotics applications. In [21], based on guided policy search (GPS), the decentralized policy for drone control is supervised by the sample trajectories collected from the model predictive control (MPC). [22] proposed a two-stage actor–critic framework for ground robots where the actor is firstly optimized by the demonstration trajectories and then transitions to the normal training of RL.

2.3. Curriculum learning and attention mechanism

Curriculum learning refers to training from the easy task to the difficult one. As for the multi-agent system (MAS), increasing the number of agents and designing more sophisticated environments are common ways to increase the difficulty of the learning task. Compared with training from scratch, researchers found that higher performance of the policy can be obtained through this curriculum [5]. Three transfer learning strategies, including the reuse of replay buffer, model reload, and curriculum distillation, have been proposed to transfer knowledge from the task with few agents to the large-scale scenario of StarCraft [23]. Besides, the credit assignment problem in multi-goal multi-agent reinforcement learning can be solved by using a credit function that is learned through a two-stage curriculum learning process. However, the objective misalignment issue needs to be addressed for each agent in the initialization of the new stage [24]. The evolutionary algorithm can be used to address the objective misalignment issue and select the best candidates for the next-stage training [25]. Another challenge in the curriculum learning of multi-agent systems (MAS) is that the neural network model trained in the previous stage, which has a fixed input dimension, cannot be directly applied in the new stage, as the observation space may have changed due to the increasing number of agents. Thus, a neural network model that can handle dynamic inputs is required. One of the most popular ways to address the challenge of dynamic input dimensions is to incorporate attention mechanisms into the neural network architecture. Attention mechanisms are inspired by the way humans focus on important areas of an image in visual tasks and have been widely used in computer vision and natural language processing (NLP) [26]. By selectively attending to relevant parts of the input, attention mechanisms can effectively handle variable-sized inputs and capture the dependencies and interactions between different agents. A Q-attention method has been proposed, which uses a hard attention module to extract the most relevant parts of the RGB and point cloud inputs for controlling robotic manipulation [27]. Moreover, the attention mechanism can either be used for learning a centralized critic [28] or simplification of the interactions [29] in a multi-agent system.

3. Problem definition

Our work focuses on navigating multiple aerial vehicles towards specific predetermined locations within an indoor environment to perform payload grasping tasks. A static payload is placed at the centre of the workspace. UAVs with a suction cup gripper (more details in Appendix C) starting from random take-off positions within the workspace are required to cover and hover over all goal positions. No obstacles are considered during the navigation task. As shown in Fig. 2, this work considers two different scenarios for the navigation task, namely targeted navigation and flexible navigation. The flexible navigation (the first scenario) where UAVs with identical roles is designed to find the efficient route to reach all the grasping point while avoiding obstacles. In targeted navigation, each drone with different roles is programmed to fly directly to a specific location or point in space where the grasping point is located.

The problem is formulated as Multi-Agent Partial Observation Markov Decision Process (MAPOMDP) described by the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with N agents. The state $\mathcal{S}$ corresponds to the states of all drones and grasping points. $\mathcal{O}$ and $\mathcal{A}$ represent a set of observations $\mathcal{O}_1, \ldots, \mathcal{O}_N$ and actions $\mathcal{A}_1, \ldots, \mathcal{A}_N$ for each agent respectively. These observations can either be obtained from onboard and external localization devices or the simulator. Each UAV i has its local observation $o_i : \mathcal{S} \mapsto \mathcal{O}_i$, and chooses

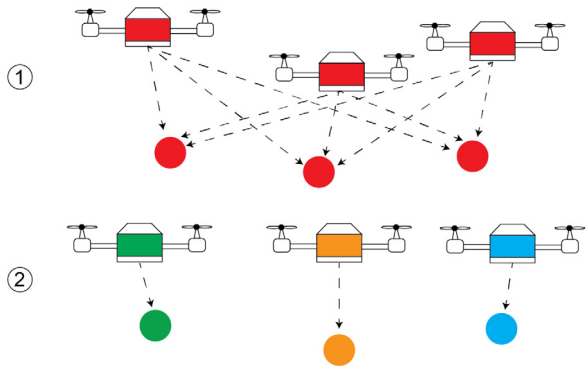


Fig. 2. Showing two types of navigation to target. ① Flexible navigation in which any UAV can approach any grasping point and ② Targeted navigation in which a UAV is programmed to fly directly to a specific grasping point. Different colour means different grasping points and assigned UAVs.

an action by a policy $a_i \sim \pi_{\theta_i}(o_i;\theta_i)$ with parameter θ_i. Then, the next state is generated by the transition probability $\mathcal{P}: \mathcal{S}\times\mathcal{A}_1\times\cdots\times\mathcal{A}_N \mapsto \mathcal{S}$. As the task is cooperative, each agent i obtains a shared reward $r: \mathcal{S}\times\mathcal{A}\mapsto\mathbb{R}$ considering states and actions of all agents. We aim to maximize the cumulative shared reward $R=\sum_{t=0}^{T}\gamma^t r_t$ where T is the time horizon and γ is the discount factor. Assume the position of each UAV i with safe distance R is p_i. In flexible navigation, grasping points are not assigned to individual UAVs. Instead, the policy selects a single goal position for each UAV, denoted as p^g. In targeted navigation, on the other hand, the goal position p^g for each UAV is pre-assigned before the task. The distance from UAV i to wall n is d_{in}^w. The required position precision for coverage is d_s. Therefore, the shared reward r is under the following constraints C_i for each agent.

$$\begin{aligned} C=\{&C_i, i=1,\ldots,N|\\ &\forall m\in[1,N], i\neq m: \|p_i-p_m\|>R,\\ &\forall n\in[1,4]: \|d_{in}^w\|>R\\ &\|p_i-p^g\|<d_s\} \end{aligned} \tag{1}$$

4. Methodology

In this section, we demonstrate our framework for the navigation and manipulation task using multiple homogeneous drones with suction cups. Our framework is based on MADDPG [10]. We first introduce the setup of reinforcement learning. Then, we illustrate demonstration-based methods for flexible navigation and curriculum-based methods for targeted navigation. Finally, we present our hierarchical control pipeline for aerial navigation and manipulation.

4.1. Reinforcement learning setup

For the low-level control, we adopt the PID velocity controller framework, while RL is used for the decision-making of linear velocities of drones. To ensure safe exploration, the drones maintain a constant hovering height and only move in the x–y Euclidean plane during navigation. Each drone is treated as an omnidirectional vehicle whose orientation does not change. The RL policy outputs linear velocities, v_x and v_y, with respect to the inertial coordinate in the x–y plane. These velocities are in the continuous action space and range from -1 to 1 m/s. We do not include extra control for the suction cup and cable, as their mass is small enough to be ignored and their impact on the drone's motion is incorporated in the learning process during interactions.

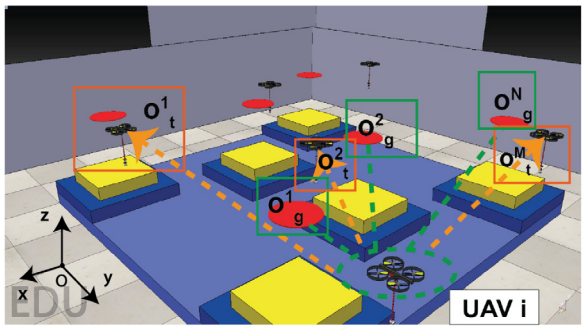


Fig. 3. Observation space of UAV i. The inertial coordinate is set in the centre of the workspace.

As shown in Fig. 3, the observation of agent i is concluded into $\boldsymbol{o}_i=[o_s^i, o_t^i, o_g^i]$ where o_s^i are the velocity and position of drone i itself, o_t^i represent relative positions and relative velocities of all neighbours respectively with respect to the agent i, and o_g^i are relative positions of all grasping points with respect to the agent i. All units are calculated under the inertial coordinate. As the navigation task is cooperative and all drones are homogeneous, every agent aims to maximize the same team reward $r=r_g+r_o$ where r_g is the dense reward computed by the negative sum of the distance d^{ji} between each grasping point j and its nearest agent i (j is iterated by all goals), r_o relates to the collision penalty (see the following (2),(3)).

$$r_g=\begin{cases}1*N, & \text{if all agents cover all goals}\\ -\sum_{j=1}^{N} min(d_{i=1:N}^{ji}), & \text{otherwise}\end{cases} \tag{2}$$

$$r_o=\begin{cases}-1*N, & \text{if any collision exists}\\ 0, & \text{otherwise}\end{cases} \tag{3}$$

To obtain a collision-avoidance policy, if any collision exists in the system, every agent is punished with a -1 reward (N is the total number of agents) and the current episode will be terminated. On the contrary, to improve the success rate of coverage, each agent is rewarded with 1 when all goals have been covered. In addition, we set the required position precision between each goal and the corresponding UAV to be 0.5 m, which is equal to the size of the drone. The episode will not be terminated after the successful coverage as drones need to learn how to keep hovering above the predefined grasping points by accumulating reward r_g. Regarding targeted navigation, some small modifications need to be made. Towards this, the state space becomes $\boldsymbol{o}_i=[o_s^i, o_t^i, o_g^1]$ and o_g^1 is the assigned goal position (not a vector). Each UAV is assigned a unique goal, and the reward function is changed to $-\sum_{j=1}^{N} d^{ji}$ where i is the corresponding index of the assigned agent for grasping point j.

We develop the proposed framework based on a multi-agent actor–critic algorithm called MADDPG. Deep deterministic policy gradient (DDPG) extends the deterministic policy μ_θ in the policy gradient formation by learning a decentralized actor together with a Q-function from Q-learning. Furthermore, MADDPG adopts a centralized Q-function (critic) considering the policies of other agents to solve the problem of the non-stationary environment in MARL. Thus, the gradient of MADDPG with policy parameter θ_i for agent i is written as,

$$\nabla_{\theta_i}J(\boldsymbol{\mu}_i)=\mathop{\mathbb{E}}_{\mathbf{x},\mathbf{a}\sim\mathcal{D}}[\nabla_{\theta_i}\boldsymbol{\mu}_i(a_i|o_i)\nabla_{a_i}Q_i^{\boldsymbol{\mu}}(\mathbf{x},\mathbf{a})|_{a_i=\boldsymbol{\mu}_i(o_i)}] \tag{4}$$

where all actions $\mathbf{a}=(a_1,a_2,\ldots,a_N)$, all observations $\mathbf{x}=(o_1,o_2,\ldots,o_N)$ and N is the agent number. Then, the centralized Q-function is optimized by,

$$\mathcal{L}(\theta_i)=\mathop{\mathbb{E}}_{\mathbf{x},\mathbf{a},r,\mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x},\mathbf{a})-(r_i+\gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}',\mathbf{a}')|_{a_j'=\boldsymbol{\mu}_j'(o_j)})^2] \tag{5}$$

Algorithm 1 MADDPG with demonstration

Input: demonstration trajectories ($\mathbf{O}_d^t$, $\mathbf{A}_d^t$, $\mathbf{O}_d^{t+1}$, $\mathbf{R}_d^t$)
Insert demonstrations into buffer D
Pre-training actors and critics
for Episode = 1:M **do**
  Reset environment
  Obtain initial observations $\mathbf{O}_t$
  for Episode length t = 1:t_{max} **do**
    For each drone, select action $a_i^t = \mu_i(o_i)$ using ϵ greedy with noise
    Use PID velocity controller to execute all actions $\mathbf{A}^t = (a_1^t, ..., a_N^t)$
    Obtain new observations $\mathbf{O}^{t+1}$ and rewards $\mathbf{R}^t$
    Store ($\mathbf{O}^t$, $\mathbf{A}^t$, $\mathbf{O}^{t+1}$, $\mathbf{R}^t$) into replay buffer D
    $\mathbf{O}^t \leftarrow \mathbf{O}^{t+1}$
    Sample a batch ($\mathbf{O}$, $\mathbf{A}$, $\mathbf{O}'$, $\mathbf{R}$) from replay buffer D
    for Drone i = 1: N **do**
      Update critics by Eq. (5)
      Update actors by Eq. (4)
      Update priorities of the sampled transitions
    end for
    Update target networks
  end for
end for

$\boldsymbol{\mu}'$ refers to the target policies with parameter θ_i' and $\mathbf{x}'$ is the next observations of all agents. Since the vanilla MADDPG has limited performance in our task, we proposed two techniques, namely demonstration learning and curriculum learning, to accelerate the learning process and improve the performance of MADDPG. These techniques enable MADDPG to better adapt to the cooperative navigation task.

4.2. Demonstration learning

4.2.1. Learning with demonstration trajectories

We choose the reciprocal collision avoidance (ORCA) method [30] based on velocity obstacle (VO) to collect the demonstration trajectories for MADDPG in the flexible navigation. The chosen expert approach acts in a decentralized way where the agent finds the collision-free path to the goal position by its own observations of other agents, which is similar to how our RL method behaves.

The velocity obstacle $VO^{\tau}_{m_i|m_j}$ is defined as the set of all relative velocities of agent m_i with respect to agent m_j that will cause the collision between agent m_i and agent m_j before time τ. We regard each drone as a disc $I(\mathbf{p}, r)$ located in the position $\mathbf{p}$ with a radius r. Thus, the formal definition is given in Eq. (7),(6)

$$I(\mathbf{p}, r) = \{\mathbf{q} | \|\mathbf{q} - \mathbf{p}\| < r\} \tag{6}$$

$$VO^{\tau}_{m_i|m_j} = \{\mathbf{v} | \exists t \in [0, \tau] :: t\mathbf{v} \in I(p_j - p_i, 2r)\} \tag{7}$$

To avoid collision for at least τ time, the collision-avoiding velocities $CA^{\tau}_{m_i|m_j}(V_j)$ for agent m_i is defined as,

$$CA^{\tau}_{m_i|m_j}(V_j) = \{\mathbf{v} | \mathbf{v} \notin VO^{\tau}_{m_i|m_j} \oplus V_j\} \tag{8}$$

where V_j is the velocity set of agent m_j and $\oplus$ refers to the operation of Minkowski sum. Therefore, $ORCA^{\tau}_{m_i|m_j}$ and $ORCA^{\tau}_{m_j|m_i}$ refer to the reciprocal collision-avoiding velocities which are maximally close to their current velocities. Following the principle of equal responsibility, in Eq. (9), the permitted velocity plane $ORCA^{\tau}_{m_i|m_j}$ of agent m_i lies in the half plane pointing from $\mathbf{v}_i^c + \frac{1}{2}\mathbf{u}$ with direction $\mathbf{n}$.

$$ORCA^{\tau}_{m_i|m_j} = \{\mathbf{v} | (\mathbf{v} - (\mathbf{v}_i^c + \frac{1}{2}\mathbf{u})) \cdot \mathbf{n} \geq 0\} \tag{9}$$

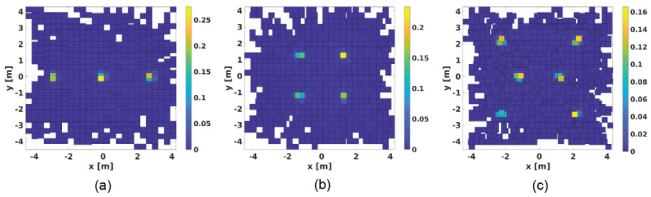


Fig. 4. Normalized 2d histogram of demonstration trajectories in all scenarios.(a) three grasping points (b) four grasping points (c) six grasping points.

where $\mathbf{v}_i^c$ is the current velocity of agent m_i, $\mathbf{u}$ is the shortest vector from the relative velocity to the boundary of velocity obstacle $VO^{\tau}_{m_i|m_j}$ and $\mathbf{n}$ represent the normal vector. The new velocity $\mathbf{v}_i^{new}$ of agent m_i is solved by *linear programming* on the convex *ORCA* planes with respect to each other drone considering its preferred velocity $\mathbf{v}_i^{pref}$.

$$\mathbf{v}_i^{new} = \{\mathbf{v} | \arg\min \|\mathbf{v} - \mathbf{v}_i^{pref}\|, \mathbf{v} \in ORCA^{\tau}_{m_i}\} \tag{10}$$

$\mathbf{v}_i^{new}$ is the expert action we need to record. To lead drones to the grasping points, each drone is assigned a goal $\mathbf{p}_g$ randomly and this assignment is fixed during one episode. Thus, the preferred velocity $\mathbf{v}_i^{pref}$ of agent m_i for ORCA is calculated by $\alpha(\mathbf{p}_g - \mathbf{p}_i)$ (α is the factor and $\mathbf{p}_i$ is the position of drone i). As this goal assignment is only executed in the demonstration setup (not in the normal training), we do not violate our assumption of flexible navigation where the goal assignment is inferred by the policy. In our scenario, the time horizon τ is set to 4.5 and the maximum velocity of a UAV is 1 m/s. The shape of a UAV is simplified into a disc with a radius of 0.3 m.

By placing UAVs at random positions in the workspace, we collect 100 episodes data for each scenario by the expert method ORCA. For the scenarios with three and four grasping points, we used a timestep of 25, while for the scenario with six grasping points, we used a timestep of 35. The normalized 2d histograms of positions of the demonstration data are shown in Fig. 4. We stored the demonstration trajectories ($\mathbf{O}_d$, $\mathbf{A}_d$, $\mathbf{O}_d'$, $\mathbf{R}_d$) of ORCA, which had the same observation and action space as MARL, in the prioritized replay buffer (PER) forever [31]. A pre-training of these demonstrations is started before the normal training of MADDPG (see Algorithm 1). As for PER, the transitions are sampled according to its priority. In MADDPG, the priority p_t of transition t is related to the TD error $\mathcal{L}(\theta_i)$ in Eq. (5) and the actor loss $Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a})$,

$$p_t = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\theta_i) + \|Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a})|_{a_i=\boldsymbol{\mu_i}(\boldsymbol{o_i})}\|^2 + \delta + \delta_d \tag{11}$$

where N is the number of agents, δ is a positive constant and δ_d is the extra parameter for the demonstration transition. To assign the high probability to the trajectories with high TD loss for all critics, we adopt the mean probability of all critics as the final transition probability. As more transitions generated by the training policy are stored in the replay buffer, the guidance ratios with different number of agents are automatically tuned.

4.2.2. Learning with behaviour cloning

We propose another method based on behaviour cloning (bc) from imitation learning for MADDPG to improve the algorithm convergence in the scenario with more grasping points and robots. Different from the previous pretraining strategy, our behaviour cloning method for MADDPG queries demonstration action $\mathbf{A}_e$ in real-time and uses it to minimize the actor loss in the early stage, which behaves like supervised learning.

$$Loss^i_{actor} = \|\mathbf{A}^i - \mathbf{A}^i_e\|^2 \tag{12}$$

Algorithm 2 MADDPG with behaviour cloning

for Episode = 1:M **do**
  Reset environment
  Obtain initial observations $\mathbf{O}_t$
  for Episode length t = 1:t_{max} **do**
    For each drone, select action $a_i^t = \mu_i(o_i)$ using ϵ greedy with noise
    Use PID velocity controller to execute all actions $\mathbf{A}^t = (a_1^t, \ldots, a_N^t)$
    Obtain new observations $\mathbf{O}^{t+1}$ and rewards $\mathbf{R}^t$
    if $M < T_{stage}$ **then**
      Query expert actions $\mathbf{A}_e^t$ by current state $\mathbf{O}^t$
      Store ($\mathbf{O}^t$, $\mathbf{A}^t$, $\mathbf{A}_e^t$, $\mathbf{O}^{t+1}$, $\mathbf{R}^t$) into replay buffer D
    else
      Store ($\mathbf{O}^t$, $\mathbf{A}^t$, $\mathbf{O}^{t+1}$, $\mathbf{R}^t$) into replay buffer D
    end if
    $\mathbf{O}^t \leftarrow \mathbf{O}^{t+1}$
    if $M < T_{stage}$ **then**
      Sample a batch ($\mathbf{O}$, $\mathbf{A}$, $\mathbf{A}_e$, $\mathbf{O}'$, $\mathbf{R}$) from replay buffer D
    else
      Sample a batch ($\mathbf{O}$, $\mathbf{A}$, $\mathbf{O}'$, $\mathbf{R}$) from replay buffer D
    end if
    for Drone i = 1: N **do**
      Update critics by Eq. (5)
      if $M < T_{stage}$ **then**
        $Loss_{actor}^i = ||\mathbf{A}^i - \mathbf{A}_e^i||^2$
      else
        Update actors by Eq. (4)
      end if
    end for
    Update target networks
  end for
end for

As shown in Eq. (12), $\mathbf{A}^i = \boldsymbol{\mu}_i(o_i)$ is the action (velocity) generated by the actor neural network of agent i. $\mathbf{A}_e^i$ is calculated by the expert controller using the same observation input of agent i. To avoid the over-fitting problem, we firstly train the actor networks using behaviour cloning and then switch to the normal MADDPG training by a threshold stage T_{stage}. In Algorithm 2, before stage T_{stage}, both the action $\mathbf{A}_e$ generated by the expert controller and $\mathbf{A}$ generated by the actor are stored in the replay buffer and then sampled for supervised learning of actors. Furthermore, in the whole process, critic networks are always trained under the normal MADDPG.

4.3. Attention-based MADDPG and curriculum learning

In this section, we adapt the attention architecture of [25] to design the attention-based actors and critics for MADDPG and curriculum learning. In our method, each agent has its own actor and critic, and the population-invariance representation of attention is set before the decision layers in the neural networks. As shown in Fig. 5, for agent i, the actor and critic nets are represented by,

$$\boldsymbol{\mu}_i(o_i) = h^i(v_t^i, v_g^i, f_1^i(o_s^i)) \tag{13}$$

$$Q_i(o_1, \ldots, o_N, a_1, \ldots, a_N) = g_i(q_i(o_i, a_i), e_i) \tag{14}$$

For the actor $\boldsymbol{\mu}_i$ ((1) of Fig. 5), the input observations o_i are classified into self-entity observation o_s^i, other-entities observation $o_t^i = \{o_{t1}^i, o_{t2}^i, \ldots, o_{tM}^i\}$ and goal observation $o_g^i = \{o_{g1}^i, o_{g2}^i, \ldots, o_{gN}^i\}$ where o_s^i is related to its own position and velocity, o_t^i is the set of relative positions and velocities of other agents, and o_g^i is the set of relative positions of all grasping points. As the number of grasping points is equal to the number of agents, thus $M = N - 1$. These observations are firstly encoded by two-hidden-layer networks f_1, f_2 and f_3 respectively with ReLU activation functions. Then, attention embeddings v_t^i and v_g^i of other-entities and goals are calculated by the Scaled Dot Product between self-entity embedding and its corresponding keys,

$$\begin{aligned} v_t^i &= attention(f_1^i(o_s^i), f_2^i(o_t^i)) \\ v_g^i &= attention(f_1^i(o_s^i), f_3^i(o_g^i)) \end{aligned} \tag{15}$$

Then, the attention outputs are followed by operations of the batch norm. Finally, a four-hidden-layer neural network h outputs either actions or state action values. Therefore, the linear velocities of a UAV are limited to (−1,1) m/s using the tangent function (Tanh). Furthermore, action inputs are combined with self-entity inputs to generate state action value for inputs of a critic net.

For the critic Q_i ((2) of Fig. 5), (o_i, a_i) is the state action pair of agent i and $(o_t^j, a_t^j)_{j=1:M}$ are state action pairs of other agents. These observation action values are firstly computed by actor nets. Then, they are processed by a self-attention module q followed by another attention module between self-attention embedding of agent i and that of other agents. The attention embedding e_i is calculated by,

$$\begin{aligned} e_i &= attention(q(o_i, a_i), q(o_t^{j=1:M}, a_t^{j=1:M})) \\ &= \sum_{j=1:M} \epsilon_{ij} q(o_t^j, a_t^j) \end{aligned} \tag{16}$$

where the attention weight $\epsilon_{ij} = softmax(\phi_{ij})$ and $\phi_{ij} = q(o_i, a_i)\, q(o_t^j, a_t^j)^T / \sqrt{d_k}$. d_k is the first dimension of $q(o_i, a_i)$. Finally, all embeddings are merged in a three-hidden-layer g to output a Q-value for a critic net of agent i. As shown in Eq. (16), the attention embedding with the fixed dimension is the weighted sum of all

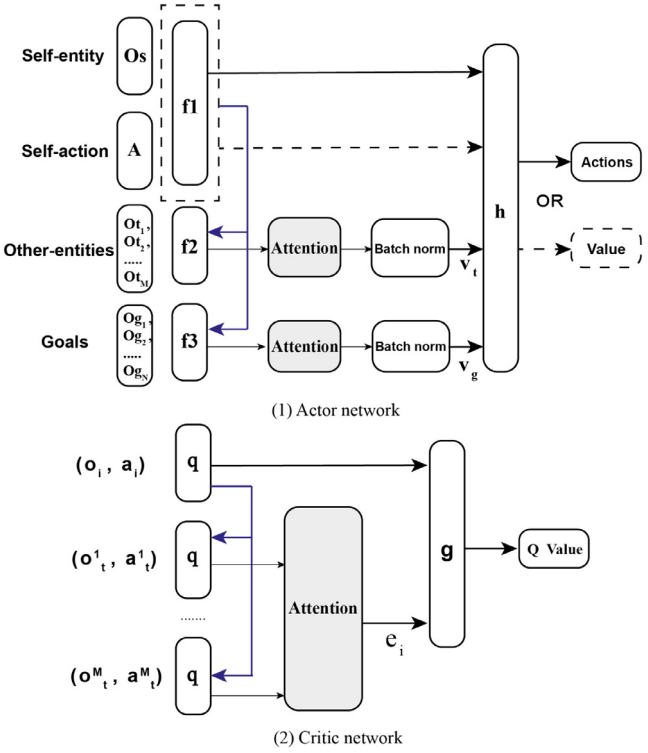


Fig. 5. The attention-based actor–critic framework.

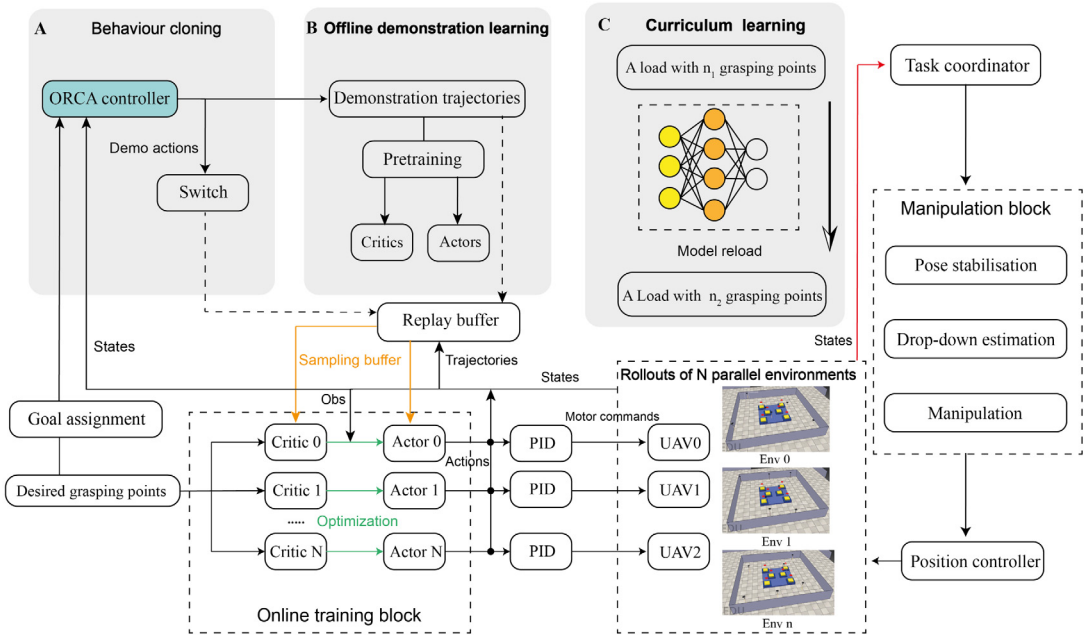


Fig. 6. Overview of the MADDPG algorithm with demonstration learning and curriculum learning.

observation-action embeddings of other agents. Therefore, the change in the number of agents will not influence the overall architecture of the neural networks. Furthermore, the attention mechanism can help the actor to find a better representation of the observation by analysing the relation between its own property and other entities, and the efficiency of the critic can be improved when it learns the centralized Q-function. The complete details of our establishment of neural networks are shown in Appendix B.7. The training process of the attention-based MADDPG is similar to that of the vanilla MADDPG. The main difference is that attention mechanism is added to the actors and critics to selectively attend to the relevant information.

Another function of the attention mechanism is to achieve the dimension-invariant architecture of our model which can be directly reloaded into the new stage without modification in the curriculum learning. The empirical experiments help us find that the performance of attention-based critics is limited. Thus, we remove the state–action values of other agents in the critics and only use the attention-based actors to create a shared att-DDPG method for better knowledge transfer. In curriculum learning, the training process is divided into several stages, with each stage gradually increasing the complexity of the task by adding more grasping points. The strategy is to first train the system on a simpler task with fewer grasping points, and then use the trained model as a starting point to train on a more complex task with more grasping points. The final parameters of the shared actor and their target networks are stored after each stage, and then reloaded to train the next stage with more grasping points. The number of stages is not limited and can be increased as needed for scenarios with more grasping points.

4.4. System overview

In Fig. 6, we demonstrate our hierarchical system based on the actor–critic framework and PID controllers. The aerial manipulation task is accomplished by the navigation and manipulation in a state machine. The navigation is achieved by multi-agent reinforcement learning and manipulation is achieved by PID position controllers. For the multi-agent system, each UAV (agent) has its own actor and critic in the decentralized execution and centralized training fashion. To improve the sample efficiency, we launch N parallel environments for collecting trajectories generated by the same policy networks, and all the trajectories are stored in the same replay buffer. The task coordinator monitors the environment states and switches to the manipulation phase once all grasping points are within the permitted range. During this phase, the UAVs first stabilize their pose to ensure the suction cups are in the correct position for manipulation. Next, drop-down positions are calculated to lift the load into the air by attaching the suction cup to the load.

5. Experiments and results

The main research problems to be explored in this section focus on two case studies: flexible navigation task and targeted navigation task. We will evaluate the performance of the demonstration learning in flexible navigation and curriculum learning in the targeted navigation respectively. The related algorithms and techniques are discussed and compared with the state-of-the-art methods to highlight the superior performance of our proposed framework. In this section, we will firstly give a description of the task and some implementation details. We briefly explore the hyper-parameters for better validation of the algorithms. Then, we will discuss the results of each experiment and compare the performance of different methods. Furthermore, a real-world pipeline to validate our algorithms with Crazyflies is shown. Finally, the related ablation experiments are conducted.

5.1. Task description

As shown in Fig. 7, we consider loads with three, four and six grasping points (centres of the yellow blocks) respectively. The loads to be transported are located in the centre of a 10 m $\times$ 10 m bounded environment. UAVs randomly initialized in the environment can move in three-dimensional space using low-level PID controllers. Before the training, the number of UAVs and grasping points are the same and UAVs need to cover all grasping points within the permitted deviation. When UAVs cover all the grasping points, they will keep that grasping formation and drop down to pick up the load using suction cups.

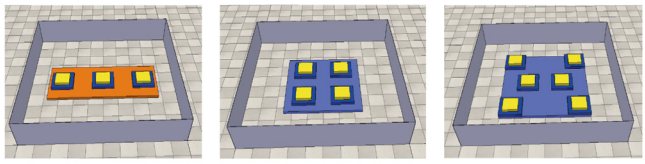


Fig. 7. Three different loads considered in our experiments.

5.2. Implementation specifications

Our simulation is implemented in Coppeliasim simulator with Pyrep [32]. The algorithm is trained on a PC (Alienware R15) with Intel-i7 and Nvidia 3070Ti using Pytorch. We set the Adam optimizer as default. The learning rate of actor and critic are set to 10^{-4} and 10^{-3} respectively. The discount factor γ is 0.95 and the size of a replay buffer is 5×10^5. Furthermore, 1024 transitions are sampled from the replay buffer for the training each time (The batch size of MADDPG with behaviour cloning is set to 256). We train 1500, 2000 and 3500 episodes for loads with three, four and six grasping points respectively. Besides, the timestep of each episode is set to 50, 60 and 70. During each episode, UAVs will take actions every 0.5 s. Before the start of one episode, UAVs firstly hover for several steps to stabilize the height before navigation.

5.3. Hyper-parameter exploration

We conducted experiments to explore suitable hyper-parametres for our algorithm. Specifically, we investigated the observation range for each UAV and the threshold stage T_{stage} of behaviour cloning. The goal was to identify hyper-parameter settings that strike a balance between feasibility and efficiency, while improving the performance of the algorithm.

5.3.1. Observation range

In our cases, the sensing thresholds of a UAV are set to 2 m, 4 m and full range respectively in a 10 m $\times$ 10 m field. The neighbouring objects (including UAVs and grasping points) which exceed this sensing distance will not be detected by a UAV. Thus, we make modifications on the full observation space $\boldsymbol{o}_i = [o_s^i, o_t^i, o_g^i]$ given in Section 4.1. Local observation space becomes $\boldsymbol{o}_i^{local} = [o_s^i, o_t^i, o_{tl}^i, o_g^i, o_{gl}^i]$. The newly added o_{tl}^i and o_{gl}^i represent the life of its neighbour UAVs and grasping points respectively. When one neighbour UAV is not detected, o_{tl}^i and its related position and velocity o_t^i are set all zeros. Otherwise, o_{tl}^i is set to 1. Furthermore, o_{gl}^i and o_g^i are set zeros when the grasping point is absent within its sensing range. We examine the performance of local and full observation in the scenario with 3 grasping points and 3 UAVs. To remove the influence of the demonstration, only vanilla MADDPG and att-MADDPG are tested. For att-MADDPG, o_{tl}^i and o_{gl}^i are incorporated into other-entity observation and goal observation respectively in the attention framework. As shown in Fig. 9, learning curves for UAVs with different observation ranges in MADDPG and att-MADDPG methods are demonstrated. All curves are calculated by three times evaluation.

Regarding the performance of MADDPG, it was found that none of the algorithms were successful, except for the method with the full observation range which showed a higher reward between episodes 50 and 100. In contrast, for att-MADDPG, the method with an observation range of 2 m failed to converge, while the methods with 4 m and full range were able to converge successfully. Notably, the full range method converged faster than the one with a 4 m range. In general, the performance of partial observation is inferior to that of full observation due to the policy's inability to receive sufficient information for decision-making in the multi-agent system. Therefore, the full observation is chosen for the design of all algorithms when the performance is evaluated in the later section.

5.3.2. Stage of behaviour cloning

We illustrate the behaviour cloning method for MADDPG in the previous section. The supervised optimization process will switch to the normal training of MADDPG when it approaches the threshold stage T_{stage}. However, how the choice of T_{stage} will influence the performance of our algorithm has not been explored. Here, we set T_{stage} to be 100, 200 and 400 timestep for the scenario with three UAVs and 200, 500 and 700 timestep for the case with four UAVs, and 1500, 2000 and 3500 steps for the case with six UAVs respectively. Fig. 8 demonstrates that increasing T_{stage} does not result in significant improvement of the final performance for the scenarios with three and four grasping points, although it reduces variance and slightly increases the reward. However, for the scenario with six grasping points, T_{stage} with small values like 1500 and 2500 fails to converge. Therefore, according to learning curves, we choose T_{stage} to be 400 for three points, 700 for four points and 3500 for six points respectively.

5.4. Flexible navigation by demonstration

In this section, we evaluate the performance of the demonstration-based algorithms we proposed in the flexible navigation. Our guided MADDPG algorithms are classified into various methods according to different strategies of demonstration. We compare them under different load scenarios with a behavioural swarm optimization algorithm and other model-free multi-agent reinforcement learning methods. The algorithms to be compared are defined as follows:

- **MADDPG**: Each agent has its independent actor and critic with the architecture of decentralized execution and centralized training.
- **att-MADDPG**: This is MADDPG with an attention mechanism.
- **bc-MADDPG**: MADDPG is firstly trained using behaviour cloning (bc), and then switched to the normal training.
- **demo-MADDPG**: Demonstration data are inserted in the prioritized replay buffer for pretraining with the attention framework.
- **MAPPO**: Multi-Agent Proximal Policy Optimization (MAPPO) is a distributed policy optimization algorithm designed for multi-agent systems [33].
- **Behavioural swarm optimization (BSO)**: Please see [7] for more information. In adapting [7] to the work in this paper, we added an additional shepherding behaviour to repel the other robots and removed the flocking behaviour to ensure multi-target tracking.

For MADDPG, the actor uses two hidden layers [64,64] with the activation function of Tanh, and the critic has the same configuration for hidden layers but with the activation function of ReLU. The att-MADDPG represents the actor and critic with attention framework shown in Section 4.3. Two demonstration methods in Section 4.2 generate two algorithms, bc-MADDPG and demo-MADDPG. However, bc-MADDPG does not utilize an attention framework. Policies of demo-MADDPG were trained using the parallel environment and attention structure. More details of the components of different algorithms used are illustrated in Table 1 where MG: **MADDPG**, att: **att-MADDPG**, bc: **bc-MADDPG** and demo: **demo-MADDPG**.

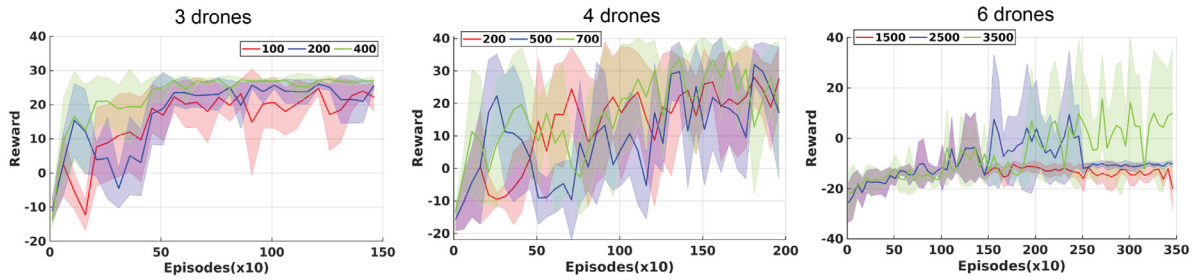


Fig. 8. Performance of different values for T_{stage}.

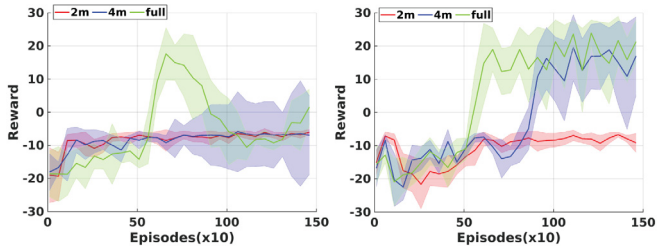


Fig. 9. Learning curves for different observation range with 3 UAVs. Left: MADDPG, Right: att-MADDPG.

Table 1
Components of different algorithms.

Components \ Methods	MG	att	bc	demo
Multi-agent	✓	✓	✓	✓
Parallel environment	×	×	×	✓
Attention mechanism	×	✓	×	✓
Behaviour cloning	×	×	✓	×
Demo buffer	×	×	×	✓

5.4.1. Evaluation of learning curves

In Fig. 10, learning curves of three algorithms (att-MADDPG, bc-MADDPG and demo-MADDPG) compared with baselines (MADDPG and MAPPO) under different scenarios are demonstrated. When we train our algorithms, the rewards are evaluated three times every 5 training episodes. We tested three scenarios as follows:

(a) A load with three grasping points: In this scenario, all the demonstration-based and attention-based algorithms successfully converged at a positive reward level but MADDPG and MAPPO failed to converge before 1500 training episodes. We also found that both bc-MADDPG and demo-MADDPG with demonstration learning converged faster than att-MADDPG. Moreover, bc-MADDPG had the smallest variance and oscillations after convergence compared with att-MADDPG (with the largest variance) and demo-MADDPG. It is observed that the learning-based methods were able to approach the reward level of demonstration as shown by the black dotted line.

(b) A load with four grasping points: In this scenario, we found that att-MADDPG could not converge and only demonstration-based methods (bc-MADDPG and demo-MADDPG) were able to. However, their convergence speed was slow and more variance was observed as the complexity of the task increased with more grasping points. We found that higher rewards were gained by the bc-MADDPG methodology when compared with the demo-MADDPG.

(c) A load with six grasping points: In this scenario, all the methods failed to finish the task except the bc-MADDPG and demo-MADDPG. Even for the demo-MADDPG, it only achieved positive rewards after 1800 episodes. We also found that demo-MADDPG achieves the best performance under the scenario with six UAVs and grasping points.

After convergence, we observed large oscillations in the curves, particularly in scenarios involving four and six grasping points. These oscillations were due to the drones' different starting positions and the fact that the reward function is related to the distance travelled. However, our focus was on maintaining a positive reward level, and the mean value of the reward function met this criterion (although oscillations could be reduced by increasing the success rate of the proposed methods).

5.4.2. Evaluation of optimal policies

To evaluate the performance of optimal policies of different algorithms obtained from the previous learning process, we test them in 20 repeated experiments for each scenario where the initialized positions of drones are randomly sampled. We record the following metrics during the cooperative navigation task: finished time, average speed, position deviation, success rate and total reward.

- **Finished time**: The time cost until all UAVs successfully finish coverage and hover for 10 steps.
- **Average speed**: The average speed of a UAV team during a successful coverage task.
- **Success rate**: Whether all UAVs have covered all grasping points within the permitted range while staying hovering for 10 steps.
- **Deviation**: Average position deviation between each drone and its corresponding grasping point during the last 10 steps.
- **Total reward**: Cumulative shared reward is measured during each episode.

The results are shown in Table 2 (The methods with blue colour are the baselines). The first three metrics are only calculated when the navigation task of UAVs is successfully finished (failed policies will have void values). For each evaluation episode, the positions of UAVs are randomly initialized and the successful condition is that UAVs covered all grasping points while keeping that formation for 10 steps within the required deviation (the evaluation episode is terminated after the successful condition). The permitted deviation range during the training is 0.5 metres.

(a) For three points, demo-MADDPG has the shortest finished time while BSO spends the largest time cost due to the random exploration. We can observe that the deviation is less than 0.3 metres, and the success rate is more than 80% for all algorithms. MAPPO and bc-MADDPG achieve the best performance with a large success rate and less deviation.

(b) For four points, the success rates of MADDPG and att-MADDPG are zero. Among all the algorithms evaluated in this scenario, bc-MADDPG outperforms the others with the highest success rate and the smallest position deviation. This indicates that the behavioural cloning approach has effectively improved the learning process.

(c) For six points, bc-MADDPG is still the best choice but the success rate of bc-MADDPG and demo-MADDPG is significantly

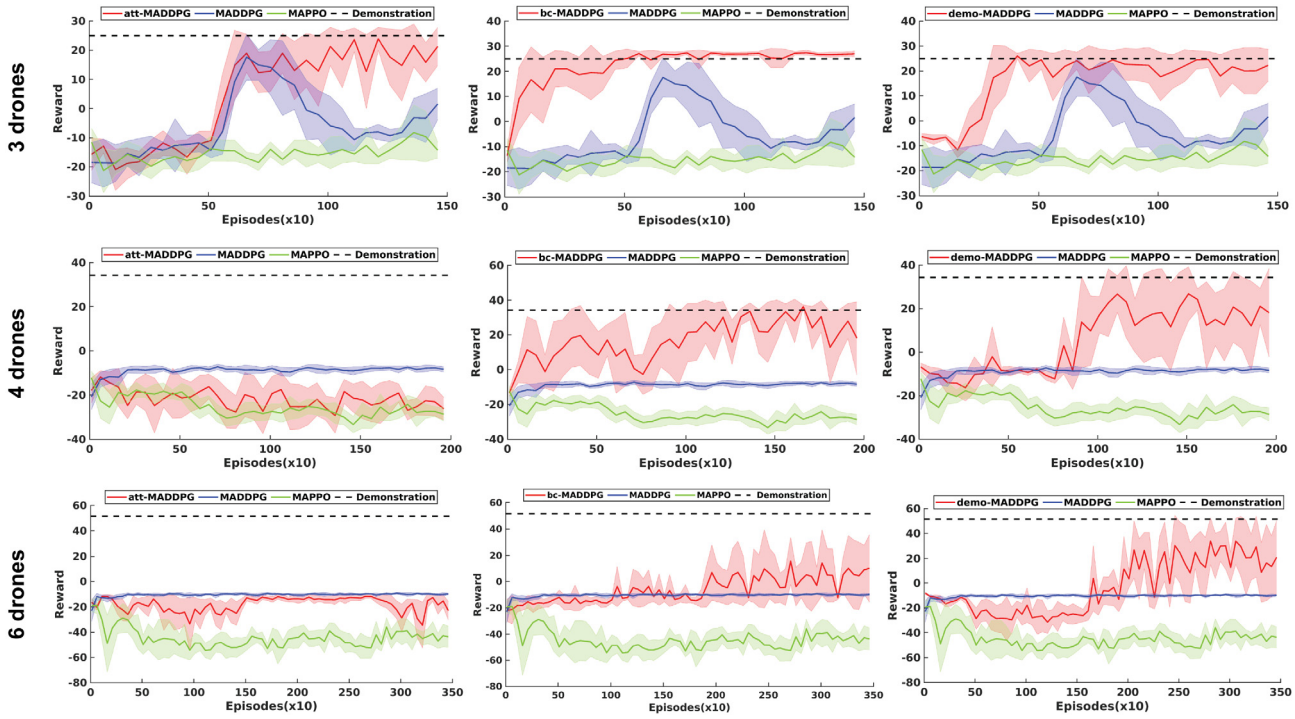


Fig. 10. Learning curves used for training different types of loads.

Table 2
Performance of simulation experiments. The blue texts represent the baseline methods.

Method	Number	Finished time	Average speed	Deviation	Success rate	Total reward
MADDPG	3	23.78 ± 2.27	0.51 ± 0.07	0.20 ± 0.03	90%	432.27
MAPPO	3	26.41 ± 3.10	0.26 ± 0.05	0.16 ± 0.04	85%	374.98
att	3	24.50 ± 3.26	0.42 ± 0.08	0.29 ± 0.02	80%	367.83
bc	3	20.44 ± 2.37	0.53 ± 0.07	0.18 ± 0.02	80%	383.60
demo	3	18.94 ± 2.04	0.45 ± 0.05	0.23 ± 0.03	90%	467.84
BSO	3	1189.06 ± 511.44	0.26 ± 0.08	0.26 ± 0.07	80%	−11115.56
MADDPG	4	–	–	–	0%	−287.62
MAPPO	4	27.2 ± 4.88	0.28 ± 0.01	0.21 ± 0.03	75%	435.07
att	4	–	–	–	0%	−488.61
bc	4	19.78 ± 2.95	0.47 ± 0.04	0.21 ± 0.04	90%	630.61
demo	4	22.15 ± 2.77	0.46 ± 0.05	0.26 ± 0.02	65%	404.66
BSO	4	1652.18 ± 908.55	0.24 ± 0.04	0.22 ± 0.05	85%	−14531.59
MADDPG	6	–	–	–	0%	−231.74
MAPPO	6	–	–	–	0%	−741.65
att	6	–	–	–	0%	−484.46
bc	6	21.75 ± 4.24	0.29 ± 0.03	0.11 ± 0.02	40%	308.34
demo	6	50.42 ± 10.86	0.37 ± 0.03	0.32 ± 0.02	35%	55.99
BSO	6	2360.50 ± 1318.12	0.15 ± 0.03	0.16 ± 0.04	50%	−26748.95

decreased to 40% and 35% due to the increasing complexity of the load. The BSO method shows a reasonably good performance with the highest success rate, but it has a longer finished time compared to other methods.

Overall, for all scenarios with different number of points, MADDPG with demonstration strategies tends to have better performance than algorithms without it. Demonstrations can aid the learning to converge even in scenarios with more grasping points. However, it also introduces more computation into the learning. The aforementioned discussion of learning curves shows that the attention mechanism can enable MADDPG to converge faster. Nevertheless, the results of metrics in att-MADDPG have not shown any further improvement compared with plain MADDPG under more challenging scenarios. According to Appendices A.2 and A.1, we find BSO takes a longer time to adapt to different scenarios but it does not need any training time. Moreover, MAPPO has good performance in scenarios with three and four grasping points, but its convergence time is longer.

To better understand the optimal policies, we visualize some trajectories from the evaluations in Fig. 12. From analysing all trajectories, it is evident that the optimal policies developed allow each drone to navigate towards goal positions while effectively avoiding collisions with other drones. However, since our focus is on maximizing the reward for maintaining the grasping formation after a successful coverage, not all trajectories necessarily result in the shortest path for coverage. The paths generated by bc-MADDPG and demo-MADDPG are close to the shortest path under different scenarios. Especially for bc-MADDPG, although the demonstrated trajectories used for behaviour cloning are not the shortest paths, the subsequent training of policies can still correct it and lead to relatively good path planning for all UAVs (highlighting the advantage of flexible navigation). Furthermore, it is apparent that BSO takes a longer travelling distance to find the grasping points compared to other methods.

We validate our algorithms using the Crazyflie 2.1 quadrotors in a bounded 3-metre by 3-metre workspace. The parameters of

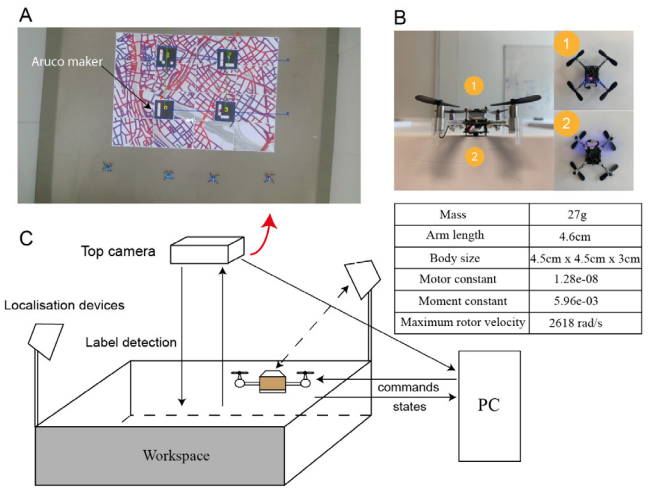


Fig. 11. Real world setup. A: A view from the top camera. Grasping locations are detected by the Aruco makers placed on the load. B: (1) is the positioning deck. (2) is the optical-flow deck consisting of a VL53L1x ToF sensor and a PMW3901 optical flow sensor. C: A brief diagram shows the control pipeline for our algorithm.

Crazyflie are shown in the table of Fig. 11. It weighs 27 g and can maximally fly for 10 min with a maximum 10 g payload. As for the small load ability, we only validate the navigation part before manipulation. As aforementioned, the observation of each UAV consists of its own positions and velocities. To give the accurate positions of UAVs, the Lighthouse localization camera and the positioning board from Bitcrazy company are utilized. The onboard positioning device is installed on the top of a Crazyflie as shown in number one of B of Fig. 11. The bottom board with number 2 is the optical flow board which is utilized for extracting velocity readings in the horizontal plane. To provide for the desirable positions of grasping points, we use the detection program from https://github.com/pal-robotics/aruco_ros to localize the Aruco makers on the load surface. A *Intel Realsense* camera is mounted on the top of the workspace to detect the makers and sends the positions of them to the central control unit. Then, these positions will be transformed into the inertial coordinate of the Lighthouse localization system. The communication and coding architecture is built on the open-source *Crazyswarm* ROS interface where the sensor data can be updated at 100 Hz. The optimal policies from different algorithms are trained in an official simulator from *Crazyswarm* to achieve the sim-to-real transfer.

As shown in (b) of Fig. 12, we test the policies in the scenarios of three and four UAVs (currently we only have four Crazyflies) and show the real-world trajectories for each Crazyflie in the last column. The first two rows try to reproduce the simulation results, and two different take-off points are validated in the last two rows. Due to the smaller size of the UAVs used in our experiments, we modified the success condition from a required precision of 0.5 m to 0.2 m. This implies that the task is complete when all the Crazyflies approach the areas that are within a 0.2-metre radius of their respective grasping points. As shown in Table 3, we observed that our proposed demonstration-based and attention-based methods perform better than the baseline MADDPG. The demo method achieved a minimal deviation of less than 0.1 cm. The bc method also achieved a good deviation, but it took more time to complete the task. We further validated the behaviour cloning method with four Crazyflies and found that the achieved deviation was less than 0.14 cm. The results obtained from our physical experiments not only matched the simulation results, but also demonstrated the ability of our methods to generalize to unseen take-off points.

Table 3
Performance of real-world experiments. The first four scenarios are with three drones and the last scenario is with four drones.

Method	Finished time	Average speed [m/s]	Deviation [m]
MG-3	579 ± 33	0.129 ± 0.001	0.168 ± 0.055
att-3	375 ± 142	0.119 ± 0.006	0.133 ± 0.010
bc-3	411 ± 68	0.118 ± 0.002	0.126 ± 0.041
demo-3	252 ± 16	0.111 ± 0.004	0.099 ± 0.040
bc-4	448 ± 198	0.115 ± 0.004	0.140 ± 0.015

Table 4
Evaluation of the reloaded architectures of the attention framework of the actor;the feasibility is validated in all scenarios.

Architecture	Feature encoders	Decision layers	Feasibility
1	Shared	not Shared	No
2	not Shared	Shared	No
3	not Shared	not Shared	No
4	Shared	Shared	Yes

5.5. Targeted navigation by curriculum learning

In this section, we explore and analyse the results of targeted navigation using curriculum learning in the scenarios of different number of grasping points. We adopt the same environment as that of the flexible navigation. In **Stage 1**, we consider an easy scenario with three grasping points. In **Stage 2**, we examine a more challenging scenario with four grasping points. Finally, in **Stage 3**, we test a load with six grasping points. In this case study, we assessed the effectiveness of attention frameworks in curriculum learning for targeted navigation (as mentioned in Section 4.3). Our empirical experiments revealed that the previous architecture of MADDPG (see Section 5.4), where each agent has its own actor and critic, is not valid for curriculum learning as the reloaded actors and critics do not facilitate the algorithm's learning (see (D) of Fig. 15). To address this issue, we validated our proposed shared att-DDPG and explored various embeddings of the attention framework of the actor (see (1) of Fig. 5) to determine which part of the reloaded model contributes the most to positive transfer. As summarized in Table 4, it indicates that sharing both the feature encoders ($f1$, $f2$, and f_3) and decision layers (h and g) among all agents during the curriculum training led to positive transfer learning.

We then compared the performance of targeted navigation with flexible navigation in all stages. For the shared att-DDPG, most parameters were kept the same as att-MADDPG. However, we slightly decreased the learning rate in stages 2 and 3 to ensure better stability, as using a large learning rate can destroy the reloaded model.

5.5.1. Comparison of learning curves

As shown in (a) of Fig. 13, in stage 1, we observed that targeted navigation converged faster than flexible navigation (almost 30 episodes early). This is likely due to the fact that goals were assigned for each agent using a lower dimensionality of feature space, thereby reducing the training complexity and eliminating the need for extra coordination among all grasping points. In stage 2, as shown in (b) of Fig. 13, compared to flexible navigation, which is trained from scratch, targeted navigation with model reloading was able to maintain a high reward level and further improve it after moving to a new stage. The flexible navigation achieved the highest reward until 1200 episodes. In stage3 ((c) of Fig. 13), the reloaded model for targeted navigation required approximately 50 episodes to adapt to the new stage, but still converged faster than flexible navigation, which converged in 2000 episodes. However, in all stages, the variance of targeted navigation was higher than that of flexible navigation. The reason

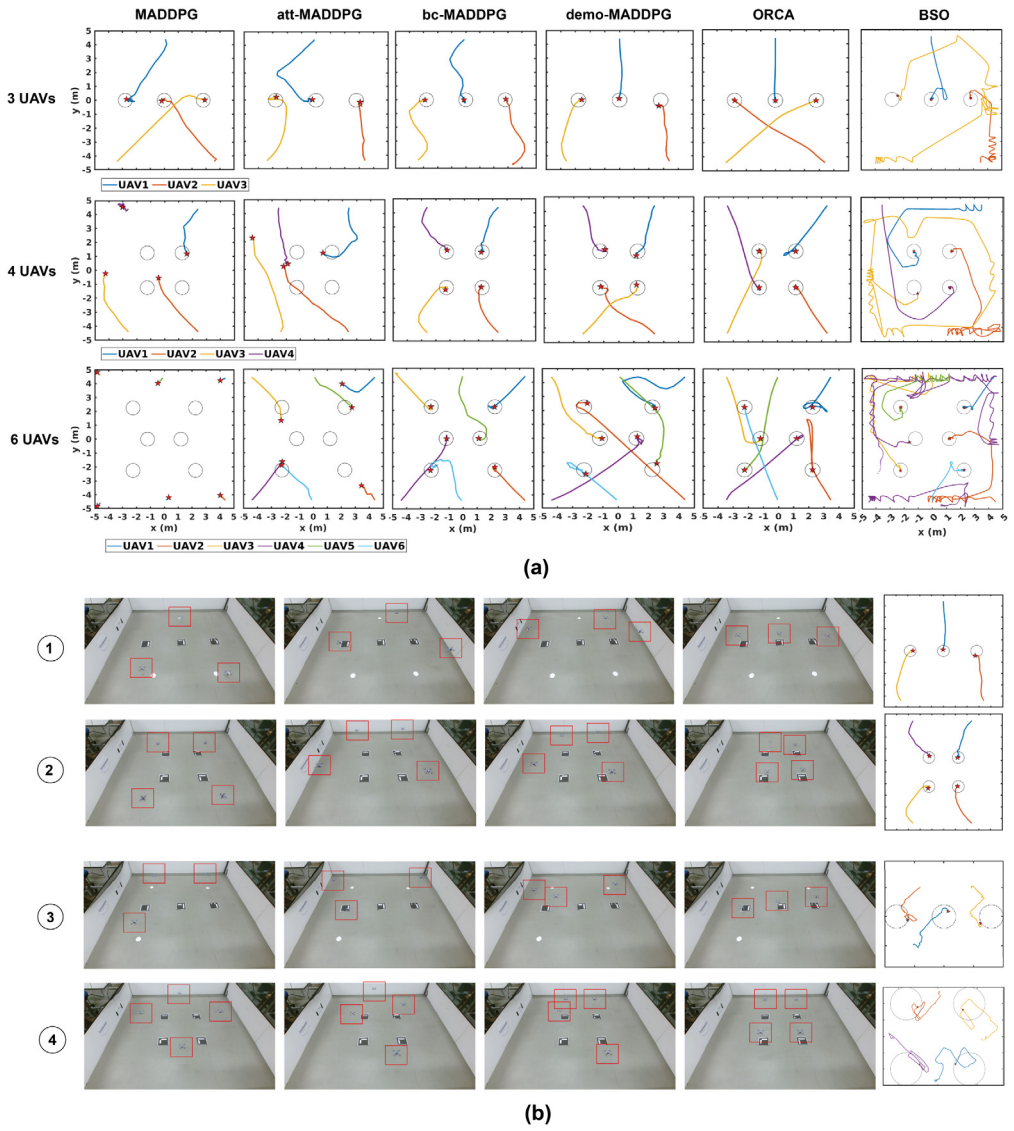
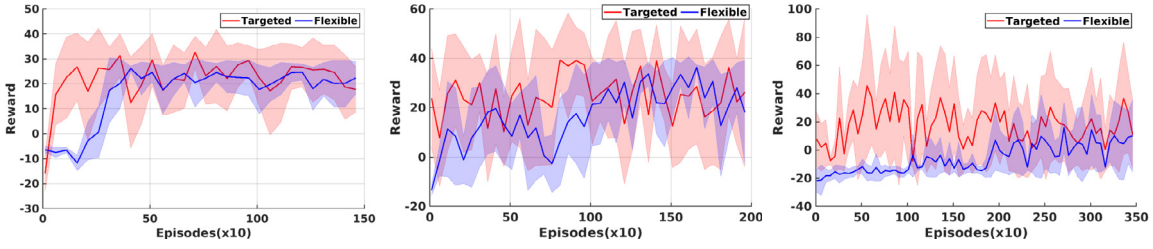


Fig. 12. Evaluation of trajectories with different methods and scenarios. (a) shows the results from simulation using different methods; For three points, UAVs are initialized at (0.0,4.5),(3.5,−4.5),(−3.5,−4.5) [m] respectively. For four points, UAVs are initialized at (3.5,4.5),(3.5,−4.5),(−3.5,−4.5),(−3.5,4.5) [m] respectively. For six points, UAVs are initialized at (4.5,4.5),(4.5,−4.5),(−4.5,4.5),(−4.5,−4.5),(0,4.5),(0,−4.5) [m] respectively. All positions are in the inertial coordinate. (b) shows the results from the real world using bc-MADDPG (Row 2,4) and demo-MADDPG (Rows 1,3).



(a) Stage 1 (b) Stage 2 (c) Stage 3

Fig. 13. Comparison of learning curves between targeted navigation and flexible navigation.

for this is that targeted navigation depends on the position it started from, while flexible navigation can better coordinate and generate new plans, resulting in lower variance.

5.5.2. Performance evaluation

Table 5 presents the performance evaluation of the two navigation tasks with random initialization of take-off positions. The metrics used are the same as in Section 5.4.2, and the results are based on 20 repeated experiments. It can be observed that flexible navigation tends to finish the navigation task faster (almost 2 time steps earlier than targeted navigation). However, the average speed of targeted navigation is faster than that of flexible navigation. The reason for this is that flexible navigation can always observe all grasping points and coordinate an optimal plan

Table 5
Performance of simulation experiments between targeted navigation and flexible navigation.

Method	Number	Finished time	Average speed	Deviation	Success rate	Total reward
Targeted navigation	3	20.22 ± 2.44	0.53 ± 0.06	0.17 ± 0.02	90%	208.41
Flexible navigation	3	18.94 ± 2.04	0.45 ± 0.05	0.23 ± 0.03	90%	467.84
Targeted navigation	4	21.07 ± 1.98	0.51 ± 0.05	0.16 ± 0.02	70%	135.64
Flexible navigation	4	19.78 ± 2.95	0.47 ± 0.04	0.21 ± 0.04	90%	630.61
Targeted navigation	6	23.28 ± 2.66	0.56 ± 0.06	0.22 ± 0.06	35%	88.91
Flexible navigation	6	21.75 ± 4.24	0.29 ± 0.03	0.11 ± 0.02	40%	308.34

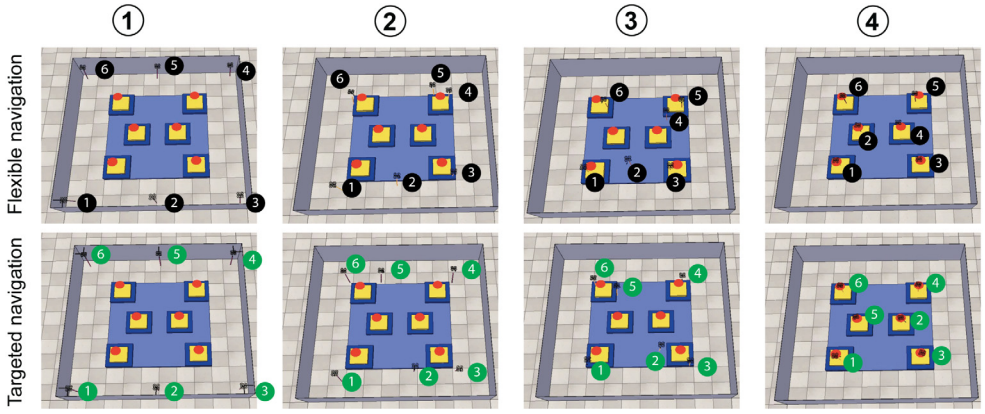


Fig. 14. Visualization of targeted navigation and flexible navigation. The solid circles represent the id of each UAV;hollow circles represent the sequence of the task from 1 to 4.

to reduce the time required to finish the task. Yet, the targeted navigation must follow a fixed assignment. In the scenarios with three and four points, the deviation of the targeted navigation is lower than that of the flexible navigation. The success rate of the two methods are almost the same. In Fig. 14, we visualized the trajectories of all UAVs using the same take-off points as in Section 5.4.2. For targeted navigation, we commanded UAV 1, 3, 4, and 6 to cover the outer four grasping points, and UAV 2 and 5 to cover the inner two grasping points. On the other hand, we can observe that flexible navigation automatically covered all grasping points with a reasonable assignment.

Overall, we find that the targeted navigation task is easier to train and its deviation is usually smaller than that of flexible navigation. However, flexible navigation often takes less time to complete the task by planning a more efficient route. In addition, flexible navigation can automatically cover all grasping points with a reasonable assignment, while targeted navigation requires a fixed assignment. Therefore, the choice between these two methods depends on the specific requirements and constraints of the task.

5.6. Navigation and load manipulation

In this section, we demonstrate how to use the navigation policy obtained from the previous training to achieve load manipulation. The scenario with six grasping points is tested by the policy of demo-MADDPG. The load weighing 532 grams to be manipulated by all UAVs is placed in the centre of a field. Each UAV weighing 512 grams with a cable-suspended suction cup can lift a load up to 80 grams (More details can be found in Appendix C). As shown in Fig. C.20 included in the appendix, UAVs firstly hover around the field. Then, they utilize their observations and the optimal control policies to achieve the coverage of all grasping points without collision. In this setup, the manipulation task is triggered by the task coordinator once the position deviation between each UAV and its corresponding goal position is less than 0.3 metres. During the task, the distance between a suction cup and a load is measured using a downward ultrasonic sensor mounted on each UAV. UAVs utilize the position PID controller to carefully lower themselves and attach their suction cups to the surface of the load. Finally, the load is lifted using the cooperative vertical force imposed by all UAVs simultaneously. The graph depicts that the pose change of the load is minimal during the manipulation, indicating that the UAVs are able to work together to maintain the load's position and orientation while it is being lifted.

5.7. Ablation experiments

In this section, we discuss the results of ablation experiments on the demonstration-based algorithm (see Section 5.4) and curriculum-based algorithm (see Section 5.5).

5.7.1. Demonstration-based algorithm

As shown in Table 1, as we combine several techniques into demo-MADDPG, we conducted ablation experiments on the priority buffer and attention framework respectively. The deployment of a prioritized replay buffer in demo-MADDPG increases the sampling probability of demonstration data when training the policy. To assess the benefits of using a priority-based sampling, we conducted an ablation experiment by replacing the prioritized replay buffer with a normal one. The results are shown in (A) of Fig. 15. In the scenario with three drones, the priority sampling enables faster convergence, while the buffer with random sampling fails to converge. However, in the scenario with four drones, there is no significant advantage of priority sampling. The buffer with priority sampling seemed to have higher rewards before 600 episodes, but the performance was not superior overall. As for the attention framework, in (B) of Fig. 15, it can be observed that the algorithm without attention framework cannot converge in the scenario with three grasping points. In the scenario with four grasping points, the attention framework led to a faster convergence speed (around 60 episodes earlier than the algorithm without attention). Overall, these results suggest that the attention framework can improve the training efficiency and effectiveness of demo-MADDPG, especially in more complex scenarios.

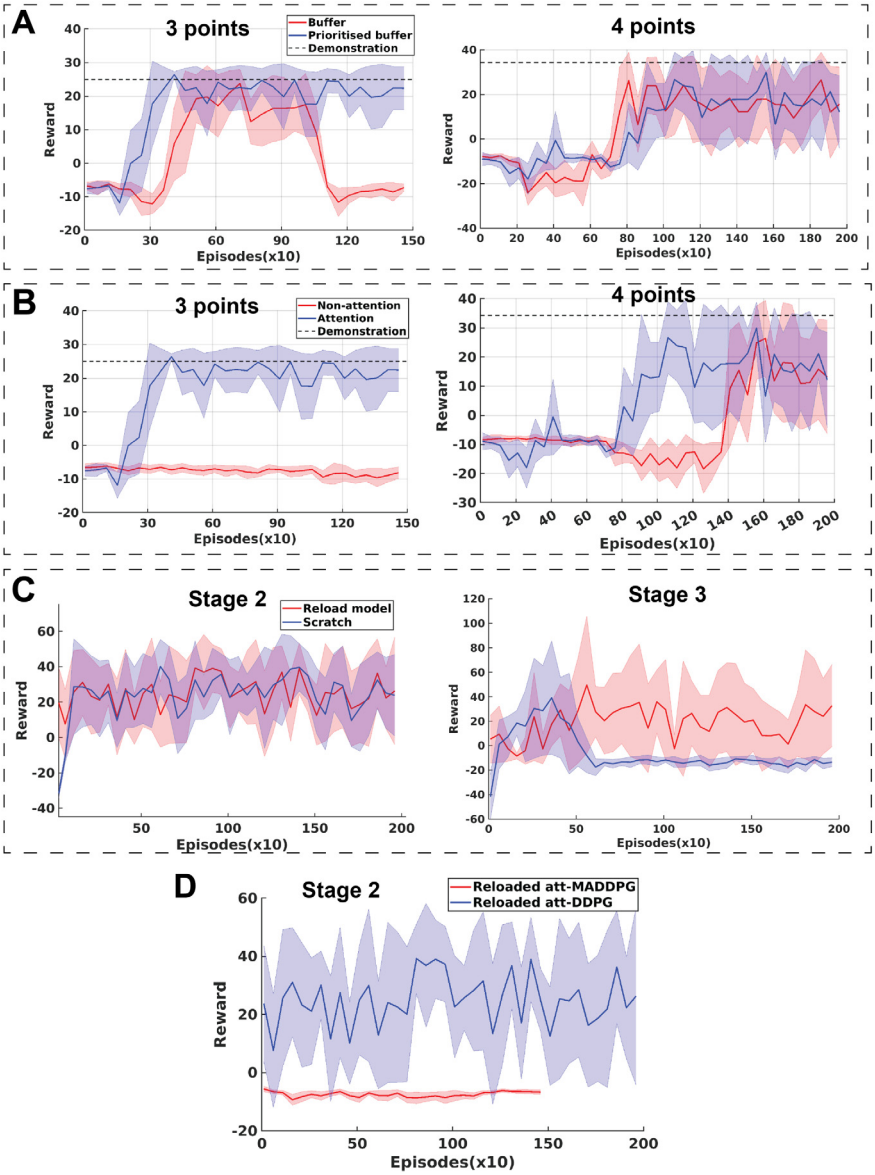


Fig. 15. Results of the ablation experiments. A: ablation analysis on the priority sampling of demo-MADDPG. B: ablation analysis on the attention framework of demo-MADDPG. C: ablation analysis on applying model reloads in curriculum learning. D: Performance between att-DDPG and att-MADDPG in stage 2.

5.7.2. Curriculum-based algorithm

(C) of Fig. 15 illustrates a performance comparison between the reloaded model and the training from scratch in Stage 2 and 3 of att-DDPG. The results show that the model reload has limited benefit in Stage 2, but in the more challenging Stage 3 with six grasping points, the model reload is able to successfully converge. Additionally, we evaluated the performance of att-MADDPG and att-DDPG when reloading the model from Stage 1 to Stage 2 as part of our exploration of curriculum learning structure. Our findings indicate that while att-MADDPG cannot converge, att-DDPG is able to successfully converge at a higher reward level.

6. Conclusion

In this paper, we propose a control system based on multi-agent reinforcement learning for aerial navigation and manipulation by UAVs targeting at flexible and targeted navigation tasks. The two demonstration techniques inspired by ORCA method are proved to be effective for scenarios with high-dimensional state space. Besides, the obtained optimal policies enable up to six UAVs to find collision-free paths to achieve the coverage of the grasping points with a reasonable success rate. In order to improve the learning efficiency, we incorporate an attention framework into MADDPG and use curriculum learning to realize the knowledge reuse from a few grasping points into more points under the aerial navigation task. The preliminary results show that based on our framework, the previous experience of a simple load can facilitate the training of a more complicated load. Though we achieved learning from a simple to complex case, our results show that in some cases, the success rate for complex loads is not high enough. Furthermore, we were not able to achieve a high coverage ability with precision when compared with some traditional methods. Nevertheless, our adaptive approach does not rely on prior known models, or high-level path planning and provides the first step for the learning-based application of aerial navigation and manipulation.

In our work, our experiments were set up in an indoor environment with the inertial coordinate created using global information. In translating to outdoor use, one approach would be to use a combination of GPS and other sensors such as accelerometers, gyroscopes and magnetometers to establish an inertial

Table A.6
The number of total parameters for different MARL algorithms.

MADDPG,bc-MADDPG	**3 points**	**4 points**	**6 points**
Total of parameters	43910	50310	67718
att-MADDPG,demo-MADDPG	**3 points**	**4 points**	**6 points**
Total of parameters	26118	26118	26118
MAPPO	**3 points**	**4 points**	**6 points**
Total of parameters	13446	16518	24966

Table B.7
Structure of neural networks with attention mechanism.

Encoders	Structure
$f_1(action)$	[4,32], ReLU, [32,16], ReLU
$f_1(value)$	[6,32], ReLU, [32,16], ReLU
f_2	[4,32], ReLU, [32,16], ReLU
f_3	[2,32],ReLU,[32,16],ReLU
$h_2(action)$	[48,32], LeakyReLu, [32,32], LeakyReLU,[32,2], Tanh
$h_2(value)$	[48,32], ReLU, [32,32],ReLU, [32,16]
q	[16,16]
g	[32,32], LeakyReLU,[32,32],LeakyReLu, [32,1]

coordinate frame. The GPS would provide the absolute position and orientation of the drone with respect to the Earth's coordinate system, while the other sensors would provide additional information about the drone's movement and orientation relative to its local environment. Furthermore, the above can be augmented by identifying and using a suitable reference point or landmark in the environment, such as a building or a tree. Translating the work we have done in this work into unstructured environments would be the focus of future work.

Data availability

No data was used for the research described in the article.

Acknowledgement

We would like to acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC), United Kingdom funding: NanoMAN (EP/V055089/1), for the work carried out in this manuscript.

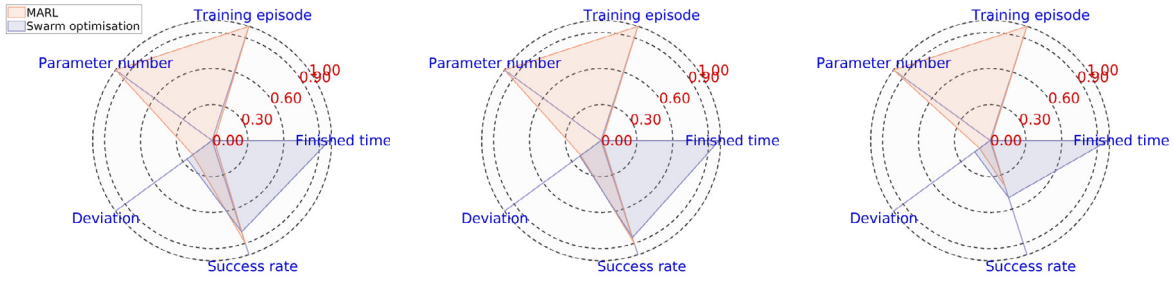
Appendix A. Supplementary materials

A.1. Swarm optimization method

Table B.8 shows the comparison between our proposed method and a behavioural swarm optimization (BSO) algorithm [7] that we modified by adding an additional shepherding behaviour to repel other robots and removing the flocking behaviour to ensure multi-target coverage. We evaluated both methods using the same environment and tasks as described in this work, using the same metrics (**Finished time** calculates the time cost in the test time and **Training episode** records the time cost in the training time), and compared it with the best MARL algorithm of Section 5.4. The results show that both methods have similar deviation and success rates. However, the BSO algorithm has fewer parameters and does not require training time, whereas the MARL algorithm requires training time but can be deployed more quickly. We normalized the metrics and plotted them on radar charts for each scenario in Fig. B.16 . From the all Tables in this part, it will be seen that the concept of no-free lunch applies. The MARL methodology takes time to train but is faster at finishing the task during runtime while the situation is the opposite for the swarm optimization algorithm. We also recorded the number of parameters for all MARL models and scenarios to evaluate model complexity (see Table A.6). It can be observed that the number of parameters increases with the number of grasping points. For att-MADDPG, the number of parameters remains constant as the attention framework can handle varying input dimensions.

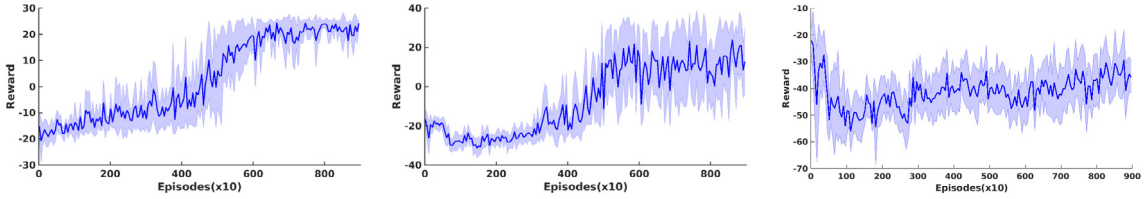
A.2. MAPPO

MAPPO extends the original PPO algorithm to the multi-agent case by incorporating a centralized critic network that can



(a) Three grasping points (b) Four grasping points (c) Six grasping points

Fig. B.16. Visualization of metrics between MARL (red colour) and swarm optimization (blue colour) in the radar charts. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) Learning curve of 3 drones (b) Learning curve of 4 drones (c) Learning curve of 6 drones

Fig. B.17. Learning curves of MAPPO in all scenarios.

Table B.8
Performance comparison between MARL and BSO in all scenarios.

Method	Number	Finished time	Training episodes	No of parameters	Deviation	Success rate
MARL	3	23.78	1500	43910	0.2	90%
BSO	3	1189.06	0	10	0.26	80%
MARL	4	19.78	2000	50310	0.21	90%
BSO	4	1652.18	0	10	0.22	85%
MARL	6	21.75	3500	67718	0.11	40%
BSO	6	2360.50	0	10	0.16	50%

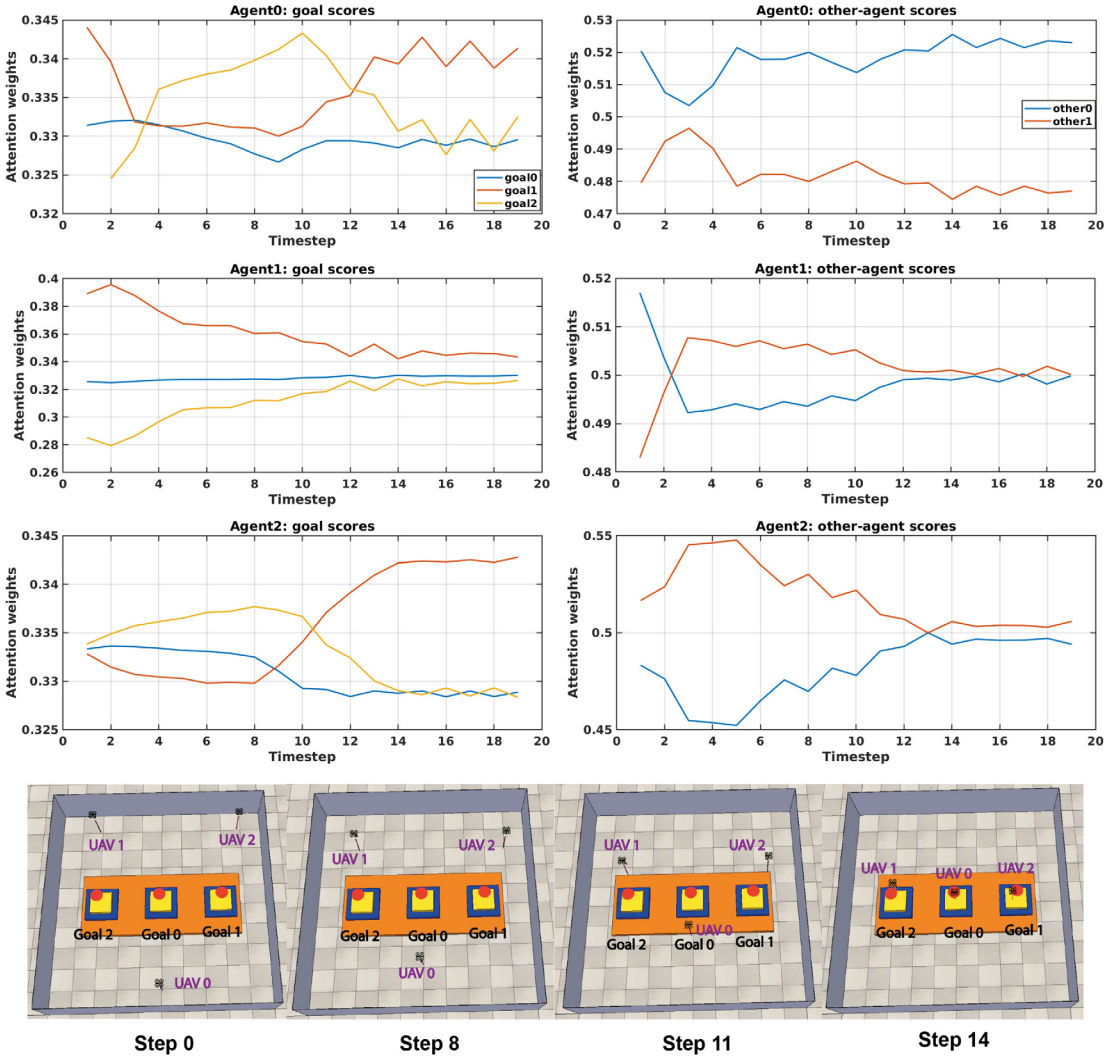


Fig. B.18. Attention scores for demo-MADDPG in the scenario of a load with three grasping points.

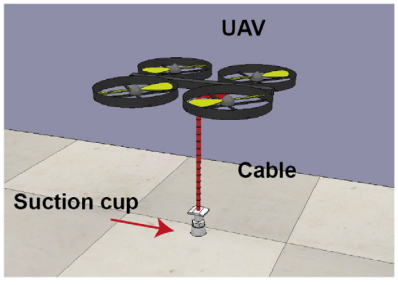


Fig. C.19. The simulation model of our proposed UAV.

estimate the state-value function for all agents. We utilized code from https://github.com/Lizhi-sjtu/MARL-code-pytorch/tree/main/1.MAPPO_MPE to implement MAPPO in our scenarios. However, we found that the algorithm did not converge in continuous action even after 9000 iterations. Therefore, we changed the continuous action of drones to discrete actions (forward, backwards, left, right, and stay) with a velocity of 0.5 m/s per action. We set the total number of training episodes to 9000. Fig. B.17 shows that MAPPO was able to converge to positive rewards in scenarios with three and four UAVs, but failed to converge with six UAVs. Furthermore, we found that MAPPO requires more than four times the number of training episodes to converge compared to our proposed methods, making it a slow process.

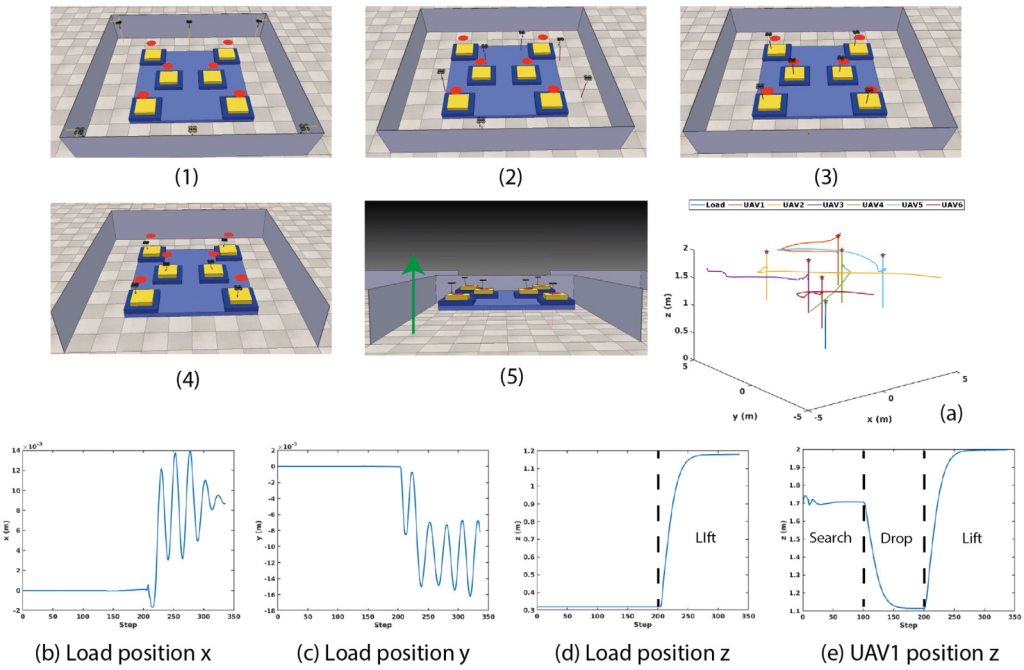


Fig. C.20. Overview of our aerial manipulation using reinforcement learning for navigation and PID controller for manipulation. (1) Start (2) UAVs coordinate and search the grasping points (3) All grasping points have been covered (4) UAVs drop down to attach their suction cups to the load (5) A load is lifting by all UAVs.

Appendix B. Attention analysis

We evaluate the attention scores of demo-MADDPG in the scenario with three UAVs. The attention score is calculated by a scaled dot product, which indicates the relationship between the query and key. In our scenario, it reveals the weight change between the agent's own observation and its corresponding keys like other-entity observations and goal observations. For example, goal scores of agent i are calculated by the attention between the self-observation o_s^i of UAV i and its goal observations o_g^i. Fig. B.18 illustrates how the weights change during a successful coverage task. The number of each goal is indicated in the bottom snapshots. The weight number of other–other agent scores relate to its neighbour UAVs. The results show the scores of different entities for each agent are not the same during the task. They keep changing in a small range and stabilize when all grasping points have been covered. We also find the goal assignment inferred by the policy does not show much relationship with the corresponding goal scores (For example, although UAV1 capture goal 2, the goal score of goal 2 for UAV1 does not show any specific rules). We deduce that this is because of the centralized training of MADDPG as agents have to consider the observations of other agents to make decisions. Thus, the observation change of all grasping points matters rather than focusing on individual goals. For the other-agent scores calculated by the self-observation o_s^i and other-entity observation o_t^i, it is observed that UAV0 located in the middle of the field has two distinct values for UAV1 and UAV2. However, for the UAV1 and UAV2 which are located on the side of the field, their weights for other agents finally converge to the same value. The reason for this difference may relate to the position of a UAV. The distance between UAV0 and the other two UAVs is the same, thus, the importance of each other UAV cannot be ignored. On the contrary, for UA1, UAV2 is too far from it and its score of UAV2 can be approximately close to UAV0.

Moreover, the parameters of the neural networks we used in the attention framework are illustrated in Table B.7.

Table C.9
Properties of UAV.

Base mass (g)	120
Propeller body mass (g/per)	100
Total drone mass m_q (g)	520
Size (m)	0.3 × 0.3 × 0.02
Arm length (m)	0.13
force constant k	8.50643e−06
moment constant b	0.016
inertial I_x, I_y, I_z	0.007, 0.007, 0.012

Table C.10
Properties of the rope and loads.

Rope size (m)	0.01 × 0.01 × 0.025
Suction size (cm)	1.8855 × 1.8855 × 3.9118
Rope Mass (g)	36
Suction cup Mass (g)	25
Maximum lift capacity (g/per)	about 100
Payload1 Mass (g)	270
Payload1 size (m)	7.5 × 3.0 × 0.2
Payload2 Mass (g)	350
Payload3 size (m)	5.0 × 5.0 × 0.2
Payload3 Mass (g)	525
Payload3 size (m)	6.0 × 6.0 × 0.2
Cable length (m)	0.448

Appendix C. Details of UAVs in the simulator

See Figs. C.19 and C.20, and Tables C.9 and C.10

References

[1] F.A. Goodarzi, T. Lee, Stabilization of a rigid body payload with multiple cooperative quadrotors, J. Dyn. Syst. Meas. Control 138 (12) (2016).
[2] G. Loianno, V. Kumar, Cooperative transportation using small quadrotors using monocular vision and inertial sensing, IEEE Robot. Autom. Lett. 3 (2) (2017) 680–687.
[3] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, S. Levine, Model-based meta-reinforcement learning for flight with suspended payloads, IEEE Robot. Autom. Lett. 6 (2) (2021) 1471–1478.

[4] I. Palunko, A. Faust, P. Cruz, L. Tapia, R. Fierro, A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots, in: 2013 IEEE International Conference on Robotics and Automation, IEEE, 2013, pp. 4896–4901.
[5] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, J. Pan, Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 6252–6259.
[6] J. Zhi, J.-M. Lien, Learning to herd agents amongst obstacles: Training robust shepherding behaviors using deep reinforcement learning, IEEE Robot. Autom. Lett. 6 (2) (2021) 4163–4168.
[7] K. Huang, J. Chen, J. Oyekan, Decentralised aerial swarm for adaptive and energy efficient transport of unknown loads, Swarm Evol. Comput. 67 (2021) 100957.
[8] R.N. Haksar, M. Schwager, Distributed deep reinforcement learning for fighting forest fires with a network of aerial robots, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2018, pp. 1067–1074.
[9] R. Polvara, S. Sharma, J. Wan, A. Manning, R. Sutton, Autonomous vehicular landings on the deck of an unmanned surface vehicle using deep reinforcement learning, Robotica 37 (11) (2019) 1867–1882.
[10] R. Lowe, Y.I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, Adv. Neural Inf. Process. Syst. 30 (2017).
[11] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P.H. Torr, P. Kohli, S. Whiteson, Stabilising experience replay for deep multi-agent reinforcement learning, in: International Conference on Machine Learning, PMLR, 2017, pp. 1146–1155.
[12] H. Mao, Z. Zhang, Z. Xiao, Z. Gong, Modelling the dynamic joint policy of teammates with attention multi-agent DDPG, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, 2019, pp. 1108–1116.
[13] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al., Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, in: International Conference on Machine Learning, PMLR, 2018, pp. 1407–1416.
[14] M. Hüttenrauch, S. Adrian, G. Neumann, et al., Deep reinforcement learning for swarm systems, J. Mach. Learn. Res. 20 (54) (2019) 1–31.
[15] D.A. Pomerleau, Alvinn: An autonomous land vehicle in a neural network, Adv. Neural Inf. Process. Syst. 1 (1988).
[16] J. Ho, S. Ermon, Generative adversarial imitation learning, Adv. Neural Inf. Process. Syst. 29 (2016).
[17] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al., Deep q-learning from demonstrations, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, No. 1, 2018.
[18] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, P. Abbeel, Overcoming exploration in reinforcement learning with demonstrations, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 6292–6299.
[19] S. Levine, A. Kumar, G. Tucker, J. Fu, Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020, arXiv preprint arXiv:2005.01643.
[20] J. Fu, K. Luo, S. Levine, Learning robust rewards with adverserial inverse reinforcement learning, in: International Conference on Learning Representations.
[21] T. Zhang, G. Kahn, S. Levine, P. Abbeel, Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search, in: 2016 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2016, pp. 528–535.
[22] D. Wang, T. Fan, T. Han, J. Pan, A two-stage reinforcement learning approach for multi-UAV collision avoidance under imperfect sensing, IEEE Robot. Autom. Lett. 5 (2) (2020) 3098–3105.
[23] W. Wang, T. Yang, Y. Liu, J. Hao, X. Hao, Y. Hu, Y. Chen, C. Fan, Y. Gao, From few to more: Large-scale dynamic multiagent curriculum learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, No. 05, 2020, pp. 7293–7300.
[24] J. Yang, A. Nakhaei, D. Isele, K. Fujimura, H. Zha, CM3: Cooperative multi-goal multi-stage multi-agent reinforcement learning, in: International Conference on Learning Representations.
[25] Q. Long, Z. Zhou, A. Gupta, F. Fang, Y. Wu, X. Wang, Evolutionary population curriculum for scaling multi-agent reinforcement learning, in: International Conference on Learning Representations.
[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. 30 (2017).
[27] S. James, A.J. Davison, Q-attention: Enabling efficient learning for vision-based robotic manipulation, IEEE Robot. Autom. Lett. (2022).
[28] S. Iqbal, F. Sha, Actor-attention-critic for multi-agent reinforcement learning, in: International Conference on Machine Learning, PMLR, 2019, pp. 2961–2970.
[29] Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, Y. Gao, Multi-agent game abstraction via graph attention neural network, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, No. 05, 2020, pp. 7211–7218.
[30] J.v.d. Berg, S.J. Guy, M. Lin, D. Manocha, Reciprocal n-body collision avoidance, in: Robotics Research, Springer, 2011, pp. 3–19.
[31] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, D. Silver, Distributed prioritized experience replay, in: International Conference on Learning Representations.
[32] S. James, M. Freese, A.J. Davison, Pyrep: Bringing v-rep to deep robot learning, 2019, arXiv preprint arXiv:1906.11176.
[33] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, Y. Wu, The surprising effectiveness of ppo in cooperative multi-agent games, Adv. Neural Inf. Process. Syst. 35 (2022) 24611–24624.

Jingyu Chen received the M.Eng. degree in electrical and electronic engineering from the University of Leicester and the M.Sc. degree in robotics from The University of Sheffield, where he is currently pursuing the Ph.D. degree with the Department of Automatic Control and Systems Engineering. His research interests include swarm robotics, swarm intelligence, and reinforcement learning.

Ruidong Ma received the B.Eng. degree in electrical and electronics engineering from the University of Liverpool and the M.Sc. degree in human and biological robotics from Imperial College London. He is currently pursuing the Ph.D. degree with the Department of Automatic Control and Systems Engineering, The University of Sheffield. His research interest includes human– robot-collaboration in complex manual manufacturing process.

John Oyekan received the M.Sc. degree in robotics and embedded systems and the Ph.D. degree in computer science and electronic engineering from the University of Essex. He is currently a Lecturer in digital manufacturing with the Department of Automatic Control and Systems Engineering, The University of Sheffield. Prior to The University of Sheffield, he was an Engineer at the Manufacturing Technology Centre, Coventry, where he developed software architectures and algorithms for autonomous systems. He has over 30 publications in the areas of swarm robotics, manufacturing informatics, bio-inspired algorithms, and sensing.