Cognitive Robotics 3 (2023) 226–256

KeAi
CHINESE ROOTS
GLOBAL IMPACT

Contents lists available at ScienceDirect

Cognitive Robotics

journal homepage: http://www.keaipublishing.com/en/journals/cognitive-robotics/

Reinforcement learning for swarm robotics: An overview of applications, algorithms and simulators

Marc-Andrė Blais*, Moulay A. Akhloufi

Perception, Robotics, and Intelligent Machines (PRIME), Department of Computer Science, Université de Moncton, Moncton, NB, Canada

ARTICLE INFO

Keywords:
Swarm robotics
Reinforcement learning
Intelligent systems
Simulators
Drones

ABSTRACT

Robots such as drones, ground rovers, underwater vehicles and industrial robots have increased in popularity in recent years. Many sectors have benefited from this by increasing productivity while also decreasing costs and certain risks to humans. These robots can be controlled individually but are more efficient in a large group, also known as a swarm. However, an increase in the quantity and complexity of robots creates the need for an adequate control system. Reinforcement learning, an artificial intelligence paradigm, is an increasingly popular approach to control a swarm of unmanned vehicles. The quantity of reviews in the field of reinforcement learning-based swarm robotics is limited. We propose reviewing the various applications, algorithms and simulators on the subject to fill this gap. First, we present the current applications on swarm robotics with a focus on reinforcement learning control systems. Subsequently, we define important reinforcement learning terminologies, followed by a review of the current state-of-the-art in the field of swarm robotics utilizing reinforcement learning. Additionally, we review the various simulators used to train, validate and simulate swarms of unmanned vehicles. We finalize our review by discussing our findings and the possible directions for future research. Overall, our review demonstrates the potential and state-of-the-art reinforcement learning-based control systems for swarm robotics.

1. Introduction

In recent years, there has been a significant increase in the number of robots, including industrial robots [1,2] and drones [3]. This rise has benefited various sectors and enabled improved safety, productivity and efficiency. For example, robotics technologies allow for a safer workplace by reducing human involvement in dangerous tasks such as moving large parts [4]. In the automotive sector, humans and robots can cooperate in an assembly line to increase production speed. Furthermore, repetitive tasks can be automated in industries to reduce costs while also increasing time efficiency [5]. The electronic manufacturing industry benefits from automation by enabling quick manufacturing, transportation and assembly of parts [6]. Robots, such as drones or quadcopters, can be utilized in search and rescue missions to autonomously navigate large regions. Swarms of drones are particularly useful in harsh or unknown environments like open seas [7] or during natural disasters [8]. Ground vehicles, like automated rovers, are advantageous in various applications such as logistics [9] and reconnaissance [10]. Additionally, ground robots can be combined with Unmanned Aerial Vehicles (UAVS) to achieve better cooperation towards a common goal [11]. However, controlling large swarms of hundreds to thousands of robots necessitates a complex control system, especially considering the expected tenfold increase in the stock of robots by the end of the decade [12].

Artificial intelligence (AI) has shown remarkable results across various applications [13–15]. It is beneficial in tasks such as flaws detection in parts [16], image classification [17] and medical anomaly detection like tumors [18]. Reinforcement learning, an AI

* Corresponding author.
E-mail addresses: emb9357@umoncton.ca (M.-A. Blais), moulay.akhloufi@umoncton.ca (M.A. Akhloufi).

https://doi.org/10.1016/j.cogr.2023.07.004
Received 24 June 2023; Received in revised form 12 July 2023; Accepted 12 July 2023
Available online 17 July 2023

paradigm, utilizes a reward system to iteratively guide the actions of an agent in an environment. It has found success in diverse applications, from video games [19] to industrial robots [20,21] and more [22]. Swarm robotics (SR) involves the decentralized and autonomous control of a group of robots using algorithms such as reinforcement learning. The behavior of each robot is self-controlled based on information such as the positions of other robots and obstacles. Swarms are able to efficiently navigate known or unknown environments by learning the desired individual and group behavior. Swarm robotics can be described as follows:

- Autonomous and simple robots
- Robots interact and modify the environment
- Cooperation towards common goal
- Limited global information and control

Despite the increasing popularity of swarm robotics and reinforcement learning, there is a limited quantity of reviews specifically focused on utilizing reinforcement learning for swarm robotics. Some reviews have explored multi-agent reinforcement learning (MARL) algorithms for cooperative, competitive or mixed scenarios games [23]. Others have focused on deep reinforcement learning for multi-agent systems [24] or theoretical results in MARL algorithms for different types of games [25]. Additionally, classical learning and non-learning methods for swarm robotics have been explored in earlier studies [26–31]. This lack of studies specifically addressing reinforcement learning in swarm robotics creates an opportunity for an in-depth review of the subject. In this review, our objective is to address this research gap by accomplishing three specific goals:

- Conducting a comprehensive review of the various applications of swarm robotics, encompassing a range of purposes from environmental to military applications. Additionally, we explore the utilization of reinforcement learning in the context of these applications.
- Presenting an in-depth study of the current state-of-the-art techniques used in swarm robotics, with a specific focus on reinforcement learning. We will analyze and compare different algorithms to provide insights into their advancements and limitations.
- Conducting an overview of different simulators available for swarm robotics, particularly those that enable the control and simulation of multiple robots using reinforcement learning. We explore the various aspects of these simulators, such as the different types of sensors, robots and parameters available.

Our review is divided as follows. Section 2 introduces the various applications of swarm robotics and their connection to reinforcement learning. Section 3 provides a brief introduction to the concept of reinforcement learning while Section 4 reviews the reinforcement learning algorithms used in swarm robotics. Section 5 refers to an overview of the different simulators suitable for swarm robotics with reinforcement learning. In Section 6, we provide a detailed discussion on the reviewed applications, algorithms and simulators. Finally, Sections 7 and 8 respectively provide the possible directions of future research and conclusion to our review.

By addressing these objectives, we aim to contribute to the advancement of swarm robotics research based on reinforcement learning. Furthermore, we intend to provide a concise overview of the field to individuals with a general interest in the subject and researchers specialized in this domain.

2. Applications

This section aims to present the various applications in which swarm robotics, combined with reinforcement learning, can be beneficial. Multiple studies have previously been conducted to review the various applications and tasks of swarm robotics [30,32,33]. Additionally, Dorigo et al. [34] reviewed the challenges and applications of swarm robotics, while Chung et al. [35] specifically focused on aerial swarm robotics. However, there is currently a lack of literature reviews on the combination of swarm robotics and reinforcement learning, hence our proposed review. This section of our study will explore a few applications that can benefit from swarm robotics, specifically focusing on the combination with reinforcement learning techniques.

Forest fire Prevention and Control One of the primary applications of swarm robotics is the prevention, surveillance and control of forest fires. According to the National Oceanic and Atmospheric Administration (NOAA), forest fires cause billions of dollars in damage annually [36]. Preventing and suppressing forest fires before they grow too large is crucial to the economy and to the health of the planet. We first explore reviews in the field of swarm robotics to combat forest fires to provide us with an overview of the current capabilities of swarms. Roland et al. [37] reviewed the different challenges and possible solutions for forest firefighting using robots. To better understand the difficulties of firefighting in the context of forest fires, they surveyed firefighters. The authors found that the preparation of vegetation is crucial for fire prevention and identified associated health risks and the lack of real-time information as important issues. The survey also revealed the efficacy of newer technology like drones in preventing, monitoring and extinguishing fires. Using this information, they proposed a concept system that utilizes a fleet of quadcopters controlled using VR headsets. The swarm is able to navigate to deliver fire suppressants, provide real-time information and prepare vegetation to control the spread of a fire.

Similarly, Akhloufi et al. [38] focused on reviewing the current frameworks for UAVs and Unmanned Ground Vehicles (UGVs) in wildland firefighting strategies. The authors compared various fire assistance systems, including swarm-based approaches with different levels of autonomy, tasks, sensors, and types/quantities of robots. Various methods to detect fires such as color segmentation and motion segmentation were also surveyed. Additionally, they explored coordination strategies for a swarm, such as centralized and decentralized approaches for multiple UAVs or a single UAV. The study also explored the benefits of combining UAVs and UGVs in a swarm for large-scale wildfires. The authors showed the extent of the field of wildfire fighting

using unmanned vehicles and highlighted the potential in this area. They presented the current possibilities and challenges in the field of wildfire fighting using unmanned vehicles, emphasizing the potential for further research. In particular, the development of artificial intelligent solutions, new large-scale aerial wildfire datasets and practical frameworks would greatly benefit this field.

To address the challenges associated with manually controlling swarms of unmanned vehicles, we explore research that utilized reinforcement learning algorithms. We specifically review algorithms that enable the optimization or control of multiple unmanned vehicles in the context of combating forest fires. For example, reinforcement learning can be employed to guide a swarm of drones to a fire spot [39], extinguish a fire [40] or to create a fire line to limit the spread [41]. Furthermore, swarms of autonomous vehicles can be used to monitor or detect fires [42–46] and to provide data for fire spread modeling [47,48]. The possibilities of swarm robotics to combat forest fires are vast, offering various applications that can provide valuable data, support, and fire extinguishment capabilities in remote areas. By combining swarms with reinforcement learning, it enables them to be optimized and controlled autonomously for an increase efficiency.

Agricultural and Forestry Applications Swarms of robots have significant potential in various agricultural and forestry management applications. To explore these applications in this domain, we first present reviews of swarm robotics followed by reinforcement learning-based research. In the agricultural sector, Santos et al. [49] reviewed the path planning algorithms focused on ground robots. They compared the point-to-point path planning algorithm and coverage path planning algorithm. Point-to-point algorithm involves planning a path from a starting point to a destination, while coverage path planning must cover a specific area. In total, the review covered 22 papers with 10 point-to-point algorithms and 11 coverage algorithms. Their finding showed that both types of algorithms could be utilized for navigation purposes as well as capturing data, such as crop maturity. Coverage path planning algorithms can also be applied to tasks such as seeding, pollination and wheat harvesting using combine tractors. Overall, the authors highlighted the potential applications for various types of robots in agriculture, such as crop monitoring, planting and harvesting. In the field of weed management, Esposito et al. [50] and Roslim et al. [51] reviewed the use of unmanned vehicles and sensors. Both papers discussed the utilization of artificial intelligence solutions to detect weeds using techniques such as images. Once detected, robots could be trained for automatic weed removal using herbicide spray or mechanical procedures. These reviews showed the ability of artificial intelligence to detect, locate and remove weed using unmanned vehicles.

In agriculture, reinforcement learning can be combined with a swarm to create a fully autonomous system. For example, reinforcement learning can be used to control a swarm of robots for weed management in crop fields [52]. Additionally, it can be used to control a swarm of UAVs for field coverage and crop monitoring [53,54]. Drones can be controlled using reinforcement learning to spray pesticides in fields [55–57] or they combined with ground vehicles such as combine tractors for efficient crop harvesting [58].

In the field of forestry, Oliveira et al. [59] conducted a review of the current state-of-the-art for swarm robotics. The authors reviewed various types of robots and their applications in forestry. First, they reviewed the use of UGVs for wildfire fighting, such as the Fire Ox, an adapted Lockheed Martin UGV. UGVs for firefighting are gaining popularity and vary in weight, size, speed, maximum payload and cost. UAVs and UGVs were found to be useful for inventory-related applications in the domain of forestry such as for delivering supplies. Drones and quadcopters can employ imaging modalities such as LiDAR or infrared to count species of animals or plants while UGVs can collect samples like biomass in inaccessible terrains. Furthermore, UGVs can be used for planting, pruning and harvesting plants of various kinds such as trees and bushes. The authors compared different planting and harvesting methods using UGVs while also comparing UAVs and UGVs for pruning. Using their review, they found that most forestry robots are based on caterpillar tracks, do not have a robotic arm and are predominantly used for inventory or monitoring purposes. Nonetheless, they showed the wide range of applications in which robots can be deployed for forestry tasks, which can be further developed into swarms.

A limited quantity of work in the field of forestry using swarm robotics combined with reinforcement learning exists. However, due to the flexibility of reinforcement learning, most applications explored in the review could be modeled in a reinforcement learning framework. For example, reinforcement learning can be combined with swarms robotics for reforestation to autonomously plant tree saplings in deforested areas after a fire or logging [60]. Overall, swarm robotics can be highly valuable in the agricultural and forestry domain with applications spanning the task of monitoring, weed and pest control, harvesting and planting. The integration of reinforcement learning with these systems enhances the efficiency and autonomy of these unmanned systems.

Logistic and service Swarms of robots, including UAVs and UGVs offer promising solutions to optimize logistics for various applications such as deliveries, services and more. We first investigate the current reviews in the field of swarm robotics for logistics, followed by an exploration of the current reinforcement learning applications in this domain. Wen et al. [61] reviewed the current challenges of swarm robotics for smart logistics. They introduced various concepts of smart logistics such as efficient transportation, smart warehousing, smart delivery, route tracking, precise supply chains and green logistics. Additionally, they identified and defined the current limitation and issues, including the communication between robots and their controller, object recognition, object picking, collision avoidance, path planning and swarm behavior. This paper gave an insightful overview of the current challenges and limitations currently faced by swarm robotics in logistics.

Many of the challenges in logistics using swarm robotics revolve around the control of vehicles, which can be addressed using intelligent solutions. Reinforcement learning offer promising solutions for autonomously controlling a single vehicle or a swarm of ground vehicles within a warehouse [62–65]. It can also be used to provide optimized routing and scheduling of deliveries based on crowd sensing, improving the overall efficiency of the system [66]. UAVs, such as quadcopters, can be autonomously

controlled for aerial delivery, facilitating transportation between warehouses or directly to customers [67,68]. Drones can also be combined with ground vehicles, such as trucks, enabling the creation of a coordinated system based on reinforcement learning [69]. Moreover, reinforcement learning can be applied to efficiently control multiple cars in autonomous driving scenarios [70]. In the context of space exploration, reinforcement learning can be used to control a multi-robot system for efficient swarm control [71]. In summary, swarms of robots, such as a fleet of drones, have an immense potential for enhancing logistics operations. By leveraging reinforcement learning, swarms of unmanned vehicles can automate the transportation of goods within warehouses, between warehouses and directly to a customer. Reinforcement learning techniques play an important role in the optimization of paths, routing and overall control of these unmanned vehicles.

Military Applications The recent conflict in Ukraine has increased the usage of unmanned vehicles crucial for surveillance or to deliver various types of payloads. Swarms of robots, including UAVs, UGVs or Unmanned Underwater Vehicles (UUVs), have significant potential for various military applications. We first explore a review in the field of swarm robotics for military purposes followed by reinforcement learning-based research. Sapaty [72] examined the current applications and trends of robotics in the military domain. They compared the applications of various types of robots such as UAVs, UGVs and UUVs and reviewed the general demands of a military robotic system. The authors presented the various types of unmanned vehicles, ranging from reconnaissance and personal support to equipment carrying and weapon systems. In their review, they emphasized the need of these systems to have a high level of autonomy while also adhering to international and ethical norms. Due to the nature of the military applications, the unmanned systems must be integrated and operated within a manned system to allow for human intervention. Overall, the review highlights the large quantity of unmanned vehicles currently available for military applications while also exploring the demands of swarm systems for military use.

As with other applications, there is a need for automated solutions to control and optimize swarm robotic systems in military applications. Reinforcement learning enables an autonomous solution for swarms of unmanned vehicles in the domain of military purposes. For instance, it can be used to control a swarm of drones for patrolling large areas or to monitor secure zones [73,74]. It can also enable drones to track and follow multiple ground targets, such as vehicles or troops [75]. Neutralizing enemy drones is a challenging task, often involving anti-drone rifles or a fleet of drones to track and eliminate the target. However, manually controlling a fleet of drones for this purpose is unrealistic due to the size and quantity of potential targets. Reinforcement learning can be employed to automatically control drones to detect, track and neutralize enemy drones [76–79]. Furthermore, reinforcement learning can aid in decision-making for maneuvering during short-range air combat providing autonomous control sequences [80–82]. In the field of military applications, reinforcement learning offers distinct advantages in the control of unmanned vehicles. Drones and other UAVs, such as quadcopters, have shown the most promise in this field by enabling extensive surveillance, monitoring of large areas and payload delivery.

Search and Rescue Swarm robotics can also play a crucial role in the support of search and rescue missions after natural disasters. Couceiro [83] reviewed the state-of-the-art in of swarm robotics for search and rescue missions. They noted that at the time, the field of search and rescue using swarm robotics was still in their early stages and more work had to be done. However, they showed various interesting proposed systems that were mostly based on theoretical frameworks and were still being developed. These frameworks included robots with different characteristics such as a common starting location as well as those deployed randomly in the search area. While the review showed the lack of research in this domain, it also showed potential of swarm robotics for search and rescue missions.

In terms of reinforcement learning, multiple techniques have been proposed in recent years for search and rescue missions using robotics. For example, a mobile unmanned ground rover can autonomously explore unknown environments with various obstacles to reach a target [84]. Similarly, UAVs, such as a single drone or quadcopter, can navigate an environment to aid in finding a target after a natural disaster [85–87]. Multiple unmanned vehicles can be combined as a swarm to more efficiently search and locate a target in an unknown environment [88–91]. Moreover, combining UAVs and unmanned surface vehicles (USVs) in a swarm has been shown to optimize marine search and rescue missions [92]. Swarm robotics has emerged as an interesting and cost-effective alternative for search and rescue applications by providing newer and cheaper alternative to finding individuals. Reinforcement learning, in particular, has shown promise in controlling unmanned vehicles to locate a target in a cluttered environment. Additionally, combining multiple unmanned vehicles and different types of robots can significantly increase the efficiency of search and rescue systems. Overall, swarm robotics, combined with reinforcement learning algorithms, continue to advance the field of search and rescue missions.

3. Reinforcement learning

Reinforcement learning is one of the three main learning paradigms within AI, alongside supervised and unsupervised learning. Unlike supervised and unsupervised learning, which relies on labeled or unlabeled data, reinforcement learning uses a trial and error approach. An agent is able to mimic the desired behavior by receiving feedback signal such as rewards or penalties. This learning approach has seen impressive results in playing simple video games [93], complex games such as StarCraft II [94] and in various robotics applications [95–97]. To better understand the field of reinforcement learning, we first must define important basic terminologies.

- **Agent** An individual or entity responsible of making decision and learning in the system.
- **Environment** The world in which an agent interacts, such as boundaries, obstacles and goals. It can be described as everything other than the agent itself in the system.

Fig. 1. Reinforcement learning process.

- **Action** A specific movement that is part of a set of possible choices that an agent can select to move within the environment.
- **State** The current representation of the environment from the perspective of an agent and consists of relevant information to the agent.
- **Reward** The feedback signal provided to the agent that is proportional to the desired behavior.
- **Policy** The decision-making process that an agent utilizes to choose an action based on a state such as a neural network or Q-table.

Reinforcement learning consists of using a reward system to iteratively move an agent in an environment based on an objective. Reinforcement learning is based on the Markov decision process (MDP) which mathematically describes the states, actions, rewards and other components of the system. Figure 1 shows the reinforcement learning process with an agent receiving a reward and next state based on an action from the current state. By iteratively trying to maximize the expected final reward, an agent is able to mimic the desired behavior using a policy. The policy is the ruleset used to choose an action a_t given a state s_t and can be deterministic or stochastic.

The deterministic policy is denoted by μ in Eq. (1) while s_t and a_t respectively represent a state and its corresponding action.

$$a_t = \mu(s_t) \tag{1}$$

In Eq. (2), the stochastic policy is denoted by π while s_t and $P(a_t|s_t)$ respectively represent a state and the probability of each action for the state.

$$P(a_t|s_t) = \pi(s_t) \tag{2}$$

The deterministic policy maps every state to an action while the stochastic policy gives a probability for each action given a state rather than a certainty for one action. Deterministic policies are often used for robot control, game playing and real-time system due to their ability to provide specific and precise actions. On the other hand, stochastic policies are more useful in scenarios where the agent must explore such as for exploration, in uncertain environments and in multi-agent systems. The choice between policies depends on the problem and in practice, a combination of both policies is beneficial to the learning process. To learn and improve the policy, an agent receives a feedback signal in the form of a reward to assess the value of an action or state. In Eq. (3), a reward r is calculated at every time step using the reward function R which changes depending on the system. It can be defined using the current state s_t and action a_t such as:

$$r = R(a_t, s_t) \tag{3}$$

It can also use the next state s_{t+1} which is defined as:

$$r = R(a_t, s_t, s_{t+1}) \tag{4}$$

The goal of an agent is to maximize the final cumulative reward $R(\gamma)$. The function $R(\gamma)$ can be generalized in Eq. (5) where γ represents an increasing discount to prioritize closer rewards while r_t represents the reward at time t.

$$R(\gamma) = \sum_{t=0}^{\infty} \gamma_t r_t \tag{5}$$

Reinforcement learning can be divided in two subcategories with model-free reinforcement learning and model-based reinforcement learning, which can be seen in Fig. 2.

Model-free algorithms can be further divided in policy-based and value-based algorithms while model-based algorithms can be divided in learning the model and given the model. In policy-based algorithms such as NPG [98], A2C/A3C [99], PPO [100] and TRPO [101], the agent directly learns the policy function to map an action to a state. Policy-based algorithms can be advantageous when working with high-dimensional data, continuous data and stochastic policies. Value-based algorithms differ slightly by using a value function to determine how good an action is when given a state such as with Q-learning [102,103], State-Action-Reward-State-Action (SARSA) [104], DQN [93] or Double-DQN [105]. Value-based algorithms are able to handle larger state space much more effectively

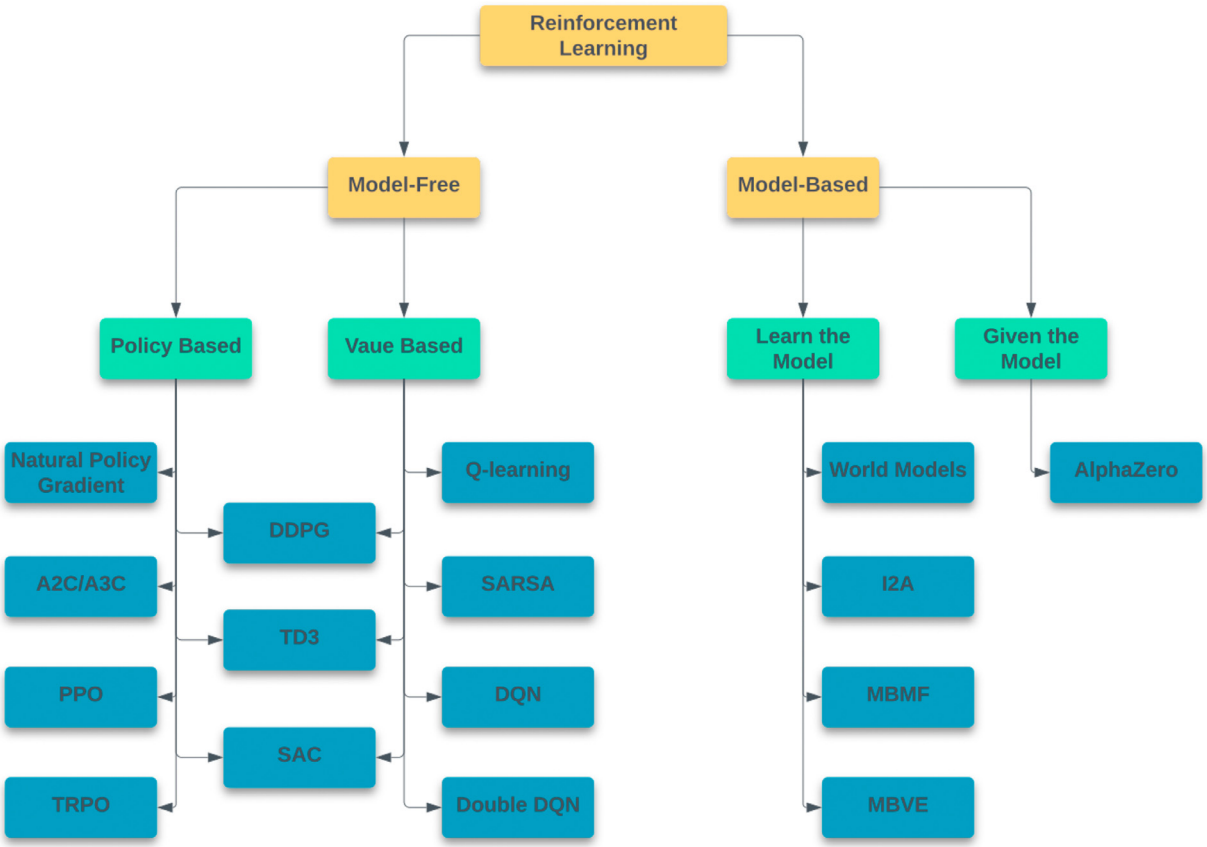


Fig. 2. Reinforcement learning categories.

while being able to handle both deterministic and stochastic policies. Algorithms such as DDPG [106], TD3 [107] or SAC [108] are hybrid algorithms that combine concepts of both value-based and policy-based algorithms. In model-based reinforcement learning, learn the model algorithms such as World models [109], I2A [110], MBMF [111] and MBVE [112] are based on the control theory and choose optimal actions based on a control function. These models learn using captured information about transitions such as the change in a state when given an action. Given the model reinforcement learning approaches such as AlphaZero [113] are algorithms that have complete knowledge of the environment and explicit information such as the reward structure.

To illustrate how reinforcement learning works in the context of swarm robotics, we focus on model-free reinforcement learning since it has been successful in recent years. More specifically, the center of attention of this review will be Temporal-Difference (TD) learning [114] which is a value-based reinforcement learning subgenre that learns by predicting a value function. The value function allows to quantify the value of a signal such as an action and varies depending on the method. The state-value function (V-function) estimates the final expected cumulative reward of a state s_t following the policy π. It can be used to determine the values of all possible states of an agent and is defined in Eq. (6). In this equation, $V_\pi(s)$ represents the expected return when starting at state s_t, following policy π where γ and r_t represent the discount and reward at time t.

$$V_\pi(s) = E_\pi\left[\sum_t^T \gamma_t r_t | s_t\right] \tag{6}$$

The action-value function (Q-function) is defined in Eq. (7) where $Q_\pi(s,a)$ represents the expected return of state s_t when taking action a_t and following policy π. Similarly, γ_t and r_t respectively represent the discount and reward at time t.

$$Q_\pi(s,a) = E_\pi\left[\sum_t^T \gamma_t r_t | s_t, a_t\right] \tag{7}$$

The Q-function returns the Q-value which determines the value of an action when given a starting state and policy to follow. It can be used to compare the expected reward of all possible actions given a state to find the next best combination.

Reinforcement learning algorithms can also be divided into two classes: on-policy algorithms and off-policy algorithms. An on-policy algorithm updates its Q-values using actions and the policy which those actions are derived from. An example of an on-policy algorithm is SARSA proposed by Sutton and Barto [104]. SARSA updates its Q-values by using the current state s_t, current action a_t,

the reward r_t, the next state s_{t+1} and the next action a_{t+1} derived from the same policy as action a_t. Using α as learning rate, γ_t as the increasing discount, the equation to find the next Q-value si defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma_t Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{8}$$

An off-policy algorithm update its Q-values using random actions or actions given by another policy such as in Q-learning [102]. Q-learning assumes the use of an optimal policy when estimating the value of state-action pairs rather than the current policy like SARSA. The Q-values are updated based on which action a gives the highest reward for state s_{t+1} and is defined in Eq. (9)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma_t \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{9}$$

In this equation, s_t represents the current state, s_{t+1} the next state, a_t the current action, α the learning rate, γ_t the discount and r_t the reward at time t.

Q-learning learns the optimal policy while SARSA learns the near-optimal policy. We can see this difference by comparing Eq. (8) and Eq. (9). SARSA can be seen as safer since it is able to use information from the current policy such as avoiding negative rewards. Since Q-learning always assumes that the next action is the best action, the algorithm could be unaware of negative rewards. However, both on-policy and off-policy algorithms are able to reproduce desired behavior using a trial and error approach.

4. Swarm robotics

In this section, we propose to explore the recent model-free reinforcement learning algorithms for swarm robotics. In the context of swarm robotics, reinforcement learning aims to control many robots by treating them like agents in a multi-agent system. We can further divide this section to better understand how each algorithm is different. The division is as follows: Section 4.1 refers to swarm robotics using the SARSA algorithm while Sections 4.2 and 4.3 respectively refers to the Q-learning and joint space Q-learning. Sections 4.4 and 4.5 explain the Deep Q-learning and Deep Deterministic Policy Gradient approaches respectively. Sections 4.6 and 4.7 present the MDDPG and MADDPG algorithms, while Sections 4.8 and 4.9 refer to the FLDDPG and MSMANN approaches. Finally, Section 4.10 refers to the use of the policy-based trust region policy optimization in a multi-agent system.

4.1. State-action-reward-state-action (SARSA)

In early works, Iiam and Kuroe [115] proposed using a swarm-based SARSA algorithm to solve shortest path planning scenarios in a two-dimensional grid. To increase the complexity of the shortest path problem, they modified the scenario such that the goal changed randomly within a specific area. In their swarm-SARSA approach, each agent could only observe their own position, could not collide and could move in any four directions. The authors compared two learning methods, individual learning and learning through exchanging information between the agents for both SARSA and Q-learning. With a static objective, they found that single-SARSA and single-Q-learning slightly outperformed the swarm-based approach with the individual-SARSA achieving the best results. With a variable objective, swarm-SARSA performed the best, closely followed by Swarm-Q-learning while the single algorithms did not reach convergence. The authors showed that using a swarm-based learning algorithms such as swarm-SARSA is beneficial for challenging tasks such as to find the shortest path with a variable target.

Kakish et al. [116] compared SARSA and Q-learning for swarm grouping using a leader and follower approach. Swarm grouping is important in swarm robotics, the leader and follower approach is an interesting solution where the followers must group themselves around the leader. The authors used the OpenAIGym environment to simulate and test the two algorithms for a grouping scenario Their experiment consisted of one leader and 10 followers to as many as 100 followers, increasing in increments of 10. Using this method, they found that SARSA converged slightly faster and better than Q-learning. They also noted that a quantity of followers over 50 increased the convergence time. To verify the scalability of the two algorithms, they modified the follower count during training and testing. The authors found that training with a smaller population decreased the performance when testing on a large population. On the other side, training with a large population and testing with a small population increased the performance. Since both algorithms achieved good theoretical results, they tested them using the Robotarium [117]. They found similar results with both algorithms achieving a good performance in grouping the followers around the leader. However, in the real-world scenario, Q-learning was significantly faster with 23 iterations compared to SARSA with 59 iterations. Overall, they found that SARSA and Q-learning can be used in both simulations and real-world experiments in a leader and follower grouping scenario.

Speck and Bucci [118] used the GM-SARSA algorithm [119] combined with a variety of behavioral swarm rules to group agents and follow waypoints. The GM-SARSA algorithm is a controller that learns multiple Q-tables derived from Reynolds Flocking behavioral swarm rules [120]. Reynolds Flocking provided three behaviors: cohesion, alignent and collision avoidance. They combined Reynolds behavioral rules with a target seek rule, an obstacle avoidance rule and object-focused (OF) learning [121]. OF learning creates copies of the tables when an obstacle or neighbor is detected to reduce the size of the Q-tables used by the algorithm. Their OF-GM-SARSA selected an action based on the weighted sum of the Q-values across the five different behavioral rules. They compared their approach against a decentralized flocking solution proposed by Price and Lamont [122] using two scenarios. The first scenario consisted of a group of nine UAVs in an urban setting while the second consisted of 25 UAVs in open terrain. In the first scenario, the UAVs had to navigate through four waypoints while avoiding obstacles such as a building. The second scenario had the UAVs navigate through six points in a dense environment of smaller obstacles to test the agility of the approach. Using both experiments, their approach achieved a better performance than the decentralized solution while maintaining a low number of collisions. SARSA was able to generalize a solution for both scenarios, while the other approach was not able to generalize between the urban and open scenario.

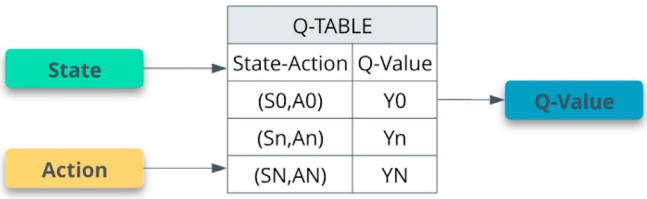


Fig. 3. Q-learning algorithm.

More recently, Luo et al. [123] proposed a deep-SARSA algorithm for a multi-UAV path planning and obstacle avoidance scenario. In multi-agent path planning scenarios, the Q-table used by the SARSA algorithm can become extremely large and inefficient. To help with the curse of dimensionality, the authors proposed replacing the Q-table with a neural network to choose actions based on a state. Their experiment consisted of a 2D grid with two agents, two obstacles moving vertically, a static obstacle and two exit positions with one for each agent. During training, they implemented an epsilon-greedy policy which allowed the algorithm to focus on exploration for the first epochs. To validate their approach, they simulated two UAVs in the Gazebo simulator using the ROS framework to control the agents. The Gazebo simulation showed that their approach was able to guide both agents towards the exit while avoiding obstacles. In summary, the agents proposed a deep-SARSA algorithm to solve a realistic multi-agent multi-goal path planning scenario and achieved good results.

4.2. Q-learning

Q-learning, originally proposed by Watkins [102], is an off-policy algorithm commonly used in reinforcement learning. The convergence of the algorithm was later proven by Watkins and Dayan [103]. This algorithm uses the Bellman optimality equation [124] to iteratively update the Q-values until the Q-function converges to its optimum. Unlike SARSA that selects the next action for s_{t+1} based on the current policy, Q-learning updates its Q-table by choosing the action that returns the maximum Q-value given s_{t+1}. The Q-learning algorithm is illustrated in Fig. 3 with a table of state-action pairs matched to a Q-value.

In recent years, Islam and Razi [39] presented a collective monitoring algorithm based on reinforcement learning to combat forest fires. In their simulated environment, multiple drones aim to reach a fire front while maintaining a safe distance from obstacles. Each drone is controlled individually, making independent movements without exchanging information with other drones. The authors utilized independent Q-learning to provide five actions to each drone: up, down, left, right or no movement. They used a target tracking system with a variable measurement error, allowing each drone to be aware of the location of obstacles, other drones and the fire front. To optimize the Q-table, the reward function was based on if the drone was flying towards obstacles or towards other agents combined with the change in position of the drone. Their Q-learning algorithm included an alternating exploitation and exploration phase to facilitate learning from new experiences. The system with two UAVS was tested on two different environments in MATLAB, one with stationary obstacles and another with mobile obstacles. In the first scenario, their approach successfully reached the target without any collisions when a low measurement error was used. They also tested the algorithm on stationary obstacles with an increased measurement error and without changing the minimum clearance that was used to detect a collision. Using this configuration, both drones reached the target, but one drone collided with an obstacle. With mobile obstacles, both drones using the algorithm reached the target without any collision. The authors compared a moving target with mobile obstacles and found that both drones successfully reached the target without any collision. The research showed the effectiveness of independent Q-learning in controlling a swarm of two UAVs towards a target in various scenarios. The algorithm yielded great results when applied to stationary obstacles, mobile obstacles and a moving target.

Cui et al. [125] continued their work [126] on a multi-agent reinforcement learning (MARL) approach to create a downlink wireless network (DWN) using a swarm of UAVs. The DWN consists of multiple ground users and UAVs, with each UAV required to simultaneously communicate with a ground user. Two air-to-ground communication methods were compared, the first being a probabilistic model while the second is based on line-of-sight (LoS). The probabilistic model is based on parameters such environmental constants, UAV elevation and the distance between the UAV and the ground user. The LOS approach relies on the locations of the UAV, ground user and the environment. A signal model based on subchannels and a signal-to-interference-plus-noise ratio was specified to reduce interference between the UAVs. By providing communication models and a signal model, the authors increased the validity and realism of their proposed system. During training, agents were rewarded based on the allocated resources, such as power and the overall performance of the swarm. Independent Q-learning was employed for each agent to control the swarm, which each perceived other agents in the system as static entities rather than dynamic. The approach was tested using simulated UAVs with a static elevation in a circular 2D plane, along with 100 uniformly distributed ground units. Various parameters were tested, including the number of agents, the number of targets and the exploration probability (ϵ). The algorithm learned more efficiently when ϵ was set between 0.2 and 0.5. Their probabilistic model and LoS model yielded similar results when comparing the average reward for two UAVs and 100 ground units. The probabilistic model with four agents and 200 ground units achieved similar results with a slightly better average reward. Finally, the MARL approach was compared against two approaches, the first was a random walk algorithm and the second approach involved complete information exchanges between UAVs (Mach). The Mach algorithm highly

outperformed both algorithms, while the MARL outperformed the random walk model. Their research demonstrated the usefulness of sharing information between agents in reinforcement learning-based swarms. Although their independent MARL algorithm could create a DQN to some extent, the Mach model, which focused on sharing information between the agents, significantly outperformed their independent approach.

Karmanova et al. [127] proposed SwarmPlay, a swarm of nano-UAVs for an interactive tic-tac-toe game using reinforcement learning algorithms. SwarmPlay consists of an interactive game where each drone represents an individual component of a game with the goal of winning against a human. The environment consisted of a game board, Crazyflie drones, a CV camera and a Vicon tracking system with 12 infrared cameras for localization. Three algorithms were compared to be used in their system, a basic state-value function (SV), Q-learning and SARSA. Each algorithm was trained for 50,000 episodes with a reward system assigning $+1.0$ for a win, -1.0 for a loss and 0.1 and 0.5 for a draw depending on who played the first turn. Time-wise, Q-learning was faster with an average training time of 38.8 s for 10,000 episodes, compared to 39.4 s for SARSA and 87.6 s for SV. Q-learning also achieved a better performance with a 28% higher total reward compared to SV and a 33% higher reward than SARSA. During training, Q-learning was able to win 32% of the games. Based on these results, Q-learning was implemented in their SwarmPlay system to play against humans. An Improved Basic algorithm (IB) was also implemented, which involved hard-coded processes to play tic-tac-toe with random actions to simulate human error. To evaluate their system, 20 participants were tasked of playing against the SwarmPlay system, 10 against Q-learning and 10 against IB. The participants evaluated the system based on excitement, engagement, latency, challenge, tiredness, stress factor and a Turing test using a Likert scale. The likert scale showed that both algorithms achieved overall good scores, with Q-learning receiving higher scores for excitement and the Turing test. A correlation was observed between excitement and game outcome, with a decrease in excitement when the human lost against the algorithm. Overall, their system demonstrated the ability of Q-learning to achieve a good performance in SwarmPlay tic-tac-toe and received a favorable human-robot interaction score.

Das et al. [128] combined Q-learning and Particle Swarm Optimization (PSO) [129] for a multi-robot path planning task. To address the issue of computational complexity of Q-learning, the author proposed improving the algorithm using a lock variable to reduce the memory needed for the storage of Q-values. The lock variable is used to only save the highest Q-value and the respective action for a state, thus reducing the size of the table. Four properties were introduced for the lock variable, the first replaced the locked pair with a higher Q-value pair if one is found. Second, when the lock variable was not set and the distance between the goal and next states favored the current state, the Q-value for the current state was assigned as the maximum Q-value amongst all possible next states. If the lock variable for the current state was set and the distance between the goal and the next states favored the next state, then the Q-value of the next state was updated as the Q-value of the current state and the lock variable for the next state was set. Lastly, when the lock variable for the next state was not set and the distance between the goal and next state favored the current state, in which case, the Q-value of the next state was updated as the maximum Q-value for all possible next states. They compared their improved Q-learning against an improved PSO (IPSO), an improved particle swarm with differentially perturbed velocity (IPSO-DV), a Q-value-based PSO-DV (QIPSO-DV) and a Differential Evolutionary (DE) algorithm [130]. In PSO, a particle moves around the search space guided by their best-known position and the best-known position of the entire swarm. The IPOS includes an adaptive inertia weight adjustment and acceleration coefficients for faster convergence. IPSO-DV further modifies the IPSO algorithm by adding a vector differential operator (DV) in the velocity update equation to reduce premature convergence. QIPSO-DV combines Q-learning, PSO and DV to dynamically choose the best action. DE is an evolutionary optimization algorithm known to effectively navigate large design spaces. The four algorithms were tested in a scenario with starting locations, destinations and obstacles in a 30×30 grid. Their first experiment consisted of five circle-shaped robots and five rectangle-shaped robots, each assigned a color and matching targets. The performance was compared using the number of turns it took for an agent to reach its target. The results showed that the number of turns increased proportionally with the number of agents while QIPSO-DV achieved the best overall performance. The authors found similar results when comparing the performance with the average total trajectory path deviation and the average untraveled trajectory target distance. Most algorithms guided the agents towards their targets, however, the QIPSO-DV consistently achieved better results. To verify the theoretical results, the algorithms were validated on two Khepera-II robots which all successfully guided the robots to their targets. Overall, the research showed that classical Q-learning/POS and their improved versions can be used for multi-robot path planning. More complex algorithms, such as QIPSO-DV, achieved better results in various experiments compared to basic Q-learning or PSO.

In precision farming, Testi et al. [58] proposed a reinforcement learning-based approach for the localization of an autonomous farming vehicle. The system was formulated as a swarm of UAVs with an autonomous farming rover moving inside a predefined square field represented by a grid. Each UAV occupied a cell in the grid and could choose one of five actions: move up, down, left, right or stay stationary. Actions that would cause collisions between UAVs or with obstacles were not allowed and would be ignored. The authors aimed to create an autonomous swarm that was able to estimate the position of the ground rover in the field. The MAQL algorithm was introduced to control the UAVs combined with a reward function based on the estimated rover distance and a geometric dilution of precision. They used independent MAQL, where each UAV updated its own Q-table while considering information from other agents. The performance of their algorithm was compared on three grid sizes: 10×10, 20×20 and 30×30. The experiments revealed that increasing the size of their environment proportionally decreased the convergence speed. Overall, Q-learning achieved a good performance on all the three grid sizes, with a similar final convergence. However, the authors noted that the success of the algorithm might not be guaranteed when increasing the size of the environment, suggesting the need for more complex algorithms such as a DQN.

Table 1
FCMAQL experimental results 3 from Sadhu et al.[133].

Method	Stick carrying (min)	Triangle carrying (min)	Box carrying (min)
FCMAQL Deterministic	0.193	0.244	0.309
FCMAQL Stochastic	**0.183**	**0.220**	**0.299**
MRLbD Deterministic	14.39	20.727	27.777
MRLbD Stochastic	18.305	25.116	33.395
ICFA Deterministic	51.085	60.723	64.633
ICFA Stochastic	52.685	60.393	63.54

Table 2
FCMAQL experimental results 4 from Sadhu et al. [133].

Method	Agent count	Average run-time (S)
Theoretical	2	**8.94**
	3	**5.96**
	5	**3.58**
Experimental	2	12.07
	3	9.84
	5	8.82

4.3. Multi-agent Q-learning

In a typical Q-learning system, each agent controls its own action independently, while in Multi-agent Q-learning (MAQL) [131], the agents are able to control a joint action in a joint space. The joint action and joint space allow the agents to communicate more directly for a better cooperation.

Wang and De Silva [132] compared single-agent Q-learning and multi-agent Q-learning for box pushing. Box pushing involves multiple agents working together to push a box to a specific destination. The reward of an agent consisted of three parts: proximity to the destination, rotational behavior and obstacle avoidance. The authors initially tested their approach in a simple environment without pre-training and found that both algorithms were able to push the box to the destination. The two algorithms were then pre-trained and the authors found interesting results. Training the single agent Q-learning led to a reduction in the average step per round by 80%. However, training the multi-agent Q-learning system showed minimal improvement in performance. Overall, the authors found that the single-agent Q-learning achieved significantly better results. The single-agent system showed improvement with training, while the multi-agent demonstrated limited improvement. In fact, the probability density for the number of steps per round was considerably reduced after training for the single-agent system. One contributing factor could be the higher probability of random actions in single-agent Q-learning compared to multi-agent Q-learning. The multi-agent system also suffered from more noise compared to the single-agent system. In conclusion, the authors determined that a single agent approach outperformed using multiple agent in this context.

Sadhu and Konar [133] improved upon classical MAQL by introducing two new properties, creating the Fast Cooperative Multi-Agent Q-learning (FCMAQL). The first property involved switching an agent to idle once it reached a target, while other agents continued exploring. If an agent received a negative reward for a joint action before it could improve its initial joint-Q-value, it would become trapped in that joint state. To address this issue, the authors proposed the second property, which involved reinitializing the joint-Q-value of a trapped agent. These two properties allowed for a better exploration of the team goal and an accelerated convergence of the Q-tables. The system could receive three rewards: maximum reward when all agents reached their goals, minimum positive reward if one agent failed to reach its goal and a negative reward if an agent violated a constraint. The authors analyzed their approach by comparing the number of joint state transitions required to reach the team goal. They tested their FCMAQL algorithm through four experiments and by comparing it against popular methods. The experiments consisted of reaching a destination in a 10×10 grid with obstacles. Various parameters were compared, such as the number of agents (2–4), policy type (deterministic/stochastic) and the number of obstacles. In the first experiment, the convergence speed of the FCMAQL was compared to other methods and the authors found that their approach achieved a faster and better convergence. The second experiment focused on multi-agent cooperation for carrying objects, such as a stick to a destination. The robots were able to carry objects by each taking a point such as in a triangle where three agents were required. Compared to a traditional MAQL, their FCMAQL succeeded in reaching the target by having the agents work together. The third experiment involved a time analysis for multi-agent path planning, comparing FCMAQL against ICFA [134], MNPSO [135], DE [130] and MLbD [136]. Table 1 provides an overview of the three best approaches using both a deterministic and stochastic policies for the third experiment. Compared to other algorithms, their approach demonstrated a considerable improved performance in terms of time.

In their last experiment, the authors used Khepera-II robots [137] for a stick-carrying scenario in real time. Their real-life scenario achieved similar experimental results from the theoretical results, which are summarized in Table 2.

Overall, the researchers found that their FCMAQL algorithm successfully controlled a swarm of robots in various scenarios, such as for stick-carrying. By improving the independent MAQL algorithm to enhance cooperation between the agents, they saw significant improvement in performance and overall better results.

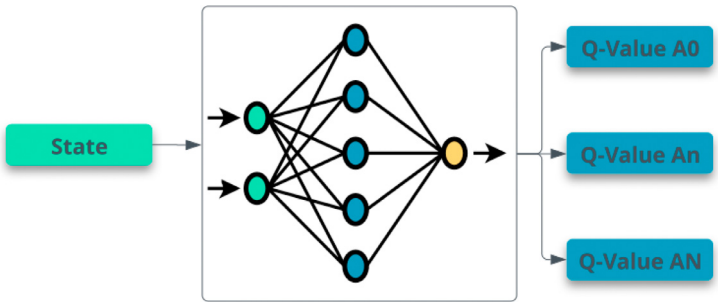


Fig. 4. DQN algorithm.

In the domain of field coverage using a swarm of drones, Pham et al. [74] introduced a distributed multi-agent approximated equilibrium algorithm based on Q-learning. Field coverage is important in various sectors, such as for agriculture, oil spill monitoring and for military application like surveillance. Unlike individual reinforcement learning algorithms such as the approach proposed by Testi et al. [58], the authors proposed a joint state and joint action approach. This approach involved having a joint state that represented the collective state of all agents and a joint action that included an action for each agent. To determine a joint action, the authors used a consensus algorithm based on correlated equilibrium [138] and inspired by Sadhu and Konar [133]. Additionally, a social convention mechanism [139] was implemented to address scenarios with multiple equilibria, where an agent may choose a suboptimal joint action over the optimal action. Two approximation techniques were implemented to reduce the computational complexity of the problem, Fixed Sparse Representation (FSR) and Radial Basis Function (RBF). Individual rewards were assigned to each agent based on the number of cells covered by the drone minus the overlapping cells, while a global reward was provided based on the percentage of the field covered by the swarm. Their environment comprised of a 3D grid of cubes and each agent could choose from six actions: up, down, North, South, East or West. Using this configuration, the authors conducted experiments in a $7 \times 7 \times 5$ field coverage scenario in MATLAB. Three UAVs were simulated to compare their MARL algorithm against a baseline algorithm based on individual performance. Their simulation results demonstrated the effectiveness of the MARL approach in controlling a swarm of UAVS for field coverage both for approximation techniques, while the baseline algorithm failed in most scenarios. This research highlighted the advantages of utilizing a joint-learning approach for Q-learning compared to individual Q-learning in a multi-agent field coverage scenario.

Recently, Chen et al. [140] developed an autonomous ground vehicle tracking algorithm utilizing a swarm of UAVs based on multi-agent reinforcement learning. The tracking of ground vehicles using UAVs, such as drones or quadcopter, has significant importance in the agricultural and military sectors. The authors proposed a system where a swarm of UAVs equipped with omnidirectional received signal strength sensors is employed to locate and track a mobile target. They compared the performance of three algorithms to locate the target using a swarm of UAVs, independent Q-learning, MAQL and constrained action-based MAQL (CA-MAQL). The CA-MAQL algorithm, derived from MAQL, integrated a searching time constraint to select more optimal actions, aiming to reduce unnecessary actions and increase performance. To compare the three algorithms, the researchers used the failure rate and searching time of a static target in a 2D grid of size 800x800. When dealing with a moving target, their CA-MAQL outperformed the others approaches, achieving a 28% reduction in search time compared to MAQL. Overall, their approach achieved a significant reduction in searching time compared to Q-learning and MAQL. Additionally, they observed that increasing the number of UAVs resulted in a decreased searching time but increased failure rate.

4.4. Deep Q-learning network

In their work, Mnih et al. [93] introduced the Deep Q-learning Network (DQN), which replaces the Q-table in Q-learning with a neural network. DQN reduces the computational power needed for large-scale reinforcement learning tasks and is illustrated in Fig. 4. In their paper, the DQN is provided with a sequence of screen captures to select an action based on a normalized reward system where positive rewards were set at $+1.0$, negative rewards at -1.0 and no change received 0.0. They compared the performance of DQN, SARSA and the Contingency algorithm (CA)[141] on the games Pong, Q^*bert, Enduro, Breakout, Seaquest, Beam Rider and Space Invaders. Since SARSA and CA are on-policy algorithms, the researchers incorporated prior knowledge about the games such as the color of objects to identify entities like obstacles. The DQN outperformed both algorithms on all games and outperformed humans on three games (Breakout, Enduro and Pong). Overall, the authors showed that their DQN approach represents a significant improvement over classical reinforcement learning algorithms.

Haksar and Schwager [40] proposed a distributed reinforcement learning algorithm for forest firefighting using a swarm of aerial robots, such as drones. The authors addressed the task of combating forest fires by introducing a forest fire model that simulated a fire behavior in a 2D grid, providing a realistic scenario. In the grid, each cell represented a tree and had three possible states: healthy, burning and burnt, while a cell could transition from a healthy to burning state if at least one neighboring tree was on fire. Within the environment, a grid cell had four neighbors (up, down, left or right), while a UAV was capable of moving in any of the four directions and diagonally for a total of eight actions. Each agent had two goals in mind, reaching a cell on fire and extinguishing it using a limit quantity of fire-retardant that had to be refilled occasionally. To control the agents, the authors compared a heuristic approach

against a Multi-Agent DQN (MADQN), where each agent is controlled by its own individual DQN. Three experiments were conducted simulating 10, 50 and 100 agents in a grid of size of 50x50, considering two fire sizes of 4×4 and 10×10. They also compared the effect unlimited fire-retardant capacity for both fire sizes and two limited quantities for a fire of size 4×4. Overall, the MADQN outperformed the heuristic approach in all scenarios, achieving a higher win percentage for both fire sizes, agent quantities and both unlimited and limited fire-retardant. However, both approaches performed poorly when overwhelmed with fires, such as with ten agents having a limited fire-retardant capacity or when ten agents faced a starting fire of size 10×10. The authors demonstrated the benefit and feasibility of a using multi-agent DQN approach to combat forest fires with various swarm sizes, fire-retardant capacity and fire sizes.

Recently, Julian and Kochederfer [45] presented a wildfire surveillance system using autonomous UAVs based on DQN. The authors defined wildfire model in a 2D grid environment, where each cell had a fuel value to represent the amount of remaining fuel and a boolean value to indicate if a cell was burning or not. A fixed-wing aircraft was used to simulate the UAVs, which had a constant speed and two possible actions, either moving left or right by an angle of five °. The state space of a UAV included the bank angle of the UAV, range to other UAVs, bearing and heading angle of the other UAVs relative to the current heading direction and the bank angle of the other UAVs. The state space was combined with one of two sensor modeling approaches, an observation-based model and a belief-based model. The observation model consisted of an image depicting the fire relative to the observed UAV and a binary layer representing if a cell was burning or not. The UAV in the observation model was penalized based on the distance from the fire front, non-burning cells nearby, high bank angles and proximity to the other UAVs. In the belief model, an image with two layers was provided, the first being a binary layer representing the belief about the cells being on fire or not and a coverage map indicating the time since each cell was last visited. The belief model was shared and updated by the swarm and the UAVs were penalized when cells were believed to be non-burning but were found to be burning upon visitation. The authors compared their observation model and belief model against a baseline receding horizon controller for two UAVs in 20 scenarios. Both the observation model and belief model achieved similar results and outperformed the receding horizon algorithm. When the scalability was tested by increasing the number of UAVs, the observation model outperformed the belief model for three and four UAVs. Additionally, the researchers compared the two models with different wildfire shapes and with wind or without wind, to assess the realism of their approach. The shape experiment revealed that the observation model struggled to generalize arc-shaped fires, while the belief model had difficulties with *T*-shaped fires. This experiment showed the importance of verifying various training parameters, such as fire shape, due to the difference in performance. Overall, the authors introduced two approaches, an observation model and a belief model to monitor wildfire fronts based on reinforcement learning. Both models demonstrated their ability in monitoring wildfires with different swarm sizes for circular fires, while different shapes highlighted the importance of testing various parameters.

Similarly, Viseras et al. [42] compared four algorithms based on DQN for monitoring wildfire fronts using a swarm of UAVs. Monitoring wildfire fronts using UAVs enables firefighting teams to allocate resources more efficiently and predict the spread and movement of fires. The task involves deploying UAVs, such as drones, to fly over a fire while adjusting their positions based on the fire spread. The authors simulated a fire scenario using a 2D grid representing a forest, where each cell had three values, fuel quantity, a binary value to represent if a cell was burning and an ignition probability. A wildfire model was used to simulate fire propagation based on these values, taking into account conditions such as the wind speed and direction. Two types of UAVs were considered for the experiments, a fixed-wing aircraft capable of turning left or right with a bank angle and a multicopter capable of moving forward, backward, left or right. Both UAVs were equipped with sensors capable of detecting burning cells with a predefined range while their goal was to monitor as many burning cells as possible. The authors compared four DQN algorithms, joint action DQN (JOINT), multiple single trained Q-learning agent (MSTA), independent DQN (INDI) and a value decomposition network(VDN). JOINT involved training a DQN with an output node for every possible combination of actions for every UAV in the swarm. MSTA utilized a model for a single agent during training, while multiple copies of the same model were used during testing with multiple UAVs. INDI trained each agent on its own DQN without direct representation of the other agents in the environment. VDN used value decomposition networks [142] to represent a joint Q-function to allow for cooperation between the agents. Each agent in these algorithms received two components, an image with two layers and a vector observation providing information about the UAVs. The image had a binary layer representing burning cells while the second layer was used as a coverage map with values ranging from 0 to 255, indicating the last time a cell was covered. Agents received rewards of $+1.0$ for each cell perceived as burning by any UAV that was not already marked as seen. To compare the algorithms, they simulated a 100×100 grid representing a forest while wind and cell fuel were provided using a uniform distribution. They measured the performance using the fire missed over time with fixed-wing UAVs to compare the four algorithms. Using this configuration, VDN and MSTA outperformed the two benchmark algorithms, with JOINT performing significantly worse. The authors also compared the performance of their two algorithms using fixed-wing UAVs against multicopter UAVs and assessed their scalability. Fixed-wing UAVs achieved slightly better performance, possibly due to their ability to move diagonally and cover more ground quickly. When evaluating the scalability, both algorithms demonstrated an improved performance when increasing the number of agents from one to three. However, the performance of the VDN decreased when further increasing the number of agents from three to nine agents, while the MSTA algorithm performed well and achieved a reduction of fire missed proportional to the number of agents. In summary, the authors proposed two new algorithms to monitor wildfire and compared them against two popular baseline algorithms with different types and quantity of UAVs. Specifically, their MSTA algorithm achieved great results in wildfire monitoring and demonstrated its scalability by comparing its performance with one to nine agents.

In their work, Li et al. [63] presented a DQN-based system for dispatching and routing of mobile robots in a warehouse environment for inventory management. Their system involved a dispatcher responsible of collecting the current location of agents, routes of active agents and the location of tasks. The authors modeled a warehouse environment using a 110×100 grid, including empty spaces,

obstacles, active agents, dispatching agents and tasks. Each dispatching agent is assigned a task and can navigate the grid by moving up, down, left or right. Two environments were used, a real warehouse generated by a robot using LiDAR and a virtual warehouse made in a simulator. Initially, the locations of the dispatching and active agents were randomized, the active agents are assigned a random task and 1 to 15 pickup locations were generated. The authors used a modified Rainbow DQN [143], which integrated various techniques such as double DQN [105], dueling DQN [144], distributional DQN [145,146] and prioritized replay [147]. The modified Rainbow DQN controlled the actions of each dispatching agent until they all reached their designated pickup location. A reward system was designed where agents received a small negative reward for every move, a large negative reward for colliding with another agent, a medium negative reward for colliding with an obstacle and a large positive reward for reaching the destination. Their algorithm was compared against the shortest travel distance (STD) dispatching algorithm, using metrics such as the mean flow time, makespan, total travel distance, and number of times agents were closer than four units from each other (traffic). Three scenarios were compared, two agents in the first environment while three and four agents were compared in the second environment. The results showed that their DQN approach was successful in reducing traffic between agents but led to an increased in the total travel distance and reduction in makespan and mean flow time of the tasks. The authors demonstrated that choosing between the DQN and STD algorithm depends on the specific application requirements, where the DQN is more beneficial when traffic must be minimized while the STD is preferred when time constraints are more important.

Yang et al. [64] proposed an improved DQN algorithm for multi-robot path planning in a warehouse dispatching system. The authors aimed to address the slow convergence speed and excessive randomness issues associated with the basic DQN by incorporating prior knowledge and rules. They utilized the A* algorithm with a single robot to quickly calculate collision-free paths from starting locations to destinations for each robot. This pre-calculation allowed the algorithm to have a basic understanding of the environment such as the obstacles, allowing for faster learning. Additionally, priori rules were introduced to guide the robots conflict occurred such as during collisions, determining which robot should move and which should wait. The improved DQN also incorporated a target network [148], which is a copy of the DQN that provides target values, thus increasing the stability of the algorithm. To evaluate their improved DQN, the authors used a 30×30 grid environment with 45 shelves, five picking areas, eight robots and 80 tasks. The authors compared their improved DQN against a basic DQN using the loss and the total picking distance against the A* algorithm. Their experiments demonstrated that their improved DQN achieved a more stable loss compared to the basic DQN while outperforming the A* algorithm when comparing the total picking distance. By using prior knowledge and rules, the improved DQN algorithm demonstrated its ability to increase convergence speed, reduce excessive randomness and reduce the total picking distance travel.

Zhou et al. [75] presented a deep reinforcement learning algorithm for tracking multiple targets using a swarm of UAVs. This application has various practical uses in many fields, such as animal tracking or military target tracking. Each agent in the swarm is modeled as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP), meaning an agent can only perceive local observations. At each time step, an agent chooses an action based on its local information while all actions are executed simultaneously to update the global state. The authors focused on fixed-wing UAVs that moved at a constant speed and altitude while they could only observe targets and communicate with nearby UAVs within a predefined range. The objective of each UAV was to track as many targets as possible while also avoiding overlapping targets to maximize the total number of targets tracked. In their research, the authors aimed to control each UAV in the swarm using a modified Deep Double Dueling Q-Network (D3QN) algorithm [149]. The UAVs were equipped with cameras that provided top-down view images and a local communication space that allowed them to exchange information with their neighbors. The images and communication space were normalized using a feature extraction algorithm, providing a cartogram feature representation of the environment to the D3QN. Since the UAVs have experiences that are independent from their identities, a single network could be trained and shared amongst them to improve computational efficiency. The authors compared their cartogram features representation against a mean-embedding representation [150] to control the swarm of UAVs. For their experiments, they used a grid size of 2000×2000 with ten UAVs, a communication and observation range of 500 units and ten targets to track. To compare the two algorithms, the authors used the average reward per episode during training and found that the cartogram representation outperformed the mean-embedding. They also conducted 100 test episodes and observed similar results, with their approach tracking more targets and achieving a better cumulative reward than the mean-embedding approach. Furthermore, the scalability of the techniques were compared by testing the previously trained models with different quantities of equal number of UAVs and targets (5, 10, 20, 50, 100). Consistently, their approach achieved better cumulative rewards and tracking ratios compared to the mean-embedding approach in every scalability experiments. Lastly, the researchers evaluated the local communication aspect of their approach by varying the communication range (100, 500, 1000, global). They found that a communication range of 500 units yielded optimal results, closely followed by 1000 units while the global range achieved moderate results. The range of 100 units performed significantly worse than any other range due to its inability to communicate properly with other UAVs. Overall, the authors proposed a decentralized MDP model to control a swam of UAVs by utilizing local and partial observations. Their approach demonstrated a good performance for tracking multiple targets compared to the mean-embedding feature representation. Furthermore, they showed the scalability of their approach using varying swarm sizes and quantity of targets, which outperformed the other method. Finally, they compared various communication ranges such as 100 units, 500 units and a global observation and found that 500 units is the optimal setting for their approach. These results demonstrate the importance of proper communication range when dealing with a swarm of unmanned vehicles.

Wei et al. [151] focused on using a DQN to move a swarm of simulated rovers between two points using images taken from a first-person view as input. In their experiment, the robots in the swarm were individually controlled without any global information or direct information about the other agents. In the swarm, each agent was initialized in the center and had to travel between the landmarks while avoiding the other rovers that were considered dynamic obstacles. To process the images, the authors used a small

Table 3
Overview of the results from Wei et al. [151].

Reward Set	Experiment 1	Experiment 2	Experiment 3
Reward A	**89.56%**	**56.44%**	**67.12%**
Reward B	57.51%	6.14%	9.81%
Reward C	21.87%	35.84%	51.65%

Convolutional Neural Network (CNN) with six output nodes to represent the six motor patterns of an agent. They compared three reward systems with 36 agents to analyze the impact of penalizing agents for being too close to other agents. Reward set A served as a baseline with no penalization, reward set B rewarded seeing other robots while reward set C heavily penalized collision. The reward of an agent at each step was calculated based on the number of pixels corresponding to a landmark or obstacle in the image. Using reward set A, they achieved good results, with 89.56% of the robots reaching their destination. However, set B achieved a completion rate of 57.51%, while set C achieved a rate of 21.78%. To investigate the effect of static obstacles on the three reward sets, two new environments with obstacles were created. Reward set A achieved the highest performance with a rate of 56.44% and 67.12%, set C followed with 35.84% and 51.65% and set B achieved 6.14% and 9.81%. The results from these three experiments with the three reward sets can be seen in Table 3.

Overall, the two experiments demonstrated that a baseline reward system, such as reward set A, outperformed more complex and specific reward systems. Reward set A achieved a good performance regardless of the presence of newly added static obstacles, reward set C showed a small improvement with obstacles, while reward set B showed a decrease performance when including static obstacles. Nonetheless, the authors showed the feasibility of using a basic CNN-based DQN to control a large swarm of ground vehicles between two landmarks.

Yasuda and Ohkura [152,153] used a DQN combined with a deep neural network (DNN) to control a swarm of ground robots. Unlike some previous work, in their system, the agents uploaded their input/output reward to a central server, which contained an experience replay memory [154]. Instead of immediately calculating the values using the DNN, the state-action pairs were temporarily stored in the experience replay model. The model could then learn using random pairs, which enables the logical separation of the learning phase and the gaining phase for an improved performance. To evaluate their approach, the authors employed a swarm of small two-wheeled robots powered by an Arduino and a Raspberry Pi. The goal of the swarm was to navigate as much as possible between two landmarks, arranged similarly to a soccer field. Their algorithm was evaluated by comparing the number of round trips achieved by the swarm and the impact of varying the frequency of experience sharing for swarms consisting of one, five and ten robots. The results showed that a higher number of robots in the swarm led to a better performance compared to a single robot. Additionally, frequent experience sharing, such as at every step, achieved a better performance than less frequent sharing. Overall, the authors demonstrated that increasing the number of agents in a swarm increases the performance while increase the frequency of sharing also increases the performance. They highlighted the importance of experience sharing in multi-agent systems, with more specific details of the experiment and results available in Yasuda and Ohkura [152,153].

Similarly, Foerster et al. [155] proposed integrating experience replay with deep multi-agent reinforcement learning to address scalability challenges. Two methods of integration were compared by the authors, an importance sampling scheme (ISS) and multi-agent fingerprinting (MAF). ISS enables off-environment learning [156], which allows learning from a target environment while continuously generating data from a different environment. Using ISS, past reward data can be used to calculate the current reward to reduce the computation time of the algorithm, increasing the efficiency. The second method, MAF, addresses and embraces the non-stationarity issue of the problem to increase the performance of the reinforcement learning algorithm. Treating other agents as stationary entities simplifies the problem but fails to consider the ever-changing policies of other agents. The authors proposed MAF, which assigned each agent the weights of all other agent networks in its observation function. However, implementing this approach with large networks and a large quantity of agents becomes impracticable. To address this, a low-dimensionality approach that utilized only the data in the replay buffer was proposed. Furthermore, instead of using the whole network, they used the sequences of policies generated for the trajectory of each agent. The researchers conducted experiments using the StarCraft II game to provide actions to a swarm of agents using each method separately and a combination of both. It should be noted that StarCraft II is a video game rather than a swarm of robots; nonetheless, the behavior and problem complexity were deemed interesting for our review. The authors compared the performance of a DQN using both a feed-forward network (FFN) and a recurrent neural network (RNN). The results showed that the integration of experience replay, of either two approaches or a combination of both, led to a better convergence than not any using experience replay. However, ISS showed an overall lower performance than other methods while MAF greatly improved the learning in the FNN. Combining both ISS and MAF outperformed the individual approaches and yielded excellent results in both the FNN and RNN. Their work demonstrated the importance of communication between agents by comparing different integrations of experience replay in multi-agent DQN.

Jiang et al. [157] used a DQN to control a swarm of small ground robots equipped with LiDAR cameras in a navigation scenario. The input of their network consisted of an occupancy map, the difference between the current and desired direction of an agent and the desired formation distance. The occupancy map captured important information about the environment such as the presence of other agents and was processed by a CNN. The output of the CNN, the difference distance and desired formation distance were all connected to fully connected layers. A command vector is generated by flattening the output of the fully connected layers to provide an action to the agent. The authors trained and evaluated their approach using the Coppelia Simulator [158] for a group

Table 4
Completion rate from Tan and Karaköse [159].

Environment	Completion rate
1	**91.5%**
2	90.6%
3	84.0%
4	77.3%
5	75.4%
6	84.7%
7	83.2%

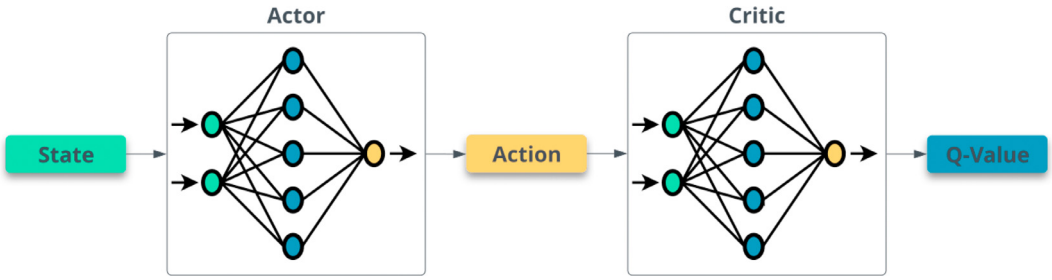


Fig. 5. DDPG algorithm.

moving scenario. Three agents were simulated and three scenarios were tested, a constant velocity and two with time-varying desired velocities scenarios. The success rate and convergence time were used to evaluate the overall performance of the algorithm using the three scenarios. The results demonstrated that their approach achieved an average success rate of 90% and 96% using a 5% and 10% error tolerance, respectively. Additionally, their method achieved a fast convergence with an average convergence time of 10 seconds. The authors showed the feasibility of using a DQN to control a group of three ground vehicles using data provided by a LiDAR camera.

More recently, Tan and Karaköse [159] focused on combining a DQN with a PPO to control a group of three small robots in a navigation task. Their objective was to enable the swarm of robots to navigate an environment while avoiding obstacles and collision between each other. The PPO had of two responsibilities, first it had to estimate the probability of each action based on the surrounding area and choose an appropriate action. The second task of the PPO was to interact with the environment using the current policy to update the Q-values, improving the overall performance of the robots. The authors trained their algorithm using a grid size of 20×20 with obstacles while the validation consisted of nine fixed environments. The completion rates achieved in seven of the presented environments can be seen in Table 4. The results showed that the proposed approach achieved completion rates ranging from 75.4% to 91.5% while an average completion rate of 83.08% was achieved across all nine environments. These results show the possibility of solving navigation tasks by combining policy-based reinforcement learning algorithms such as PPO with value-based algorithms such as a DQN.

4.5. Deep deterministic policy gradient

Hüttenrauch et al. [160] proposed controlling a swarm of rovers using Deep Deterministic Policy Gradient (DDPG) in a simulated 2D environment with agents inspired by Kilobot robots. Kilobots are small unmanned robots with limited sensing abilities and capture images using a first-person view. The authors compared two reinforcement learning algorithms, a DQN and a DDPG which are shown in Figs. 4 and 5 respectively. The DDPG is an extension of the DQN that combines an actor model with a critic model to achieve a better performance. The actor model, which represents the policy, uses a neural network to calculate the best action given a state, similar to a DQN. Compared to a DQN, a critic model is also provided to calculate a reward when given a state-action pair. DDPG can be described as having one network providing the action and another network providing the reward for that action. The authors proposed a guided DDPG, where the critic network also evaluated a joint action. They also incorporated a target network and experience replay in their approach to increase the stability of the networks. In this work, each agent had a detection radius to simulate the communication distance that was used to detect if a target was in range. The objective of their research was to group a swarm ranging from two to eight agents while also localizing a target. The results showed that training with seven or eight agents resulted in noisy predictions and poor performance. However, when evaluating with few agents, the model performed better when the models were trained with a similar quantity of agents rather than more agents. Overall, their system was able to guide a swarm of small ground rovers towards a centralized objective with a swarm between two and six agents.

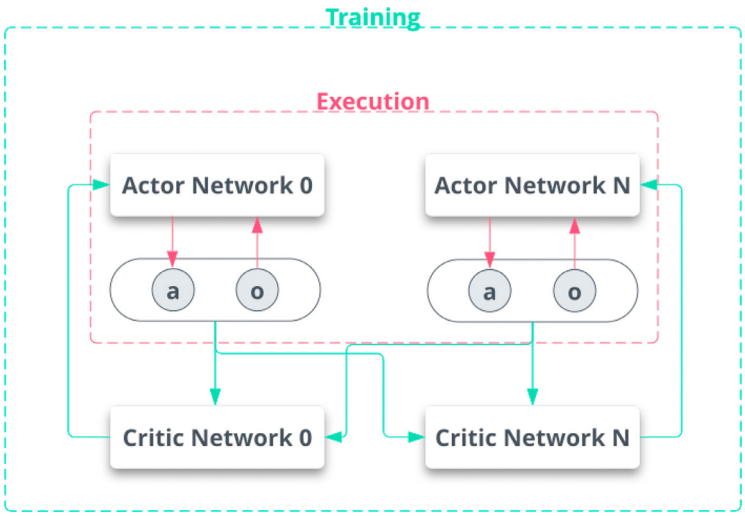


Fig. 6. MADDPG algorithm.

4.6. Multi deep deterministic policy gradient

Yang et al. [161] focused on controlling a swarm of agents with a continuous action space, using a modified version of DDPG called Multi-DDPG (MDDPG). The MDDPG proposed by the authors also enabled concurrent learning from both images and sensors for each agent, providing more information to the agents. In a regular DPPG, a single critic model and a single actor model are used while MDDPG consists of a critic model with multiple actor model. To account for this modification, the critic model was modified to have multiple state-action output values, one for each actor model. The authors used multi-layer perceptron convolutional (MLPCONV) layers [162] to combine data from images and sensors, reducing the dimensionality by 75%. MLPCONV layers replaced the first, third and last layer of the actor models and were frozen during training, allowing only the convolutional layers to be trained. The last MLPCONV layer enabled the networks to seamlessly share parameters between them. The authors compared their MDDPG with a DDPG proposed by Lillicrap et al. [163] on 12 movement-based control tasks in the Gazebo simulator. They found that their approach achieved similar results to a regular DDPG while using considerably fewer parameters, making it more effective. They then conducted tests on 12 tasks to verify the performance of their MDDPG and achieved good results. Overall, the authors demonstrated that the MDDPG was capable of achieving results similar to DDPG with significantly fewer parameters, thus making it more efficient.

4.7. Multi-agent deep deterministic policy gradient

Singh et al. [164] used reinforcement learning to efficiently capture a target using multiple agents in a cooperative manner. They used a Multi-Agent Deep Deterministic Policy Gradient (MADDPG), which is an extension of DDPG that enables better coordination between the agents and was originally proposed by Lowe et al. [165]. MADDPG enables the system to achieve a better performance by incorporating a centralized critic training and decentralized actor execution. During training, the current states of all agents are shared to calculate the reward using the centralized critic model. The centralized reward model allows for better cooperation while the decentralized execution enables the agents to make independent decisions. The authors tested their approach in a simulated 2D environment with a single non-stationary target and multiple agents. The target moved at twice the speed of the agents and moved around using an algorithm based on the Voronoi regions. After 350 iterations, their MADDPG achieved an impressive capture rate of nearly 100%. Their research demonstrated the effectiveness of their MADDPG in capturing a moving target in a simple environment. However, comparing their approach against other algorithms such as a DDPG using various scenarios would increase the validity of their proposed approach (Fig. 6).

Huang et al. [71] focused on using the MADDPG to increase the performance and self-sustainability of deep-space exploration missions using robots. Space exploration using robots presents various demanding challenges, such as the cost, complexity and the unknown nature of the environment. In this paper, the authors proposed using MADDPG to address these challenges by simulating a swarm of ground rovers on a Mars mission. In their approach, the algorithm used multiple sub-policies during training to increase the robustness of the algorithm. The authors developed an experience sample optimizer to select the best sub-policy rather than choosing a sub-policy randomly. To evaluate their approach, they simulated a Mars mission involving ground rovers and UAV/satellites providing an overhead view of the rovers. The goal of the mission was to explore all targets while simultaneously avoiding collisions with other agents and obstacles. The results of their experiments demonstrated that a regular MADDPG with random sub-policy selection can be used to extent for deep-space exploration. Their sample optimizer was found to significantly improve the performance of the MADDPG algorithm, this increase in performance can be seen in Table 5.

Table 5
Sample optimizer effect on performance from Huang et al.[71].

Agent	Reward increase	Time decrease
2	1.42%	2.26%
3	3.08%	2.50%
4	4.71%	**2.69%**
5	**8.87%**	1.79%

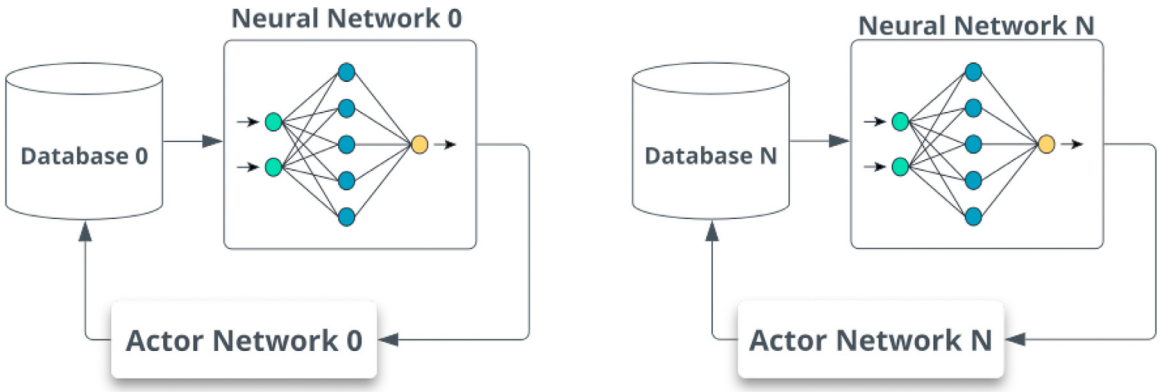


Fig. 7. IDDPG algorithm.

Ho et al. [166] proposed an approach based on the MADDPG for swarm robotics in a warehouse setting. In this paper, the authors proposed a grid-based system compromising multiple robots, a pick-up location, a drop-off location and a charging station. They also considered important factors such as incorporating a kinetic model, an energy model and a communication model for a more realistic system. The objective of each individual robot was to navigate from the pick-up location to the destination while occasionally recharging at the charging station. To accomplish this task, the authors modified the MADDPG approach to have three outputs, one for each individual robot. These outputs included the movement of the robot, resource allocation and task assignment for a fully autonomous system. The researchers compared their MADDPG against a standard DDPG and found that both approaches achieved good results, with the MADDPG outperforming the DDPG. Four energy optimization techniques were compared for the MADDPG, exhaustive search, single agent DDPG, max power and fixed bandwidth. The results showed that using the optimal energy optimization technique achieved the best energy efficiency, with the MADDPG and DDPG algorithms achieving a similar efficiency. Overall, the authors demonstrated the adaptability of reinforcement learning algorithms, such as MADDPG and DDPG, in the context of warehouse logistics. By combining more complex information, such as a kinetic model, the algorithms were also able to optimize the energy efficiency of the robots.

4.8. Federated learning deep deterministic policy gradient

In recent years, Na et al. [167] introduced a novel federated learning (FL) DDPG algorithm (FLDDPG) for navigating a swarm of robots while avoiding collisions. Federated learning involves distributing the training of an algorithm across multiple devices where the data is also stored on these devices. The authors selected FL since maintaining a constant strong connection between a central server and all robots is unrealistic. By adopting a distributed approach, the agents are able to communicate and process information without relying on a central server; rather they rely on themselves. They compared their FLDDPG approach with three DDPG-based approaches: individual DDPG (IDDPG), shared network DDPG (SNDDPG) and shared experience DDPG (SEDDPG). In IDDPG, each agent had its own DDPG and local database without a central server or communication between them. SNDDPG allowed for better communication by utilizing a shared DDPG and a shared database through a central server. SEDDPG was a mixed approach that employed a shared database but individual DDPGs, enabling agents to communicate their positions. FLDDPG utilized an individual DDPG and database for each agent, periodically sharing the model with a central server for fine-tuning. This reduced the communication required between the agents and the central server while enabling information sharing. The IDDPG, SNDDPG, SEDDPG and FLDDPG can be seen in Figs. 7–10 respectively.

To compare the approaches, the authors trained the algorithms using navigation and collision avoidance scenarios in the Gazebo simulator. Six environments were created with a variety of obstacles while six agents simulating “TurtleBot3” robots were deployed. TurtleBot3 are small ground robots with a three-wheeled differential drive system and a 360-degree range finder. Three experiments were performed to validate the FLDDPG, different weights averaging, soft weight update and a comparison of the FLDDPG against the three DDPG-based algorithms. A navigation reward, a collision avoidance reward and a smooth trajectory reward were combined to create a reward system. The performance of various parameters and algorithms were compared using the completion rate, completion

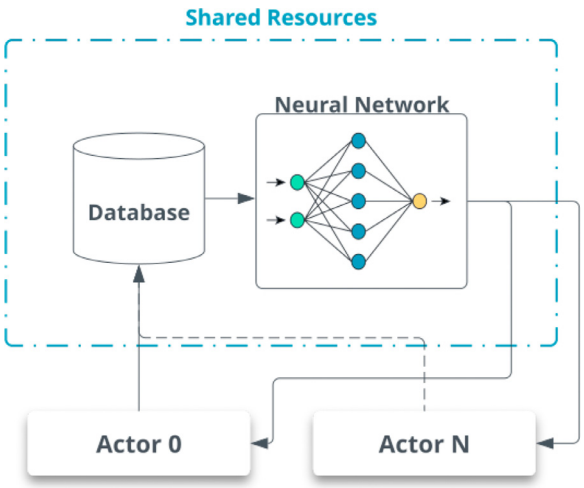


Fig. 8. SNDDPG algorithm.

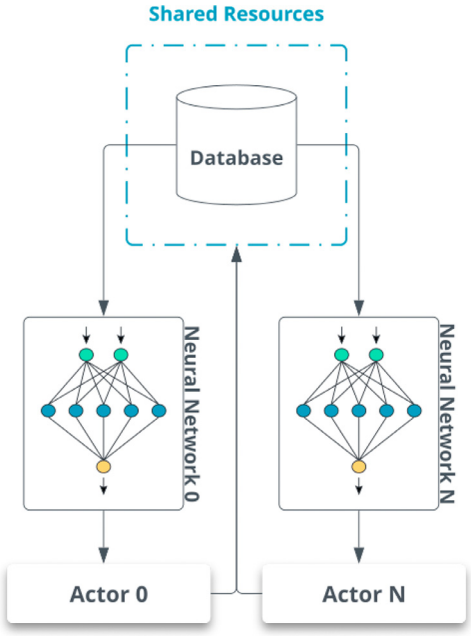


Fig. 9. SEDDPG algorithm.

Table 6
Comparison of results from Na et al. [167].

Method	Completion rate	Completion time	Trajectory efficiency
IDDPG	17%	14.52 s	93.65%
SEDDPG	24%	11.86 s	90.74%
SNDDPG	34%	11.66 s	92.08%
FLDDPG	**96%**	**11.19 s**	**98.97%**

time and trajectory efficiency. FLDDPG demonstrated impressive results across all metrics as can be seen in Table 6. It achieved a faster convergence compared to other algorithms, with a minimum difference of 15 epochs compared to SNDDPG and maximum difference of 195 epochs compared to SEDDPG. FLDDPG also achieved the highest completion rate at 96%, while the other methods achieved a rate of 17% (IDDPG), 24% (SEDDPG) and 34% (SNDDPG). It also demonstrated a faster completion time with a time of 11.19 s for FLDDPG, 11.66 s for SNDDPG, 11.86 s for SEDDPG and 14.52 s for IDDPG. Additionally, the FLDDPG achieved better trajectory efficiency with a rate of 98.97% compared to 93.65% for IDDPG, 92.08% for SNDDPG and 90.74% for SEDDPG. Overall, the authors

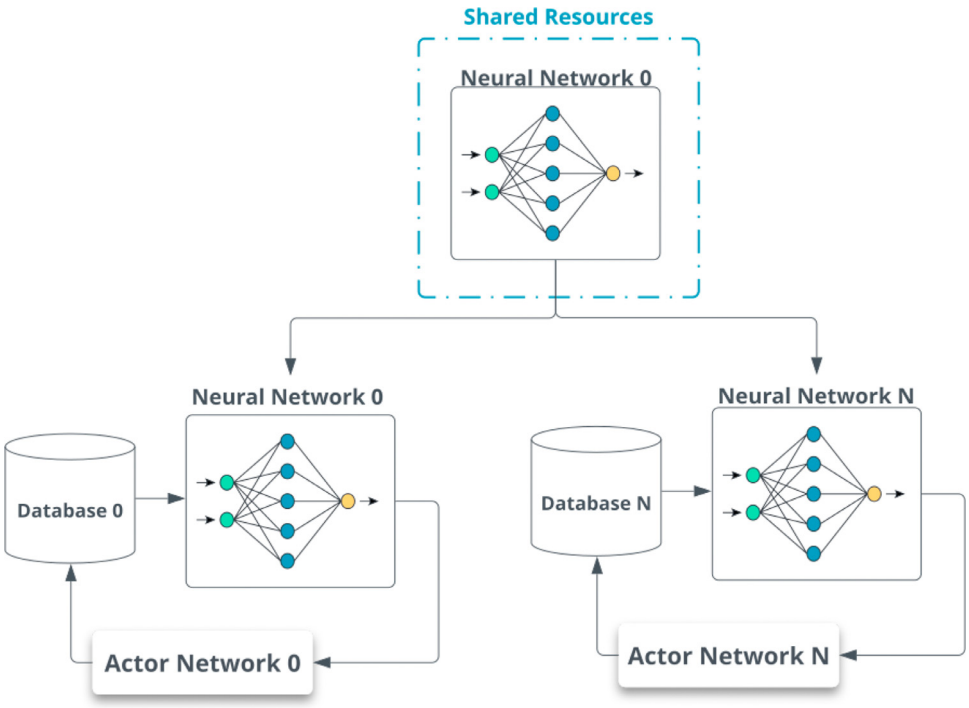


Fig. 10. FLDDPG algorithm.

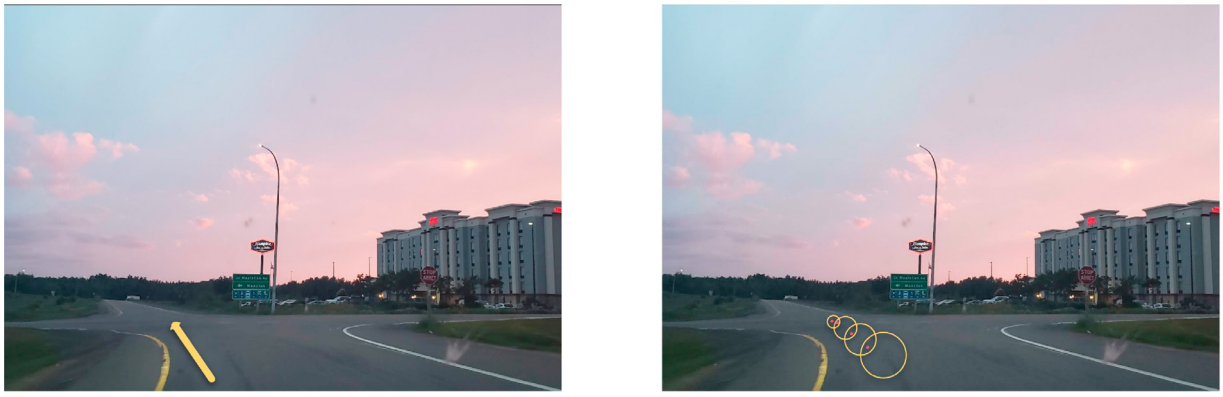
introduced a federated learning DDPG algorithm, named FLDDPG, to control a swarm in a warehouse setting and demonstrated its ability to achieve a good performance. The authors compared their novel federated learning-based algorithm with three popular DDPG algorithms using various performance metrics, such as the completion rate. FLDDPG demonstrated impressive performance improvement on two metric while it performed on par with the other methods for the third metric. The authors showed the potential of using federated learning on popular reinforcement learning algorithms to improve their performances while reducing the need for a central server.

4.9. Multi-step, multi-agent neural network

Li et al. [168] introduced a Multi-Step, Multi-Agent Neural Network (MSMANN) to learn control policies using a neural network. Their neural network, known as a distributed policy network (DPN), utilized a fixed-size vector of inputs and outputs. Each agent in the system followed the distributed policy and had multiple tasks: interpreting the observations of an agent, receiving communications, determining the best action and broadcast this information to other agents. To reduce the complexity of the algorithm, a neighbor discretization was used to normalize the received communication into a fixed-length vector. They utilized supervised learning to construct their distributed policy based on a pre-defined centralized control policy. Two scenarios were used to evaluate their approach, a rendezvous scenario with limited visibility and a particle assignment task. In the rendezvous scenario, multiple agents needed to be grouped within a specific region on a 2D plane. The results demonstrated that the MSMANN outperformed all state-of-the-art methods in this scenario. However, it was noted that the performance was reduced when the number of agents was increased. The particle assignment task involved multiple targets that all had to be covered by at least one agent. The MSMANN approach was able to efficiently cover every target; however, their method suffered from a convergence issue when a large number of agents were employed. Overall, their proposed MSMANN algorithm demonstrated a good generalization in both a rendezvous scenario and particle assignment task. However, the authors identified scalability challenges of MSMANN when dealing with a high number of agents in both scenarios.

4.10. Trust region policy optimization

Hüttenrauch et al. [169] proposed utilizing a Trust Region Policy Optimization (TRPO) [101] for swarm robotics using reinforcement learning. TRPO is a popular optimization method, alongside gradient descent, which is commonly used as a linear search in artificial intelligence. Unlike gradient descent, TRPO locates the best position within a specific radius, enabling the control of the learning speed by expanding or shrinking the search radius. The linear search from gradient descent and trust region search can be seen in Fig. 11(a) and (b) respectively.



(a) Linear search (b) Trust region search

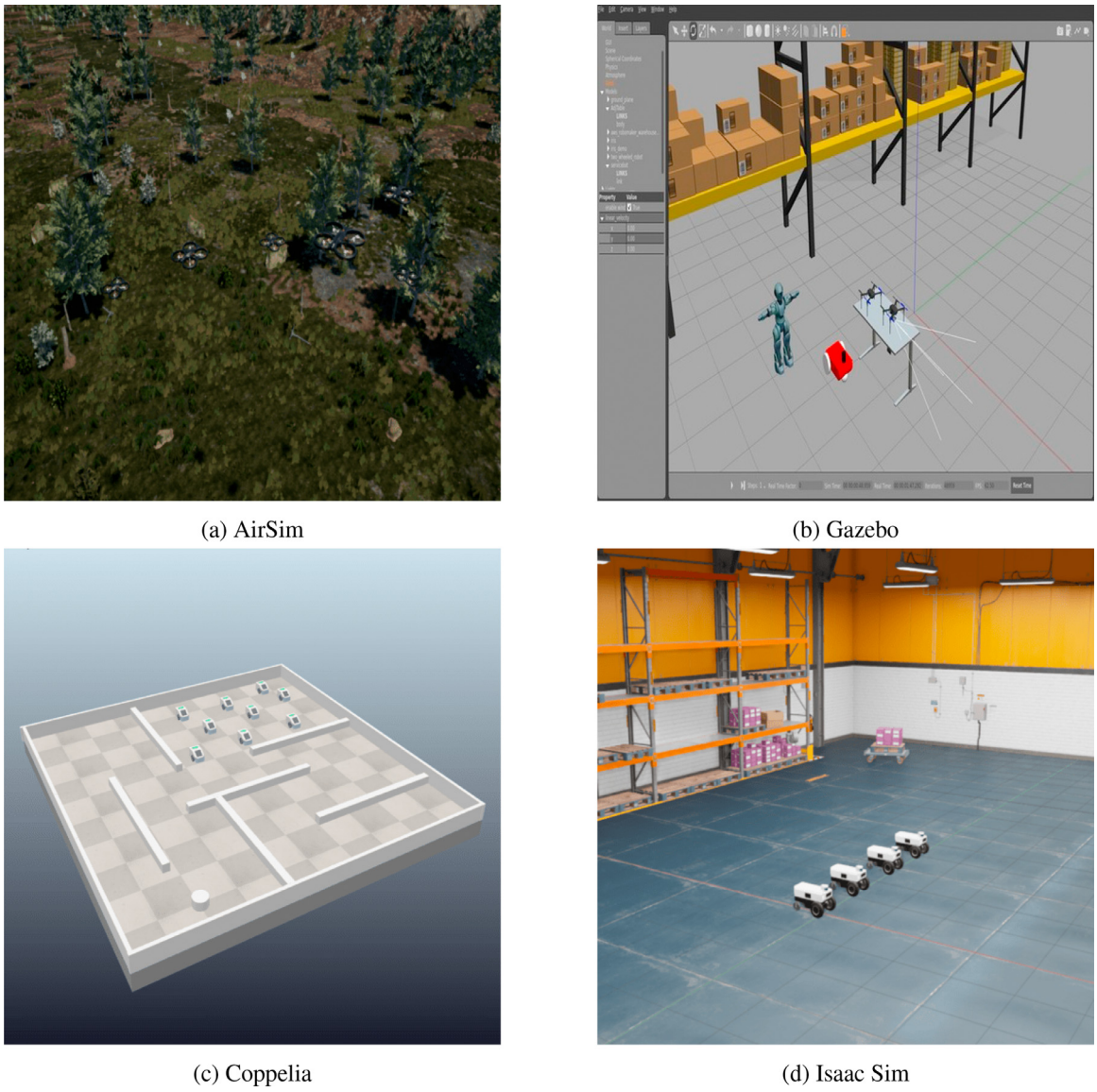
Fig. 11. Examples of searches.

To utilize TRPO with reinforcement learning, the authors adapted a multi-agent learning system similar to the one proposed by Gupta et al. [170]. Their policy utilized a DNN combined with various information encoding protocols. The authors compared two histograms as communication protocols to encode information and reduce complexity. Two types of histogram were compared: One-dimensional histogram and multidimensional histogram. The researchers conducted two experiments to evaluate their approach and compare the two histograms. In the first experiment, the objective was to maintain a specific distance between the robots and their environment, as well as between the robots themselves. The second experiment involved utilizing the short communication range of the robots to establish a communication link between two distant points. Both experiments were conducted in a small 1 m × 1 m region with ten randomly initialized agents. The results of the first experiment demonstrated a good performance for both the one-dimensional and multidimensional histogram approaches. The authors found that a histogram with a length of two achieved the best results, with the performance decreasing as the length increased. Similarly in the second experiment, a multidimensional histogram with a length of two yielded the best results by constructing the shortest path between two points. Overall, the authors showcased the possibility of using policy-based reinforcement learning algorithms to control a swarm of robots. They compared two histogram approaches to reduce dimensionality, a one-dimensional and multidimensional histograms, and found that a histogram with a length of two achieved the best results. These results demonstrate the benefit of sharing information between agents using an appropriate amount of data.

Subsequently, Hüttenrauch et al. [150] continued their work on deep reinforcement learning with TRPO. Their focus was the comparison of different state representations in a pursuit and evasion scenario involving multiple agents. Similar to their previous research, they employed TRPO along with a mean embedding of distributions to reduce the dimensionality of the data into a low space vector based on the work of Smola et al. [171]. The authors proposed comparing four embedding methods: neural network, histograms, Radial Basis Functions (RBFs) and other pooling methods. They also compared a constant length feature space with variable one. To implement their algorithm, a DQN was combined with a Local Observation Model (LOM), which provided information such as velocity amd object distance. They also incorporated a Local Communication Model (LCM) to facilitate the communication between agents using the LOM. Their first experiment involved a rendezvous scenario where agents had to group themselves by minimizing the distance between each other. All embedding methods achieved good generalization with 20 agents, with the neural network achieving the best performance. The neural network with the extended set of information from the LOM demonstrated a 25% faster convergence compared to the neural network and RBFs with the basic set. The second scenario focused of a pursuit and evasion with ten agents and a single non-stationary evader in an infinite 2D plane. The configuration of the environment made trapping the evader in a corner technically impossible, encouraging better cooperation between the agents. In this experiment, every embedding method achieved good results, while the neural network with the base and extended set achieved the best performance. Their third experiment consisted of a pursuit and evasion with multiple agents and multiple evaders, further complicating the task. To adapt their model, the evaders were embedded with the agents, enabling the agents to extract information directly from the evaders. This experiment deployed 50 agents and five evaders. The neural network achieved the best results, particularly with the extended set of information. Overall, the authors demonstrated that using a neural network for embedding allowed for better performance to other approaches. Additionally, they showed the importance of information sharing between the agents for a more efficient swarm system. The authors emphasized the significance of high-dimensional vectors over lower-dimensional vectors when embedding the information in multi-agent systems.

5. Simulators

In previous sections, we saw how crucial reinforcement learning is in the field of controlling swarms of unmanned vehicles. However, creating an efficient control system for a swarm using reinforcement learning algorithms requires extensive training and



(a) AirSim (b) Gazebo

(c) Coppelia (d) Isaac Sim

Fig. 12. Examples of simulators.

testing. Since reinforcement learning is based on an iterative trial and error approach, data collection is a complicated task for unmanned vehicles. Using autonomous robots such as cars or drones in populated areas is dangerous, while building a closed circuit is expensive and unrealistic. Simulators have been proposed as a solution to remove the need of real-life data collection, allowing to completely remove the safety concern while reducing cost enormously [172]. In this section we present nine simulators: AirSim, Sim4CV, Gazebo, Kilombo, ARGoS, OpenAIGym, Isaac Sim, Coppelia, PyBullet and other relevant tools. We compare the simulators based on the physics engine, robots available, simulated sensors and the various environmental parameters. Additionally, we compare them on their abilities to train reinforcement learning algorithms with libraries such as Pytorch [13] and Tensorflow [173]. Visual examples of four simulators can be seen in Fig. 12 while an overview of all simulators can be found in Table 7.

AirSim [174] is a photorealistic simulator used for a variety of vehicles such as quadcopters or cars. This open-source simulator was created by Microsoft using Unreal Engine 4 [175]. Unreal Engine is a real-time 3-D toolkit defined as a photorealistic and physic realistic game engine. It was created to help supplement missing data in machine learning, deep learning and reinforcement learning. The simulator offers premade environments such as 12 km of rural roads, 20 blocks of an urban setting, a windmill farm, a dense redwood forest or ”Africa” which offers uneven terrain and animated animals. Users are also able to create custom environments using the Unreal Engine 4 platform for their specific needs. The time of day and weather can also be adjusted to add rain, road wetness, snow, leaves, dust, fog and wind. Various sensors and imaging modalities can be used such as distance sensors, barometers, inertial measurement unit (IMU), LiDAR and infrared imagery. The simulator enables

Table 7
Overview of simulators.

Simulator	Engine	Robot types	Sensors	Environment	Languages
AirSim [174]	Unreal Engine 4	Drones (Multirotors), Cars	RGB, IR, LiDAR, GPS, Barometer, IMU	Urban, Rural, Wind, Rain, Snow, Fog, leaves, Dust	ROS, C+, C#, Java, Python
Sim4CV [178]	Unreal Engine 4	Drones, Cars	RGB, Pixel-level Segmentation	Rural, Urban, Suburban, Indoor	C+, Python, MATLAB
Gazebo [177]	Open Dynamic Engine	Drones, UAVs, Cars, Boats	RGB, IR, LiDAR, IMU, GPS, Force, Distance, Altimeter, Depth, RFID	Urban, Rural, Farm, Warehouses	Python, C+ and Java through ROS
Kilombo [182]	Custom Engine	Kilobots	Promixity, Distance	Simulated graph	C
ARGoS [184]	Multi-engine	Simple ground robots	RGB, Altitude, Proximity	Wind, Orbit, Water, Flow, Terrain	C+
OpenAIGym [185]	MuJoCo physics engine	Drones/UAVs, Cars, Boats, Industrial robots	RGB, Depth, Distance, Simulated IR, Speed	Urban, Rural, Wind, Rain, Snow, Dust, Storms	Python
Isaac Sim [189]	NVIDIA Omniverse	Industrial robots, Ground robots	RGB, Infrared, LiDAR, IMU	Warehouses, Houses, Fully customizable	C, C+, Python, Java, Lua, MatLab, Octave
Coppelia [158]	Multi-Engine	RGB, Infrared, LiDAR, Proximity	Various industrial robots, Ground vehicles	Warehouses, Lighting, Fog	ROS, ROS2, Python
PyBullet [192]	Bullet	ground and aerial vehicles, industrial robots	RGB, Segmentation, Depth, Optical flow	Fully customizable	C, C+, Python, Java, Lua, MatLab, Octave

swarm systems to be simulated using tools such as a Robot Operating System (ROS) [176]. This software can also import models from other simulators such as the Gazebo simulator [177]. In terms of training reinforcement learning algorithms, the AirSim Application Programming Interfaces (APIs) based on the RPC library allows for an easy connection. The APIs are accessible through languages such as ROS, C++, C#, Java and Python which facilitates the use of libraries such as Pytorch and Tensorflow. Due to the popularity of this simulator, a large quantity of Python examples can be found online to train reinforcement learning algorithms.

Sim4CV [178] is a realistic simulator used for autonomous driving, autonomous drone flying and aerial object tracking. The simulator is based on Unreal Engine 4 to provide realistic physics, realistic visuals and a fully customizable environment. The environment can be adapted to any criteria such as for driving scenarios, group understanding, pose estimation, multi-agent collaboration and more. The lighting, shadows, textures, objects and the time of day can all be modified using the simulator to provide realistic data. Currently, the simulator is limited to RGB images, depth, segmentation, and ground truth labeling with drones and cars available. The authors provide C++, Python and MATLAB Socket Interfaces to train algorithms using libraries such as Pytorch and Tensorflow.

Gazebo [177] is a physic-realistic simulator built on the Open Dynamic Engine (ODE). Realistic rendering with dynamic lighting, textures and shadows can easily be created with this simulator. Similar to AirSim, different sensors to perceive a variety of data such as images and distances can be simulated. A simulation can be run slower, at the same speed or faster than real-life by adjusting a time parameter. Large numbers of vehicles such as drones, cars, planes and underwater vehicles can be simulated. This simulator also offers specific tools to reproduce industrial scenarios such as industrial robots and warehouses. A ROS interface based on the Player Robot Device Server [179] is provided to interact with the simulator. Since ROS is language agnostic, the simulator can be used with any languages such as Python, C++ and Java to train algorithms using Pytorch or Tensorflow. The Gazebo simulator has been used in a variety of challenges such as the Virtual RobotX Competition [180] and the DARPA Subterranean Challenge [181].

Kilombo [182] is a lesser-known and more specific simulator used on Kilobots. Kilobots are small ground robots that use rigid legs with vibrations to move in any direction. These robots are able to communicate with each other and calculate the distance between them using infrared sensors. The authors created their simulator ”Kilombo” for multi-agent collaboration with three goals in mind. First it had to simulate the real functions of the robots, second it needed to run significantly faster than the real robots and lastly the transfer from simulator to robots had to be simple. Using a variety of testing methods, they found that their simulator can effectively simulate Kilobots. Kilombo is written in C which supports a limited quantity of artificial intelligence libraries such as Tensorflow and FANN [183]

ARGoS [184] is a physic-realistic simulator for large-scale ground robot simulations. Dynamic environment aspects such as water, flow and wind can easily be simulated to offer realistic environments. ARGoS is limited to simple ground robots with modifiable components such as mechatronics, sensors and actuators. The robots can communicate with each other using a medium plugin that enables a user to easily implement a suitable collaboration algorithm such as wifi, stigmergy through RFID or vision techniques. The authors underlined tree goals for their simulator; it had to be highly accurate, flexible and efficient. ARGoS is not built on any specific engine; rather, a plugin feature allows the user to select any engine. An environment can also be divided in multiple regions which can each be controlled using their individual engine. The simulator is written in C++, supports multi-threads for a better performance and can employ libraries such as Tensorflow and Pytorch.

OpenAIGym [185] is a toolkit used to train, validate and test reinforcement learning algorithms. This simulator is more geared towards reinforcement learning for video games but can be adapted for swarm robotics. This Python-based simulator can

simulate a variety of scenarios such as video games, board games and 3D physic simulations. OpenAIGym is built on the open-source MuJoCo physics engine which can simulate a variety of dynamics. The OpenAIGym API simplifies the comprehensibility of the simulator and the usage of libraries such as Tensorflow and Pytorch. Additionally, public libraries allow to easily import drones [186], cars [187] and industrial robots [188] for swarm designs.

NVIDIA Isaac Sim [189] is a newer photorealistic robotic simulation and data generation toolkit. The toolkit is powered by the NVIDIA Omniverse platform [190] which enables real-time, multi-GPU and multi-node simulations. The Omniverse platform is based on the Universal Scene Description (USD) by Pixar, Material Definition Language (MDL) and the PhysX physics engine. The toolkit is currently focused on industrial applications with unmanned ground vehicles and industrial robots and support sensors such as RGB-D, LiDAR and IMU. The environment is fully customizable and can simulate many scenarios such as a warehouse or a house. Isaac Sim supports ROS, ROS2 [191] and Python natively for the manipulation of robots. Libraries such as Tensorflow, Pytorch and Caffe can easily be implemented through Python to train reinforcement learning algorithms. The toolkit offers a variety of modules such as the Cortex decision framework, the GYM reinforcement learning tool and the Omnigraph visual programming tool. A user is also able to easily transfer simulations from Gazebo to Isaac Sim and vice versa.

Coppelia Simulator [158] is a robotic simulation toolkit used primarily for industrial robots. The simulator offers various robot types such as SCARA robots and other types of ground vehicles. Since Coppelia is built on a distributed control architecture, each entity can be controlled individually with embedded scripts, ROS, plugins or remote API clients. This makes the simulator highly versatile for swarm applications such as factory automation with various types of robots. The controllers can be written in different languages such as C/C++, Python, Java, Lua, MatLab and Octave which offer libraries such as Tensorflow or Pytorch. The simulator also offers four physics engines (Bullet Physics, ODE, Newton and Vortex Dynamics) to simulate various mechanics such as collisions. Coppelia offers various sensors such as RGB, infrared and LiDAR while also providing parameters such as ambient light or fog for realistic scenarios.

PyBullet [192] is a simulator that provides a Python interface for the Bullet Physics Library [193]. The library offers forward dynamics simulations, inverse dynamics computation, forward/inverse kinematics and collision detection. Due to the Bullet Physics library, PyBullet offers realistic physics such as rigid and soft body collisions. The environments can be rendered using methods such as a CPU renderer or OpenGL visualization for efficient 3D graphics. Various robots such as ground vehicles, drones, quadcopters,industrial robots and more can easily be implemented. Since the simulator is built using Python, reinforcement learning libraries such as Tensorflow and Pytorch can easily be applied.

Others Robotarium [117] is a swarm of small ground robots which can be accessed remotely for testing. This remote swarm provided by Georgia Tech is free to use and provides a few dozen ground vehicles. The Robotarium uses Vicon motion-capture cameras [194] and unique patterns of Vicon Pearl markers [195] to identify and locate the vehicles. They also provide a high-resolution video camera to capture the experiments and provide feedback to the user. The swarm can be controlled using an API with either Python or MATLAB while a simulator is provided to verify the scripts. Overall, this remote swarm provides the user with an easy and reliable test bed for a swarm of ground vehicles.

StarCraft II [196] is a real-time military strategy game using an axonometric projection view of a battlefield. The goal of the game is to collect diverse resources to develop an increasingly complex military force to eliminate all enemies. The user must simultaneously build an army, workers and a defense system to defend and attack the enemies. Three races are available to play (Terran, Zerg and Protoss) each with different units and mechanics. A player is declared the winner when all the enemy forces, bases and resources are exhausted and the enemy player is forced to forfeit the game. This game is a complex strategy game which requires both a micro and macro-management system of the resources similar to a swarm.

6. Discussion

In Sections 2, 4 and 5 we respectively saw the various applications, algorithms and simulators for swarm robotics based on reinforcement learning. First, we explored applications in various fields that can benefit from swarm robotics, focused on reinforcement learning-based control systems. We surveyed applications in the sectors of forest fire prevention and control, agriculture, forestry, logistics, military and search and rescue missions. These applications have demonstrated the flexibility of swarm systems such as a swarm of UAVs, UGVs or a combination of both for a more efficient system. This section of our review has also established the viability of reinforcement learning in controlling swarms for simple and complex tasks. By comparing the various applications of reinforcement learning-based swarm robotics, it provided us with an overview of the current state of the field. We continued our review by defining important reinforcement learning terms and classification of algorithms such as learning the model and given the model. We focused our review on swarm robotics utilizing value-based learning the model algorithms due to the extensive work in this field in recent years. We first explored the use of SARSA to control a swarm of agents in various scenarios such as path planning, grouping around a leader, obstacles avoidance and navigation. Research proposed by Speck and Bucci [118] demonstrated the benefit of combining SARSA with behavioral rules such as Reynolds Flocking to increase the performance on navigation and collision avoidance scenarios. Similarly, research demonstrated that Q-learning can be used for a variety of tasks such as a path planning to reach a fire, creating a downlink wireless network or for the localization of a ground vehicle using UAVs. Q-learning has demonstrated its ability to easily be adapted to be used in real-life to play tic-tac-toe against human using small drones. Das et al. [128] demonstrated that combining Q-learning with PSO and a vector differential operator outperformed basic Q-learning in various experiments. Both Q-learning and SARSA have demonstrated their performances in multi-robot systems, while improvements have increased the general performance of the algorithms. We continued our review by exploring four multi-agent Q-learning systems that utilized joint spaces to enable a more direct communication between the agents. Multi-agent Q-learning has achieved a good performance for covering a field, tracking

multiple ground targets using UAVs and for carrying objects such as sticks, triangle and boxes. In early works, Wang and De silva [132] found individual Q-learning to be more effective than multi-agent Q-learning in a box pushing scenario due to the complexity of the task. However, recent work in the field of multi-agent Q-learning [74,133,140] have demonstrated multi-agent Q-learning to significantly outperform individual Q-learning. Sadhu and Konar [133] demonstrated an important increase in performance for various objects carrying tasks using multi-agent Q-learning algorithm compared to other algorithms. Additionally, the authors tested the viability of the algorithm using Khepera-II robots for a stick-carrying scenario and achieved similar results from the theoretical experiments. Chen et al. [140] showed that integrating improvements such as a searching time constraint to multi-agent Q-learning reduced the average search time of the algorithm by 28%. Similar to Q-learning and SARSA, we demonstrated that multi-agent Q-learning benefited from improvements by increasing the overall performance on various experiments. Due to the computational complexity of multi-agent reinforcement learning systems, Q-table-based algorithms such as Q-learning or SARSA are inefficient for complex tasks. Mnih et al. [93] proposed DQN, a novel approach replacing the Q-table with a neural network to address the challenges faced by classical reinforcement learning algorithms. DQNs have been proposed as a solution for various swarm problems, such as path planning for a swarm of UAVs, forest fire extinguishment, multi-target tracking and more. Various improvements have been proposed to increase the performance and scalability of DQNs such as joint spaces and value decomposition networks for wildfire monitoring [42]. Similarly, Li et al. [63] increased the performance of a DQN in a warehouse routing scenario by integrating improvements such as double DQN, dueling DQN, experience replay and prior knowledge. Yasuda and Ohkura [152,153] achieved good results in utilizing a physical swarm of two-wheeled robots to navigate between two landmarks which demonstrated the validity of DQN in swarm robotics. Overall, the reviewed papers using the DQN algorithm showed the importance of improvements in controlling a swarm of robots for various tasks. DDPG has been proposed as an extension of DQN by combining an actor model to provide an action given a state and a critic model to provide the reward given a state-action pair. Hüttenrauch et al. [160] achieved a good performance in controlling a swarm of virtual Kilobots utilizing a guided DDPG combined with a joint space. Yang et al. [161] proposed the MDDPG that involved a single critic model and an actor model for each agent. The authors were able to achieve similar performance to a DDPG on 12 tasks but with significantly fewer parameters. MADDPG has been proposed as an extension to a regular DDPG to enable a more direct communication between the agents in the swarm system. Huang et al. [71] demonstrated that integrating a sample optimizer with the MADDPG enabled an increase in reward by upwards of 8.87% and a reduction in time by upwards of 2.69% compared to the baseline MADDPG. Na et al. [167] proposed the FLDDPG to handle the challenges associated with the communication range and strength of unmanned vehicles, The authors compared their federated learning communication approach against an individual, shared network and shared experience versions of the DDPG algorithm. Using a navigation scenario simulated in Gazebo, their FLDDPG outperformed the other algorithms using the completion rate, completion time and trajectory efficiency. The FLDDPG was able to achieve a significant improvement in completion rate, achieving a rate of 96% compared to the SNDDPG at 34%, SEDDPG at 24% and the IDDPG at 17%. Hüttenrauch et al. [150,169] demonstrated the ability of utilizing the TRPO algorithm for a navigation scenario and the creation of a communication link. To reduce computational complexity, the authors compared various communication encoding protocols such as histograms, neural networks, RBFs and pooling methods. The authors found that encoding protocols based on neural networks achieved the overall best results while also demonstrating the importance of information sharing between the agents.

Overall, the reviewed papers showed the potential of controlling multi-robot systems using different reinforcement learning algorithms for various tasks. Simple algorithms such as Q-learning or SARSA achieved good results on simple tasks using smaller swarm, while more complex algorithms such as DQN or DDPG were able to solve more complex problems with larger swarms. Our review also demonstrated the importance of improvements such as information sharing using joint spaces or by integrating a central server to allow a more direct communication between the agents. Furthermore, our review of reinforcement learning algorithms for swarm robotics revealed a lack of test using real swarms of unmanned vehicle to demonstrate the validity of the algorithms. An overview of the reviewed algorithms in Section 4 is presented in Tables 8 and 9.

Since most of the discussed papers utilized simulators to conduct their experiments, we also presented an overview of the available simulators. Nine simulators were deemed interesting for our review based on their abilities to simulate multiple robots, realism, ease-of-use and accessibility. Simulators such as AirSim, IsaacSim and Gazebo offered more realistic environments while others such as OpenAIGym, Kilombo, Pybullet were more efficient. We compared the abilities of simulating different types of robots, sensors, image modalities and environment configuration such as fog, wind, rain and more. Certain simulators offered APIs in different programming languages which enabled reinforcement learning algorithms to be trained easily using Pytorch or Tensorflow. We also presented a remote swarm of small ground rovers and Starcraft II, a real-time military strategy video game, due to their possible use in the context of swarm robotics. Overall, we demonstrated that choosing the correct simulator depends on the application, type of unmanned vehicle desired, realism and the ability of using external libraries.

7. Future research directions

To build upon our findings, we present a list of potential research directions in the field of swarm robotics based on reinforcement learning.

- **Scalability** As mentioned in Section 6, the reviewed swarm robotic systems were limited to only a few agents, which limits the reliability and scalability of the proposed algorithms. Scalability is a crucial aspect in swarm robotics, as swarms can vary significantly in size, ranging from just a few vehicles to potentially thousands of unmanned vehicles. Therefore, it is important to develop or adapt current algorithms that can effectively handle a swarm with a large number of agents. By tackling the issue

Table 8
Overview of algorithms.

Algorithm	Policy	Approach description	Achieved tasks
SARSA referenced in Iima and Kuroe [115], Kakish et al. [116], Speck and Bucci [118], Luo et al. [123]	On-policy	Uses the current state-action pair and next state-action pair derived from the current policy to optimize itself.	• Achieved swarm grouping with up to 100 targets • Drone swarm grouping in urban and rural settings • Path tracing
Q-learning referenced in Islam and Razi [39], Testi et al. [58], Cui et al. [125,126], Karmanova et al. [127], Das et al. [128]	Off-policy	Uses a Q-table to determine next best state-action pair. Assumes the use of an optimal policy.	• Drone Swarm Control • Downlink Wireless Network • Tic-Tac-Toe SwarmPlay • Multi-agent path planning • Khepera-II robot simulation • UAV-based ground vehicle localization
Multi-agent Q-learning referenced in Wang and De Silva [132], Sadhu and Konar [133], Chen et al. [140]	Off-policy	Enables a more direct communication between agents by utilizing joint states and joint actions for Q-learning	• Object carrying tasks • Field coverage • UAV-based ground vehicle localization
DQN referenced in Haksar and Schwager [40], Viseras et al. [42], Julian and Kochenderfer [45], Li et al. [63], Yang et al. [64], Zhou et al. [75], Vinyals et al. [94], Wei et al. [151], Yasuda and Ohkura [152], Jiang et al. [157], Tan and Karaköse [159]	Off-policy	Replaces the Q-table with a neural network to get the next action given a state. Provides a probability for each action, the maximum is taken.	• Fire extinguishment & surveillance • Warehouse inventory management • • Multi-robot path planning • Multi-target tracking • First-person view navigation • Central server for experience replay • Multi-agent fingerprinting and sampling scheme • LiDAR-based control system
DDPG referenced in Hüttenrauch et al. [160]	Off-policy	Uses two neural networks, an actor model to get the next action and a critic to determine the value of the action.	• Kilobots swarm control • Experience replay

Table 9
Overview of algorithms 2.

Algorithm	Policy	Approach description	Achieved tasks
MDDPG referenced in Yang et al. [161]	Off-policy	Similar to DDPG, with the addition of multiple actor models connected via MLPCONV layers combined with a singular critic model.	• Gazebo simulator-based control • Combination of data using MLPCONV layers
MADDPG referenced in Huang et al. [71], Singh et al. [164], Ho et al. [166]	Off-policy	Combines multiple actor models similar to MDPPG with the addition of multiple critic models.	• Multi-agent swarm control for non-stationary targets • Simulated MARS rover mission to explore an unknown environment
IDDPG referenced in Na et al. [167]	Off-policy	Uses multiple individual DDPGs such that each actor in a swarm controls its own DDPG.	• Navigation and collision avoidance scenario
SNDDPG referenced in Na et al. [167]	Off-policy	Each agent in the swarm uses a shared DDPG and shared database to store histories.	• Navigation and collision avoidance scenario
SEDDPG referenced in Na et al. [167]	Off-policy	Each agent shares a central database to store histories but they each have their individual neural networks.	• Navigation and collision avoidance scenario
FLDDPG referenced in Na et al. [167]	Off-policy	Each agent has its individual histories and neural network but also periodically share a central neural network.	• Navigation and collision avoidance scenario
MSMANN referenced in Li et al. [168]	Off-policy	This approach trains a distributed policy network using supervised learning and an individual policy for each agent.	• Distributed Network policy • Swarm control to reach multiple targets using multiple agents
TRPO referenced in Hüttenrauch et al. [150,169]	On-policy	Policy-based reinforcement learning algorithm that iteratively updates the policy within a specific trust region to learn the desired behavior.	• Distributed Network policy • Swarm control to reach multiple targets using multiple agents

of scalability, it would increase the validity of reinforcement learning techniques in controlling swarms of unmanned vehicles. This involves exploring techniques to efficiently control large swarms with a focus on robustness, adaptability and efficiency. Additionally, providing a comparative analysis on the scalability of different algorithms on various applications would provide important insights. By addressing the scalability potential of various algorithms, it would enhance their credibility in the domain of swarm robotics.

- **Practical experiments** Currently, the majority of swarm robotic systems that utilize reinforcement learning algorithms are trained and validated using simulators. Simulators offer a safer and more cost-effective alternative to real robots when training and evaluating swarm systems. However, simulator-based swarm systems may reduce the credibility of the algorithms due to the ideal and controlled nature of the simulated environment. Conduction practical experiments using real robots is crucial to demonstrate the viability of the algorithms in real-world scenarios. By comparing the behavior of different algorithms on physical robots, a more accurate and realistic analysis could be provided. Utilizing physical robots enables to evaluate the algorithms when faced with uncertainties such as a sudden change in wind when using drones. By combining simulators to provide training and real robot for evaluation, a compromise can be found between safety, cost and realism.
- **Efficiency comparison** In the field of swarm robotics based on reinforcement learning, metrics such as the completion rate are often used to compare the algorithms. While these metrics provide a good overview of the performance, they lack the ability to compare them for real-world scenarios. The speed and efficiency of processing data to control a swarm is crucial when transferring the swarm from a simulator to physical robots. Comparing the speed and processing time of the different algorithms would be highly beneficial in the field of swarm robotics. Efficiency is particularly important in time-sensitive applications such as for forest fire fighting, surveillance or search and rescue mission. Therefore, comparing the time to provide an action to a vehicle would provide valuable information to choose the appropriate algorithm for a task.
- **Multi-type swarm** The majority of papers in the field of reinforcement learning-based swarm robotics have focused on utilizing a single type of robot in a swarm. However, a swarm can combine multiple types of vehicles such as UAVs and UGVs to provide a more efficient solution, such as for search and rescue mission. For search and rescue, air vehicles can be used to locate areas of interest while ground units can be utilized to search the area. By developing and adapting reinforcement learning algorithms for heterogeneous swarms, it can provide more efficient solutions to complex tasks. This direction has enormous potential in yielding interesting results in a wide range of applications.

8. Conclusion

In conclusion, swarm robotics consists of utilizing multiple unmanned vehicles, such as drones or rovers, to solve various tasks. Controlling large swarms of unmanned vehicle for complex applications is a challenging task and doing it manually is unrealistic and inefficient. Reinforcement learning, a subgenre of artificial intelligence, has seen promise in the field of controlling robots of different types and quantities. It consists of iteratively moving an agent in an environment to learn the desired behavior based on a trial and error approach. We introduced the concept of reinforcement learning-based swarm robotics by presenting an overview of the various applications based on this control system. Our review demonstrated the extent of swarm robotics to solve simple problems such as delivering a payload to more challenging tasks such as automating a warehouse. To understand how reinforcement learning can be beneficial to swarm robotics, we presented an overview of important reinforcement learning terminologies. Various reinforcement learning algorithms for swarm robotics were then reviewed to provide an overview of the current state-of-the-art. This overview demonstrated the importance of reinforcement learning algorithms such as DQN and DDPG over basic algorithms such as Q-learning and SARSA. DQNs and DDPGs enabled the control of multiple types of swarm for various complex applications. Additionally, a recurrent observation is the benefit of various improvements over the baseline algorithms. Information sharing showed the most potential as an improvement by enabling a more direct communication between the agents. In fact, multi-agent Q-learning outperformed classical Q-learning consistently in newer research due to the integration of joint spaces. FLDDPG also showed the benefit of information sharing by significantly improving the performance of the DDPG algorithm. Our review demonstrated the importance of testing different algorithms with different configurations and improvements. Since most algorithms were trained and evaluated on simulators, we also reviewed the available simulators and compared them using their realism, type and quantity of robots available and environment configurations. Overall, we demonstrated the potential of reinforcement learning-based swarm robotics systems for a plethora of applications in various fields and domains. However, we found a lack reliability in terms of applying and evaluating the control systems on real swarm systems for complex applications. More work in this field is also required for larger swarm systems, since most reviewed research papers are currently limited to a few robots while real system may require thousands of agents.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was enabled in part by support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2018-06233.

References

[1] A. McCain, Market size, growth, and biggest companies, 2022, Available at https://www.zippia.com/advice/robotics-industry-statistics/, (accessed) Sept. 1st, 2022.
[2] M. Placek, Industrial robots worldwide, Statista, 2022, Available at https://www.statista.com/topics/1476/industrial-robots/, (accessed) Sept. 1st, 2022.
[3] G.V. Research, Commercial Drone Market Size & Share Report, 2021–2028, Technical Report, Grand View Research Inc, 2022. https://www.grandviewresearch.com/industry-analysis/global-commercial-drones-market.
[4] The National Institute for Occupational Safety and Health (NIOSH), Robotics and workplace safety, 2021, Available at https://www.cdc.gov/niosh/newsroom/feature/robotics-workplace-safety.html, (accessed) Sept. 1st, 2022.
[5] B. Powrozek, Efficiency and lower costs: Is automation the solution?, 2021, Available at https://claytonmckervey.com/efficiency-lower-costs-industrial-automation-solution/, (accessed) Sept. 1st, 2022.
[6] A. Kingatua, Robots and automation in electronics manufacturing, 2020, Available at https://medium.com/supplyframe-hardware/robots-and-automation-in-electronics-manufacturing-a77f177585eb, (accessed) Sept. 1st, 2022.
[7] D. Nicholson, Equipping Drones for At-Sea Search and Rescue, 2012, Available at https://www.practical-sailor.com/safety-seamanship/equipping-drones-for-at-sea-search-and-rescue, (accessed) Sept. 1st, 2022.
[8] R.D. Arnold, H. Yamaguchi, T. Tanaka, Search and rescue with autonomous flying robots through behavior-based cooperative intelligence, J. Int. Humanit. Action 3 (1) (2018) 1–18.
[9] Waredock, What is amazon robotic fulfillment center?, 2019, Available at https://www.waredock.com/magazine/what-is-amazon-robotic-fulfillment-center, (accessed) Sept. 1st, 2022.
[10] Rheinmetall, Rheinmetall mission master family, 2018, Available at https://www.rheinmetall.ca, (accessed) Sept. 1st, 2022.
[11] Q. Gu, T. Fan, F. Pan, C. Zhang, A vehicle-UAV operation scheme for instant delivery, Comput. Ind. Eng. 149 (2020) 106809.
[12] Oxford Economics, How robots change the world, Econ. Outlook 43 (3) (2019) 5–8, doi:10.1111/1468-0319.12431.
[13] R. Abduljabbar, H. Dia, S. Liyanage, S.A. Bagloee, Applications of artificial intelligence in transport: an overview, Sustainability 11 (1) (2019) 189.
[14] N.C. Eli-Chukwu, Applications of artificial intelligence in agriculture: a review, Eng., Technol. Appl. Sci. Res. 9 (4) (2019) 4377–4383.
[15] S. Dilek, H. Cakir, M. Aydin, et al., Applications of artificial intelligence techniques to combating cyber crimes: a review, Int. J. Artif. Intell. Appl. 6 (1) (2015) 21–39, doi:10.5121/ijaia.2015.6102.
[16] R. Li, M. Jin, V.C. Paquit, Geometrical defect detection for additive manufacturing with machine learning models, Mater. Des. 206 (2021) 109726.
[17] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012).
[18] A. Işin, C. Direkoğlu, M. Şah, Review of MRI-based brain tumor image segmentation using deep learning methods, Procedia Comput. Sci. 102 (2016) 317–324.
[19] K. Shao, Z. Tang, Y. Zhu, N. Li, D. Zhao, A survey of deep reinforcement learning in video games, arXiv preprint arXiv:1912.10944 (2019).
[20] R. Meyes, H. Tercan, S. Roggendorf, T. Thiele, C. Büscher, M. Obdenbusch, C. Brecher, S. Jeschke, T. Meisen, Motion planning for industrial robots using reinforcement learning, Procedia CIRP 63 (2017) 107–112.
[21] M.A.-M. Khan, M.R.J. Khan, A. Tooshil, N. Sikder, M.A.P. Mahmud, A.Z. Kouzani, A.-A. Nahid, A systematic review on reinforcement learning-based robotics within the last decade, IEEE Access 8 (2020) 176598–176623.
[22] Y. Li, Reinforcement learning applications, 2019, Available at https://medium.com/@yuxili/rl-applications-73ef685c07eb, (accessed) Sept. 1st, 2022.
[23] L. Buşoniu, R. Babuška, B.D. Schutter, Multi-agent reinforcement learning: an overview, in: Innovations in Multi-Agent Systems and Application 1, 2010, pp. 183–221.
[24] T.T. Nguyen, N.D. Nguyen, S. Nahavandi, Deep reinforcement learning for multiagent systems: areview of challenges, solutions, and applications, IEEE Trans. Cybern. 50 (9) (2020) 3826–3839.
[25] K. Zhang, Z. Yang, T. Başar, Multi-agent reinforcement learning: a selective overview of theories and algorithms, in: Handbook of Reinforcement Learning and Control, 2021, pp. 321–384.
[26] Y. Mohan, S.G. Ponnambalam, An extensive review of research in swarm robotics, in: 2009 world Congress on Nature & Biologically Inspired Computing (Nabic), IEEE, 2009, pp. 140–145.
[27] L. Bayindir, E. Şahin, A review of studies in swarm robotics, Turkish J. Electr. Eng. Comput. Sci. 15 (2) (2007) 115–147.
[28] Z. Shi, J. Tu, Q. Zhang, L. Liu, J. Wei, A survey of swarm robotics system, in: International Conference in Swarm Intelligence, Springer, 2012, pp. 564–572.
[29] M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, Swarm robotics: a review from the swarm engineering perspective, Swarm Intell. 7 (1) (2013) 1–41.
[30] L. Bayindir, A review of swarm robotics tasks, Neurocomputing 172 (2016) 292–321.
[31] N. Nedjah, L.S. Junior, Review of methodologies and tasks in swarm robotics towards standardization, Swarm Evol. Comput. 50 (2019) 100565.
[32] N. Nedjah, L.S. Junior, Review of methodologies and tasks in swarm robotics towards standardization, Swarm Evol. Comput. 50 (2019).
[33] M.H.A. Majid, M.R. Arshad, R.M. Mokhtar, Swarm robotics behaviors and tasks: a technical review, Stud, Syst., Decis. Control (2021) 99–167.
[34] M. Dorigo, G. Theraulaz, V. Trianni, Reflections on the future of swarm robotics, Sci. Robot. 5 (49) (2020) eabe4385.
[35] S.-J. Chung, A.A. Paranjape, P.M. Dames, S. Shen, V.R. Kumar, A survey on aerial swarm robotics, IEEE Trans. Robot. 34 (2018) 837–855.
[36] Center for Climate and Energy Solutions (C2ES), Wildfires and climate change, 2020, Available at https://www.c2es.org/content/wildfires-and-climate-change/, (accessed) Sept. 1st, 2022.
[37] J.J. Roldán-Gómez, E. González-Gironda, A. Barrientos, A survey on robotic technologies for forest firefighting: applying drone swarms to improve firefighters' efficiency and safety, Appl. Sci. 11 (1) (2021) 363.
[38] M.A. Akhloufi, A. Couturier, N.A. Castro, Unmanned aerial vehicles for wildland fires: sensing, perception, cooperation and assistance, Drones 5 (1) (2021) 15.
[39] S. Islam, A. Razi, A path planning algorithm for collective monitoring using autonomous drones, in: 2019 53rd Annual Conference on Information Sciences and Systems (CISS), IEEE, 2019, pp. 1–6.
[40] 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), IEEE, 2018, pp. 1067–1074.
[41] T. Hammond, D.J. Schaap, M. Sabatelli, M.A. Wiering, Forest fire control with learning from demonstration and reinforcement learning, in: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–8.
[42] A. Viseras, M. Meissner, J. Marchal, Wildfire front monitoring with multiple UAVs using deep Q-learning, IEEE Access (2021).
[43] F.H. Panahi, F.H. Panahi, T. Ohtsuki, A reinforcement learning-based fire warning and suppression system using unmanned aerial vehicles, IEEE Trans. Instrum. Meas. 72 (2022) 1–16.
[44] K.A. Ghamry, M.A. Kamel, Y. Zhang, Cooperative forest monitoring and fire detection using a team of UAVs-UGVs, in: 2016 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2016, pp. 1206–1211.
[45] K.D. Julian, M.J. Kochenderfer, Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning, J. Guid., Control, Dyn. 42 (8) (2019) 1768–1778.
[46] H.X. Pham, H.M. La, D. Feil-Seifer, M.C. Deans, A distributed control framework of multiple unmanned aerial vehicles for dynamic wildfire tracking, IEEE Trans. Syst., Man, Cybern. 50 (4) (2018) 1537–1548.
[47] S.G. Subramanian, M. Crowley, Learning forest wildfire dynamics from satellite images using reinforcement learning, in: Conference on Reinforcement Learning and Decision Making, Ann Arbor MI, 2017.
[48] S. Ganapathi Subramanian, M. Crowley, Using spatial reinforcement learning to build forest wildfire dynamics models from satellite images, Front. ICT 5 (2018) 6.
[49] L.C. Santos, F.N. Santos, E.J.S. Pires, A. Valente, P. Costa, S. Magalhães, Path planning for ground robots in agriculture: ashort review, in: 2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), IEEE, 2020, pp. 61–66.

[50] M. Esposito, M. Crimaldi, V. Cirillo, F. Sarghini, A. Maggio, Drone and sensor technology for sustainable weed management: a review, Chem. Biol. Technol. Agric. 8 (1) (2021) 1–11.
[51] M.H.M. Roslim, A.S. Juraimi, N.N. Che'Ya, N. Sulaiman, M.N.H.A. Manaf, Z. Ramli, M. Motmainna, Using remote sensing and an unmanned aerial system for weed management in agricultural crops: a review, Agronomy 11 (9) (2021) 1809.
[52] D. Albani, J. IJsselmuiden, R. Haken, V. Trianni, Monitoring and mapping with robot swarms for agricultural applications, in: 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), IEEE, 2017, pp. 1–6.
[53] Z. Zhang, J. Boubin, C. Stewart, S. Khanal, Whole-field reinforcement learning: a fully autonomous aerial scouting method for precision agriculture, Sensors 20 (22) (2020) 6585.
[54] N. Marwah, V.K. Singh, G.S. Kashyap, S. Wazir, An analysis of the robustness of UAV agriculture field coverage using multi-agent reinforcement learning, Int. J. Inf. Technol. (2023) 1–11.
[55] Z. Hao, X. Li, C. Meng, W. Yang, M. Li, Adaptive spraying decision system for plant protection unmanned aerial vehicle based on reinforcement learning, Int. J. Agric. Biol. Eng. 15 (4) (2022) 16–26.
[56] Z. Qin, Y. Chen, C. Fan, Density constrained reinforcement learning, in: International Conference on Machine Learning, PMLR, 2021, pp. 8682–8692.
[57] A. Amarasinghe, V.B. Wijesuriya, D. Ganepola, L. Jayaratne, A swarm of crop spraying drones solution for optimising safe pesticide usage in arable lands, in: Proceedings of the 17th Conference on Embedded Networked Sensor Systems, 2019, pp. 410–411.
[58] E. Testi, E. Favarelli, A. Giorgetti, Reinforcement learning for connected autonomous vehicle localization via UAVs, in: 2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), IEEE, 2020, pp. 13–17.
[59] L.F.P. Oliveira, A.P. Moreira, M.F. Silva, Advances in forest robotics: a state-of-the-art survey, Robotics 10 (2) (2021) 53.
[60] P.D. Siedler, Dynamic collaborative multi-agent reinforcement learning communication for autonomous drone reforestation, arXiv preprint arXiv:2211.15414 (2022).
[61] J. Wen, L. He, F. Zhu, Swarm robotics control and communications: imminent challenges for next generation smart logistics, IEEE Commun. Mag. 56 (7) (2018) 102–107.
[62] H. Lee, J. Jeong, Mobile robot path optimization technique based on reinforcement learning algorithm in warehouse environment, Appl. Sci. 11 (3) (2021) 1209.
[63] M.P. Li, P. Sankaran, M.E. Kuhl, R. Ptucha, A. Ganguly, A. Kwasinski, Task selection by autonomous mobile robots in a warehouse using deep reinforcement learning, in: 2019 Winter Simulation Conference (WSC), IEEE, 2019, pp. 680–689.
[64] Y. Yang, L. Juntao, P. Lingling, Multi-robot path planning based on a deep reinforcement learning DQN algorithm, CAAI Trans. Intell. Technol. 5 (3) (2020) 177–183.
[65] H. Bae, G. Kim, J. Kim, D. Qian, S. Lee, Multi-robot path planning method using reinforcement learning, Appl. Sci. 9 (15) (2019) 3057.
[66] X. Chen, H. Wang, Z. Li, W. Ding, F. Dang, C. Wu, X. Chen, Deliversense: Efficient delivery drone scheduling for crowdsensing with deep reinforcement learning, in: Adjunct Proceedings of the 2022 ACM International Joint Conference on Pervasive and Ubiquitous Computing and the 2022 ACM International Symposium on Wearable Computers, 2022, pp. 403–408.
[67] A. Faust, I. Palunko, P. Cruz, R. Fierro, L. Tapia, Automated aerial suspended cargo delivery through reinforcement learning, Artif. Intell. 247 (2017) 381–398.
[68] G. Muñoz, C. Barrado, E. Çetin, E. Salami, Deep reinforcement learning for drone delivery, Drones 3 (3) (2019) 72.
[69] G. Wu, M. Fan, J. Shi, Y. Feng, Reinforcement learning based truck-and-drone coordinated delivery, IEEE Trans. Artif. Intell. (2021).
[70] P. Palanisamy, Multi-agent connected autonomous driving using deep reinforcement learning, in: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–7.
[71] Y. Huang, S. Wu, Z. Mu, X. Long, S. Chu, G. Zhao, A multi-agent reinforcement learning method for swarm robots in space collaborative exploration, in: 2020 6th International Conference on Control, Automation and Robotics (ICCAR), IEEE, 2020, pp. 139–144.
[72] P. Sapaty, Military robotics: latest trends and spatial grasp solutions, Int. J. Adv. Res. Artif. Intell. 4 (4) (2015) 9–18.
[73] J. Zheng, S. Mao, Z. Wu, P. Kong, H. Qiang, Improved path planning for indoor patrol robot based on deep reinforcement learning, Symmetry 14 (1) (2022) 132.
[74] H.X. Pham, H.M. La, D. Feil-Seifer, A. Nefian, Cooperative and distributed reinforcement learning of drones for field coverage, arXiv preprint arXiv:1803.07250 (2018).
[75] W. Zhou, Z. Liu, J. Li, X. Xu, L. Shen, Multi-target tracking for unmanned aerial vehicle swarms using deep reinforcement learning, Neurocomputing 466 (2021) 285–297.
[76] A. Bonnet, M.A. Akhloufi, Uav pursuit using reinforcement learning, in: Unmanned Systems Technology XXI, vol. 11021, SPIE, 2019, pp. 51–58.
[77] M.A. Akhloufi, S. Arola, A. Bonnet, Drones chasing drones: reinforcement learning and deep search area proposal, Drones 3 (3) (2019) 58.
[78] M. Zaier, W. Miled, M.A. Akhloufi, Vision based UAV tracking using deep reinforcement learning with simulated data, in: Autonomous Systems: Sensors, Processing and Security for Ground, Air, Sea and Space Vehicles and Infrastructure 2022, vol. 12115, SPIE, 2022, pp. 92–108.
[79] C.D. Hsu, H. Jeong, G.J. Pappas, P. Chaudhari, Scalable reinforcement learning policies for multi-agent control, in: 2021 IEEE/RSJ international conference on intelligent robots and systems (IROS), IEEE, 2021, pp. 4785–4791.
[80] Q. Yang, J. Zhang, G. Shi, J. Hu, Y. Wu, Maneuver decision of UAV in short-range air combat based on deep reinforcement learning, IEEE Access 8 (2019) 363–378.
[81] Y.-f. Li, J.-p. Shi, W. Jiang, W.-g. Zhang, Y.-x. Lyu, Autonomous maneuver decision-making for a UCAV in short-range aerial combat based on an MS-DDQN algorithm, Def. Technol. 18 (9) (2022) 1697–1714.
[82] J. Hu, L. Wang, T. Hu, C. Guo, Y. Wang, Autonomous maneuver decision making of dual-UAV cooperative air combat based on deep reinforcement learning, Electronics 11 (3) (2022) 467.
[83] M.S. Couceiro, An overview of swarm robotics for search and rescue applications, Artif. Intell. (2017) 1522–1561.
[84] F. Niroui, K. Zhang, Z. Kashino, G. Nejat, Deep reinforcement learning robot for search and rescue applications: exploration in unknown cluttered environments, IEEE Robot. Autom. Lett. 4 (2) (2019) 610–617.
[85] J.G.C. Zuluaga, J.P. Leidig, C. Trefftz, G. Wolffe, Deep reinforcement learning for autonomous search and rescue, in: NAECON 2018-IEEE National Aerospace and Electronics Conference, IEEE, 2018, pp. 521–524.
[86] S. Kulkarni, V. Chaphekar, M.M.U. Chowdhury, F. Erden, I. Guvenc, UAV aided search and rescue operation using reinforcement learning, in: 2020 SoutheastCon, vol. 2, IEEE, 2020, pp. 1–8.
[87] A. Peake, J. McCalmon, Y. Zhang, B. Raiford, S. Alqahtani, Wilderness search and rescue missions using deep reinforcement learning, in: 2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), IEEE, 2020, pp. 102–107.
[88] D.S. Drew, Multi-agent systems for search and rescue applications, Curr. Robot. Rep. 2 (2021) 189–200.
[89] J.P.n. Queralta, J. Taipalmaa, B.C. Pullinen, V.K. Sarker, T.A.N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, T. Westerlund, Collaborative multi-robot systems for search and rescue: coordination and perception, arXiv:2008.12610 (2020).
[90] R.D. Arnold, M. Osinski, C. Reddy, A. Lowey, Reinforcement learning for collaborative search and rescue using unmanned aircraft system swarms, in: 2022 IEEE International Symposium on Technologies for Homeland Security (HST), 2022, pp. 1–6.
[91] A.A. Rahman, A. Bhattacharya, T. Ramachandran, S. Mukherjee, H. Sharma, T. Fujimoto, S. Chatterjee, AdverSAR: adversarial search and rescue via multi-agent reinforcement learning, in: 2022 IEEE International Symposium on Technologies for Homeland Security (HST), 2022, pp. 1–7.
[92] Y. Wang, W. Liu, J. Liu, C. Sun, Cooperative USV–UAV marine search and rescue with visual navigation and reinforcement learning-based control, ISA Trans. (2023).
[93] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013).

[94] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D.H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., Grandmaster level in starcraft II using multi-agent reinforcement learning, Nature 575 (7782) (2019) 350–354.
[95] H. Nguyen, H. La, Review of deep reinforcement learning for robot manipulation, in: 2019 Third IEEE International Conference on Robotic Computing (IRC), IEEE, 2019, pp. 590–595.
[96] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: a survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.
[97] A.T. Azar, A. Koubaa, N. Ali Mohamed, H.A. Ibrahim, Z.F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A.M. Khamis, I.A. Hameed, et al., Drone deep reinforcement learning: a review, Electronics 10 (9) (2021) 999.
[98] S.M. Kakade, A natural policy gradient, Adv. Neural Inf. Process. Syst. 14 (2001).
[99] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine learning, PMLR, 2016, pp. 1928–1937.
[100] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).
[101] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine learning, PMLR, 2015, pp. 1889–1897.
[102] C. Watkins, Learning from Delayed Rewards, King's College, 1989 Ph.D. thesis.
[103] C.J. Watkins, P. Dayan, Q-lLearning, Mach. Learn. 8 (3) (1992) 279–292.
[104] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[105] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, 2016.
[106] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971 (2015).
[107] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International Conference on Machine learning, PMLR, 2018, pp. 1587–1596.
[108] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International Conference on Machine Learning, PMLR, 2018, pp. 1861–1870.
[109] D. Ha, J. Schmidhuber, World models, arXiv preprint arXiv:1803.10122 (2018).
[110] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, et al., Imagination-augmented agents for deep reinforcement learning, Adv. Neural Inf. Process. Syst. 30 (2017).
[111] S. Bansal, R. Calandra, K. Chua, S. Levine, C. Tomlin, Mbmf: model-based priors for model-free reinforcement learning, arXiv preprint arXiv:1709.03153 (2017).
[112] V. Feinberg, A. Wan, I. Stoica, M.I. Jordan, J.E. Gonzalez, S. Levine, Model-based value estimation for efficient model-free reinforcement learning, arXiv preprint arXiv:1803.00101 (2018).
[113] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., Mastering chess and Shogi by self-play with a general reinforcement learning algorithm, arXiv preprint arXiv:1712.01815 (2017).
[114] R.S. Sutton, Learning to predict by the methods of temporal differences, Mach. Learn. 3 (1) (1988) 9–44.
[115] H. Iima, Y. Kuroe, Swarm reinforcement learning algorithms based on Sarsa method, in: 2008 SICE Annual Conference, IEEE, 2008, pp. 2045–2049.
[116] Z. Kakish, K. Elamvazhuthi, S. Berman, Using reinforcement learning to herd a robotic swarm to a target distribution, in: International Symposium Distributed Autonomous Robotic Systems, Springer, 2021, pp. 401–414.
[117] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, M. Egerstedt, The robotarium: globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems, IEEE Control Syst. Mag. 40 (1) (2020) 26–44.
[118] C. Speck, D.J. Bucci, Distributed UAV swarm formation control via object-focused, multi-objective SARSA, in: 2018 Annual American Control Conference (ACC), IEEE, 2018, pp. 6596–6601.
[119] N. Sprague, D. Ballard, Multiple-goal reinforcement learning with modular Sarsa (0), in: International Joint Conferences on Artificial Intelligence, 2003.
[120] C.W. Reynolds, Flocks, herds and schools: A distributed behavioral model, in: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, 1987, pp. 25–34.
[121] L.C. Cobo, C.L. Isbell, A.L. Thomaz, Object focused Q-learning for autonomous agents, in: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems, in: AAMAS '13, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2013, pp. 1061–1068.
[122] I.C. Price, G.B. Lamont, Ga directed self-organized search and attack UAV swarms, in: Proceedings of the 2006 Winter Simulation Conference, IEEE, 2006, pp. 1307–1315.
[123] W. Luo, Q. Tang, C. Fu, P. Eberhard, Deep-Sarsa based multi-UAV path planning and obstacle avoidance in a dynamic environment, in: Advances in Swarm Intelligence: 9th International Conference, ICSI 2018, Shanghai, China, June 17–22, 2018, Proceedings, Part II 9, Springer, 2018, pp. 102–111.
[124] R. Bellman, Dynamic programming, Science 153 (3731) (1966) 34–37.
[125] J. Cui, Y. Liu, A. Nallanathan, Multi-agent reinforcement learning-based resource allocation for UAV networks, IEEE Trans. Wirel. Commun. 19 (2) (2019) 729–743.
[126] J. Cui, Y. Liu, A. Nallanathan, The application of multi-agent reinforcement learning in UAV networks, in: 2019 IEEE International Conference on Communications Workshops (ICC Workshops), IEEE, 2019, pp. 1–6.
[127] E. Karmanova, V. Serpiva, S. Perminov, A. Fedoseev, D. Tsetserukou, Swarmplay: Interactive tic-tac-toe board game with swarm of nano-UAVs driven by reinforcement learning, in: 2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN), IEEE, 2021, pp. 1269–1274.
[128] P.K. Das, H.S. Behera, B.K. Panigrahi, Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity, Eng. Sci. Technol., Int. J. 19 (1) (2016) 651–669.
[129] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-International Conference on Neural Networks, vol. 4, IEEE, 1995, pp. 1942–1948.
[130] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[131] B. Banerjee, S. Sandip, J. Peng, Fast concurrent reinforcement learners, in: International Joint Conference on Artificial Intelligence, vol. 17, Lawrence Erlbaum Associates LTD, 2001, pp. 825–832.
[132] Y. Wang, C.W. De Silva, Multi-robot box-pushing: Single-agent Q-learning vs. team Q-learning, in: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2006, pp. 3694–3699.
[133] A.K. Sadhu, A. Konar, Improving the speed of convergence of multi-agent Q-learning for cooperative task-planning by a robot-team, Robot. Auton. Syst. 92 (2017) 66–80.
[134] A.K. Sadhu, P. Rakshit, A. Konar, A modified imperialist competitive algorithm for multi-robot stick-carrying application, Robot. Auton. Syst. 76 (2016) 15–35.
[135] M.L. Littman, Value-function reinforcement learning in Markov games, Cogn. Syst. Res. 2 (1) (2001) 55–66.
[136] Z. Zhang, D. Zhao, J. Gao, D. Wang, Y. Dai, Fmrq–a multiagent reinforcement learning algorithm for fully cooperative tasks, IEEE Trans. Cybern. 47 (6) (2016) 1367–1379.
[137] AAI Canada, Inc, Intelligent robots Khepera II, 2003, Available at http://www.aai.ca/robots/khep_2.html, (accessed) Sept. 1st, 2022.
[138] A. Greenwald, M. Zinkevich, P. Kaelbling, Correlated Q-learning (2003).
[139] L. Busoniu, R. Babuska, B. De Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Trans. Syst., Man, Cybern., Part C (Appl. Rev.) 38 (2) (2008) 156–172.
[140] Y.-J. Chen, D.-K. Chang, C. Zhang, Autonomous tracking using a swarm of UAVs: aconstrained multi-agent reinforcement learning approach, IEEE Trans. Veh. Technol. 69 (11) (2020) 13702–13717.
[141] M. Bellemare, J. Veness, M. Bowling, Investigating contingency awareness using Atari 2600 games, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 26, 2021, pp. 864–871, doi:10.1609/aaai.v26i1.8321.

[142] P. Sunehag, G. Lever, A. Gruslys, W.M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J.Z. Leibo, K. Tuyls, et al., Value-decomposition networks for cooperative multi-agent learning, arXiv preprint arXiv:1706.05296 (2017).
[143] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: combining improvements in deep reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.
[144] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling network architectures for deep reinforcement learning, in: International Conference on Machine Learning, PMLR, 2016, pp. 1995–2003.
[145] S.C. Jaquette, Markov decision processes with a new optimality criterion: discrete time, Ann. Stat. 1 (3) (1973) 496–505.
[146] M.G. Bellemare, W. Dabney, R. Munos, A distributional perspective on reinforcement learning, in: International Conference on Machine learning, PMLR, 2017, pp. 449–458.
[147] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv preprint arXiv:1511.05952 (2015).
[148] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[149] Y. Huang, G. Wei, Y. Wang, VD D3QN: the variant of double deep Q-learning network with dueling architecture, in: 2018 37th Chinese Control Conference (CCC), IEEE, 2018, pp. 9130–9135.
[150] M. Hüttenrauch, S. Adrian, G. Neumann, et al., Deep reinforcement learning for swarm systems, J. Mach. Learn. Res. 20 (54) (2019) 1–31.
[151] Y. Wei, X. Nie, M. Hiraga, K. Ohkura, Z. Car, Developing end-to-end control policies for robotic swarms using deep Q-learning, J. Adv. Comput. Intell. Intell. Inform. 23 (5) (2019) 920–927.
[152] T. Yasuda, K. Ohkura, Collective behavior acquisition of real robotic swarms using deep reinforcement learning, in: 2018 second IEEE International Conference on Robotic Computing (IRC), IEEE, 2018, pp. 179–180.
[153] T. Yasuda, K. Ohkura, Sharing experience for behavior generation of real swarm robot systems using deep reinforcement learning, J. Robot. Mechatron. 31 (4) (2019) 520–525.
[154] L.-J. Lin, Reinforcement Learning for Robots Using Neural Networks, Carnegie Mellon University, 1992.
[155] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P.H.S. 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.
[156] K. Ciosek, S. Whiteson, Offer: off-environment reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, 2017, doi:10.1609/aaai.v31i1.10810.
[157] C. Jiang, Z. Chen, Y. Guo, Learning decentralized control policies for multi-robot formation, in: 2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), IEEE, 2019, pp. 758–765.
[158] E. Rohmer, S.P.N. Singh, M. Freese, V-REP: a versatile and scalable robot simulation framework, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 1321–1326, doi:10.1109/IROS.2013.6696520.
[159] Z. Tan, M. Karaköse, Proximal policy based deep reinforcement learning approach for swarm robots, in: 2021 Zooming Innovation in Consumer Technologies Conference (ZINC), IEEE, 2021, pp. 166–170.
[160] M. Hüttenrauch, A. Šošić, G. Neumann, Guided deep reinforcement learning for swarm systems, arXiv preprint arXiv:1709.06011 (2017).
[161] Z. Yang, K.E. Merrick, H.A. Abbass, L. Jin, Multi-task deep reinforcement learning for continuous action control, in: IJCAI, vol. 17, 2017, pp. 3301–3307.
[162] M. Lin, Q. Chen, S. Yan, et al., Network In Network, 2014, arXiv:1312.4400
[163] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, et al., Continuous control with deep reinforcement learning, (2019). arXiv:1509.02971.
[164] G. Singh, D.M. Lofaro, D. Sofge, Pursuit-evasion with decentralized robotic swarm in continuous state space and action space via deep reinforcement learning, in: ICAART (1), 2020, pp. 226–233.
[165] 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).
[166] T.M. Ho, T.T. Nguyen, K.-K. Nguyen, M. Cheriet, Deep reinforcement learning for URLLC in 5g mission-critical cloud robotic application, in: 2021 IEEE Global Communications Conference (GLOBECOM), IEEE, 2021, pp. 1–6.
[167] S. Na, T. Krajník, B. Lennox, F. Arvin, Federated reinforcement learning for collective navigation of robotic swarms, arXiv preprint arXiv:2202.01141 (2022).
[168] Q. Li, X. Du, Y. Huang, Q. Sykora, A.P. Schoellig, Learning of coordination policies for robotic swarms, arXiv preprint arXiv:1709.06620 (2017).
[169] M. Hüttenrauch, A. Šošić, G. Neumann, Local communication protocols for learning complex swarm behaviors with deep reinforcement learning, in: International Conference on Swarm Intelligence, Springer, 2018, pp. 71–83.
[170] J.K. Gupta, M. Egorov, M. Kochenderfer, Cooperative multi-agent control using deep reinforcement learning, in: International Conference on Autonomous Agents and Multiagent Systems, Springer, 2017, pp. 66–83.
[171] A. Smola, A. Gretton, L. Song, B. Schölkopf, A hilbert space embedding for distributions, in: International Conference on Algorithmic Learning Theory, Springer, 2007, pp. 13–31.
[172] W. Slagter, Why simulation is a driving force for autonomous vehicles, 2020, Available at https://www.ansys.com/blog/simulation-drives-autonomous-vehicles, (accessed) Sept. 1st, 2022.
[173] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning, in: Osdi, Volume 16, Savannah, GA, USA, 2016, pp. 265–283.
[174] S. Shah, D. Dey, C. Lovett, A. Kapoor, Airsim: high-fidelity visual and physical simulation for autonomous vehicles, in: M. Hutter, R. Siegwart (Eds.), Field and Service Robotics, Springer International Publishing, Cham, 2018, pp. 621–635.
[175] Epic Games, Unreal engine 4, 2014, Available at https://www.unrealengine.com/en-US/, (accessed) Sept. 1st, 2022.
[176] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Ng, ROS: an open-source robot operating system, in: ICRA Workshop on Open Source Software, vol. 3, 2009, p. 5.
[177] N. Koenig, A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, in: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), vol. 3, IEEE, 2004, pp. 2149–2154.
[178] M. Müller, V. Casser, J. Lahoud, N. Smith, B. Ghanem, SIM4CV: a photo-realistic simulator for computer vision applications, Int. J. Comput. Vis. 126 (9) (2018) 902–919.
[179] B. Gerkey, R.T. Vaughan, A. Howard, et al., The player/stage project: Tools for multi-robot and distributed sensor systems, in: Proceedings of the 11th International Conference on Advanced Robotics, vol. 1, Citeseer, 2003, pp. 317–323.
[180] RoboNation, Vrx competition 2022, 2022, Available at https://robotx.org/programs/vrx-competition-2022/, (accessed) Sept. 1st, 2022.
[181] DARPA, Subterranean challenge, 2021, Available at https://www.subtchallenge.com/, (accessed) Sept. 1st, 2022.
[182] F. Jansson, M. Hartley, M. Hinsch, I. Slavkov, N. Carranza, T.S.G. Olsson, R.M. Dries, J.H. Grönqvist, A.F.M. Marée, J. Sharpe, et al., Kilombo: a kilobot simulator to enable effective research in swarm robotics, arXiv preprint arXiv:1511.04285 (2015).
[183] S. Nissen, et al., Implementation of a Fast Artificial Neural Network Library (FANN), Report, Department of Computer Science University of Copenhagen (DIKU), 2003.
[184] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, et al., Argos: a modular, parallel, multi-engine simulator for multi-robot systems, Swarm Intell. 6 (4) (2012) 271–295.
[185] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540 (2016).
[186] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, A.P. Schoellig, Learning to fly–a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control, 2021, Issue: 1 Pages: 1–8 Publication Title: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Volume: 1 original-date: 2020-08-10T07:38:09Z, https:www//github.com/utiasDSL/gym-pybullet-drones.
[187] I. Gilitschenski, Multi-Car Racing Gym Environment, 2022, Available at https://github.com/igilitschenski/multi_car_racing, (accessed) Sept. 1st, 2022.

[188] M. Lucchi, F. Zindler, S. Mühlbacher-Karrer, H. Pichler, Robo-gym–an open source toolkit for distributed deep reinforcement learning on real and simulated robots, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 5364–5371.
[189] NVIDIA, Isaac sim, 2022a, (Available at https://developer.nvidia.com/isaac-sim, (accessed) Sept. 1st, 2022.
[190] NVIDIA, Omniverse platform for virtual collaboration, 2022b, Available at https://www.nvidia.com/en-us/omniverse/, (accessed) Sept. 1st, 2022.
[191] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot operating system 2: design, architecture, and uses in the wild, Sci. Robot. 7 (66) (2022) eabm6074, doi:10.1126/scirobotics.abm6074.
[192] Admin, Bullet real-time physics simulation, 2022, https://pybullet.org/wordpress/.
[193] Bulletphysics, Bulletphysics/bullet3: Bullet physics SDK: Real-time collision detection and multi-physics simulation for VR, games, visual effects, robotics, machine learning etc., https://github.com/bulletphysics/bullet3.
[194] Vicon Industries, Inc, Vicon cameras, 2022a, Available at https://www.vicon.com/hardware/cameras/, (accessed) Sept. 1st, 2022.
[195] Vicon Industries, Inc, Vicon pearl markers, 2022b, Available at https://www.vicon.com/hardware/accessories/, (accessed) Sept. 1st, 2022.
[196] Blizzard Entertainment, Starcraft II, 2010, Available at https://starcraft2.com/en-us/, (accessed) Sept. 1st, 2022.

Marc-Andrè Blais received a B.Sc.A. degree in computer science (cooperative program) in 2021 from Universitè de Moncton, New-Brunswick, Canada. He is currently completing a M.Sc. in computer science also from Universitè de Moncton. His research interests include artificial intelligence, deep learning, intelligent systems, reinforcement learning, swarm robotics, robotics and their applications.

Moulay Akhloufireceived his M.Sc. and a Ph.D. in Electrical Engineering from Ecole Polytechnique of Montreal and Laval University (Canada), respectively. He also holds an MBA from Laval University. He is currently Professor in the University of Moncton (Canada) and Head of the Perception, Robotics, and Intelligent Machines Group. Before Joining the University of Moncton in 2016, he worked in the industry and technology transfer in the areas of computer vision and robotics for over 15 years. He is a Senior Member of IEEE and member of SPIE.