Distributed Reinforcement Learning for Flexible UAV Swarm Control with Transfer Learning Capabilities

Federico Venturini*, Federico Mason*, Francesco Pase*,
Federico Chiariotti*, Alberto Testolin*†, Andrea Zanella*, Michele Zorzi*
*Department of Information Engineering, University of Padova - Via Gradenigo, 6/b, Padova, Italy
†Department of General Psychology, University of Padova - Via Venezia, 8, Padova, Italy
E-mail:{venturin, masonfed, pasefran, chiariot, testolin, zanella, zorzi}@dei.unipd.it

ABSTRACT

Over the past few years, the use of swarms of Unmanned Aerial Vehicles (UAVs) in monitoring and remote area surveillance applications has become economically efficient thanks to the price reduction and the increased capabilities of drones. The drones in the swarm need to cooperatively explore an unknown area, in order to identify and monitor interesting targets, while minimizing their movements. In this work, we propose a distributed Reinforcement Learning (RL) approach that scales to larger swarms without modifications. The proposed framework can easily deal with non-uniform distributions of targets, drawing from past experience to improve its performance. In particular, our experiments show that when agents are trained for a specific scenario, they can adapt to a new one with a minimal amount of additional training. We show that our RL approach achieves favorable performance compared to a computationally intensive look-ahead heuristic.

CCS CONCEPTS

• **Computing methodologies** → **Multi-agent reinforcement learning**; *Mobile agents*; • **Computer systems organization** → *Robotic control*;

KEYWORDS

Surveilling, UAV networks, Distributed Deep RL, multi-agent RL

ACM Reference Format:
2020. Distributed Reinforcement Learning for Flexible UAV Swarm Control with Transfer Learning Capabilities. In *The 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DroNet'20), June 19, 2020, Toronto, ON, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3396864.3399701

1 INTRODUCTION

The high data rate achievable with modern wireless communications and the increasing computational power of embedded systems, along with the sharp price reduction of commercial Unmanned Aerial Vehicles (UAVs), have enabled the use of swarms of drones for Smart City services [**?**]. Thanks to their size, flexibility and flight ability, these swarms represent a new solution for a plethora of different applications, such as remote surveillance, distributed sensing, wireless coverage extension and object tracking [**?**].

Over the past few years, researchers have proposed several UAV-based systems [**?**], but achieving an efficient distributed control is a complex problem, whose solution is often task-dependent. In this context, it is important to properly define the different sub-tasks of surveillance, monitoring, mapping and tracking [**?**]. In this work, we assume that targets are static, but occupy random positions in the monitored area. Moving UAVs are equipped with sensors that can detect targets within a limited sensing range, and need to explore the area and find the targets.

Once the initial exploration has been performed, it is easy to extend the problem to include moving targets, so an efficient initial exploration is of interest even in this larger class of problems. The problem of identifying fixed targets arises in several practical situations, ranging from the generation of real-time flood maps [**?**] to the detailed tracking of weeds in agriculture [**?**]. Note that an efficient exploration of unknown areas to locate fixed targets is also of interest for more complex scenarios, e.g., with moving targets. One such example is wildfire monitoring in dry regions [**?**], although the benefits of efficient exploration might be limited if the spread of the fires is faster than the UAVs' movement.

The dynamic nature of these problems, in which actions can have long-term consequences and affect the future evolution of the environment in complex ways, makes them a natural application area for Reinforcement Learning (RL) techniques. However, due to the curse of dimensionality, tackling the problem in a centralized fashion (i.e., using a single controller) is feasible only for very small swarms. In order to design a scalable system, Multi-Agent Reinforcement Learning (MARL) techniques need to be used, but the non-stationarity of the environment [**?**] complicates the system design and the agent training. This additional complexity makes MARL an open research field, and the different possible degrees of centralization and communication between agents make the configuration of the learning system rather complex. An extensive taxonomy of these solutions was developed in [**?**]. The general approaches adopted to solve the MARL problem can be cast into one of these four frameworks: (1) a single agent architecture that interacts with multiple copies of itself generating emergent behaviors; (2) communication learning to improve coordination; (3) cooperation learning through local actions and observations; and (4) modeling other agents' behaviors and planning a response [**?**].

The authors in [**?**] study the first of these four approaches and use the tabular Q-learning algorithm to guide drones to survey an unknown area, showing that even the simplest MARL algorithm can

DroNet'20, June 19, 2020, Toronto, ON, Canada

ACM ISBN 978-1-4503-8010-2/20/06…$15.00
https://doi.org/10.1145/3396864.3399701

improve the overall system rewards. Similarly, in [**?**] and [**?**] the MARL framework is applied to a more complex problem in which a UAV network is adopted to provide flexible wireless communication. However, in these works the MARL algorithm is used to optimize resource allocation instead of guiding drones, where a coordinated exploration strategy is still missing.

An interesting research direction for communication learning in MARL is pioneered in [**?**], which uses Deep Neural Networks (DNNs) to represent and learn more complex Q-functions [**?**]. At first, the authors study the performance of one network trained for all agents, which then share the same parameters during the execution phase (this is also our approach). A second proposed system uses the Differentiable Inter-Agent Learning (DIAL) framework, in which agents learn meaningful real-valued messages to be exchanged in order to improve cooperation: this allows for faster training, but the model is limited to a very small number of agents.

Other works use RL in the practical scenarios discussed above: in [**?**], the authors adopt a MARL approach to control a flood-finding swarm of UAVs. However, the model only considers a swarm with a fixed number of drones, and the experimental results are not compared to state-of-the-art heuristics. In [**?**], the reinforced random walk model is exploited to map weeds in an agricultural setting, taking noisy acquisition into account and solving the issue with collective observations. Random walks are then biased based on the positions of the already discovered targets, which have to be properly mapped, along with the distances from other drones in the network. In this case, the authors considered swarms of variable sizes, but the random walk needs to be manually tuned for each setting. Another recent study [**?**] considers wildfire spread monitoring, considering a known starting point and checking how the fire evolves and spreads in the map. The authors define the problem as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) and report several experiments, as well as comparisons against a greedy heuristic (similar to the look-head method we studied in our experiments).

In this work, we consider a general MARL framework for the initial exploration and surveillance. Our scheme follows the framework in which observations of other agents are used to make decisions and to avoid collision, thus encouraging cooperation. We define a Deep Q-Network (DQN) algorithm and demonstrate its efficiency with limited training, comparing it to a benchmark look-ahead heuristic and showing that the MARL scheme can explore the environment and reach the targets faster. We also perform a transfer learning experiment, showing that agents trained on a different map can learn to adapt to a completely new scenario much faster than restarting the training from scratch.

The rest of the paper is divided as follows: the system model and MARL algorithm are presented in Sec. 2. The experimental results, including transfer learning experiments, are reported in Sec. 3, while Sec. 4 concludes the paper and presents some possible avenues of future work.

2 SYSTEM MODEL

Our aim is to find a flexible Machine Learning (ML) strategy to explore and monitor a certain area with a swarm of UAVs. Performance is determined by the ability of the drones to find and reach the targets, which are located in unknown positions. We adopted a general framework, using a grid-world representation and making a limited number of assumptions on the nature of the task. In the following, we first present the environment definition. We then model the scenario as a Networked Distributed Partially Observable Markov Decision Process (ND-POMDP) and present the algorithm by which the UAVs learn to maximize the system performance.

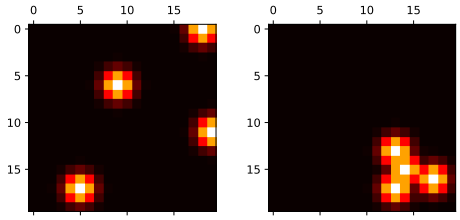

Figure 1: Two examples of the sparse (left) and cluster (right) target distributions with $d = 8$ for the sparse scenario.

2.1 Environment

The system environment consists of a square grid of size $M \times M$. Each cell of the grid is identified by its coordinates $\mathbf{x} \in \mathcal{M}^2$, where $\mathcal{M} = \{0, ..., M-1\}$. Each cell is associated with a specific value $\phi(\mathbf{x})$, which represents the "value" of the location, which increases with the proximity to a target. In practice, we generate a set of K bivariate Gaussian functions over the grid, with the same covariance matrix $\Sigma = \left(\begin{smallmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{smallmatrix}\right)$. The mean $\mathbf{z}_k = (x_k, y_k)$ corresponds to the coordinates of the target. Note that the Gaussian functions do not represent actual distributions, but rather the full view of the UAVs, which can see a target from afar. The value of a generic cell $\mathbf{x}$ is given by the maximum among all the Probability Density Functions (PDFs) of the K distributions, normalized in such a way that the target locations have values equal to 1:

$$\phi(\mathbf{x}) = \max_{k \in \{0,\dots,K-1\}} e^{-\frac{1}{2}((\mathbf{x}-\mathbf{z}_k)^{\mathrm{T}}\Sigma(\mathbf{x}-\mathbf{z}_k))} \tag{1}$$

If $\phi(\mathbf{x})$ is smaller than 0.01, it is set to 0, as the UAVs cannot see the target from that far. Under these conditions, the most valuable cells coincide with the mean of each Gaussian distribution, which represents one of the targets in the considered scenario. While the environment is static, a set of U UAVs move within the map with the aim of positioning themselves over the targets as fast as possible.

In this work, we consider two different scenarios, namely *sparse* and *cluster*, which are characterized by different correlations among the target positions. In both cases, the first target is randomly placed on the grid following a 2D uniform distribution: $z_0 = (x_0, y_0)$ can take any value in $\mathcal{M}^2$ with equal probability. The other targets are then placed sequentially, according to the following rules.

In the sparse scenario, the position z_i of the i-th target is randomly chosen in the set $\mathcal{M}_i^{\mathrm{s}} = \{\mathbf{x} \in \mathcal{M}^2 : ||\mathbf{x} - \mathbf{z}_j||_2 > d \; \forall j < i\}$, with probability mass distribution

$$P_{\mathrm{s}}(\mathbf{z}_i = \mathbf{x}) = \frac{||\mathbf{x} - \mathbf{z}_0||_2}{\rho_i^{\mathrm{s}}} \prod_{j=1}^{i-1} u(||\mathbf{x} - \mathbf{z}_j||_2 - d). \tag{2}$$

where $u(\cdot)$ is the step function and the normalization factor ρ_i^{s} ensures that $\sum_{\mathbf{x} \in \mathcal{M}_i^{\mathrm{s}}} P_{\mathrm{s}}(\mathbf{z}_i = \mathbf{x}) = 1$. Hence, the other targets tend

to be distributed far from the first, with a minimum distance d between each other, where d is a scalar parameter depending on the scenario.

In the cluster scenario, instead, the i-th target can take any position in the set $\mathcal{M}_i^c = \{\mathbf{x} \in \mathcal{M}^2 : ||\mathbf{x} - \mathbf{z}_j||_2 > 1 \; \forall j < i\}$ with probability mass distribution

$$P_c(\mathbf{z}_i = \mathbf{x}) = \frac{1}{(1 + ||\mathbf{x} - \mathbf{z}_0||_2)\rho_i^c} \prod_{j=1}^{i-1} u(||\mathbf{x} - \mathbf{z}_j||_2 - 1). \quad (3)$$

where ρ_i^c is chosen in such a way that $\sum_{\mathbf{x} \in \mathcal{M}_i^c} P_c(\mathbf{z}_i = \mathbf{x}) = 1$. In this case, the targets tend to cluster around the first one, but cannot occupy adjacent cells, since the minimum distance must be greater than 1. An example of the two target placements is shown in Fig. 1.

We consider multiple *episodes*: in each episode, the targets are redistributed in the map, and the swarm must locate them in as few steps as possible. We consider discrete time slots, so that each drone can move by a single cell at each time step n and has a limited Field of View (FoV) with a radius of ζ cells. This framework allows us to represent many different applications and scenarios by changing the size of the grid, the number of drones, the number of targets, and the variance parameter σ. It can also be easily extended to dynamic targets and more complex scenarios, which can include obstacles such as no-fly zones and tall buildings.

At the beginning of each episode, each UAV only knows the values of the cells within the swarm's FoV ζ. The drones assume that all unexplored points of the map are associated with the maximum $\phi(\mathbf{x})$. Then, each UAV can move independently at each time step n: as the swarm explores the environment, each drone discovers the values of the map locations that it has covered, and updates its information according to $\phi(\mathbf{x})$. We highlight that the knowledge about the map is shared, which means that each drone receives the observations that all the other drones have acquired. The objective of the swarm for each episode is to position each of its UAVs above a target as quickly as possible.

2.2 ND-POMDP formulation

The described scenario is modeled as an ND-POMDP [**?**], i.e., a Markov Decision Process (MDP) where the system state in not directly observable and is influenced by the actions of multiple agents, whose behavior is not centrally coordinated. Indeed, the swarm only has limited knowledge of the map, and the UAVs can take actions independently and have independent rewards. Formally, an ND-POMDP is identified by a 4-tuple, including a state space $\mathcal{S}$, a joint action space $\mathcal{A}$, an observation space $\mathcal{O}$, and a reward map $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^U$, where U is the number of distributed agents.

In our case, each UAV can either stay over the same cell or move to one of the four adjacent cells. Hence, the action space for the swarm is the set $\mathcal{A} = \{(0,0), (0,1), (1,0), (0,-1), (-1,0)\}^U$. The complete system state s is given by three matrices: the $2 \times U$ matrix $\mathbf{X}$ with the UAVs' positions, the $M \times M$ value matrix $\mathbf{\Phi}$, whose elements correspond to the value $\phi(\mathbf{x})$ of each cell, and the $M \times M$ observed value matrix $\hat{\mathbf{\Phi}}$, whose elements are equal to $\phi(\mathbf{x})$ if the cell has been explored and 1 otherwise. The observation o that is available to the drones is given by $\mathbf{X}$ and $\hat{\mathbf{\Phi}}$, as the UAVs can exchange information about the explored portions of the map but do not know the value of unexplored areas.

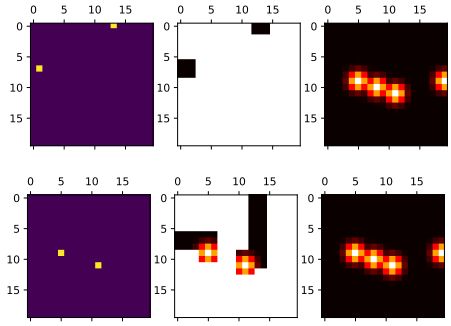

Figure 2: Drone positions (left), known map (center), real map (right). Beginning (above) and end (below) of an episode.

Fig. 2 depicts the system state at the beginning and in an advanced stage of an episode, with two drones and four targets located in a 20×20 map. In particular, the drones' positions are shown on the left (in yellow), the observed value map is in the center, and the real value map is on the right. In the figure, darker cells are associated with lower values and brighter cells are associated with higher values. In the figure, the observed state o for all UAVs corresponds to the two maps on the left and in the center. It is easy to see how the swarm gains knowledge during the episode, as the drones explore the map and look for targets. In this case, the UAVs found two targets relatively quickly, and a significant portion of the grid remained unexplored.

We give reward 1 to a UAV if it is directly above a target, reward $-\theta$ if it tries to go outside the map, reward $-\psi$ if it is in the same cell as another drone, and reward 0 in any other case. The UAVs should learn quickly to avoid actions that take them outside the map, so the exact value of θ does not affect the final performance, but the value of ψ affects how far away from each other the drones try to keep: if ψ is low, the drones will get close to each other if the targets are very close. Naturally, if there is a collision risk when the drones are in the same cell, the value of ψ should be high. The reward depends on $\mathbf{X}$, as well as on the action vector $\mathbf{a}$. Indicating with $\mathbf{x}_u$ and $\mathbf{a}_u$ the position and action of drone u, we now define the collision variable $\chi_u(\mathbf{X}, \mathbf{a})$ as

$$\chi_u(\mathbf{X}, \mathbf{a}) = I\left(\sum_{v \neq u} \delta(\mathbf{x}_u + \mathbf{a}_u - \mathbf{x}_v - \mathbf{a}_v) > 0\right), \quad (4)$$

where $I(E)$ is equal to 1 if the event E is true, and 0 otherwise, while $\delta(\mathbf{x})$ is equal to 1 if $\mathbf{x} = (0,0)$, and 0 otherwise. In short, $\chi_u(\mathbf{X}, \mathbf{a})$ has value 1 if one or more drones move to the same cell as drone u, and 0 otherwise. The collision variable depends on the moves of other agents, so the problem is distributed. The reward function for UAV u, denoted as $r_u(\mathbf{X}, \mathbf{a})$, is given by:

$$\begin{aligned} r_u(\mathbf{X}, \mathbf{a}) =& (1 - \chi_u(\mathbf{X}, \mathbf{a})) \sum_{k=0}^{K-1} [\delta(\mathbf{x}_u + \mathbf{a}_u - \mathbf{z}_k)] \\ & - \theta I(\mathbf{x}_u + \mathbf{a}_u \notin \mathcal{M}) - \psi \chi_u(\mathbf{X}, \mathbf{a}). \end{aligned} \quad (5)$$

In our model, the state transitions and the system observations are both deterministic; therefore, both the state evolution and the observation are not affected by random events, but only

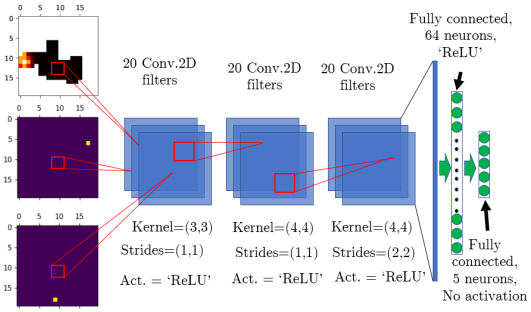

Figure 3: Architecture of the DQN.

by the agents' decisions. We define a policy π as a mapping between observations and action probabilities. Under these assumptions, the goal of each drone u is to find the policy π^* that maximizes the cumulative expected future discounted reward $R_{u,t}(\pi) = \mathbb{E}\left[\sum_{\tau=t}^{+\infty} \gamma^{\tau-t} r_{u,\tau} | o(t), \pi\right]$, where $\gamma \in [0, 1)$ is named *discount factor*.

2.3 Distributed Deep Q-Learning

To make the agents learn the best strategy to achieve this goal, we adopt a Distributed Deep Q-Learning (DDQL) approach. Each agent leverages a DQN, i.e., a Neural Network (NN) that takes as input the last system observation o_t and returns the Q-values of the possible actions that can be taken, i.e., $Q(o_t, a) \; \forall \; a \in A$. In Q-learning, the function $Q(o, a)$ is an estimate of the expected long-term reward R that will be achieved by choosing a as the next action and following the learnt policy. In our case, we maintain a single DQN during the training phase, whose values are shared by all the agents.

In this work, we follow the approach from [?] and leverage a *replay memory* to store the agent experience $e_t = (o_t, a_t, r_t, o_{t+1})$. Whenever the agent carries out a training step, a batch of B_{size} elements is picked from the replay memory, allowing to separate the algorithm training from the experience acquisition. The replay memory is shared between the agents during a training phase, and a new batch is used to train the agent at every step.

In addition, we exploit the *double Q-learning* technique to remove biases from the Q-value estimation and speed up the algorithm convergence [?]. This means that, during the training, we use two different Q functions $Q_A(o, a)$ and $Q_B(o, a)$, where $Q_A(o, a)$ is used to select actions and $Q_B(o, a)$ is used to evaluate actions. In particular, the model is updated as

$$Q_B^{\text{new}}(o_t, a_t) = (1-\alpha) Q_B(o_t, a_t) + \alpha \left[r_t + \gamma \max_a Q_A(o_{t+1}, a) \right], \quad (6)$$

and every n_q training steps the weights of $Q_A(o, a)$ are updated with the weights of $Q_B(o, a)$. The learning rate parameter α is set automatically by the Rectified Adam (RAdam) optimizer [?].

In our model, the observed state of the system for each agent can be represented by three matrices, representing the agent position, the locations of the other agents, and the known map, respectively. Therefore, our system approximates the function $Q(o, a)$ by a Convolutional Neural Network (CNN), whose architecture is described in Fig. 3. In particular, we consider a CNN exploiting three convolutional layers followed by two fully-connected layers. The dimension of the last layer is identical to the number of actions, so that each output element can be associated to a different action $a \in A$.

Parameter	Value	Description
M	20	Grid size
U	$\{2, 3\}$	Number of UAVs
K	4	Number of targets
σ^2	1	Targets variance
ζ	3x3	Field of View
d	8	Minimum target distance (sparse scenario)
θ	1	Outside penalty
ψ	0.8	Collision penalty
γ	0.9	Discount factor
α	Chosen by RAdam	Learning rate
N_e	{250, 1000}	Training episodes
N_s^t	300	Steps per training episode
N_s^p	40	Steps per test episode
N_t	{125, 250}	Transfer learning episodes
N_p	500	Test episodes

Table 1: Simulation settings.

Hence, each agent trains its own model following (6), so that the CNN output can converge to the Q-values $Q(o, a) \; \forall \; a \in A$. We implement the well-known ε-greedy policy to allow the agents to explore the action space during the training phase, which is carried out by simulating a sequence of episodes.

3 SIMULATION SETTINGS AND RESULTS

In this section, we describe the simulations by which we evaluated the performance of the designed system. All the results are derived through a Monte Carlo approach, where multiple independent simulations are carried out to obtain reliable statistical data. In particular, the algorithms' training is executed by carrying out a total of N_e episodes for each studied scenario (sparse or cluster), where each episode is given by N_s^t steps. Training episodes are far longer than test episodes, which have length N_s^p, as we need to allow the agents to explore the map fully.

Before training, we initialize the replay memory by executing $N_e^m = 100$ episodes of N_s^t steps each, to allow agents to immediately start the learning procedure. Moreover, we apply *transfer learning* to allow the agents trained in the sparse environment to quickly adapt to the cluster scenarios; to this goal, additional N_t training episodes are carried out. Finally, the performance of the proposed strategy is tested in a total of N_p episodes. The exploration rate ε follows a decreasing staircase-like curve during the training and is set to 0 in the test episodes. The training and testing processes are performed independently 5 times to verify the robustness of the DDQL scheme, and the complete simulation settings are reported in Tab.1.

To assess the performance of our DDQL scheme, we compare it with a heuristic strategy inspired by Model Predictive Control (MPC), by which drones can explore the map and reach the targets. Such a strategy is named *look-ahead* and is used as a benchmark for our analysis. In the look-ahead strategy, at each time-step each drone u receives the reward

$$r_u^{(\ell)}(\mathbf{X}, \mathbf{a}) = \frac{\phi(\mathbf{x}_u + \mathbf{a}_u)}{\eta(\mathbf{x}_u + \mathbf{a}_u)} - \theta I(\mathbf{x}_u + \mathbf{a}_u \in \mathcal{M}^2), \quad (7)$$

where $\eta(\mathbf{x})$ is the number of UAVs located in $\mathbf{x}$. To decide its next action, each drone u tries to maximize its expected cumulative reward over the following n_ℓ steps, assuming that none of the other drones move. Practically, the look-ahead strategy makes each drone select the action $\mathbf{a}^*$ that maximizes

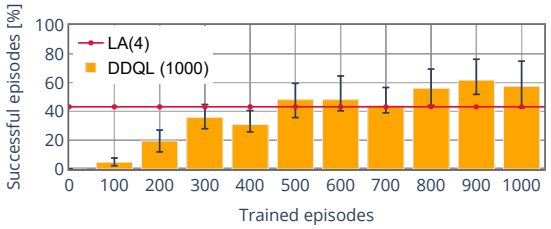

Figure 4: Success probability over the training phase in the cluster scenario with 2 UAVs.

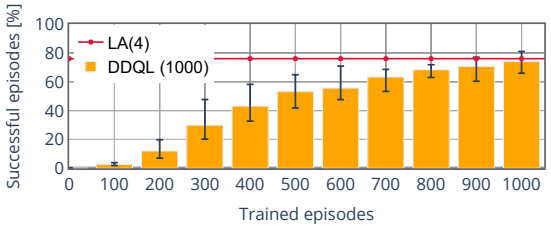

Figure 5: Success probability over the training phase in the sparse scenario with 2 UAVs.

$$\max_{\mathbf{A}\in\tilde{\mathcal{A}}^{n_\ell}} \sum_{i=0}^{n_\ell-1} r_u^{(\ell)}\left(\mathbf{X}+\sum_{j=0}^{i-1}\mathbf{A}^j,\mathbf{a}^i\right). \tag{8}$$

In (8), $\tilde{\mathcal{A}}^{n_\ell}$ is the set of ordered sequences $\mathbf{A}$ of action vectors $\mathbf{A}^0$, $\mathbf{A}^1$, ..., $\mathbf{A}^{n_\ell-1}$, so that $\hat{\mathbf{a}}_u^0 = a^*$ and $\mathbf{a}_v^i = (0,0)\ \forall\ i \in \{0,...,n-1\}$ and $\forall\ v \neq u$, i.e., the set of possible move sequences of u while the other UAVs are static. At the beginning of an episode, each drone u assumes that all the map values $\phi(\mathbf{x})$ outside its FoV are equal to 1; therefore, look-ahead forces u to continuously explore the map. However, as soon as it finds a target, u will stop over the target center. The target is then eliminated from other agents' perception, as it is already covered by a UAV. We highlight that the performance of look-ahead mainly depends on the n_ℓ parameter: as n_ℓ increases, drones can make more foresighted decisions, but at a greater computational cost.

3.1 Results: training and convergence

We now analyze the performance over 1000 training episodes, in a scenario with 2 UAVs and 4 targets in a 20×20 map. The look-ahead approach is abbreviated as LA(4), as we set $n_\ell = 4$. This already had a significant computational cost, and in our simulation each look-ahead decision takes approximately 15 times longer than running a trained DDQL agent. Fig. 4 shows the success probability in the cluster scenario as a function of the training set size: the DDQL approach catches up with the look-ahead in less than 500 episodes, converging to a success probability between 0.5 and 0.6 after approximately 800 episodes. The error bars show the best and worst result over 5 test phases, showing that the performance improves as the UAVs gain more experience. The performance boost over the look-ahead approach is due to the DDQL scheme's ability to exploit the correlation among the target positions, quickly finding the other targets after the first one has been spotted.

The opposite happens in the sparse scenario, as Fig. 5 shows: since targets are often far apart, the look-ahead approach can identify slightly more targets in time. In this case, more training would probably bring the DDQL approach to the same level. In general, both approaches have more success than in the cluster scenario, as finding the cluster and reaching the targets is not always possible in the limited duration of an episode.

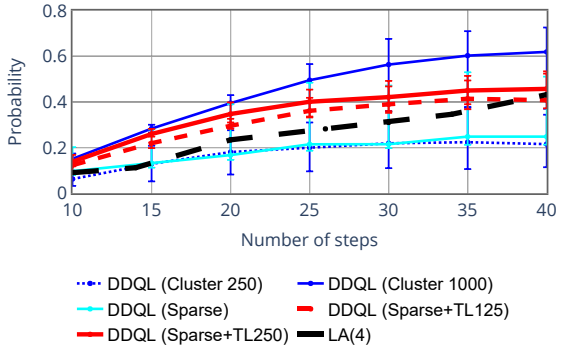

Figure 6: CDF of the episode duration for different algorithms in the cluster scenario with 2 UAVs.

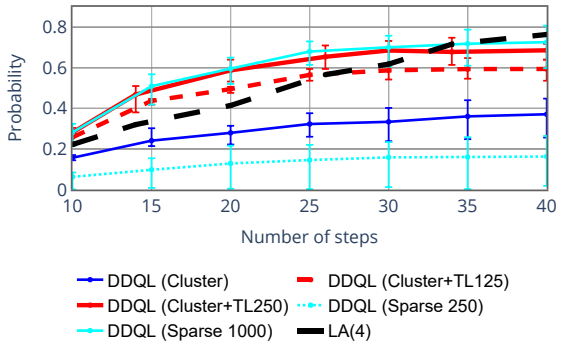

Figure 7: CDF of the episode duration for different algorithms in the sparse scenario with 2 UAVs.

3.2 Results: exploration and transfer learning

In order to show the flexibility of the DDQL approach, we show the Cumulative Distribution Function (CDF) of the episode duration, i.e., the time until both drones reach targets or the episode limit of 40 steps is reached, for the look-ahead benchmark and DDQL.

Fig. 6 shows the results obtained in the cluster scenario when considering different training options for the DDQL algorithm: a training of $N_e = 250$ or $N_e = 1000$ episodes in the cluster scenario (Cluster 250/1000); a training of $N_e = 1000$ episodes in the sparse scenario (Sparse); a pre-training of $N_e = 1000$ episodes in the sparse scenario, followed by a training of additional $N_t = 125$ or $N_t = 1000$ episodes in the cluster scenario (Sparse+TL125/250), according to the transfer learning approach. 250 episodes of training in the cluster scenario are not enough for the agents to learn how to explore the map, while with 1000 episodes the DDQL algorithm reaches more targets in fewer steps than the look-ahead LA(4). As expected, a training in the sparse scenario does not yield good performance. However, retraining the CNN on a smaller number of episodes in the correct (cluster) scenario allows the algorithm to beat the look-ahead approach.

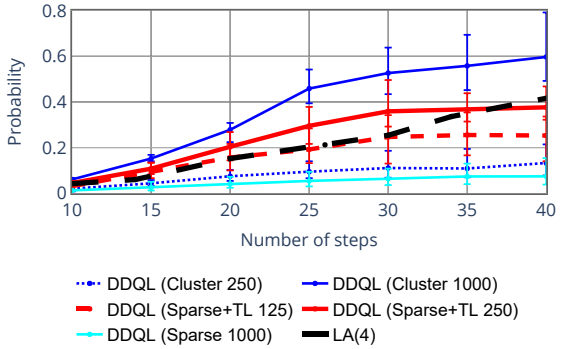

Figure 8: CDF of the episode duration for different algorithms in the cluster scenario with 3 UAVs.

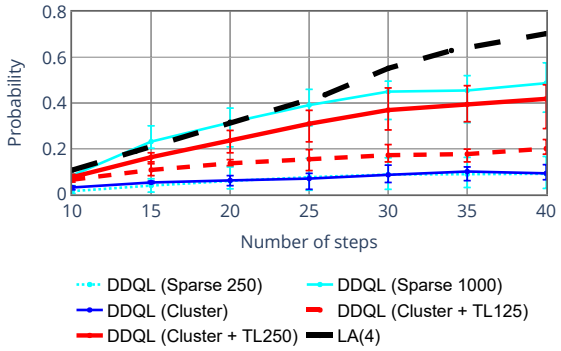

Figure 9: CDF of the episode duration for different algorithms in the sparse scenario with 3 UAVs.

We repeated the experiment for the sparse scenario and we report the obtained results in Fig. 7. In this case, the look-ahead exploration can reach slightly more targets, but the DDQL approach is much faster. In this case, transfer learning is even more effective, as a 250 episode re-training on a network trained on the cluster scenario can achieve almost the same results as a network trained on the correct scenario.

We also show the results for a scenario with 3 UAVs in Fig. 8 and Fig. 9: in this case, transfer learning is slightly less effective in the cluster scenario, as Fig. 8 shows, but can still achieve approximately the same results as the much more computationally expensive look-ahead approach. In the sparse scenario (Fig. 9), transfer learning is effective, but the swarm needs more training to understand how to act, and the look-ahead approach outperforms it.

4 CONCLUSIONS AND FUTURE WORK

In this work, we faced the problem of area monitoring and surveillance with a swarm of drones. We modeled the environment with a 2D grid and cast the problem into the theoretical framework of ND-POMDP. By our experiments we showed how our DDQL solution brings interesting benefits in terms of scalability, exploration speed, and transfer capability.

Important research directions include the study of the system performance in case of real wireless communications, which may affect the quality of the observations. To explore scalability, a larger map with only partial communication between UAVs might also be interesting: exploiting transfer learning in scenarios with different constraints will enable the design of new adaptive UAV systems. Finally, the introduction of physical obstacles and dynamic targets would be an important step to increase the scenario's realism.

ACKNOWLEDGMENTS

This work has been partially supported by the US Army Research Office under Grant no. W911NF1910232, "Towards Intelligent Tactical Ad hoc Networks (TITAN)."

REFERENCES

[] Dario Albani, Daniele Nardi, and Vito Trianni. 2017. Field coverage and weed mapping by UAV swarms. In *International Conference on Intelligent Robots and Systems*. IEEE, 4319–4325.

[] David Baldazo, Juan Parras, and Santiago Zazo. 2019. Decentralized multi-agent deep reinforcement learning in swarms of drones for flood monitoring. In *European Signal Processing Conference (EUSIPCO)*. EURASIP.

[] Lucian Busoniu, Robert Babuška, and Bart De Schutter. 2010. Multi-agent Reinforcement Learning: An Overview. *Innovations in Multi-Agent Systems and Applications* 310 (Nov. 2010), 113–147.

[] Soon Jo Chung, Aditya Avinash Paranjape, Philip Dames, Shaojie Shen, and Vijay Kumar. 2018. A Survey on Aerial Swarm Robotics. *IEEE Transactions on Robotics* 34, 4 (2018), 837–855.

[] Jingjing Cui, Yuanwei Liu, and Arumugam Nallanathan. 2019. The application of multi-agent reinforcement learning in UAV networks. In *International Conference on Communications Workshops (ICC)*. IEEE.

[] Jingjing Cui, Yuanwei Liu, and Arumugam Nallanathan. 2020. Multi-Agent Reinforcement Learning-Based Resource Allocation for UAV Networks. *IEEE Transactions on Wireless Communications* 19, 2 (Feb. 2020), 729–743.

[] Jakob N. Foerster, Yannis M. Assael, Nando De Freitas, and Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*. 2145–2153.

[] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. 2017. A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity. In *3rd International Workshop on Conflict Resolution in Decision Making (COREDEMA)*.

[] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. 2019. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 33, 6 (Nov. 2019), 750–797.

[] Naser Hossein Motlagh, Tarik Taleb, and Osama Arouk. 2016. Low-Altitude Unmanned Aerial Vehicles-Based Internet of Things Services: Comprehensive Survey and Future Perspectives. *IEEE Internet of Things Journal* 3, 6 (Sept. 2016), 899–922.

[] Kyle D. Julian and Mykel J. Kochenderfer. 2019. Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning. *Journal of Guidance, Control, and Dynamics* 42, 8 (Aug. 2019), 1768–1778.

[] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2019. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265* (Aug. 2019).

[] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533.

[] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. 2005. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *19th Conference on Artificial Intelligence*, Vol. 5. AAAI, 133–139.

[] Reza Shakeri, Mohammed Ali Al-Garadi, Ahmed Badawy, Amr Mohamed, Tamer Khattab, Abdulla Khalid Al-Ali, Khaled A. Harras, and Mohsen Guizani. 2019. Design Challenges of Multi-UAV Systems in Cyber-Physical Applications: A Comprehensive Survey and Future Directions. *IEEE Communications Surveys and Tutorials* 21, 4 (June 2019), 3340–3385.

[] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. 2019. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access* 7 (April 2019), 48572–48634.

[] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with Double Q-learning. In *30th Conference on Artificial Intelligence*. AAAI.

[] Riccardo Zanol, Federico Chiariotti, and Andrea Zanella. 2019. Drone mapping through multi-agent reinforcement learning. In *Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–7.