IET Computer Vision

Special Issue: Computer Vision for Smart Cameras and Camera Networks

IET Journals
The Institution of Engineering and Technology

Drone swarm patrolling with uneven coverage requirements

ISSN 1751-9632
Received on 5th December 2019
Revised 7th April 2020
Accepted on 1st May 2020
E-First on 14th October 2020
doi: 10.1049/iet-cvi.2019.0963
www.ietdl.org

Claudio Piciarelli[1] ✉, *Gian Luca Foresti*[1]

[1]*Università degli Studi di Udine, Via delle Scienze 206, 33100 Udine, Italy*

✉ *E-mail: claudio.piciarelli@uniud.it*

Abstract: Swarms of drones are being more and more used in many practical scenarios, such as surveillance, environmental monitoring, search and rescue in hardly-accessible areas and so on. While a single drone can be guided by a human operator, the deployment of a swarm of multiple drones requires proper algorithms for automatic task-oriented control. In this study, the authors focus on visual coverage optimisation with drone-mounted camera sensors. In particular, they consider the specific case in which the coverage requirements are uneven, meaning that different parts of the environment have different coverage priorities. They model these coverage requirements with relevance maps and propose a deep reinforcement learning algorithm to guide the swarm. This study first defines a proper learning model for a single drone, and then extends it to the case of multiple drones both with greedy and cooperative strategies. Experimental results show the performance of the proposed method, also compared with a standard patrolling algorithm.

1 Introduction

During the last years, the use of multicopter drones has gained large popularity in many practical application fields, such as agriculture [1], surveillance [2], disaster management [3], search and rescue [4], environmental data acquisition [5], medicine delivery [6] and so on. This interest in commercial, scientific and social fields led to a drastic change in the way drones are used if compared to previous applications, which were mostly confined to video acquisition. While a single drone, manually controlled by a human operator, was a typical scenario up to few years ago, current applications often rely on multiple drones (swarms) autonomously cooperating to perform a given task. This justifies the increment of scientific works on topics such as automatic drone control, path planning, smart resource management and so on.

In this work, we focus on the problem of drone visual coverage: multicopter drones are equipped with cameras to observe a portion of the environment for specific tasks (e.g. surveillance) and the observed area must be optimised according to given criteria, under the assumption that the environment is too large to get a simultaneous full coverage. For example, in a surveillance application, a basic requirement could be that no area is left uncovered for more than a given amount of time, in order to avoid 'blind spots' in the surveillance application.

A task like the one just described can be accomplished with uniform coverage, given that enough drones are available. In this paper, however, we want to investigate scenarios with uneven coverage requirements. This means that not all the areas of the environment have the same importance: some parts should be observed more frequently, or require immediate coverage, if compared to other zones. Each point of the environment thus has a given priority (or *relevance*, using the term that will be adopted in the rest of the paper), and the goal of the system is to find a patrolling strategy that optimally covers the environment according to these relevance requirements. For example, in a disaster management context such as a flooding or fire in extended rural areas, the zones around known buildings should have priority for visual inspection in order to quickly identify people in danger.

We propose to model the problem of prioritised visual coverage as a Markov decision process, where each drone is an agent that can actuate several actions to change its state (e.g. moving forward, rotating, zooming the camera etc.) and it gets a reward (either positive or negative) in terms of coverage quality as a consequence of its actions. This model allows to search for a patrolling strategy using reinforcement learning, thus letting the drone learn from experience rather than explicitly giving a patrolling algorithm. The reinforcement learning algorithm will be implemented using a neural network, thanks to the Deep Q-Network architecture. The proposed model is theoretical, meaning that the set of actions and the state space do not consider all the variables of a real-world system (such as drone movement inertia, power consumption constraints etc.), however it can be used as a reference framework for possible implementations. Our basic idea of reinforcement-learning-based patrolling was already presented in [7], however in that work only the single-drone case was considered. In this paper, we extend our previous work by improving the patrolling model (see (23)) and by considering the novel scenario of multi-drone swarms. In this case, we propose two different strategies, a greedy and a cooperative one, depending on the type of information shared between drones. Compared to [7], also the experimental results have been extended, even for the single-drone scenario.

The paper is organised as follows. In Section 2, we review some of the most relevant works in literature regarding drone coverage tasks, while in Section 3 we recap the basic theory behind reinforcement learning and its deep learning implementations. In Section 4, the Markov decision process model for a single drone is described by defining the state space, the action space and the reward function in terms of visual coverage of relevance maps. A patrolling strategy is then implemented using the given model, which is then extended to the case of multiple drones in Section 5, where two swarm patrolling strategies are proposed. Experimental results are given in Section 6.

2 Related works

The problem of drone control for patrolling tasks has been studied by several authors. A survey of coverage path planning algorithms in robotics can be found in [8], while [9] is specifically focused on drones. Here, the authors propose a taxonomy of the cited methods, ranging from simple geometric flight patterns to more complex grid-based solutions considering full and partial information about the area of interest. The authors mostly focus on how to cover areas with complex geometrical shapes but, in contrast with the proposed method, none of the surveyed works explicitly considers uneven coverage requirements, neither addresses the problem using neural networks.

In [10], the authors give a survey on dynamic reconfiguration of camera networks, which is a superset of the considered problem. They explicitly discuss coverage optimisation methods and drone deployment strategies, although the two topics are separated: surveyed coverage-oriented methods are mostly focused on PTZ camera networks (in which the camera cannot translate), while the analysed drone reconfiguration works are more focused on resource management (e.g. as in [11, 12]).

The joint task of area coverage and resource management has been studied in [13], where the authors give a deep mathematical formulation of 3D coverage and they propose a resource-aware algorithm that shifts the bulk of spatial redistribution onto less constrained agents. A similar topic is addressed in [14], where the problem of coverage-driven path planning is studied from the point of view of resource management such as energy consumption. In [15], the problem is analysed from the novel point of view of path planning in adversarial environments, where the efficient use of chaotic behaviours copes with enemy entities. The work proposed in [16] is again oriented to energy-efficient algorithms for coverage optimisation, although in this case the authors deal with communication coverage, where drones are used as a communication infrastructure to deal with emergency situations. Other works focus on the decentralised aspects of the task, such as in [17]: in the case of a swarm of drones, distributing the overall computation over all the agents allows efficient implementations that do not rely on a single point of failure. All these works are thus mostly focused on resource awareness, a topic that we do not address in this work.

To the best of our knowledge, few works have been published dealing with drone coverage problems using neural networks and/or reinforcement learning. The work presented in [18] uses a deep-sarsa approach, thus adopting a reinforcement-learning-based approach as in our work, however it is focused on target-based guidance with collision avoidance rather than on patrolling. In [19], the authors use reinforcement learning for attitude control, and thus they are more focused on short-term stability-oriented tasks rather than mission-level, long-term objectives like in our case.

3 Reinforcement learning essentials

Reinforcement learning is a learning strategy in which an agent in a given state executes an action and gets an immediate reward (or penalty) as a consequence of that action. The goal is to learn from this experience, figuring out the best actions that will eventually lead to a maximisation of the total reward on the long term. This approach can be easily applied to coverage-oriented drone patrolling problems, since the drone (the agent) can take several actions (e.g. moving forward, zooming in etc.) that will impact its visual coverage of the environment. It must be noted that trivial solutions, such as choosing the action that will immediately maximise the coverage, are not suited for problems with uneven coverage requirements. In this case, the best short-term action does not necessarily lead to a good long-term solution, e.g. when the drone must necessarily cross a low-priority area in order to reach an high-priority zone. This justifies the use of reinforcement learning as a technique to find the optimal patrolling strategy.

Formally, reinforcement learning models the problem as a Markov decision process $\mathcal{P} = \{S, A, \tau, r, \gamma\}$, where S is a finite set of agent states and A is a finite set of actions. $\tau(s'|s,a)$ is the transition probability from state s to state s' given that action a is executed, $r(s,a)$ is the reward obtained by executing action a in state s and γ is a discount factor, modelling the importance of immediate, short term rewards with respect to past rewards. A fundamental property of Markov decision processes is that the transition probability is defined only in terms of s, s' and a, thus meaning that the next state will be affected only by the current state and action, and not by the history of previous states. A *policy* is a probability function $\pi(a|s)$ denoting the probability for an agent in state s to take action a, and the goal of reinforcement learning is to find the optimal policy π^* that maximises the expected total discounted reward

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{\tau,\pi}\left[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)\right] \tag{1}$$

where $\mathbb{E}_{\tau,\pi}$ means that the expected value is computed assuming that the states sequence is distributed according to τ and the actions are chosen according to π. Several methods have been proposed to solve (1), one of the most popular being *Q-learning*.

3.1 Q-learning

The Q-learning algorithm is based on the definition of a function Q_π modelling the total discounted reward that can be achieved following the policy π if action a is chosen in state s

$$Q_\pi(s,a) = \mathbb{E}_{\tau,\pi,s_0=s,a_0=a}\left[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)\right]. \tag{2}$$

Equation (2) can be written recursively as

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s',a'} \tau(s'|s,a)\pi(a',s')Q_\pi(s',a') \tag{3}$$

and it can be simplified if the optimal policy π^* is considered. In fact, $\pi^*(a,s)$, because of its optimality, has a binary nature: it has value 1 for the best action possible and 0 for any other action, thus the Q^* function for the optimal policy reduces to

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} \tau(s'|s,a) \max_{a'} Q^*(s',a') \tag{4}$$

If we restrict our analysis to deterministic problems, in which also $\tau(s'|s,a)$ can assume only binary values, (4) further simplifies to the so-called Bellman equation [20]

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} Q^*(s',a') \tag{5}$$

where s' is the state reached from s by executing action a, in other words it is the only state such that $\tau(s'|s,a) = 1$.

The Q-learning algorithm computes Q^* using (5) and dynamic programming. It starts by filling a $|S| \times |A|$ table with initial random values $Q_0^*(s,a)$ for each possible combination of states and actions, and then updates them iteratively until convergence according to the following equation:

$$Q_{i+1}^*(s,a) = r(s,a) + \gamma \max_{a'} Q_i^*(s',a'). \tag{6}$$

In order to apply (6), Q-learning requires the knowledge of the reward $r(s,a)$. This knowledge comes from experience: in the training phase, the agent is requested to perform actions in order to measure the corresponding reward and accumulate enough data to update Q^* until convergence. In theory, the actions could be always chosen randomly, but in practice it is more effective to draw some of them from the partially-learned policy in order to evaluate the most promising parts of the state-action space. This approach is called exploration-exploitation strategy, where exploration refers to the random choice of actions to explore previously unseen state-action pairs and exploitation refers to exploiting the current estimate Q_i^* in order to choose the action that maximises it in the current state. A typical training starts with exploration only, and the exploitation becomes more and more frequent as the iterative process progresses.

Once Q^* is computed, finding the optimal policy is trivial, since the best action to be taken in state s is the one that maximises $Q^*(s, \cdot)$.

3.2 Deep Q-networks

The dynamic programming approach described in Section 3.1 requires the memorisation of a $|S| \times |A|$ table which is often

impractical, since $|S|$ could be large (or even infinite, if we extend the definition of Markov decision processes to infinite state spaces, e.g. with continuous rather than discrete values). Deep Q-Networks approaches [21, 22] solve the problem by using deep neural networks as function approximators for Q^*.

Let $Q^*(s;\theta)$ be a neural network with parameters θ, taking as input a state s and giving as output the values $Q^*(s;\theta)|_k, k \in \{1 \dots |A|\}$ representing the Q^* value for each possible action in state s. Then, when an experience tuple (s, a, r, s') is acquired, it can be used to train the network by minimising the following MSE loss function:

$$L(\theta) = \mathbb{E}\left[\left(Y - Q^*(s;\theta)|_a\right)^2\right] \tag{7}$$

representing the difference between the current estimate $Q^*(s;\theta)|_a$ of the value of action a in state s and the new estimate that can be computed from the experience, defined as

$$Y = r(s,a) + \gamma \max_k Q^*(s';\theta)|_k \,. \tag{8}$$

The computation of the loss function (7) can lead to bias issues in practical implementations. Neural networks are typically trained in small batches, and if the experience data of a batch come from the same experiment (e.g. subsequent steps of the same agent) they can lead to biased computation of the expected value. In order to avoid this problem, a *replay memory* can be used, which consists in a large set of experience tuples. During the training phase the tuples, rather than being directly used to train the network, are stored in the replay memory. Batches are then built by sampling the replay memory with uniform distribution, thus avoiding to build batches composed only of highly correlated tuples.

Another problem in the computation of the loss function comes from the network parameters θ which are used both for action selection in (8) and for action evaluation in (7). It has been proven that this could lead to biased results [23], which can be avoided by decoupling selection and evaluation using two different networks. Two of the most popular approaches are the Target Network approach [21], in which the new estimate is defined as

$$Y = r(s,a) + \gamma \max_k Q^*(s';\theta^-)|_k \tag{9}$$

and the Double DQN approach [23]:

$$Y = r(s,a) + \gamma Q(s';\theta^-)|_{\text{argmax}_k Q(s';\theta)|_k} \tag{10}$$

where θ^- are the parameters of a second deep Q-network. This second network, rather than being trained independently, is generally defined in terms of the first network, either via hard update (θ^- is set to θ at every fixed number of epochs) or via soft update at each epoch i, according to a temporal smoothness factor $\alpha \in [0, 1]$

$$\theta_i^- = (1-\alpha)\theta_{i-1}^- + \alpha\theta_i \,. \tag{11}$$

4 Single drone model

In order to apply reinforcement learning techniques to drone patrolling tasks, we must model the drone as a Markov decision process agent. This implies defining its possible states, the actions and a reward function giving a positive or negative feedback to each action.

4.1 State space

The state space should consider all the relevant parameters that define the drone setup at a given time instant. We identified six parameters that directly influence the visual coverage of a drone. Formally, a drone state is a tuple $s = \{x, y, z, \psi, \phi, f\}$ defined as

- x, y, z: spatial coordinates of the drone;
- ψ: camera orientation angle;
- ϕ: camera tilt angle;
- f: camera focal length.

The spatial coordinates x, y, z are referred to a world reference system, and are limited by the borders of the area to be monitored and by the maximum flying height the drone can reach. The camera orientation angle $\psi \in [0, 2\pi]$ is the camera azimuth, expressed as the angle between the camera frontal axis and the X-axis of the world reference system. The camera tilt angle $\phi \in [0, \pi/2]$ describes the elevation of the camera, where $\phi = 0$ is the camera pointed at the horizon and $\phi = \pi/2$ is a nadiral view. Finally, the focal length f is included in the state to model zoom cameras, and its range is hardware dependent.

Observe that we did not include the drone azimuth in the state space, as this information is not relevant. The proposed work is focused on multicopter drones, which can move freely in the three spatial dimensions (as opposed to fixed-wing drones), thus identifying a frontal axis is unnecessary: in the proposed framework, a drone aiming north and moving frontward is equivalent to a drone aiming east and moving leftward.

4.2 Action space

As done for the state space, we identified a set of drone actions that directly influence the visual coverage of the drone. The action space consists of a total of 12 actions:

- Move {Forward | Backward | Left | Right | Up | Down };
- Rotate {Left | Right};
- Tilt {Down | Up};
- Zoom {In | Out}.

The *Move* actions translate the drone in the 3D space and influence the $\{x, y, z\}$ components of the drone state. Without loss of generality, the front direction is assumed to be the camera orientation angle: as described in Section 4.1, there is no need to decouple drone and camera azimuths. The camera orientation is defined by the *Rotate* and *Tilt* actions, which, respectively, influence the $\{\psi, \phi\}$ parameters. Finally, the *Zoom* actions change the focal length f and thus the zoom level of the camera.

Note that the proposed actions do not quantify the amount of requested change, e.g. how much the drone should move when a MoveForward action is executed. Ideally, those values should be continuous, and the action space would be infinite. However, the reinforcement learning techniques described in Section 3 require a finite action space, and thus the actions must be discretised. The amount of parameter change caused by each action is thus fixed and defined a priori. For example, the experimental results discussed in Section 6 have been obtained with a *Rotate* step of $\pi/16$, meaning that the camera performs a full 2π rotation after 32 *Rotate* actions in the same direction.

4.3 Visual coverage

In order to evaluate the visual coverage quality of a drone, we need a way to compute the portion of the environment observed by the camera. Let us consider a reference system as the one depicted in Fig. 1, where the origin lies on the centre of the camera optics, the Y-axis points upwards and the Z-axis is initially aligned with the camera optical axis when the camera points at the horizon with no rotation. The effect of actions *Rotate {Left | Right}* can be modelled by a rotation matrix $\boldsymbol{R}_\psi$ around the Y-axis

$$\boldsymbol{R}_\psi = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix} \tag{12}$$

while the actions *Tilt { Up | Down}* are modelled by a rotation around the X-axis

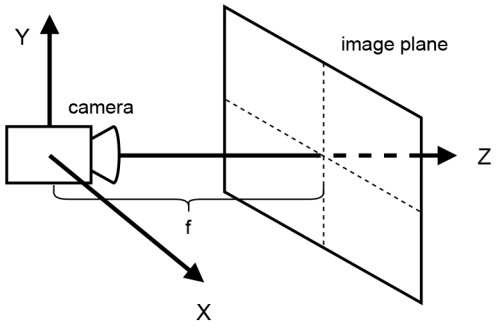

Fig. 1 *Camera reference system*

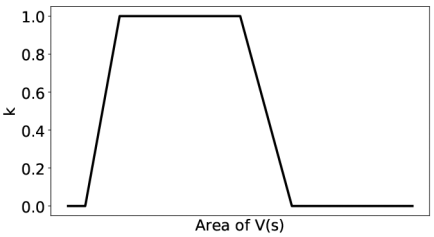

Fig. 2 *Penalty function to penalise large visual coverage*

$$\boldsymbol{R}_{\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}. \tag{13}$$

Any camera configuration can thus be modelled by a combination $\boldsymbol{R}_{\psi\phi}$ of the two matrices, observing that the tilt rotation must be applied first, in order to correctly model the motion of a gimbal camera

$$\boldsymbol{R}_{\psi\phi} = \boldsymbol{R}_{\psi}\boldsymbol{R}_{\phi} = \begin{bmatrix} \cos\psi & \sin\psi\sin\phi & \sin\psi\cos\phi \\ 0 & \cos\phi & \sin\phi \\ -\sin\psi & \sin\phi\cos\psi & \cos\phi\cos\psi \end{bmatrix}. \tag{14}$$

By using the matrix (14), it is possible to project on the ground plane any point on the image plane, given the camera rotation and tilt angle. Let $\boldsymbol{p} = (x, y)$ be a pixel in the image plane. According to the camera pinhole model, the focal length f is the distance between the optics and the image plane itself, and thus the coordinates of the pixel in the camera reference frame are $\hat{\boldsymbol{p}} = (\delta_x x, \delta_y y, f)$, where δ_x, δ_y are the pixel sizes of the imaging sensor. If the camera is rotated and tilted, point $\hat{\boldsymbol{p}}$ is moved to $\hat{\boldsymbol{p}}_{\psi\phi} = \boldsymbol{R}_{\psi\phi}\hat{\boldsymbol{p}}$. By switching to a world reference system, where the camera has coordinates $\boldsymbol{C} = (C_x, C_y, C_z)$, the parametric equation of a line parallel to vector $\hat{\boldsymbol{p}}_{\psi\phi}$ and passing through $\boldsymbol{C}$ is

$$L(t) = \boldsymbol{C} + t\hat{\boldsymbol{p}}_{\psi\phi}. \tag{15}$$

To find the intersection of L with the ground plane, it is sufficient to set the Y world coordinates to zero

$$\begin{aligned} C_y + t\hat{\boldsymbol{p}}_{\psi\phi,y} &= 0 \\ t &= -C_y/\hat{p}_{\psi\phi,y} \end{aligned} \tag{16}$$

and by substituting (16) into (15) we get the projection p_g of point p on the ground plane

$$\boldsymbol{p}_g = \boldsymbol{C} - \frac{C_y}{\hat{p}_{\psi\phi,y}}\hat{\boldsymbol{p}}_{\psi\phi}. \tag{17}$$

By using (17), it is possible to project on the ground plane any point $\boldsymbol{p}$ in the image plane given the camera angles ψ, ϕ. If the equation is applied to the four corners of the image plane, it is thus possible to compute the corners of the trapezoidal projection of the image plane on the ground plane, which is the portion of the environment observed by the camera (we here assume that no point lies above the horizon). We define this zone the *visual coverage* of the drone in state s, for now on denoted as $V(s)$.

4.4 Reward function

As stated in Section 1, this works deals with the case of uneven coverage requirements. This means that not all the portions of the environment have the same priority, and some areas are more important than others and require immediate visual inspection. In [24], the authors used this approach to focus a surveillance system on the areas with highest activity, while in [25] the high priority areas were identified by audio sensors. In general, the definition of these areas is extremely context-dependent, and here we just assume that such a definition exists in the form of a *relevance map*. A relevance map $\mathscr{M}(x, y): \mathbb{R}^2 \to [0, 1]$ is thus a function taking as input the (x, y) coordinates of a point in the world reference system and returning a value in the range $[0, 1]$ denoting the relevance of that point, i.e. the relative importance of getting visual coverage of that point with respect to the rest of the map.

It is now possible to define the *observed relevance* $\rho(s)$ of a drone in state s as the total relevance within its visual coverage

$$\rho(s) = \int\int_{V(s)} \mathscr{M}(x, y)\, \mathrm{d}A \tag{18}$$

or, in the likely case that $\mathscr{M}$ is discretised in a matrix

$$\rho(s) = \sum_{(x,y) \in V(s)} \mathscr{M}(x, y) \tag{19}$$

where $V(s)$ is the visual coverage of the drone, as defined in Section 4.3.

The drone reward function could thus be defined in terms of its observed relevance, giving a positive reward to actions that increase $\rho(s)$. However, additional constraints are needed in order to avoid extreme cases, such as the drone flying at the maximum possible height trying to cover the entire area. Despite in small areas this could be a viable solution, in larger environments it is most probably a useless configuration from a practical point of view because of the very low spatial resolution (pixel per meter) of the acquired images. We thus enforce a constraint on the size of the visual coverage by defining a penalty function $k: \mathbb{R} \to [0, 1]$ such as the one shown in Fig. 2 that penalises coverages which are either too small or too large for practical applications; the shape and tuning of k is of course application dependent. We thus define the *constrained observed relevance* (COR) as

$$\hat{\rho}(s) = k(V(s))\rho(s). \tag{20}$$

The drone reward function $r(s, a)$ can now be defined as

$$r(s, a) = \hat{\rho}(s') - \hat{\rho}(s) \tag{21}$$

that is, if the execution action a in state s leads to state s', the corresponding reward is defined as the difference of COR between the two states s' and s. By using such a function, the reinforcement learning system is positively rewarded each time the agent chooses actions that lead to an increase of observed relevance. Since reinforcement learning algorithms maximise the total reward on the long term, this means that the drone will try to cover high-relevance areas, even if this requires to move through low-relevance zones in the short term.

4.5 Patrolling

Training a reinforcement learning network with the reward function described in Section 4.4 is not sufficient to get a patrolling algorithm. The reward function just forces the drone to move to more relevant areas in order to increase the total reward, and the process will stop once the highest possible value has been reached: the algorithm is actually a state-space explorer trying to find a path to the global maximum of the COR function.

In order to get a sensible patrolling behaviour, the temporal aspect must be introduced. More specifically, we assume that an area that is under observation by the drone should have its relevance reduced, in order to give the drone the opportunity to move to other still unexplored zones. At the same time, it is reasonable to require that, given two areas with the same relevance, higher priority should be given to the one that has been unobserved for the longest time.

In order to model the temporal aspect, we introduce the *temporal relevance map*

$$\mathscr{M}_s^t(x,y) = \mathscr{M}(x,y)T_s^t(x,y) \tag{22}$$

defined as a combination of the static map $\mathscr{M}$ and a temporal mask $T_s^t: \mathbb{R}^2 \to [0,1]$. The temporal mask at time instant t for a drone in state s is defined as

$$T_s^t(x,y) = \begin{cases} 1 & \text{for } t = 0 \\ \min\left(1, T_s^{t-1}(x,y) + \delta_+\right) & \text{for } t > 0 \wedge (x,y) \notin V(s) \\ \max\left(0, T_s^{t-1}(x,y) - k(V(s))\delta_-\right) & \text{for } t > 0 \wedge (x,y) \in V(s) \end{cases} \tag{23}$$

where k is the penalty function defined in Section 4.4, $V(s)$ is the visual coverage of the drone in state s defined in Section 4.3 and δ_-, δ_+ are constant decreasing and increasing factors. The decreasing factor is multiplied by the penalty function $k(V(s))$ because we do not want to model the contribution of drones that do not satisfy the area constraints, as already done in the definition of the COR (20).

With this definition, the temporal relevance map $\mathscr{M}_s^t$ dynamically changes depending on the time since last observation for each point of the map. Applying the reinforcement learning algorithm on this map forces the drone to continuously move around the area searching for high-relevance areas that have not been observed since a long time, thus implementing an efficient patrolling algorithm.

Finally, observe that the temporal map should now be part of the state, since it is dependent on the chosen actions rather than being static. The reward function (21) should now be rewritten as

$$r(\{s, \mathscr{M}_s^t\}, a) = \hat{\rho}(\{s', \mathscr{M}_{s'}^{t+1}\}) - \hat{\rho}(\{s, \mathscr{M}_s^t\}) \tag{24}$$

and the definitions of $\hat{\rho}, \rho$ are consequently adapted so that the summation in (19) is performed over $\mathscr{M}_s^t$ rather than $\mathscr{M}$.

5 Swarm model

The model proposed in Section 4 describes how a single drone can be modelled as a Markov decision process agent, and its patrolling strategy can be defined using reinforcement learning and a proper reward function. However, the model can be extended to a swarm of drones, where multiple drones share the task of patrolling a given area trying to maximise the overall visual coverage of relevant areas. We here propose two approaches, the *greedy* strategy and the *cooperative* one.

5.1 Greedy strategy

The greedy strategy consists in applying the single-drone model to each drone of the swarm. This way, every agent will try to maximise its own total reward in a greedy way; this is independent from other swarm members. However, a naive application of this strategy would eventually lead all the drones to cover the same areas of the map, namely the ones with highest relevance. In order to turn the greedy strategy into a sensible patrolling algorithm, it is sufficient to request that the temporal relevance map is shared among all the swarm. With map sharing, each drone will naturally avoid the areas already observed by other members of the swarm because the temporal relevance of those areas will be decreasing due to coverage. The approach is still greedy since each drone does not explicitly know about the presence of other drones in the area, however this knowledge is indirectly modelled by the shared map.

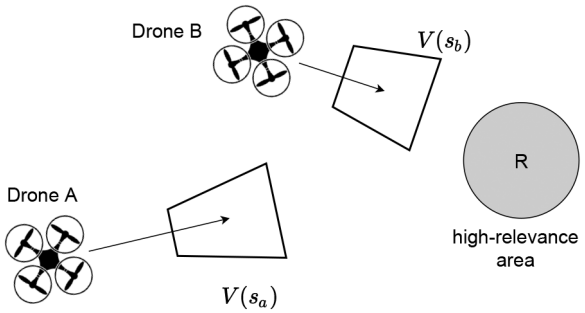

Fig. 3 *Two competing drones for the same high-relevance area*

The shared temporal relevance map for a swarm of N drones with states $\boldsymbol{S} = \{s_i\}_{i=1}^N$ is defined as

$$\mathscr{M}_{\boldsymbol{S}}^t(x,y) = \mathscr{M}(x,y)T_{\boldsymbol{S}}^t(x,y) \tag{25}$$

where the temporal mask follows the same principle of (23), but extended to all the drones in the swarm

$$T_{\boldsymbol{S}}^t(x,y) = \begin{cases} 1 & \text{for } t = 0 \\ \min\left(1, T_{\boldsymbol{S}}^{t-1}(x,y) + \delta_+\right) & \text{for } t > 0 \wedge (x,y) \notin \bigcup_{s \in \boldsymbol{S}} V(s) \\ \max\left(0, T_{\boldsymbol{S}}^{t-1}(x,y) - \sum_{s \mid (x,y) \in V(s)} k(V(s))\delta_-\right) & \text{for } t > 0 \wedge (x,y) \in \bigcup_{s \in \boldsymbol{S}} V(s) \end{cases} \tag{26}$$

This way, the temporal mask increases if the point (x,y) is not observed by any drone, but it is decreased multiple times if it is observed by several drones, to encourage the multiple coverage of zones where the relevance is particularly high. The reward function (24) is consequently redefined as

$$r(\{s, \mathscr{M}_{\boldsymbol{S}}^t\}, a) = \hat{\rho}(\{s', \mathscr{M}_{\boldsymbol{S}'}^{t+1}\}) - \hat{\rho}(\{s, \mathscr{M}_{\boldsymbol{S}}^t\}) \tag{27}$$

The computation of the shared map can be done by a central processing node and sent at each time interval to all the swarm. This is particularly suitable if the entire computation is done offline: in this case the algorithm is just used to pre-compute a patrolling strategy which is subsequently executed. If the algorithm must be applied online, a distributed approach would be preferred; this case is covered in the next section.

5.2 Cooperative strategy

In the case of cooperative strategy, each drone is aware of the rest of the swarm and their states, so that collaborative strategies can be implemented directly, rather than indirectly via the shared map as in the greedy strategy. However, rather than explicitly defining the cooperative models, we rely on reinforcement learning to automatically learn them from experience.

The idea is to use $\boldsymbol{S} = \{s_i\}_{i=1}^N$, the set of all the states of the drones, as a new global state. The implementation is straightforward, since the reward function for agent in state s and taking action a leading to state s' is defined as

$$r(\{\boldsymbol{S}, \mathscr{M}_{\boldsymbol{S}}^t\}, a) = \hat{\rho}(\{s', \mathscr{M}_{\boldsymbol{S}'}^{t+1}\}) - \hat{\rho}(\{s, \mathscr{M}_{\boldsymbol{S}}^t\}) \tag{28}$$

However, this is the same as (27), except for $\boldsymbol{S}$ being made explicit in the function input. The difference between the cooperative and greedy strategy in fact does not lie in the reward function, but in the way rewards are related to states, and this is what

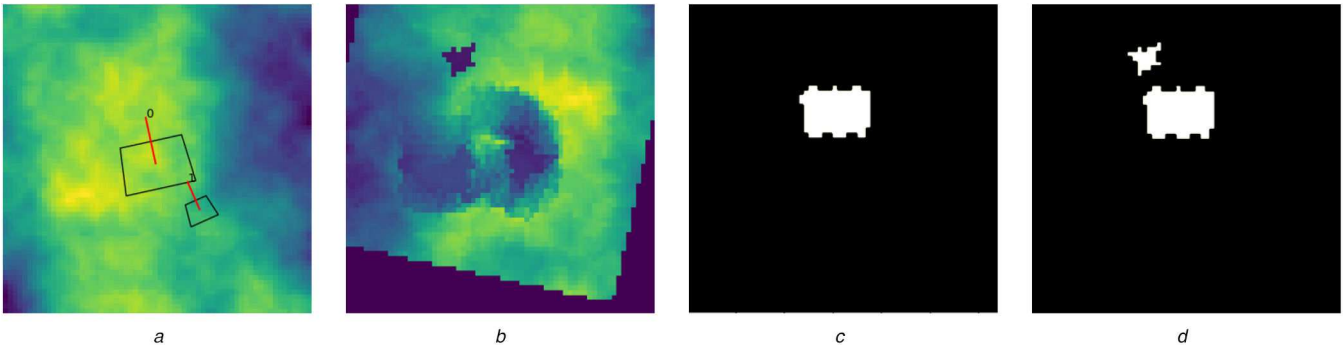

Fig. 4 *Network input example*

(a) Randomly-generated relevance map and two drones with their visual coverage, ***(b)*** Shared temporal relevance map, centred and rotated on drone 0, ***(c)*** Visual coverage of drone 0, ***(d)*** Visual coverage of all drones, as seen from drone 0

Network input for the greedy strategy is composed of (b) and (c) stacked; network input for the cooperative strategy also adds (d) to the stack

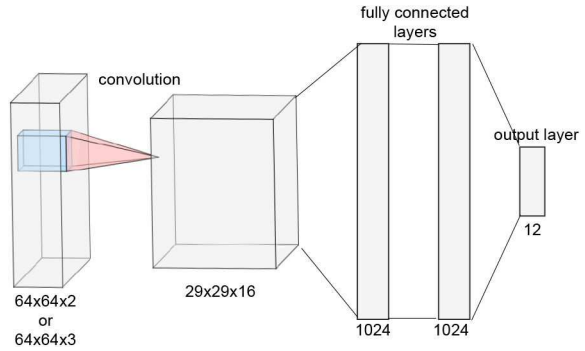

Fig. 5 *Proposed network structure*

reinforcement learning tries to learn. The Q-function (see Section 3.1) learned by the greedy approach is $Q(\{s, \mathscr{M}^t_S\}, a)$ while the one learned by the cooperative approach is $Q(\{\boldsymbol{S}, \mathscr{M}^t_S\}, a)$. In order to clarify the difference, consider the example of Fig. 3, showing two drones A and B, their visual coverages $V(s_a), V(s_b)$ and an high-relevance area R. With the greedy strategy, both drones will move towards R, however B will reach it first since it is closer. Once B covers R, the shared temporal relevance will decrease because of temporal update, and A will probably switch to another, more relevant target. With the cooperative approach instead, by explicitly knowing the state of B, drone A could predict that R will be covered by the closer drone and immediately switch to another target. In other words, despite the reward function not changing between the two algorithms, the greedy approach relies on *observing* changes in the temporal map $\mathscr{M}^t_S$, while the cooperative approach could *predict* them thanks to the explicit knowledge of $\boldsymbol{S}$. Of course the example shows that this form of reasoning is possible, but since we do not model any explicit cooperative strategy, we need to prove with experimental results that reinforcement learning can automatically infer this kind of strategies from experience.

Regarding practical implementations, each drone needs to know the state of the entire swarm at each time interval. This could be achieved by using a central processing node, as already discussed in Section 5.1, and the knowledge of $\boldsymbol{S}$ can also be used by each drone to independently reconstruct the entire map $\mathscr{M}^t_S$. A distributed approach could be implemented by drones exchanging state vectors each other when they are within communication range, thus propagating the state information across the entire swarm network, however this solution is not detailed here.

5.3 Deep learning implementation

In order to implement the greedy and cooperative techniques, we adopted the Deep Q-Network approach described in Section 3.2 with a linearly decreasing ratio between exploration and exploitation steps. In order to avoid training biases, the Double DQN model has been used (10) with replay memory and soft update of the second network.

For the greedy strategy, the network input is $\{s, \mathscr{M}^t_S\}$, this is the drone state and the current shared temporal relevance map, and the output is a $\mathbb{R}^{12}$ vector with the estimated $Q(\{s, \mathscr{M}^t_S\}, a)$ values for each possible action a. However, rather than modelling the state s as a tuple, we propose to represent it visually, with a binary image showing the visual coverage $V(s)$ of the drone. This way, the information about the observed area is immediately available from the input data and does not have to be estimated from the agent state tuple, thus achieving a simpler model and a faster convergence rate. In order to simplify the input, the two images are centred on the drone position, rotated by the drone orientation angle, and cropped to a fixed size. This way, the network input represents the surrounding of the drone, aligned with its orientation. The two images are finally stacked to form a $64 \times 64 \times 2$ input tensor (Fig. 4).

For the cooperative strategy, we adopted a similar representation to model the network input $\{\boldsymbol{S}, \mathscr{M}^t_S\}$. A new binary image is created, representing the visual coverage of all the drones, and it is added to the input with the same centring, rotation and cropping procedures described above, in order to create a $64 \times 64 \times 3$ input tensor. The choice of keeping the representations of $V(s)$ and $V(\boldsymbol{S})$ separately (second and third elements in the input stack, respectively) rather than relying on $V(\boldsymbol{S})$ alone (which contains $V(s)$) is used to feed the network with an explicit information to discriminate between the drone on which the network is being executed and the remaining ones. The final input for cooperative mode is shown in Fig. 4.

The rest of the network is the same for the two approaches, and consists of a single convolutional layer, composed of 16 8×8 filters with stride = 2. The convolutional layer is followed by two 1024×1 fully connected layers and a 12×1 output layer representing the Q values for all the possible drone actions. All the layers use ReLU activation functions, except for the output layer, where the activation function is linear. The network topology is shown in Fig. 5.

The training is done by creating a random relevance map and a random number of drones, with random initial states, at each epoch. The system then evolves for 20 steps (i.e. 20 actions for each drone), either by choosing random actions or by using the partially trained network (exploration/exploitation). During this procedure, old and new states, actions and rewards are stored in the replay memory. After 20 steps, a random batch of samples is extracted from the replay memory and used to compute the loss function and train the network. The process is repeated for every epoch.

6 Experimental results

The proposed model has been implemented and tested in Python using Tensorflow and the Keras library. As described in Section 5.3, the adopted model is a Double Deep Q-Network with soft update of the second network and replay memory. Each epoch

Table 1 Deep Q-Network training hyperparameters

Hyperparameter	Value
discount factor γ (1)	0.99
replay memory size	10^4
batch size	64
optimiser	Adam
Adam learning rate	10^{-4}
epochs (greedy)	10^5
epochs (coop)	3×10^5
soft update α (11)	0.001

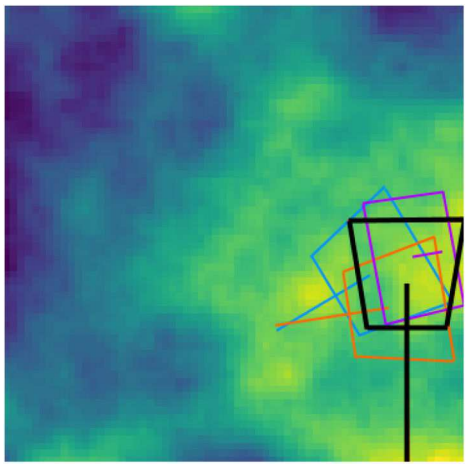

Fig. 6 *Optimal configuration found by brute force (black) and some approximations computed by the proposed method (other colours). For each configuration the visual coverage (trapezoidal area) and the line connecting the drone with the projection of the image centre are shown*

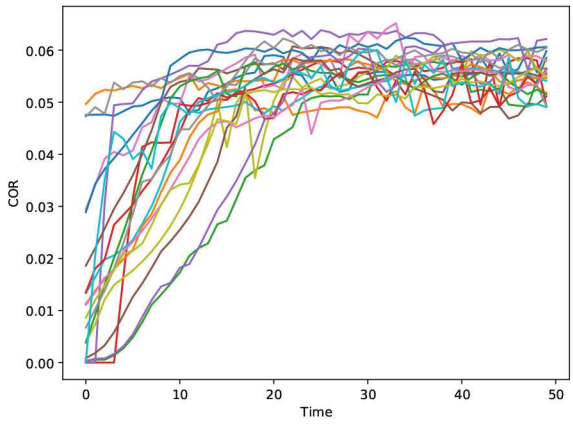

Fig. 7 *COR values of 20 drones running on the same map for 50 steps (no temporal update of the map)*

consists in the creation of a random relevance map and the instantiation of a random number of drones, between 1 and 5, with random valid initial states. The system evolves for 20 steps, during which each done collects the tuples $(s, s', a, r(s, a))$ and stores them in the replay memory. The actions are chosen according to the exploration/exploitation strategy, starting with 100% exploration and linearly decreasing it until 10% exploration is used at the end of the training. At the end of each epoch, a batch set of tuples is extracted from the replay memory with uniform probability distribution and it is used for training. The full set of training hyperparameters is shown in Table 1. Training and all the experiments have been computed on a dual Xeon E5-2660 CPU, 224 GB RAM, 1 Tesla K40 and 2 Titan XP GPUs.

6.1 Convergence to high-relevance zones

We started the evaluation process by checking if the proposed approach can really control a drone so that it can always converge to high-relevance zones. In order to do the test, we relied on the greedy network and used it in a single-drone configuration with the temporal map update disabled. This way, the expected behaviour is that the drones will converge towards the zone with highest relevance and, once reached, they will keep patrolling the same zone.

In order to have a reference measure, as a first experiment we generated a random relevance map (Fig. 6) and searched for its optimal COR (20) using a brute-force, exhaustive search. Since each one of the six state parameters has been discretised in 32 values, the total amount of possible states is 32^6 and its exhaustive search would require ∼280 h if performed on the adopted hardware. In order to make the problem tractable, we halved the number of discretisation steps of each state parameter, thus finding the brute-force optimal COR = 0.079 in ∼2.8 h.

We then executed the proposed method 100 times on the same map, with random initial conditions and measured the final COR to check if it was similar to the brute force one. On average, the final COR was 0.075, thus 95% of the best possible result. The proposed method can thus find drone configurations with a coverage that is close to the best possible one, although in a fraction of time: on average, each run found the optimal COR in 0.61 s. Fig. 6 shows the best result as well as some of the approximated solutions.

It is also interesting to measure the efficiency of the proposed method in finding the best solution efficiently, this is in a small number of actions. We thus measured the number of actions required by each run in order to find the final solution, and divided it by the minimum number of steps possible, which is easily computed once the initial and ending states are known. The average ratio is 1.54, thus we can expect the proposed method to reach the optimal configuration in ∼50% more steps than the best possible path in the state space.

The proposed experiment however used a single relevance map, and it is not suited for computing robust results, since each brute force computation is extremely time consuming. We thus developed another test that can be more easily applied on several maps. On each map, we ran 20 agents for 50 time intervals and measured the COR of each drone. The starting state of each drone is chosen randomly. If the proposed algorithm works, we expect all the COR values to converge to a similar final score, meaning that every drone has reached an approximation of the global maximum. Fig. 7 shows this behaviour, where it is clearly seen that after roughly 20 steps all the drones reached stable and similar COR values. Small fluctuations still exist, since our model do not explicitly consider the 'do nothing' action (to avoid getting stuck) and thus the drone keeps moving around the found optimal area.

In order to measure this behaviour numerically, we analysed the final COR values by normalising them so that their mean is set to 1 and by computing their standard deviation σ. The normalisation step is required to get comparable results between different maps. We repeated the test on 50 different relevance maps, and the results are shown in Table 2. As it can be seen, the final standard deviation is relatively small, with an average value of $\sigma = 0.13$. This means that the majority of the final COR values lie within their mean ± 13%, thus proving the capability of the system to reach high-relevance areas.

6.2 Single drone patrolling

As mentioned in Section 4.5, patrolling behaviour is achieved by enabling the temporal update of the relevance map. With temporal update, observed areas gradually decrease their relevance, thus making more convenient to move to a different state in search for higher rewards. With patrolling enabled, a drone should thus avoid static behaviours in which it keeps monitoring always the same zone. Fig. 8*a* shows the path of a drone image centre, projected on the ground plane, with patrolling enabled. As it can be seen, the non-static behaviour is evident.

Table 2 Standard deviations of 20 normalised COR values on 40 different relevance maps

Test no.	σ	Test no.	σ	Test no.	σ	Test no.	σ
0	0.0547	10	0.1074	20	0.0749	30	0.0978
1	0.2317	11	0.2127	21	0.1111	31	0.0721
2	0.1001	12	0.0888	22	0.1852	32	0.1821
3	0.1335	13	0.1210	23	0.1509	33	0.0705
4	0.1966	14	0.1501	24	0.1745	34	0.1955
5	0.1049	15	0.0753	25	0.1471	35	0.1264
6	0.0909	16	0.1074	26	0.0933	36	0.1667
7	0.1646	17	0.0975	27	0.1346	37	0.1798
8	0.0729	18	0.1966	28	0.1887	38	0.1800
9	0.0658	19	0.1106	29	0.2126	39	0.0795

In order to measure the overall quality of the patrolling path, we define a global measure of patrolling quality, called *global coverage* G with values in [0, 1], as

$$G^t = 1 - \sum_{(x,y)} \frac{\mathscr{M}_s^t(x,y)}{\mathscr{M}_s^0(x,y)}. \tag{29}$$

The summation part of (29) is the ratio between the current relevance of the entire map at time t and the initial total relevance (i.e. the relevance map without temporal updates). The ratio thus has a maximum value of 1, and lower values denote a good patrolling, since it means that the drone covered high-relevance areas making their temporal relevance decrease. G^t is then defined in order to have higher scores for good patrolling patterns. Fig. 8*b* shows the global coverage score for the experiment shown in Fig. 8*a*. After 200 iterations, G^t reaches a stable value around 0.15, meaning that the total temporal relevance for $t > 200$ is ∼85% of the original relevance. This measure alone is not meaningful, since it depends on many factors such as the size of the map, the speed of temporal updates and so on, however it is useful for comparative results.

The proposed system is thus compared with a standard, naive zigzag patrolling scheme. In this case, the drone follows a predefined path, disregarding the relevance values, uniformly spanning all the environment following a path like the one shown in Fig. 9, alternating vertical and horizontal scans. Fig. 10 shows the global score over the same relevance map of both the proposed approach and the zigzag pattern. As it can be seen, the proposed method outperforms the naive strategy, with an average $G^t = 0.184$, more than 50% better than the average $G^t = 0.119$ obtained by the naive patrolling. Table 3 shows the results of 50 tests on different maps, reporting the average G^t for both methods and the consequent performance boost. The behaviour shown in Fig. 10 is confirmed, and the average achieved performance boost of the proposed method is 41.95%.

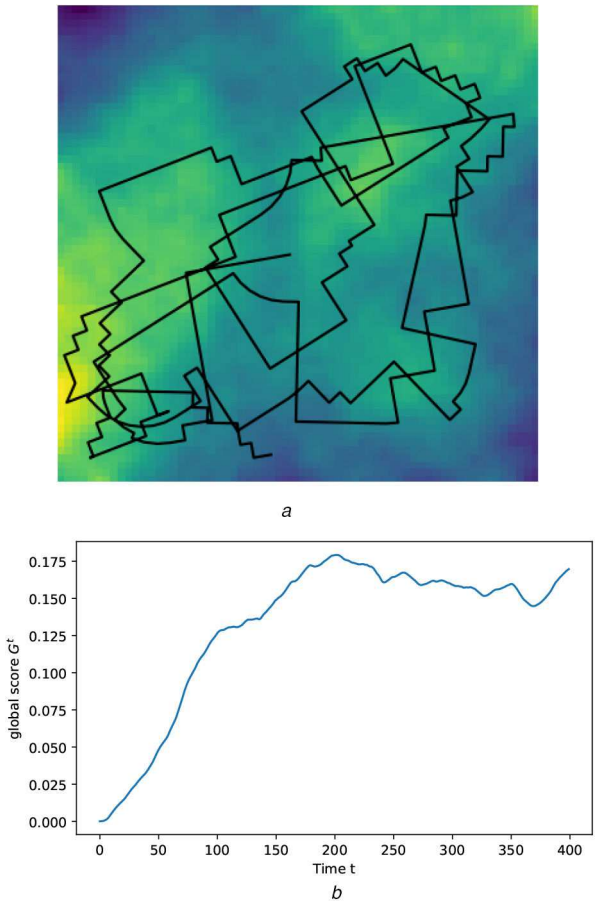

Fig. 8 *Example of single drone patrolling*
(a) Temporal evolution of the image centre of a drone, projected on the ground plane, while in patrolling mode; ***(b)*** Corresponding global coverage G^t

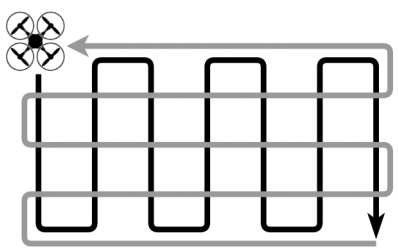

Fig. 9 *Naive patrolling pattern. After a sequence of vertical (black) and horizontal (grey) scans, the drone returns to its starting position and repeats the same pattern*

6.3 Swarm patrolling

In Section 5, two swarm patrolling models have been proposed, a greedy approach and a cooperative one. The greedy approach consists of each drone acting independently from the others, since the only collaboration happens indirectly by sharing the same temporal relevance map. The cooperative one instead uses an explicit representation of the states of any other drone that can be used to explain and predict temporal map changes and thus plan a better patrolling strategy. Since this predictive feature has not been explicitly coded, we rely on reinforcement learning to learn it from experience, and thus comparative results are required to show if a performance gap between the two approaches actually exists.

The global coverage (29) can be used again as a quality metric of the computed patrolling paths. Fig. 11 shows the global coverages of the greedy and cooperative approaches on the same map, with four drones starting from the same initial states. The cooperative approach indeed seems to perform better than the basic greedy approach, with an average global coverage of 0.505 and 0.355, respectively.

In order to confirm this result, more experiments have been conducted, as shown in Table 4. In this case, we considered a swarm of 2, 3, 4 and 5 drones, respectively. For each case, we ran 20 tests on random relevance maps. In each test, both the greedy

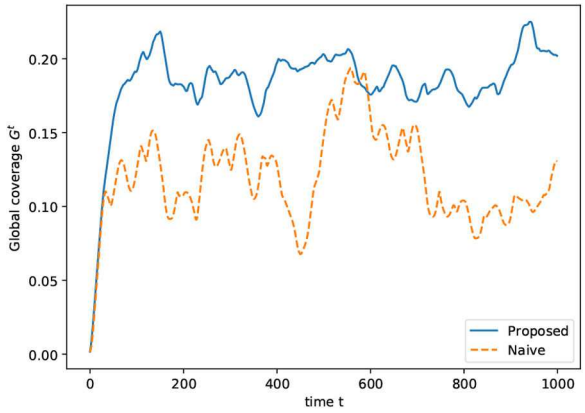

Fig. 10 *Global coverage scores G^t for the proposed and zigzag patrolling schemes (single drone)*

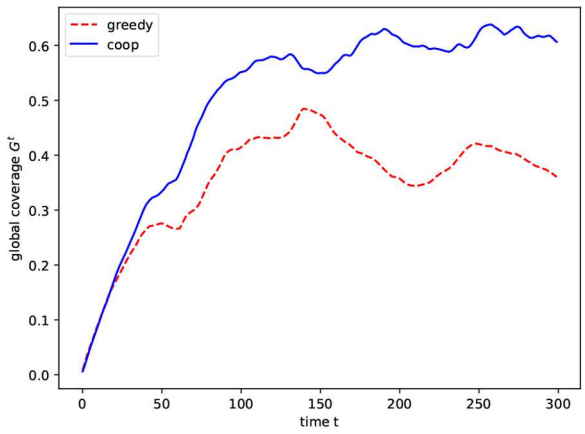

Fig. 11 *Global coverage scores G^t of the greedy and cooperative swarm models with four drones, on the same map and same initial drone states*

and cooperative methods were applied for 500 steps before measuring the average global coverage. The table reports the results of the two approaches, as well as the boost improvement of the cooperative approach versus the greedy one. As it can be seen, in case of multiple drones the cooperative approach always outperforms the greedy one by a factor of 40–50% on average.

In the end, the only advantage of the greedy approach is its faster training time. The cooperative approach, in order to process the additional input properly, required three times the number of epochs of the greedy one (see Table 1). The greedy network however is still useful for single-drone scenarios.

6.4 Computational load

Since the proposed theoretical framework is currently implemented as a Python simulation, we cannot measure the timings of a full sense-act cycle. However, we can measure the computational time required to choose an action given a state input, in order to understand if a practical implementation is actually feasible. Choosing an action requires to run the input through the neural network. This operation in our experiments consistently requires 1.5 ms on a CPU-only implementation running on an Intel(R) Core(TM) i7-9700 K 3.60 GHz processor. We believe this time is negligible if compared to the time required to actually execute the chosen action.

7 Conclusions

In this work, we proposed a theoretical model to compute efficient patrolling paths for single drones and swarms of drones. The problem of uneven coverage requirements was explicitly considered: the environment zones are associated to different priority (relevance) scores, expressing the importance for an area to be visually covered by a drone. The proposed implementation uses reinforcement learning to train a deep network that selects the best action that will most likely lead to a good coverage in the long run. The proposed system was extensively tested and showed good performances, also compared to a standard patrolling scheme.

Table 3 Comparison of the average global coverage of the proposed method and naive zigzag patrolling schemes on 50 different maps

Test no.	Ours	Zigzag	Boost%	Test no.	Ours	Zigzag	Boost%
1	0.1836	0.1161	58.16	26	0.1546	0.1192	29.66
2	0.1671	0.1162	43.73	27	0.1581	0.1192	32.67
3	0.1421	0.1176	20.89	28	0.1649	0.1250	31.92
4	0.1800	0.1180	52.56	29	0.1648	0.1213	35.81
5	0.1788	0.1258	42.08	30	0.1557	0.1284	21.28
6	0.1551	0.1229	26.17	31	0.1701	0.1219	39.51
7	0.1650	0.1164	41.71	32	0.1481	0.1177	25.77
8	0.2005	0.1146	74.97	33	0.1870	0.1213	54.24
9	0.1555	0.1132	37.39	34	0.1679	0.1235	35.99
10	0.2203	0.1137	93.88	35	0.1473	0.1173	25.61
11	0.1580	0.1162	35.92	36	0.1675	0.1263	32.65
12	0.1832	0.1191	53.87	37	0.1790	0.1162	54.06
13	0.1697	0.1220	39.08	38	0.1571	0.1199	31.06
14	0.1315	0.1230	6.86	39	0.1693	0.1203	40.68
15	0.1759	0.1188	48.09	40	0.1563	0.1251	24.97
16	0.1715	0.1251	37.03	41	0.1492	0.1163	28.35
17	0.1545	0.1119	38.04	42	0.1685	0.1236	36.36
18	0.1817	0.1213	49.80	43	0.1594	0.1190	33.91
19	0.1566	0.1176	33.13	44	0.1861	0.1156	61.01
10	0.2098	0.1129	85.80	45	0.1749	0.1208	44.79
21	0.1671	0.1137	46.98	46	0.1592	0.1200	32.69
22	0.1705	0.1191	43.15	47	0.1659	0.1217	36.36
23	0.1565	0.1198	30.59	48	0.1973	0.1248	58.05
24	0.1877	0.1159	61.92	49	0.1868	0.1236	51.14
25	0.1853	0.1232	50.42	50	0.1735	0.1179	47.19

Each test lasts 500 steps. The boost% column shows the improvement of the proposed versus zigzag approach.

Table 4 Comparison of the average global coverage of the greedy and cooperative swarm patrolling schemes

Test no.	2 drones			3 drones			4 drones			5 drones		
	Coop	Greedy	Boost%	Coop	Greedy	Boost%	Coop	Greedy	Boost%	Coop	Greedy	Boost%
1	0.3097	0.2050	51.06	0.4028	0.2878	39.94	0.4459	0.3505	27.22	0.5241	0.3571	46.75
2	0.3111	0.2604	19.46	0.4877	0.2655	83.70	0.4940	0.3761	31.34	0.5767	0.3591	60.56
3	0.3079	0.1901	61.95	0.4488	0.3231	38.91	0.5569	0.3857	44.40	0.5617	0.3771	48.96
4	0.2714	0.2037	33.23	0.4524	0.2668	69.59	0.5137	0.3866	32.89	0.5363	0.4203	27.58
5	0.2854	0.2159	32.18	0.4453	0.3417	30.33	0.5236	0.3581	46.22	0.5491	0.3413	60.88
6	0.3407	0.1680	102.77	0.3911	0.2126	83.94	0.5276	0.3481	51.59	0.5294	0.3944	34.22
7	0.3389	0.2374	42.76	0.3278	0.2721	20.48	0.5553	0.3975	39.69	0.5608	0.3789	48.02
8	0.2268	0.1595	42.16	0.4549	0.3640	24.97	0.4761	0.2807	69.63	0.5823	0.4830	20.56
9	0.3039	0.2301	32.06	0.4484	0.3238	38.45	0.5392	0.3862	39.62	0.6006	0.3642	64.89
10	0.3200	0.1828	75.07	0.4072	0.2503	62.69	0.4843	0.3707	30.67	0.5946	0.3728	59.49
11	0.2859	0.1879	52.15	0.3909	0.2389	63.65	0.5410	0.3705	46.01	0.5232	0.3900	34.16
12	0.2210	0.1962	12.66	0.4443	0.3374	31.67	0.5035	0.3415	47.46	0.5566	0.3287	69.36
13	0.3431	0.2040	68.13	0.4229	0.3160	33.82	0.5300	0.3463	53.02	0.5823	0.4426	31.56
14	0.3128	0.2626	19.11	0.4398	0.3021	45.57	0.4804	0.3563	34.82	0.6036	0.4150	45.46
15	0.3099	0.2562	20.94	0.4323	0.2670	61.92	0.5219	0.3475	50.18	0.5862	0.4667	25.59
16	0.3353	0.2006	67.19	0.4381	0.2733	60.27	0.5184	0.3411	51.98	0.5707	0.3717	53.55
17	0.2623	0.1571	66.93	0.3967	0.2701	46.85	0.4918	0.3797	29.51	0.5870	0.4846	21.13
18	0.3047	0.1575	93.44	0.4562	0.3159	44.41	0.5076	0.2988	69.88	0.5806	0.4320	34.42
19	0.2933	0.2193	33.74	0.4581	0.3183	43.91	0.5227	0.3910	33.70	0.5618	0.3552	58.17
20	0.3011	0.1917	57.10	0.4704	0.2786	68.81	0.5088	0.3836	32.62	0.5473	0.3095	76.84
Avg.	0.2992	0.2043	49.20	0.4308	0.2912	49.69	0.5121	0.35982	43.12	0.5657	0.3922	46.10

Tests have been made with a swarm of 2, 3, 4 and 5 drones. Each test lasts 500 steps. The boost% column show the improvement of the cooperative versus greedy approach.

The proposed work can be a reference framework for real-world implementations, although in that case several additional constraints should be considered, such as different topological altitudes of the environment, presence of obstacles, battery life and so on.

8 Acknowledgment

This work is partially supported by the PNRM project 'Proactive Counter-UAV' (a2018.045).

9 References

[1] Tripicchio, P., Satler, M., Dabisias, G., *et al.*: 'Towards smart farming and sustainable agriculture with drones'. 2015 Int. Conf. on Intelligent Environments, Prague, Czech Republic, 2015, pp. 140–143

[2] Motlagh, N.H., Bagaa, M., Taleb, T.: 'UAV-based IOT platform: a crowd surveillance use case', *IEEE Commun. Mag.*, 2017, **55**, (2), pp. 128–134

[3] Erdelj, M., Natalizio, E., Chowdhury, K.R., *et al.*: 'Help from the sky: leveraging UAVs for disaster management', *IEEE Pervasive Comput.*, 2017, **16**, (1), pp. 24–32

[4] Silvagni, M., Tonoli, A., Zenerino, E., *et al.*: 'Multipurpose UAV for search and rescue operations in mountain avalanche events', *Geomatics Natural Hazards Risk*, 2017, **8**, (1), pp. 18–33

[5] Cruzan, M.B., Weinstein, B.G., Grasty, M.R., *et al.*: 'Small unmanned aerial vehicles (micro-UAVs, drones) in plant ecology', *Appl. Plant Sci.*, 2016, **4**, (9), p. 1600041

[6] Thiels, C.A., Aho, J.M., Zietlow, S.P., *et al.*: 'Use of unmanned aerial vehicles for medical product transport', *Air Med. J.*, 2015, **34**, (2), pp. 104–108

[7] Piciarelli, C., Foresti, G.L.: 'Drone patrolling with reinforcement learning'. Proc. of the 13th Int. Conf. on Distributed Smart Cameras, Trento, Italy, 2019

[8] Galceran, E., Carreras, M.: 'A survey on coverage path planning for robotics', *Robot. Auton. Syst.*, 2013, **61**, (12), pp. 1258–1276

[9] Cabreira, T.M., Brisolara, L.B., Ferreira, P.R.Jr.: 'Survey on coverage path planning with unmanned aerial vehicles', *Drones*, 2019, **3**, (1), p. 4

[10] Piciarelli, C., Esterle, L., Khan, A., *et al.*: 'Dynamic reconfiguration in camera networks: a short survey', *IEEE Trans. Circuits Syst. Video Technol.*, 2016, **26**, (5), pp. 965–977

[11] Yanmaz, E., Yahyanejad, S., Rinner, B., *et al.*: 'Drone networks: communications, coordination, and sensing', *Ad Hoc Netw.*, 2018, **68**, pp. 1–15

[12] Wischounig-Strucl, D., Rinner, B.: 'Resource aware and incremental mosaics of wide areas from small-scale UAVs', *Mach. Vis. Appl.*, 2015, **26**, (7), pp. 885–904

[13] Bentz, W., Panagou, D.: '3D dynamic coverage and avoidance control in power-constrained UAV surveillance networks'. 2017 Int. Conf. on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 2017, pp. 1–10

[14] Di Franco, C., Buttazzo, G.: 'Energy-aware coverage path planning of UAVs'. 2015 IEEE Int. Conf. on Autonomous Robot Systems and Competitions, Vila Real, Portugal, 2015, pp. 111–117

[15] Curiac, D.I., Volosencu, C.: 'Path planning algorithm based on Arnold Cat map for surveillance UAVs', *Def. Sci. J.*, 2015, **65**, (6), pp. 483–488

[16] Liu, C.H., Chen, Z., Tang, J., *et al.*: 'Energy-efficient UAV control for effective and fair communication coverage: a deep reinforcement learning approach', *IEEE J. Sel. Areas Commun.*, 2018, **36**, (9), pp. 2059–2070

[17] Zargar, R.R., Sohrabi, M., Afsharchi, M., *et al.*: 'Decentralized area patrolling for teams of UAVs'. 2016 4th Int. Conf. on Control, Instrumentation, and Automation (ICCIA), Qazvin, Iran, 2016, pp. 475–480

[18] Luo, W., Tang, Q., Fu, C., *et al.*: 'Deep-SARSA based multi-UAV path planning and obstacle avoidance in a dynamic environment', in Tan, Y., Shi, Y., Tang, Q. (Eds): '*Advances in swarm intelligence*' (Springer International Publishing, USA, 2018), pp. 102–111

[19] Koch, W., Mancuso, R., West, R., *et al.*: 'Reinforcement learning for UAV attitude control', *ACM Trans. Cyber-Phys. Syst.*, 2019, **3**, (2), pp. 22:1–22:21

[20] Russell, S.J., Norvig, P.: '*Artificial intelligence: a modern approach*' (Pearson Education Limited, UK, 2016)

[21] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.*: 'Human-level control through deep reinforcement learning', *Nature*, 2015, **518**, (7540), p. 529

[22] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.*: 'Playing ATARI with deep reinforcement learning'. NIPS Deep Learning Workshop, Lake Tahoe, NV, USA, 2013

[23] Van Hasselt, H., Guez, A., Silver, D.: 'Deep reinforcement learning with double Q-learning'. Association for the Advancement of Artificial Intelligence (AAAI), Phoenix, AZ, USA, 2016, vol. 2

[24] Piciarelli, C., Micheloni, C., Foresti, G.L.: 'Automatic reconfiguration of video sensor networks for optimal 3D coverage'. Int. Conf. on Distributed Smart Cameras, Ghent, Belgium, 2011

[25] Piciarelli, C., Canazza, S., Micheloni, C., *et al.*: 'A network of audio and video sensors for monitoring large environments', in Pal, S.K., Petrosino, A., Maddalena, L. (Eds): '*Handbook on soft computing for video surveillance*' (CRC Press, USA, 2012), pp. 287–315