Drone patrolling with reinforcement learning

Claudio Piciarelli
claudio.piciarelli@uniud.it
University of Udine
Udine, Italy

Gian Luca Foresti
gianluca.foresti@uniud.it
University of Udine
Udine, Italy

ABSTRACT

When a camera-equipped drone is assigned a patrolling task, it typically follows a pre-defined path that evenly covers the whole environment. In this paper instead we consider the problem of finding an ideal path under the assumption that not all the areas have the same coverage requirements. We thus propose a reinforcement learning approach that, given a relevance map representing coverage requirements, autonomously chooses the best drone actions to optimize the coverage.

1 INTRODUCTION

This paper is focused on finding an optimal patrolling strategy for unmanned aerial vehicles (UAVs, or drones). In particular, the proposed model is suitable for multicopter UAVs, which have a better maneuverability than other types of drones, e.g. fixed wing. The problem consists of patrolling a given area in order to give visual coverage with UAV-mounted camera sensors of the whole environment. Since the environment is typically too large to be fully covered, the UAV must adopt a patrolling strategy to optimize the partial coverage through time. This generally implies following generic, pre-defined paths (e.g. zig-zag sweep pattern) but this approach does not take into account the specific requirements of the patrolling task. In particular, we assume that the visual coverage of some parts of the environment is more important, or has an higher priority, that other zones, and we express this requirement with a *relevance map*, as described in sec. 3.1. The proposed system relies on a deep-learning network that, given in input the relevance map and the current drone state, chooses the best action to optimize a reward function expressed in terms of area coverage. Being an action-reward scenario, the network has been trained using reinforcement learning.

The problem of UAV control for patrolling has been studied by several authors. A good initial survey can be found in [2], where the authors propose a taxonomy of the cited methods. 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-optimization methods and UAV-deployment strategies, although the topics are separated: coverage-oriented methods have been mostly developed for PTZ camera networks (in which the camera cannot transalte), such as in [5, 12], while UAV reconfiguration works are typically more focused on resource management [15, 16]. The joint task of area coverage and resource management has been studied in [1], where the authors give a deep mathematical formulation of 3D coverage and they propose a resource-aware algorithm that shift the bulk of spatial redistribution onto less constrained agents. In [3] the problem is analyzed from the novel point of view of path planning in adversarial environments, where the efficient use of chaotic behaviors copes with enemy entities. Other works focus on the decentralized 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. To the best of our knowledge, very few works have been published dealing with UAV coverage problems using neural networks and/or reinforcement learning. The work presented in [6] 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 [4] 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.

2 ALGORITHMS FOR REINFORCEMENT LEARNING

Many reinforcement learning algorithms assume that the problem to be solved can be modeled as a Markov Decision Process (MDP). Formally a MDP is a 5-tuple $\{S, A, \tau, r, \gamma\}$ where S is a finite set of states and A is a finite set of actions. The model is described by the transition probability $\tau(s'|s, a)$ with $s, s' \in S, a \in A$ expressing the probability to move from state s to state s' given that action a was chosen. If τ can have only binary values, the process is considered deterministic. The basic property of MDPs thus consists in the future states being a function of the current state and action only, not depending on previous history of the process. A reward function $r(s, a)$ is also defined, giving a score to the choice of action a while in state s. An agent can thus act in a MDP context according to a specific policy and get a feedback on the quality of its choices, thus allowing the reinforcement or weakening of its strategies (hence the name reinforcement learning). Formally, a policy is a probability function $\pi(a|s)$ expressing the probability for an agent to choose action a while being in state s. The goal of a reinforcement learning algorithm is to find the optimal policy π^* that optimizes 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 γ is the discount factor, a parameter of the MDP denoting how much recent rewards must be weighted with respect to older

ICDSC'19, September 09–11, 2019, Trento, Italy

ACM ISBN 978-1-4503-7189-6/19/09…$15.00
https://doi.org/10.1145/3349801.3349805

ones. In order to simplify the notation, with $\mathbb{E}_{\tau,\pi}$ we mean that the expected value is computed over a sequence of states s and actions a distributed according to τ and π respectively.

Since finding the optimal policy from eq. (1) is hard, the Q-value function $Q(s,a)$ is introduced, intuitively denoting the total value (also considering future developments) of choosing action a while in state s, and defined as:

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

Eq. (2) can be written in a recursive definition as:

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

which, in the case of the optimal policy π^*, reduces to the Bellman equation [13]:

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

and, in case of deterministic systems, further reduces to:

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

where s' is the state resulting from the transition $s \xrightarrow{a} s'$ and the maximum is computed over all the possible actions a' in state s'.

The Q-learning algorithm exploits eq. (5) to compute $Q^*(s,a)$ by starting from a random choice for $Q_0^*(s,a)$ for each possible combination of states and actions, and iteratively updating it up to convergence:

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

Observe that, once Q^* is computed, obtaining the optimal policy is straightforward, since the best action is the one maximizing Q:

$$\pi^*(a|s) = \begin{cases}1 & \text{if } a = \text{argmax}_a\, Q^*(s,a)\\ 0 & \text{otherwise}\end{cases} \tag{7}$$

Exploiting eq. 6 however requires the reward function r, which is generally unknown. Q-learning solves the problem by learning from experience: if the agent is in state s, it executes an action a to learn the value of $r(s,a)$ and thus update $Q^*(s,a)$. The action to be executed is typically chosen using the exploration/exploitation approach: exploration means that the actions are chosen randomly to explore the state-action space and learn new reward values, while exploitation exploits the current estimate π_i^* of the optimal policy to sample the best action for the given state. Q-learning searches for a compromise between the two approaches, typically by using exploration at the first stages of learning, and then relying more and more on exploitation. There are several ways to do it, the most common being the ϵ-greedy approach where action a_i is defined as:

$$a_i = \begin{cases}a_i^* \sim \pi_i^* & \text{if } \rho > \epsilon\\ \text{random action} & \text{if } \rho \le \epsilon\end{cases} \tag{8}$$

where $\rho \in [0,1]$ is a random value and $\epsilon \in [0,1]$ decreases through time.

2.1 Deep Q-Networks

Implementations of the Q-learning algorithms typically rely on a dynamic programming approach, where Q values are explicitly stored in tables with size $|S| \times |A|$. This approach however is not practical when the number of states is very large, or even continuous. In this case neural networks can be used as *function approximators* for Q, as in the case of Deep Q-Networks (DQN) [7, 8]. Here, the experience (s,a,r,s') gained at each action is exploited to train a neural networks with parameters θ_i at iteration i according to the following loss function:

$$L(\theta_i) = \mathbb{E}\left[(Y_i - Q(s,a;\theta_i))^2\right] \tag{9}$$

which intuitively represents the difference between the current Q estimate $Q(s,a;\theta_i)$ and the new estimate that can be computed by knowing the reward $r(s,a)$, defined as:

$$Y_i = r(s,a) + \gamma\max_{a'}Q(s',a';\theta_i). \tag{10}$$

The expected value in eq. (9) is ideally computed over all the experience tuples (s,a,r,s'), and in practice over a limited mini-batch of samples. However, if the mini-batch is built of subsequent samples, it lacks of the necessary randomness required to estimate the expected value. For this reason, DQN algorithms implement a *replay memory*, where all the samples (or anyway a large number of them) are stored. The mini-batches are then built by randomly extracting experience tuples from the replay memory with a uniform distribution.

Another problem of eq. (9) is that the same network parameters θ_i are used both for action selection and evaluation. This could lead to biased estimates for Q, as proven in [14]. a possible solution is to decouple selection and evaluation using two different networks, either adopting the Target Network approach [8]:

$$Y_i = r(s,a) + \gamma\max_{a'}Q(s',a';\theta_i^-) \tag{11}$$

or the Double DQN approach [14]:

$$Y_i = r(s,a) + \gamma Q(s',\text{argmax}_a\, Q(s',a;\theta_i);\theta_i^-) \tag{12}$$

where θ_i^- are the parameters of a second Deep Q-Network. Rather than training the second network independently from the first one, θ_i^- is often either hard-updated to θ_i every a fixed number of epochs, or soft-updated at each epoch, according to a smoothness factor $\alpha \in [0,1]$:

$$\theta_i^- = (1-\alpha)\theta_{i-1}^- + \alpha\theta_i \tag{13}$$

3 PROPOSED SYSTEM

In this section we will model the problem of controlling a UAV to optimize a quality score as a reinforcement learning task. The agent is a camera-equipped UAV and the actions are the possible UAV controls regarding its position, camera configuration, etc.. A score is defined to boost or penalize actions depending on their outcome.

3.1 Relevance maps

In this work we rely on the notion of *relevance maps*, initially adopted in the context of camera network reconfiguration [11, 12]. A relevance map is a bi-dimensional, rectangular map of the environment to be monitored, whose values represent the relevance of a specific area, this is, the importance of its observation, or the

cost for the system if that area is not observed. Formally, given an environment with size $W \times H$ meters, a relevance map $\mathbf{R}_{m \times n}$ is a $m \times n$ matrix whose element $r_{ij} \geq 0$ represents the relevance of the rectangular cell in the area $[iW/n, (i+1)W/n[\times[jH/m, (j+1)H/m[$. We do not describe here how the relevance maps can be computed, since this is extremely dependent on the specific application. The system only requires that a relevance can be assigned to different portions of the environment. For example, in [11] the authors use the detection of moving targets by a wide-field-of-view camera to identify the regions of high activity where PTZ cameras should be focused, while in [9] the relevance maps are defined using audio sensors in order to identify areas requiring visual inspection. Here we assume that relevance maps are used to define a priority in the need of visual coverage of parts of the environment by cameras mounted on UAVs. In other words, areas with highest relevance require immediate UAV displacement to guarantee their monitoring.

3.2 Agent model

In this work it is assumed that a camera-equipped UAV can fly over the environment to be monitored to optimize the visual coverage of high-relevance areas. In order to model the position and camera orientation of the UAV, we model it as an agent whose state is a 6-tuple $\mathbf{s} = \{x, y, z, \psi, \phi, f\}$ defined as follows:

- $x \in [0, W], y \in [0, H], z \in [0, Z]$ are the spatial coordinates of the UAV, assuming it cannot fly outside the range of the environment with size $W \times H$ (as defined in section 3.1) and Z is the maximum flying height;
- $\psi \in [0, 2\pi]$ is the orientation angle of the UAV (angle between the frontal UAV axis and the x axis);
- $\phi \in [0, \pi/2]$ is the camera tilt angle, $\phi = \pi/2$ is a nadiral view;
- f is the focal length of the camera optics, the range is hardware-dependent.

The agent can actuate the following 12 actions:

- *MoveForward, MoveBackward, MoveLeft, MoveRight*: move the agent in the four main directions by a step δ. Formally, given an initial position (x, y), the new position (x', y') is defined as:
$$x' = x + \delta\cos(\alpha), y' = y + \delta\sin(\alpha) \tag{14}$$
where $\alpha \in \{\psi, \psi + \pi, \psi + \pi/2, \psi - \pi/2\}$ depends on the moving direction;
- *MoveUp, MoveDown*: increment or decrement the flying height z;
- *RotateLeft, RotateRight*: increment or decrement the orientation angle ψ modulo 2π;
- *TiltDown, TiltUp*: increment or decrement the camera tilt angle ϕ;
- *ZoomIn, ZoomOut*: increment or decrement the focal length f.

All the actions have no effect if they would force a state parameter to violate the range constraints.

3.3 Reward function

Before describing the reward function, we first need to define the ground coordinates of a generic point observed by the agent's camera. Let us consider the camera reference frame depicted in Figure 1. A point $p(x, y)$ on the image plane is described by the vector $v = [\hat{x}, f, \hat{y}]$ in the camera reference system, where $\hat{x} = s_x x, \hat{y} = s_y y$ and s_x, s_y are the pixel sizes of the imaging sensor (i.e. size of a single CCD cell). The effect of the camera pan and tilt rotations can be described by proper rotation matrices:

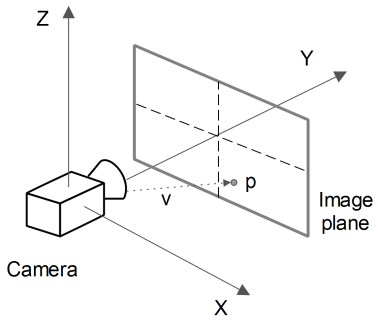


Figure 1: The camera reference system.

$$R_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos -\phi & -\sin -\phi \\ 0 & \sin -\phi & \cos -\phi \end{bmatrix} \quad R_p = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

thus the rotated vector aiming at point p is defined as:

$$v_{pt} = R_p R_t v \tag{16}$$

Let $C = [c_x, c_y, c_z]$ be the coordinates of the camera in the *world* reference system. Then, the projection of p on the ground plane is defined as:

$$p_0 = C + d v_{pt} \tag{17}$$

where d is a scaling factor defined as $d = -c_z/(v_{pt} \cdot [0, 0, 1])$, thus enforcing the requirement of p_0 lying on the ground plane.

Using equation (17), applied to the four corners of the images acquired by the camera, it is thus possible to define the ground projection of the image plane, this is, a trapezoidal shape T enclosing the portion of ground plane observed by the agent, as shown in Figure 2. We here assume that no image point is pointing above the horizon, otherwise T would be an open, infinite shape.

We can now define the total observed relevance ρ of an agent in state s as the sum of all the relevance, as defined by the relevance map $\mathbf{R}$, lying within the observed area $T(s)$:

$$\rho(s) = \sum_{(x,y) \in T(s)} \mathbf{R}(x, y). \tag{18}$$

The reward function will be defined in terms of total observed relevance, meaning that actions leading to an increase of such a value will be positively rewarded. However, we require some additional constraints, otherwise reward maximization will always lead to "extreme" agent states where the entire relevance map is enclosed in $T(s)$, e.g. by flying at very high altitude. We thus define the Constrained Total Observed Relevance (CTOR) as:

$$\hat{\rho}(s) = k(T(s))\rho(s) \tag{19}$$

where k is a penalty function on the area of T with values in $[0, 1]$ penalizing observed areas that are either too small or too large. This can be seen as a image resolution constraint, since it avoids

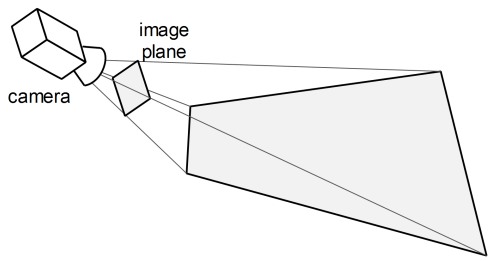


Figure 2: A rectangular image plane has a trapezoidal projection on ground plane (assuming that no point is projected above the horizon).

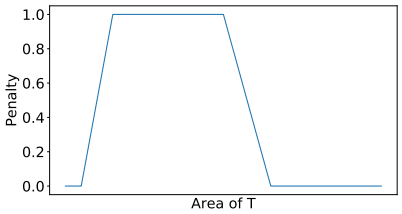


Figure 3: Reward penalty function, depending on the size of the observed area.

depicting very large areas where the pixel-per-meter resolution is low, and can be defined depending on the specific agent task: for example, target recognition typically requires a higher resolution than simple target detection tasks. The definition of k can thus be tuned on the specific needs of the system; in this paper we adopted a double-ramp function as shown in Figure 3.

It is now possible to define the reward function of the Markov Decision Process. If the action a, applied on state s, leads to a new state s', the reward function is defined as:

$$r(s,a) = \hat{\rho}(s') - \hat{\rho}(s) \tag{20}$$

thus the action is rewarded proportionally with the CTOR increment. By applying reinforcement learning algorithms on such a system model, we expect to find an approximately optimal control policy that maximizes the total discounted reward, thus controlling the agent in order to observe the zones with highest relevance.

3.4 Deep learning implementation

As discussed above, we model the problem of UAV control to maximize the coverage of relevant zones as a Markov Decision Process. The set of states and actions are defined in section 3.2, the reward is defined in 3.3 and the transition probability is trivial since the system is assumed to be deterministic. The proposed system has been implemented as a Double DQN with soft update, as described in section 2.1. The network input consists in the relevance map and the current agent state. However, rather than representing the state as a tuple (as described in section 3.2), we adopted a visual representation of the observed area as a binary mask, as shown in Figure 5b. This way, the information about the observed area is immediately available form the input data and does not have to be

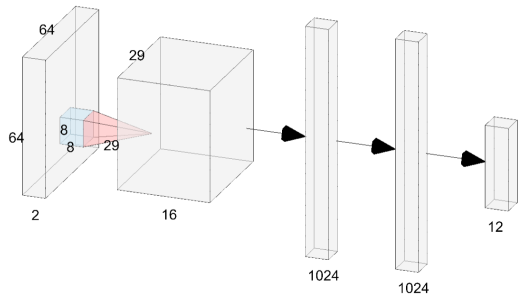


Figure 4: Network topology.

estimated from the agent's parameters, thus achieving a simpler model and a faster convergence rate. In order to further simplify the input, the two images are centered on the agent position, rotated by the agent's orientation angle, and cropped to a fixed size 64×64. This way, the input represents the surroundings of the agent, aligned with its orientation, as shown in Figure 5. The two images are finally stacked together to form a $64 \times 64 \times 2$ input tensor.

Network structure is extremely simple: the input is fed into 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 agent actions. All the layers use ReLU activation functions, except for the output layer, where the activation function is linear. The full network topology is shown in Figure 4.

The network is trained using the Double DQN approach shown in equation (12), with soft-update and replay memory. At each epoch, samples are collected by letting the agent actuate 20 actions, chosen according to an ϵ-greedy approach with exponential decay to balance between exploration and exploitation needs.

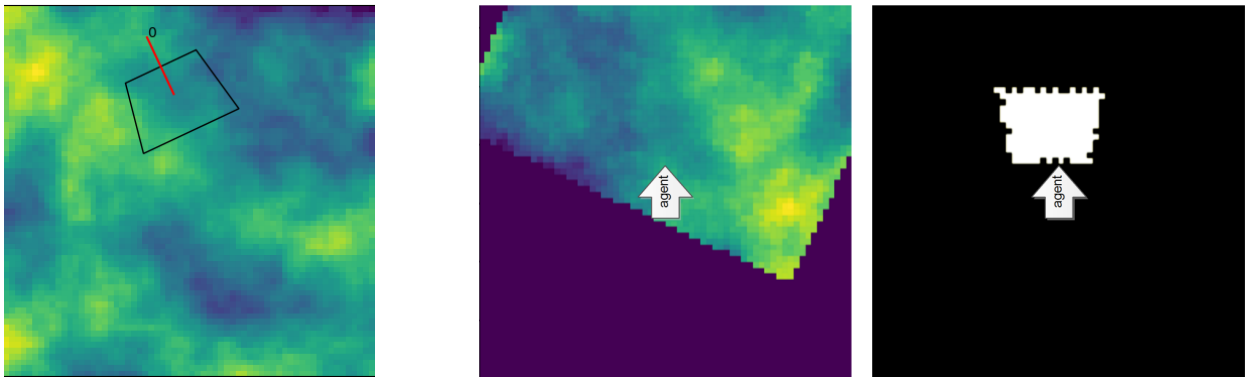
3.5 Patrolling with dynamic maps

The approach described in the previous sections is not a full patrolling algorithm, since it merely guides the agent to the configuration that maximizes the CTOR. However, it does not consider the temporal aspect, meaning that two different observation of the same area always result in the same CTOR score, independently of the time passed between the two observations. This is an unwanted behavior, because a highly-relevant zone that has not been monitored since a long time should have priority over other high-relevance zones that have been recently observed. In order to model this temporal aspect, we introduce a dynamic relevance map $\mathbf{R}^t$, which is obtained by element-wise multiplication of the static relevance map $\mathbf{R}_{m \times n}$ and a temporal mask $\mathbf{S}^t_{m \times n}$. The temporal mask is defined so that $0 < s_{ij} \in \mathbf{S}^t \leq 1$ and it is updated as follows:

$$s_{ij} = \begin{cases} 0.1 & \text{if relevance cell } (i,j) \text{ lies within} \\ & \text{the observation area of an agent} \\ \min(1, s_{ij} + \delta) & \text{otherwise} \end{cases} \tag{21}$$

where δ is an update factor expressing the speed at which the relevance of recently observed areas should increase towards its initial value.

The use of a dynamic maps implies that an agent cannot statically observe always the same region, thus enforcing a continuous



(a) (b)

Figure 5: (a): A relevance map with superimposed the agent position (marked with a 0) and observed area; (b): the corresponding network input, consisting in a portion of the relevance map aligned with agent position and orientation and the binary mask of the observed area (agent position superimposed for better clarity).

dynamic patrolling. At each time interval t, the agent chooses the best action to move toward the best configuration for map $\mathbf{R}^t$, but the map itself keeps updating based on the agent actions.

4 EXPERIMENTAL RESULTS

In order to evaluate the proposed model with static maps, we compared the best CTOR scores (eq. 19) computed by reinforcement learning (RL) search with the optimal one found by an exhaustive, brute force (BF) search over the entire state space. In our implementation, the discretization steps have been chosen so that each one of the six state parameters can assume 32 different values. With this setup, we estimate that an exhaustive search of the 32^6 possible states would require ~180 hours to be completed with a dual Xeon E5-2660 CPU, 224 GB RAM, 1 Tesla K40 and 2 Titan XP GPUs. We thus doubled the step sizes to reduce the computational time to ~2.8 hours in order to obtain the result shown in figure 6a, where CTOR=0.079. Figures 6b–d show the results obtained by the RL algorithm on the same relevance map with different runs starting from random initial states. As it can be seen, the proposed algorithm found solutions that are very close to the optimal one in terms of CTOR (0.075, 0.072, and 0.076 respectively) although the final states are very different: the same area can be observed from many different points of view.

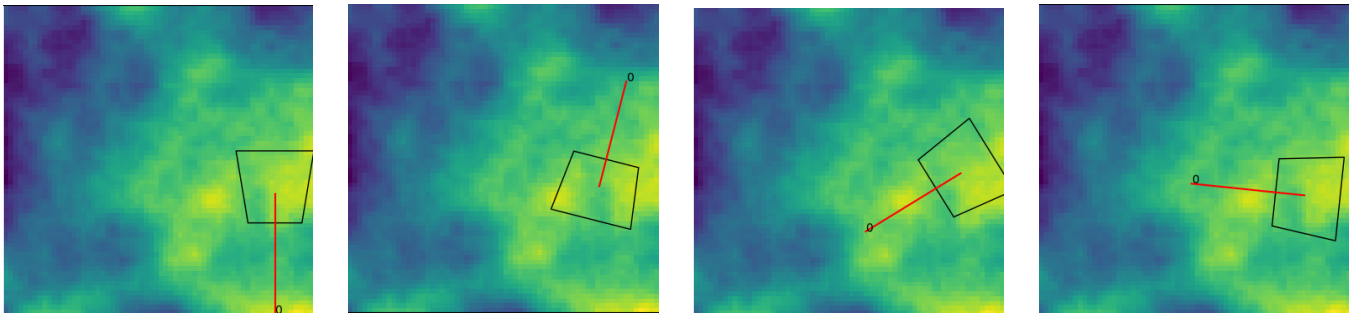
In order to get statistically meaningful results, we further doubled the step sizes in exhaustive search to reduce the computational time to ~150 seconds and we performed 100 tests with randomly-generated relevance maps. For each map, we have searched for the optimal solution using BF and we have run the proposed RL method 50 times, each time starting from a random initial state. Table 1 contains the results for the first 10 tests (the remaining ones are omitted for lack of space), showing the BF CTOR and the average RL CTOR over 50 runs on the same map (also expressed as a percentage w.r.t. the BF result). On average, the CTOR found by RL is 92.27% the BF one. The proposed method is thus able to find good solutions, close to the optimal ones, although in a fraction of time: the best RL CTOR is typically found in 0.1–0.2 seconds, thus making the computational time negligible in a real-world application, where it would be overwhelmed by the time required for the UAV to physically execute the actions.

test #	BF CTOR	RL CTOR	Actions Ratio
0	8.20	7.11 (86.61%)	1.70
1	6.30	6.05 (96.11%)	1.74
2	7.95	7.60 (95.53%)	1.90
3	6.83	6.43 (94.18%)	1.72
4	8.71	7.49 (85.94%)	1.75
5	10.06	8.91 (88.59%)	1.66
6	7.13	6.66 (93.46%)	1.62
7	7.97	7.36 (92.27%)	1.90
8	8.06	7.63 (94.74%)	1.59
9	7.68	7.50 (97.62%)	1.72

Table 1: Results on 10 tests. The table shows the CTOR found by brute force search and the average CTOR over 50 runs of reinforcement learning searches from random initial states (values multiplied by 100 for ease of notation). Last column shows the ratio between the number of actions used by RL and the optimal one.

Regarding the efficiency of the RL algorithm, we also compared the number of actions required to reach the best CTOR with the minimum number of actions possible (which is easily computed once we know both the initial and final state). Table 1 shows the ratio between the two values: on average, the number of actions executed by the RL algorithm is 1.71 times the minimum one.

Finally, we evaluated the patrolling performances of the proposed algorithm. We defined a coverage score $S = \sum_{ij} \mathbf{R}^t_{ij} / \sum_{ij} \mathbf{R}_{ij}$, this is the ratio between total relevance in the dynamic map at time t and the total relevance of the uncovered map. Low values denote good performances, because it means that high-relevance areas are kept under observation; a totally uncovered environment has a score $S = 1$. The absolute value of S is not particularly meaningful because it depends on the size of the environment w.r.t. the maximum area the drone can cover efficiently, however it can be used to compare results with other patrolling strategies. Figure 7 shows the score S



(a) (b) (c) (d)

Figure 6: Solutions found by (a): exhaustive search of the state space; (b)–(d): proposed method.

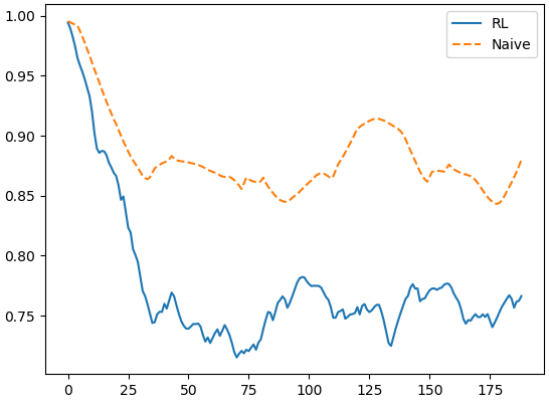


Figure 7: Coverage score S for RL and naive patrolling. Lower values are better.

for two runs on the same map, one with RL and one with a naive zig-zag patrolling pattern. As it can be seen, the proposed algorithm manages to get a constantly lower score than the naive approach thanks to its prioritization of high-relevance areas coverage. A test on 100 different maps confirm the result, and on average the RL approach is ~20% more efficient than naive patrolling.

5 CONCLUSIONS AND FUTURE WORKS

In this paper we presented a reinforcement learning approach to train a deep neural network to find optimal patrolling strategies for UAV visual coverage tasks. Differing from the majority of other works, the proposed method explicitly considers different coverage requirements expressed as relevance maps. Experimental results show that our relevance-aware method leads to improved patrolling results if compared with standard zig-zag patterns. As a future development, we plan to apply the proposed approach to swarm of drones. We will also investigate the improvements needed to deal with real-world challenges, such as different topological altitudes or the presence of obstacles.

ACKNOWLEDGMENTS

This work is partially supported by the Regione Friuli Venezia Giulia under the SUPReME project (L.R. 20/2015) and the Italy-Singapore bilateral technological cooperation PRESNET project.

REFERENCES

[1] W. Bentz and D. Panagou. 2017. 3D dynamic coverage and avoidance control in power-constrained UAV surveillance networks. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 1–10.
[2] Tauã M. Cabreira, Lisane B. Brisolara, and Paulo R. Ferreira Jr. 2019. Survey on Coverage Path Planning with Unmanned Aerial Vehicles. *Drones* 3, 1 (2019).
[3] Daniel-Ioan Curiac and Constantin Volosencu. 2015. Path Planning Algorithm based on Arnold Cat Map for Surveillance UAVs. *Defence Science Journal* 65, 6 (Nov. 2015), 483–488.
[4] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. 2019. Reinforcement Learning for UAV Attitude Control. *ACM Trans. Cyber-Phys. Syst.* 3, 2, Article 22 (2019), 22:1–22:21 pages.
[5] K.R. Konda, N. Conci, and F. De Natale. 2016. Global Coverage Maximization in PTZ-Camera Networks Based on Visual Quality Assessment. *IEEE Sensors Journal* 16, 16 (2016), 6317–6332. cited By 10.
[6] Wei Luo, Qirong Tang, Changhong Fu, and Peter Eberhard. 2018. Deep-Sarsa Based Multi-UAV Path Planning and Obstacle Avoidance in a Dynamic Environment. In *Advances in Swarm Intelligence*, Ying Tan, Yuhui Shi, and Qirong Tang (Eds.). Springer International Publishing, Cham, 102–111.
[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*.
[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
[9] Claudio Piciarelli, Sergio Canazza, Christian Micheloni, and Gian Luca Foresti. 2012. A network of audio and video sensors for monitoring large environments. In *Handbook on Soft Computing for Video Surveillance*, S.K. Pal, A. Petrosino, and L. Maddalena (Eds.). CRC Press, 287–315.
[10] Claudio Piciarelli, Lukas Esterle, Asif Khan, Bernhard Rinner, and Gian Luca Foresti. 2016. Dynamic Reconfiguration in Camera Networks: A Short Survey. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 5 (2016), 965–977. https://doi.org/110.1109/TCSVT.2015.2426575
[11] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti. 2010. Occlusion-aware multiple camera reconfiguration. In *Proceedings of the International Conference on Distributed Smart Cameras*. 88–94.
[12] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti. 2011. Automatic reconfiguration of video sensor networks for optimal 3D coverage. In *International Conference on Distributed Smart Cameras*.
[13] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.
[14] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.. In *AAAI*, Vol. 2. Phoenix, AZ, 5.
[15] Daniel Wischounig-Strucl and Bernhard Rinner. 2015. Resource aware and incremental mosaics of wide areas from small-scale UAVs. *Machine Vision and Applications* 26, 7 (01 Nov 2015), 885–904.
[16] Evşen Yanmaz, Saeed Yahyanejad, Bernhard Rinner, Hermann Hellwagner, and Christian Bettstetter. 2018. Drone networks: Communications, coordination, and sensing. *Ad Hoc Networks* 68 (2018), 1 – 15. Advances in Wireless Communication and Networking for Cooperating Autonomous Systems.
[17] R. R. Zargar, M. Sohrabi, M. Afsharchi, and S. Amani. 2016. Decentralized area patrolling for teams of UAVs. In *2016 4th International Conference on Control, Instrumentation, and Automation (ICCIA)*. 475–480.