POLITECNICO DI TORINO
Repository ISTITUZIONALE

Comparison between Helical Axis and SARA Approaches for the Estimation of Functional Joint Axes on Multi-Body Modeling Data

Original
Comparison between Helical Axis and SARA Approaches for the Estimation of Functional Joint Axes on Multi-Body Modeling Data / De Benedictis, C.. - In: APPLIED SCIENCES. - ISSN 2076-3417. - ELETTRONICO. - 12:3(2022), p. 1274. [10.3390/app12031274]

Availability:
This version is available at: 11583/2956209 since: 2022-03-09T10:00:38Z

Publisher:
MDPI

Published
DOI:10.3390/app12031274

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

16 June 2026

RL-based Path Planning for Autonomous Aerial Vehicles in Unknown Environments

Gianpietro Battocletti *, Riccardo Urban †, Simone Godio ‡ and Giorgio Guglieri §

Unmanned Aerial Systems (UASs) have become a relevant sector in the aerospace industry. In the last decade, the increase in the capabilities of Unmanned Aerial Vehicles (UAVs), paired with a drop in their price, has led them to be used in many different applications where they are employed for their versatility and efficiency. A challenge that is being addressed in this field is that of *autonomous UAVs fleets*, i.e., the coordinated use of multiple UAVs to perform a common task. A particularly interesting application of UAV fleets is their use in the exploration and mapping of unknown or critical environments. This topic brings with it a significant number of challenges, from the design of the policy used to coordinate the fleet to the path planning algorithm that each UAV uses to move in the environment while exploring it. In this paper, a Reinforcement Learning (RL)-based approach for the cooperative exploration of unknown environments by a fleet of UAVs is presented. Two RL agents are trained to address this problem: the first has the task of coordinating the exploration task, optimizing the way the UAVs spread in the unknown area by assigning some waypoints to them. The waypoints are placed in order to optimize the distribution of the fleet and to maximize the exploration process efficiency. The second RL agent is a path planning algorithm and is used by each UAV to move in the environment to reach the region pointed by the first agent. The combined use of the two agents allows the fleet to coordinate in the execution of the exploration task.

I. Introduction

In the last few decades, Unmanned Aerial Vehicles (UAVs) have become a relevant field in the aerospace sector. In particular, much attention has been directed towards the field of *autonomous* UAVs. Thanks to the recent developments in the field of Artificial Intelligence (AI), and in particular in that of Deep Learning (DL) [1], the capabilities of autonomous UAVs have grown exponentially. Therefore, they have attracted significant interest from the academic and scientific community. As their capacities increased and costs dropped, one particular area on which part of the research has focused is that of autonomous UAV fleets. The idea is to create fleets of UAVs capable of coordinating and collaborating in the pursue of a common objective. This task comes with several challenges. A number of different solutions have been proposed - or are still being researched - to solve this problem [2]. The techniques involved in the design of fleet coordination algorithms are various, and range from the use of mechanical models to the implementation of bio-inspired methods, up to the use of Deep Learning approaches [3]. A particular application of autonomous UAV fleets is their use in the exploration of unknown or critical environments (i.e., where it could be dangerous for a human operator to go). This particular topic constitutes the main focus of this paper. Before getting to the description of the model proposed to tackle this problem, a brief review of some state-of-art techniques used to solve similar problems will be discussed.

A first approach used for fleet coordination is that of *flocking algorithms* [4]. In this case, fleets are usually organized to have a leader which guides the action of all the components. The algorithm is distributed, i.e., each component of the fleet takes its own decisions, but the leader has a prominent role in the decision-making centralizing part of the fleet control. Often, flocking algorithms aim at maintaining a desired formation, i.e., a given relative position between the fleet leader and the other components, and in some cases also between the components themselves. To obtain this goal, different strategies can be enacted [5–7]. Flocking algorithms are well suited to control fleets that need to maintain a formation following a leader unit. However, they are not efficient for exploration and coverage tasks, since the single components lack the independence necessary to quickly cover all the area in the environment.

*MSc Graduate Student, Politecnico di Torino, Torino, Italy, gianpietro.battocletti@studenti.polito.it
†MSc Graduate Student, Politecnico di Torino, Torino, Italy, riccardo.urban@studenti.polito.it
‡PhD Student, Department of Mechanical and Aereospace Engineering, Politecnico di Torino, Torino, Italy, simone.godio@polito.it
§Full Professor, Department of Mechanical and Aereospace Engineering, Politecnico di Torino, Torino, Italy, giorgio.guglieri@polito.it

A second group of methods developed to assess this problem is that of *swarming algorithms* [8, 9]. Swarming algorithms are based on a decentralized intelligence approach. Each component of the swarm follows the same set of rules, which are usually quite simple and regard only the decisions taken by a single member of the swarm. The application of those rules by all the members of the fleet leads to the emergence of more complex behaviors and strategies which can not be predicted from the basic rules. These kinds of behaviors are called *emerging behaviors* and stem from the concept of *collective intelligence* [10]. These methods are often bio-inspired, as they get inspiration from the observation of the behavior of swarms of bees or insect colonies. Fleets designed following this approach are usually leaderless, i.e., they have no hierarchical organization. Every component of the swarm takes decisions on its own on the basis of its current state and of the information it has about the other members. This information can be obtained by sensing the surrounding environment or through direct communication with the other components of the swarm. Communication (both direct or indirect, i.e., obtained by leaving a trail or marks in the environment) is one crucial feature of swarming algorithms, as the swarm components rely on information about the other members to take their decisions. A particular type of swarming methods based on indirect communication is that of *stigmergy*-based algorithms [11–13]. In this approach, each member of the fleet releases a virtual substance (usually called "digital pheromone" from its biological inspiration) as it moves in the environment. The pheromone remains where it is released for a certain time, diffusing slowly in the nearby area. While it persists, it can be used to gain information on the past location of the other members of the fleet, and be exploited to give rise to a collective exploration algorithm. If every member of the fleet tends to go toward the lowest concentration of pheromone, it is possible to obtain an emerging coverage algorithm suitable for unknown environments. The main issue in a practical implementation of this method for a fleet of UAVs is in the simulation of the release and diffusion dynamics of the pheromone. In fact, all these operations must be performed inside the algorithm and can result in time-consuming. Moreover, the release of the substance must be communicated *directly* between the UAVs, losing all the advantages of a method based on indirect communication. Attempts have been made to use a *real* substance released in the air by the UAVs, equipping them with a sensor to detect its presence. However, this kind of approach brings with it a series of complications about the management of this substance, rendering the method sub-optimal.

A different and still unexplored approach to swarming algorithms is obtained by exploiting learning-based models, i.e., models obtained through a Deep Learning (DL) process. In this case, the rules determining the individual behavior of each component of the fleet are learned through a dedicated training process. The goal is to generate the best set of individual rules to obtain the desired collective behavior. This kind of method fits in the domain of Multi-Agent Reinforcement Learning models (MARL models), which are a very recent field of study in Artificial Intelligence [14–16]. MARL models are usually built using special architectures to design the training process, optimized to drive the agent to learn the desired policy. The application of the same policy by all the members of the fleet is what leads to the desired emerging behavior. There is relatively little literature on this approach allied to the autonomous exploration problem [17–19]. The goal of this paper is to explore this subject by implementing a Reinforcement Learning-based swarming algorithm for exploration and coverage of unknown environments. The design of a DL-based swarming algorithm is not straightforward. In fact, it is not easy to design an effective learning process able to train an agent (i.e., a member of the fleet acting according to an *individual behavior*) on the basis of an *emerging behavior-based objective*. For this reason, it has been decided to divide into two parts the exploration algorithm. The first is the Coverage agent and is based on the idea just discussed: a single policy will be trained to obtain a desired collective behavior. The second part of the algorithm is the Path Planning algorithm. Ideally, the two could be integrated into a single entity. However, due to the initial difficulties typical of the design and training process of a DL agent, it has been decided to keep them separated.

The paper is organized in the following way. In Section II the proposed algorithm is illustrated in detail, with a focus on the design of the Reinforcement Learning agents that constitute its core. In Section III some simulations of the algorithm performances are shown. The results obtained from the simulations are discussed in more detail in section IV. Finally, the conclusions of the work, its main issues and future developments are discussed in Section V.

II. Proposed model

A Reinforcement Learning (RL)-based swarming model is proposed to address the exploration problem. The goal is to design an algorithm capable of driving a fleet of autonomous UAVs in the exploration of an unknown environment, having little (or no) a-priori information about it. The proposed model is composed of two RL agents that work together. The first agent is called the *Coverage* agent and has the task of coordinating the exploration task. To do so, the Coverage

agent computes a temporary goal (also called *waypoint*) in the environment for each component of the fleet; each UAV has then to reach its designed waypoint. The waypoints are generated in order to optimize the speed of the exploration task: therefore, their positioning will tend to favor spread fleet formations, with each UAV having to explore a specific region of the environment. The second agent is a *Path Planning* agent. Its task is to compute an efficient trajectory to lead each UAV to its target waypoint. The challenges that the Path Planning agent has to overcome are represented by the presence of obstacles, both fixed (objects in the environment) and mobile (at the moment, only other UAVs are considered as mobile obstacles), that need to be avoided. The Path Planning agent relies on a numerical model of the environment to compute the trajectory. The model is built iteratively, adding information about obstacles and environment properties as they get discovered during the exploration. The model is built as an Artificial Potential Field (APF) model [20]. A potential value $U(x)$ is assigned to each point of the environment, quantifying the "risk" of being in that point. A high potential value is associated to obstacles, while a lower potential is associated to a safe point. The model is built to have its minimum in the temporary goal. In the original implementation of the APF algorithm, the goal is found by just following the negative gradient of the potential, i.e. $-\nabla U(x)$. However, this solution has some issues: first of all, the trajectories produced are non-smooth and therefore sub-optimal from a dynamical standpoint. In addition, the APF model suffers from the presence of local minima that, especially in the original discussion, can be quite problematic from the path planning point of view and need a dedicated solution (in fact, a UAV following only the gradient to find the goal would get attracted and trapped by the local minima) [21, 22]. The proposed Path Planning agent is able to overcome all these issues thanks to a proper training of the RL agent.

The focus of this paper is on the RL agents design and training. In fact, the agents are the core of the proposed algorithm. The use of two RL agents instead of one is a significant difference with respect to other works in the same field, as the already-cited [17, 19]. It is expected that the use of two agents will increase the performances of the algorithm since they will both be more specialized in their role (this will come at the cost of a bit more computational power - needed to execute two agents instead of one - and the requirement of two distinct training processes). The algorithm also includes some utility function, as for example the ones that update and manage the APF environment model. Each of the two agents has been defined using two Neural Networks (NNs) and has been trained using a Deep Deterministic Policy Gradient (DDPG) learning algorithm [23]. DDPG is a model-free, off-policy learning algorithm suitable to work in continuous action space. It exploits two NNs, called *critic* and *actor*, to concurrently learn a Q-value function and a policy. The training of the two RL agents has been performed in a custom-designed training environment. A group of functions has been developed to simulate all the interactions of the agents with the environment, such as flight, communication, sensing, and vision operations. A scheme of the training process is shown in Figure 1. The relationships between the RL agents, the DDPG learning algorithm and the simulated environment are displayed.

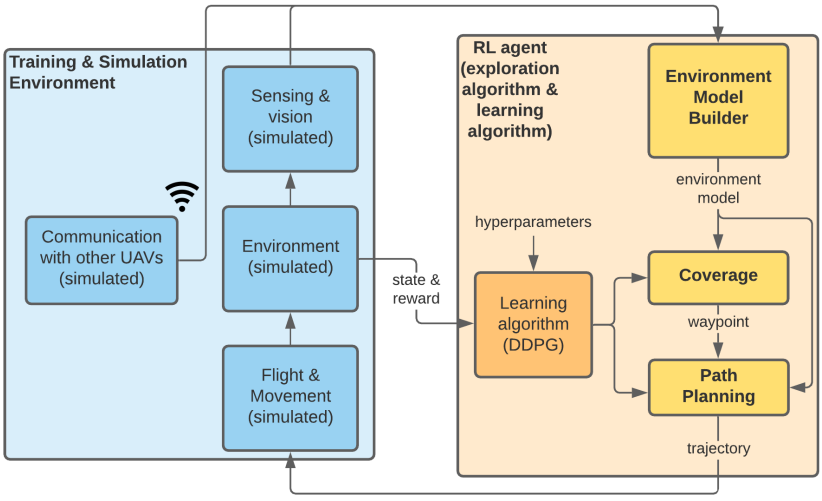

Fig. 1 Block scheme of the algorithm agents integrated in the training process.

In the training process of Figure 1, at each time step, the Model Builder receives some sensor data from the environment and uses them to update the APF model. The model is used by the Coverage and Path Planning agents to choose which actions to apply. The actions are executed in the environment, whose state is updated depending on

their effect. The learning algorithm works alongside this process. At each time step, the action $a = a(t)$ chosen by the agent, the current state $s = s(t)$, the next state $s' = s(t+1)$ (i.e., the state of the environment after the action a has been performed) and the reward $r = r(t)$ are collected. The tuple (s, a, r, s') is stored in a memory buffer and used to perform the DDPG learning operations. The training process continues iteratively in this fashion; at each time step, the state s' obtained in the previous iteration is used as the new input state.

A. Coverage agent design

As explained previously, the task of the coverage agent is to provide a temporary objective for each UAV. The motion of each UAV toward its designed waypoint leads to the observation of new regions of the environment, contributing to the exploration process. The simultaneous use of the coverage algorithm by each member of the fleet leads to the collective exploration of the environment. A first requirement on the Coverage agent is that the UAV fleet must be leaderless. This means that the algorithm is distributed, and each UAV computes its own waypoint. This increases the flexibility of the Coverage agent, since it is not required that the UAVs can communicate with each other for it to work. A second requirement regards the scalability of the Coverage algorithm. The Coverage agent is trained to work for any number of UAVs in the fleet (at least in a reasonable range, e.g. between 2 and 10 UAVs). This requirement directly translates into a constraint on the input of the agent NN. In fact, the input must be independent from the number of UAVs, but at the same time, it must contain all the necessary information for the agent to take into account all the other UAV positions.

The requirements discussed above lead to the design of a state composed of two parts. The first is a 2-node dense input layer containing the normalized current position of the UAV whose waypoint is being computed. The second is a $n \times m$ matrix containing all the necessary information on the map. The matrix is built in the Environment Model Builder as a matrix having the same dimension of the environment. Then (1) obstacles are represented with ones, (2) other UAVs are represented with a 2D Gaussian function, (3) explored parts of the environment are indicated with a value η and (4) unexplored regions are represented with zeros. The resulting matrix contains all the required information about the environment status. The shape associated to other UAVs is intended to represent their "area of influence", i.e. the region they are more likely to explore. This way, exploration paths should not overlap. An example of the resulting input state is displayed in Figure 2. The output layer of the agent is composed of 2 neurons holding a sigmoidal activation function. The two outputs correspond to the computed position of the UAV temporary objective normalized with respect to the environment size. It is sufficient to multiply the outputs for the environment dimension to obtain the position of the waypoint. It is worth making a remark about the use of the environment dimensions in this part of the algorithm. In fact, in general, these data are not known a-priori. However, at least an upper bound on the environment dimension must be known for the algorithm to work. Therefore, at least a conservative value of the environment size is always available.

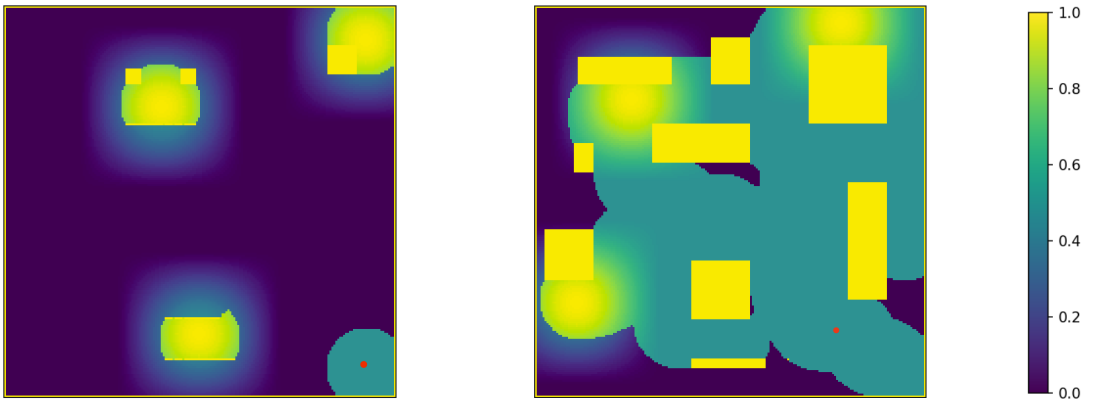

Fig. 2 Example of two Coverage agent input states. The matrices displayed have dimension [200 × 200], representing an environment of size [20m × 20m] with a resolution of 0.1m. The color bar shows the value of each point. In parallel to this matrix state, the agent position (indicated by the red dot, which is not part of the actual matrix state) is supplied as input to the Coverage NN.

The NN designed to build the Coverage agent is shown in Figure 3. In the image, the main sections composing the NN can be observed. The input state, divided into the two channels described above, is processed by some initial layers. Here, the most relevant part is represented by the convolutional layers that elaborate the matrix input. The convolutional layer applies some "filters" to the input to extract relevant features. These features are passed to the dense layers section, where the two inputs are merged and elaborated together. All the neurons in the dense layer are associated with a ReLU activation function.

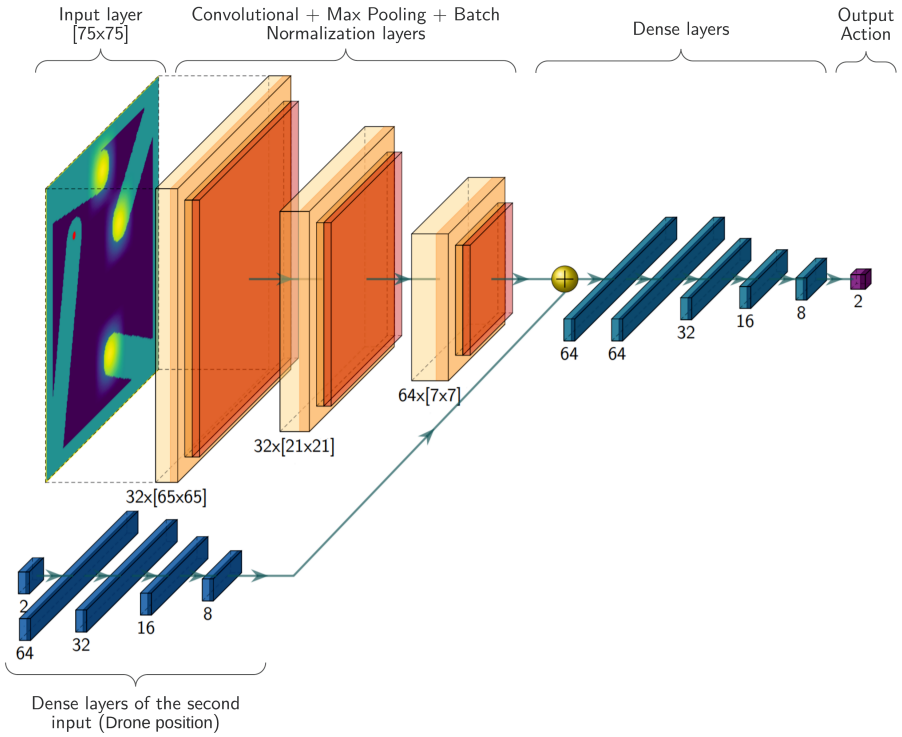

Fig. 3 Representation of the Coverage agent *actor* Neural Network structure.

A relevant feature in the training process is the reward function $r = \mathcal{R}(s, a)$ which associates action a taken in the state s to a value r which measures the goodness of the action choice itself. The function $\mathcal{R}$ can be arbitrarily defined, but it must be carefully designed since the reward values drive the learning of the NN. In general terms, actions which are rewarded positively are more likely to be repeated when a similar input state is received. The reward function used to train the Coverage agent is:

$$r = \mathcal{R}(s, a) = \begin{cases} -5 & \text{if } x_g \text{ is in an illegal location} \\ -1 & \text{if } x_g \text{ is in an unwanted location} \\ w_1 \Delta & \text{else,} \end{cases} \tag{1}$$

where x_g is the waypoint position and Δ is the size of the region explored while moving toward a valid waypoint, which is multiplied by a tunable-weight w_1. A temporary goal x_g is considered to be in an *illegal* location if it is over a known obstacle. An *unwanted* location, instead, is defined as an already explored point, or a point under the area of influence of another UAV. While the first two lines of the reward function are intended to prevent unwanted behaviors, the third is the one that prizes the agent for exploring new regions of the environment. The reward in this case is proportional to the size of the explored region, encouraging the agent to find smart ways to explore more space. The area is computed by simulating the movement of the agent toward the waypoint. This is done by performing a certain number of steps using an already-trained Path Planning agent. A property of the Coverage algorithm that is worth noticing is that the agent is trained to be *selfish*. In fact, it receives no reward for the other UAVs exploration.

B. Path Planning agent design

Inside each UAV, once the Coverage agent has computed the temporary objective, this is passed to the Path Planning agent. The task of this agent is to compute a suitable trajectory to reach the waypoint. As already introduced at the beginning of Section II, the Path Planning agent relies on an APF environment model to get all the information necessary to perform its computations. The APF model is updated over time by the Model Builder in order to take into account changes in the goal position (since goals are temporary and are cyclically updated by the Coverage agent) as well as to integrate any information about obstacles that are discovered as the exploration proceeds. At each time step, a portion $s(t)$ of the APF model is extracted and passed as input state to the Path Planning agent $\mu_\theta(s(t))$. The state has fixed dimension (e.g. $[75 \times 75]$ cells, representing a $[7.5m \times 7.5m]$ area with a resolution of $0.1m$) and represents the environment in the neighborhood of the agent. The input state is always centered on the agent current position. The output of the path planning agent is a single scalar value ψ, corresponding to the optimal movement direction in the plane. This kind of output is quite limited in itself since the UAV would have to move in a straight line until the next execution of the Path Planning agent. Therefore, the routine of Algorithm 1 is performed to compute the desired path.

Algorithm 1 Path Planning routine

1: compute the starting state s_0 in the current position x_0
2: **for** i = 1...n **do**
3: compute the motion direction as $\psi = 2\pi \cdot \mu(s_{i-1})$
4: move from position x_{i-1} of a distance δ in direction ψ to obtain position x_i
5: compute the new state s_i in position x_i
6: **end for**
7: compute the trajectory by applying a fitting function to the n points $(x_0, x_1 \dots x_n)$
8: pass the trajectory to the controller to follow it

The main limit of the Path Planning routine is the fact that the distance between each step is fixed (δ). A significant improvement could be obtained by adding a second output node to the agent, representing the desired speed. This way, the agent output would be a velocity vector, and it would be possible to compute more dynamically efficient trajectories. However, this improvement in the NN structure has not been introduced due to the difficulty to define an effective reward function for the second node output. Therefore, this enhancement is postponed to future developments. The NN designed to build the Path Planning agent is shown in Figure 4. Similarly to the one designed for the Coverage agent, it is composed of a section of convolutional layers that processes the input matrix as if it was an image, extracting relevant features by mean of some trained filters. Afterward, the features are passed to a dense layers section where the "abstracted" features are elaborated to produce the output value.

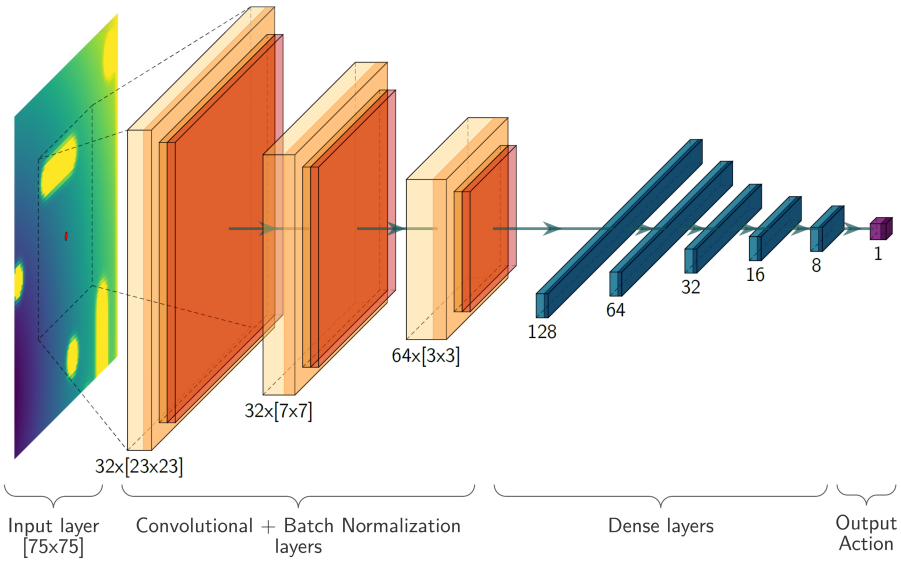

Fig. 4 Representation of the Path Planning agent *actor* Neural Network structure.

As for the Coverage agent, it is worth devoting a few words to the definition of the reward function used in the training of the Path Planning agent. The reward function used is:

$$r = \mathcal{R}(s,a) = \begin{cases} -10 & \text{if an obstacle is hit} \\ 10 & \text{if goal } x_g \text{ is reached} \\ w_1 \Delta U - w_2 \Delta\psi - w3\tau & \text{else,} \end{cases} \tag{2}$$

The first piece of the reward function is intended to punish unwanted and potentially dangerous behaviors. The second one assigns a large prize to the reaching of the goal, which is the final objective of the path planning agent. The third line is the one that helps the agent learn how to reach the goal. As in the Coverage agent case, the reward is composed of a term proportional to a variable performance index, which in this case is the difference between the current potential value and the potential at time $t-1$, that is, $\Delta U = U(x(t)) - U$. If the difference is positive, the agent is positively rewarded. If not, the reward is negative. The weight w_1 is defined to vary with the sign of ΔU: when $\Delta U > 0$ the weight w_1 becomes bigger, punishing more the agent if it goes "against" the potential (but not too much, since in some situations going against the potential is the correct thing to do). The second term of the third line, $\Delta\psi$, punishes the agent with a negative reward if the variation of direction ψ between two subsequent steps is too large. This term is meant to incentive long-term behaviors that produce smooth trajectories. With proper tuning of the weight w_2, this term could be used to take in account the dynamical properties of the agent and produce optimal paths from the dynamical point of view. The term τ is constant and represents a time penalty. It urges the agent to find the goal quickly to avoid getting this negative reward. This is the term through which the agent should learn to get out from APF local minima, since once inside them this small negative reward would be continuously obtained for not doing anything. The constant pieces of the reward, as well as the weights in the third line, are tuned through a trial-and-error process.

III. Agent training and simulation

Several simulations have been run to inspect the agents behavior. Different Path Planning and Coverage agents have been tested in multiple validation environments and with various fleet dimensions. It is worth discussing a few of the simulations since some relevant results can be observed. In this section, a qualitative analysis of the results is performed. A quantitative evaluation and comparison of the results can be found in the next section. All the trainings and validations have been performed in a custom-designed simulation environment. This simulation environment has been implemented using Python 3.8.3. Some relevant libraries used to develop the code are Tensorflow 2.3.0, Keras 2.4.3 and OpenCV 4.4.0. Some of the code developed can be found in the GitHub repository https://github.com/gbattocletti-riccardoUrb. The three validation maps used to perform all the simulations and test shown in this document are displayed in Figure 5. They are used both for Path Planning and Coverage evaluation.

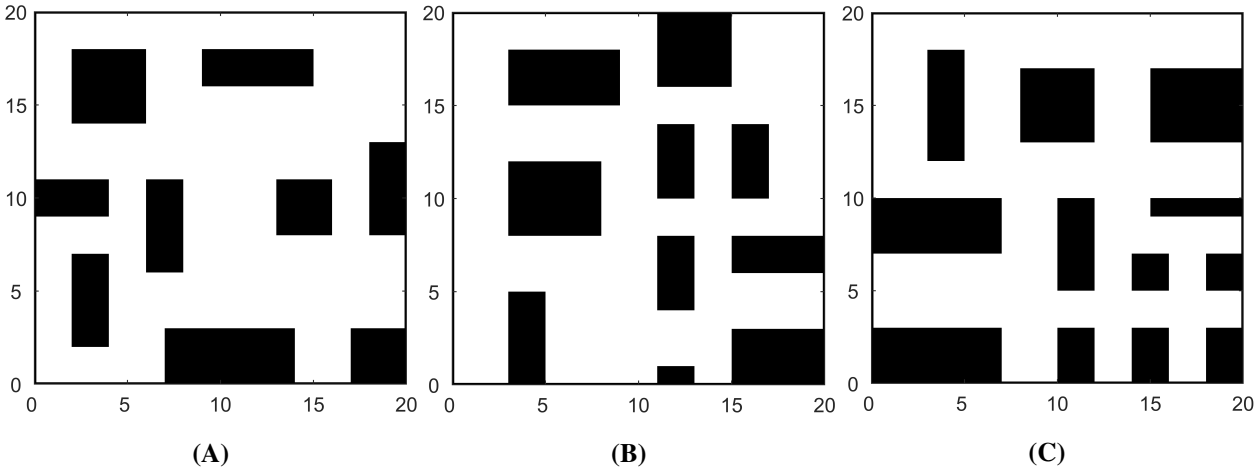

Fig. 5 Validation environments used to generate the test scenarios. Dimensions are expressed in meters, resolution is 0.1m. Environments are ordered on the base of the growing complexity. Map A is 27.8% occupied by obstacles. Map B and map C are covered by obstacles respectively 30.3% and 34.0% of their area.

A. Path Planning agent training and simulations

The first agent trained was the Path Planning agent. The results of the training process performed on the Path Planning agent described in Section II are displayed in Figure 6 in terms of average reward and of episode length.

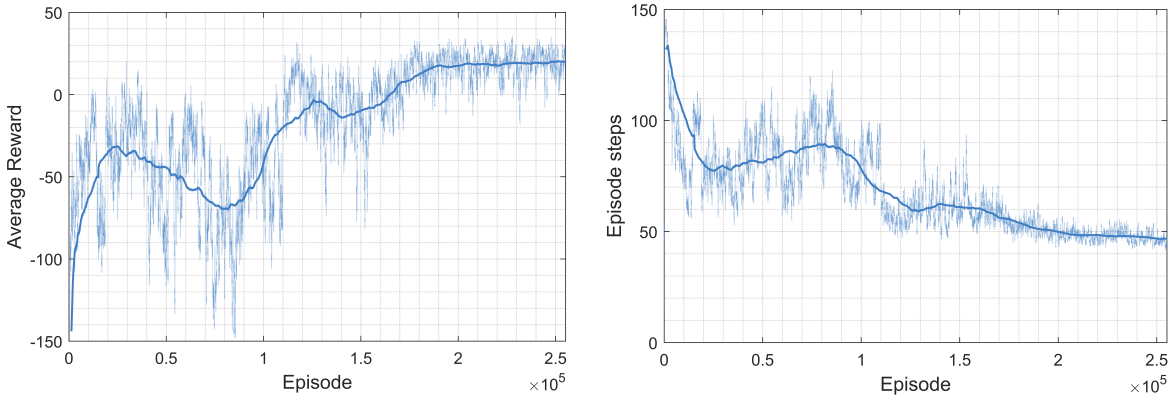

Fig. 6 Some evaluation metrics of the training process of the Path Planning agent.

In the left figure, the average episode reward is plotted over the episodes. The thin line represents the singular episode rewards, whereas the thick one represents the reward computed through a moving average. It can be observed that after about 2×10^5 episodes the NN parameters go to convergence. The average reward stabilizes around the value +20 while its variance diminishes significantly. This indicates that the agent has learned an efficient policy to deal with the training environments and is able to consistently obtain positive rewards for reaching the goal. This is also indicated by the fact that the average episode length becomes shorter, as illustrated by the image on the right. Since the agent has learned how to act efficiently in the environment, it takes less time for it to reach the goal (or, the agent spends less time trying to find an effective way to reach it). The model obtained from the training shown above has been deployed for testing in a set of validation environments. The graphical results of the computed trajectories are shown in Figure 7.

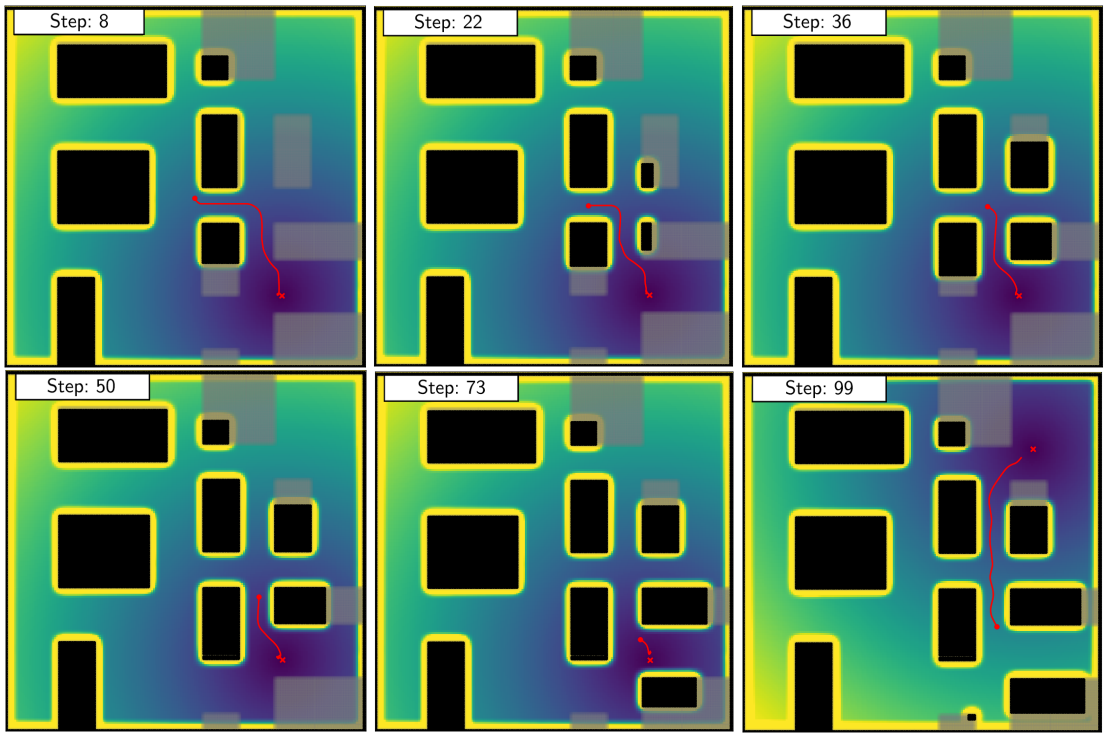

Fig. 7 Simulation of the UAV movement using the trained Path Planning agent (in validation map B).

Each image represents an instantaneous configuration of the simulation environment. The red dot represents the UAV, the red cross is its objective (randomly generated, since there is not a trained Coverage agent yet) and the red line connecting the two is the computed trajectory. Black regions represent obstacles known by the agent, whereas the grey ones indicate the locations of other obstacles which are still unknown. The colored background represents the APF model value in each point. Blue tones indicate a low potential (i.e., the goal neighborhood) whereas yellow ones represent obstacles and the area near them. The images are taken at different moments of the exploration of a validation environment. Time is represented by the *step* counter located in the upper left corner. The counter starts from 0 and increases by 1 at each step performed by the agent in the simulated environment. At each time step, the agent re-computes the trajectory from its current status using Algorithm 1 and performs a single simulated movement step. A movement step corresponds to a movement of 0.1m along the trajectory. As can be observed in the images, as the agent moves it discovers new obstacles,* whose influence is added to the APF model and updates its trajectory accordingly. In the last image (step = 99) the APF background is different from that of the other images; in fact, between step 73 (5th frame) and 99 (6th frame), the goal has been reached. For this reason, a new objective has been computed in the upper region of the environment, and the agent computes a new trajectory to reach it. The fact that the trajectory does not reach exactly the goal is due to the fact that the trajectory planning of Algorithm 1 uses a finite number n of intermediate points to fit to obtain the final path. In this case, the maximum number of steps has been reached without getting to the goal. This does not represent a problem, since the agent is still able to go toward the objective safely, and will update the trajectory to reach it in the future.

An issue encountered in the model validation has been its management of local minima. As introduced in Section II, local minima are a common problem in APF-based algorithms. The model presented above, in a situation where it found itself near some particularly difficult local minima problems ($< 10\%$ of the local minima encountered), has not been able to consistently reach the goal. For this reason, in the final implementation it has been paired with a second agent (named "*assisting*" agent), trained on different environments and with a slightly different reward function (in which the penalty for going against the potential was lowered and the time penalty was increased). This second agent, while producing - in general - sub-optimal trajectories, is able to effectively manage all local minima in which the "main" agent would get trapped. One goal for future developments is to merge the two agents into a single one. This could be obtained by performing two successive training processes, the first one to learn the generic Path Planning operations (the one carried out by the "main" agent) and the second one to specialize in the management of local minima (i.e., to have the agent learn to do the operations of the "assisting" algorithm). The performances of the local-minima-avoidance agent (i.e., the "assisting" one) are displayed in Figure 8.

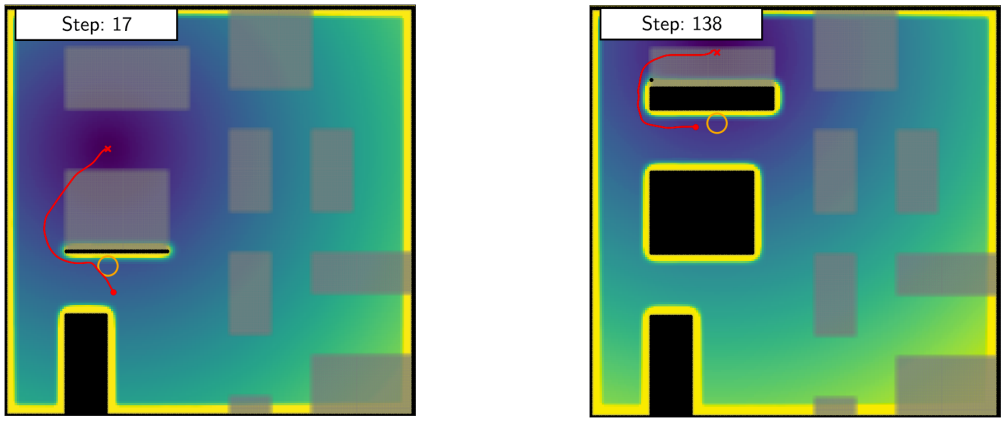

Fig. 8 Examples of trajectories computed by the "assisting" agent, i.e., the one devoted to local minima avoidance. Two different local minima in map B are considered.

*it is worth mentioning that, for simplicity, in all the training and validation environments only rectangular obstacles have been considered. This limitation has been introduced to allow the implementation of a shape-prediction algorithm (based on OpenCV) to estimate the shape and dimension of the obstacles without knowing all their contour points locations. This allowed to speed up the exploration process and to manage more easily obstacles. An extension of the shape-prediction algorithm for any closed shape is possible, but has not been implemented in the work yet.

The trajectories displayed in Figure 8 show an effective avoidance of the local minima by the assisting agent. The environment representation is the same as Figure 7; the only difference is that in this case the location of local minima is indicated by an orange circle. As can be observed, the agent is able to detect the local minima and avoid them by computing a suitable trajectory to reach the goal. In both the images the trajectory passes through an unknown obstacle: this is not a problem, since once the presence of the obstacle will be discovered the trajectory will be updated to avoid it (in the image on the right, also the goal location will be re-computed to place it outside the obstacle).

B. Coverage agent training and simulations

For what regards the Coverage agent, different combinations of NN design, reward function and input state have been considered for the training. The best combination that has been found up to now is the one discussed in Section II. However, no one of the trained agents has reached parameter convergence yet (not even the one that will be discussed in this section). In fact, as can be observed in Figure 9, the agent average reward is still negative and growing, and the reward variance is still high, indicating that the training process is not completed (it is worth noting that the plot shows only a small number of episodes. This is due to the very slow speed at which the Coverage agent proceeds, as will be explained next). This has multiple reasons. The first is that the problem is more complex than the one cast to build the Path Planning agent, and therefore it requires more time and more trials to find a proper combination of NN architecture and reward. A second reason is that the simulation of the Coverage operations (which includes also the Path Planning computations) is quite slow. This leads to much longer and slower training sessions, so the progress in the development of this agent is slower too. The last reason is that the training process is not completely optimized for a multi-agent scenario. Some improvements suggested by other already-cited papers about Multi-Agent RL [14–16], have not been implemented yet, in an attempt to maintain the highest possible flexibility for the algorithm with respect to the swarm size. These improvements regard mainly the generation of the reward and the structure of the *critic* NN. If the current agent architecture will reveal to be unable to learn the desired policy, those changes will be integrated.

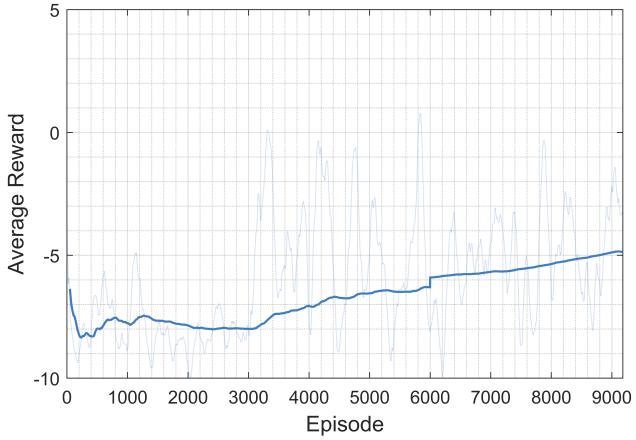

Fig. 9 Average episode reward during the training process of the Coverage agent.

The coverage agent development is not to be considered completed and will be further refined in the future. The results obtained up to now are displayed in this section and discussed from a quantitative standpoint. A sequence of frames from an exploration simulation, performed with a fleet of 4 UAVs, is displayed in Figures 10 and 11. The exploration process is visualized through a series of two different representations. Each couple of frames represents the same environment state in two different ways. The image on the left displays the goal and trajectory of each of the 4 UAVs (represented with different colors). The representation style is the same as the Path Planning agent simulations in Figure 7. The APF model visualized in the background is the one of the red UAV; it is worth noting that the other UAVs are represented as small obstacles, i.e., as mobile obstacles, corresponding to a peak in the APF value. In the top-left corner, the *step number* is reported, indicating the time advance. The right image of each couple represents the *exploration status* of the environment. The position and goal of each UAV are indicated in the same way they are in the left-side image. Discovered obstacles are displayed in black, while the *explored regions* are colored in blue. In the top-left corner, the percentage of the explored area (comprehending discovered obstacles) with respect to the total environment size is reported.

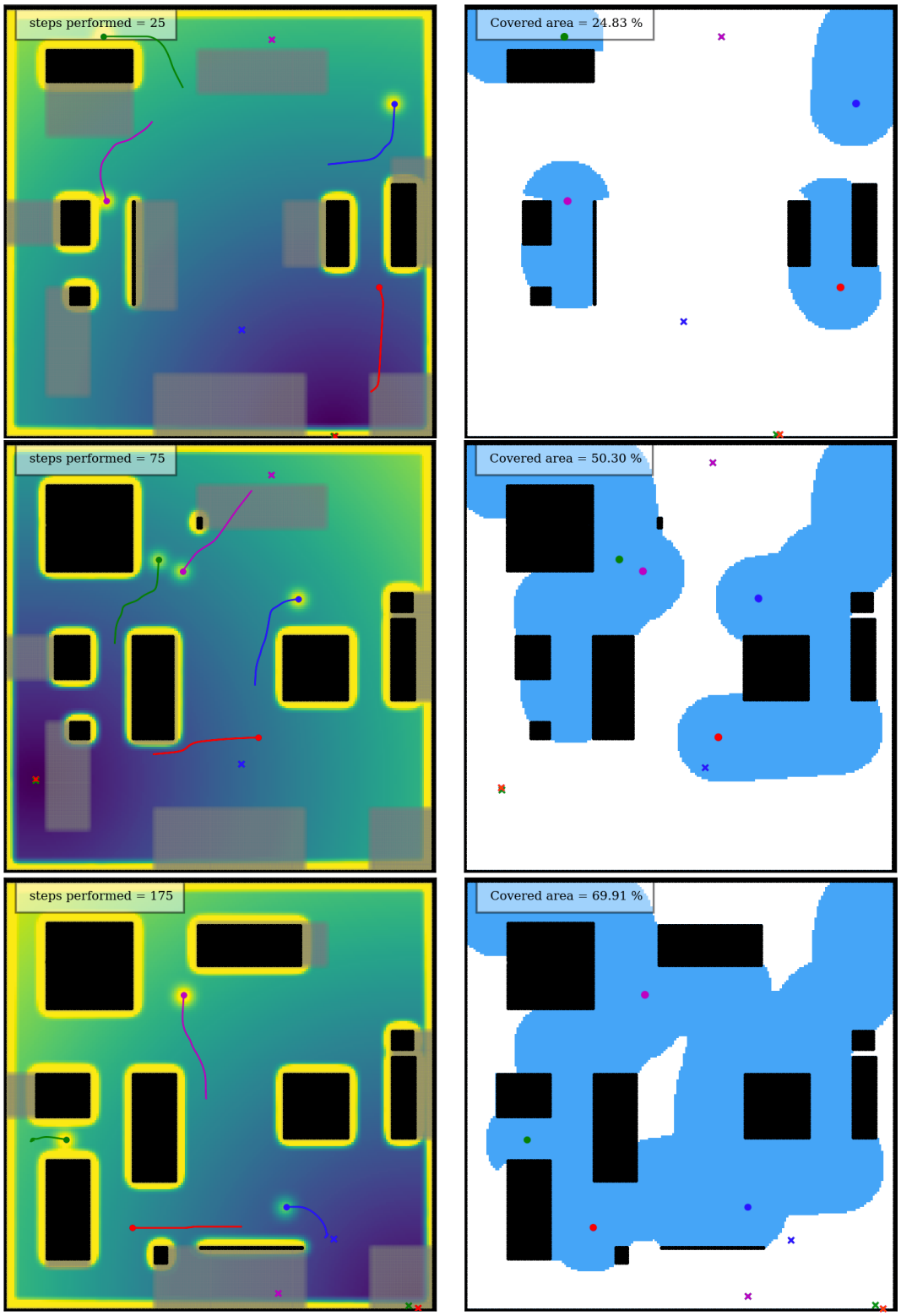

Fig. 10 [Part 1] Frames from the simulation of the exploration process driven by the Coverage agent in map A. The coverage is performed by a fleet of 4 UAVs.

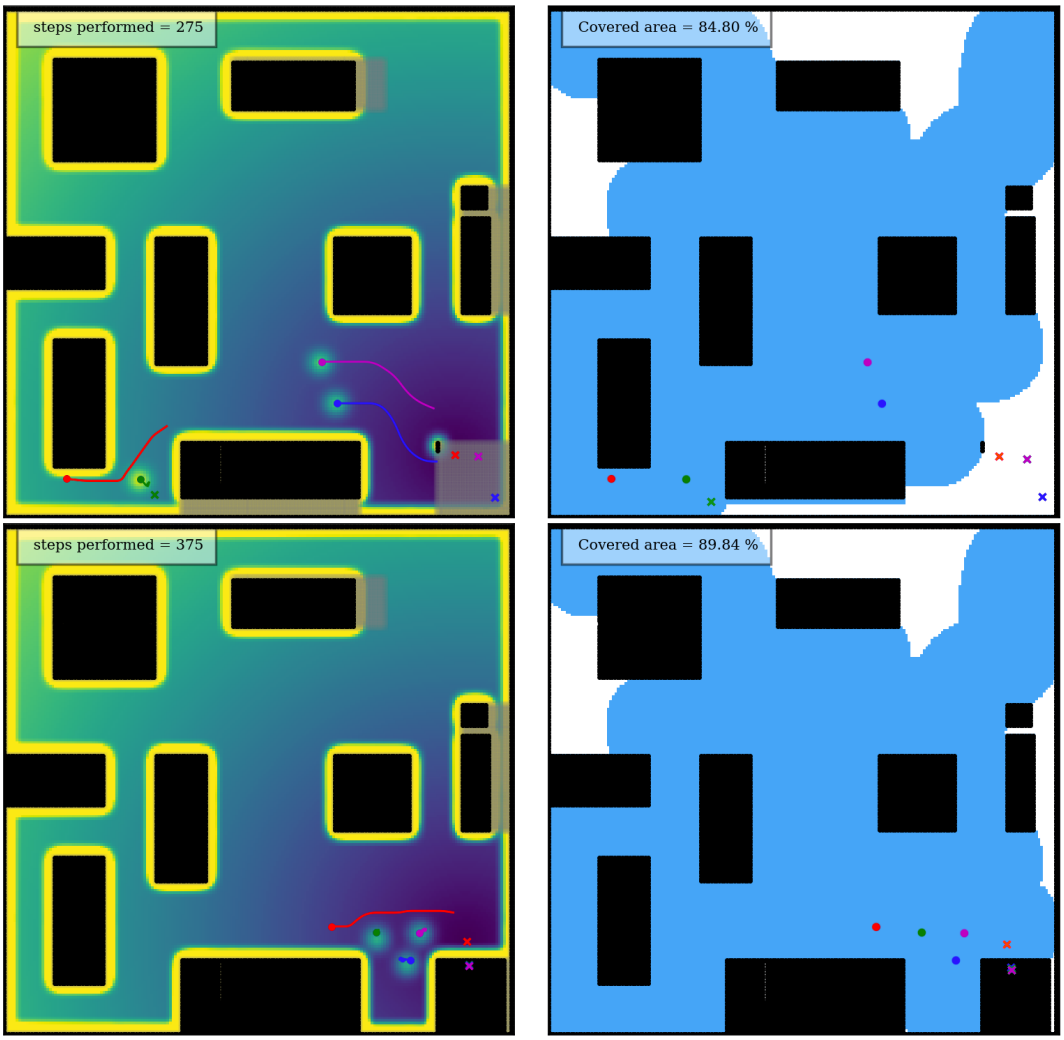

Fig. 11 [Part 2] Frames from the simulation of the exploration process driven by the Coverage agent in map A.

In the simulations of Figures 10 and 11, each UAV of the fleet starts from a corner of the unexplored environment. A goal is computed immediately by each one of them through the Coverage agent. The first 3 frames show an interesting behavior by the agent. The waypoint distribution is not optimal, but all the UAVs follow a trajectory that leads them to explore a significant amount of the environment, especially around its center. The green UAV is probably the one that computes the worst waypoints, which are almost always coincident with the red UAV ones (but are far more efficient for the red agent). The first part of the desired collective behavior, that is, to move toward the largest unexplored areas in the environment, emerges satisfyingly. The waypoint distribution is still not optimal, and often the trajectory of two UAVs cross each other, rendering the exploration process sub-optimal. However, the results are promising and encouraging toward further development of the Coverage agent. The last two frames represented show the part of the exploration process that the agent has not learned yet. In fact, after reaching an exploration value of $\sim 70\%$, the goal starts to be placed in the same area by all the UAVs (4th frame) which end up getting stuck in the same location (5th frame). This behavior has been observed in almost all the simulations after a certain exploration threshold has been reached. The reason for this is that the capacity to detect the small, unexplored areas that remain at the end of the exploration is learned by the agent only after getting a grasp of the generic behavior. Since the agent has been learning only for a limited number of episodes due to the slowness of the Coverage training process (as shown in Figure 9), this behavior has not yet been mastered.

IV. Results

In this section, the performances of the Path Planning and Coverage agents will be discussed from a quantitative standpoint. Some metrics have been defined to perform this analysis. Numerical results have been collected about them by running the agents in different test scenarios created using validation maps presented in Section III.

A. Path Planning results

The Path Planning agent performances have been compared with those of other path planning algorithms. In particular, A* and the original implementation of APF[†] have been used. The code used for A* was adapted from https://github.com/AtsushiSakai. The metrics through which each algorithm performances are evaluated are:

- *Goal Reached*: a boolean value indicating whether or not a trajectory reaching the goal has been found;
- ε: the minimum measured distance between the agent and any obstacle point;
- t: the time required by the algorithm to compute the trajectory;
- $\bar{\Delta}_{IO}$: the average value of Δ_{IO}, which is the difference between the "input angle" and "output angle" computed at each point of the trajectory. $\bar{\Delta}_{IO}$ quantifies how sharp are - on average - the turns in a trajectory. A low value of $\bar{\Delta}_{IO}$ is associated with a more difficult trajectory from the dynamical and energetic point of view (i.e., once the UAV dynamics is taken into account the trajectory is more difficult to follow, requiring lower speed or a higher actuation force).

Results from the path planning simulations in the test scenarios are listed in Table 1. For each test performed, the corresponding validation map is indicated (as "A", "B" or "C", with reference to Figure 5) along with the starting position, the goal position (expressed in meters with respect to a reference frame having the origin in the bottom-left corner of each map), and the approximate length of the trajectory l. The value of l is computed as an average of the lengths of the trajectories computed by the three algorithms. The value of the evaluation metrics is reported in the table for each of the Path Planning algorithms considered.

Table 1 Numerical results obtained from the Path Planning simulation in the test environments.

Test	Map	Start/End Point [m]	l [m]	Algorithm	Goal Reached	ε [m]	$\bar{\Delta}_{IO}$ [deg]	t [s]
1	B	$x_0 : (7, 3)$ $x_g : (16, 18)$	20.00	RL	yes	0.72	6.06	1.06
				A*	yes	0.45	10.43	1.66
				APF	yes	0.76	6.81	-
2	C	$x_0 : (17, 4)$ $x_g : (17, 18)$	19.00	RL	yes	0.67	14.17	0.92
				A*	yes	0.42	8.64	0.69
				APF	no	-	-	-
3	A	$x_0 : (17, 4)$ $x_g : (14.5, 13)$	9.80	RL	yes	0.53	10.01	0.49
				A*	yes	0.36	9.84	0.42
				APF	no	-	-	-
4	B	$x_0 : (2, 13)$ $x_g : (14, 9)$	14.00	RL	yes	0.61	15.82	0.70
				A*	yes	0.36	20.61	0.52
				APF	yes	0.60	17.50	-
5	C	$x_0 : (2, 18.8)$ $x_g : (6, 17.2)$	4.75	RL	yes	0.70	8.70	0.258
				A*	yes	0.40	20.54	0.05
				APF	yes	0.58	16.58	-
6	A	$x_0 : (10, 10)$ $x_g : (5, 5)$	8.00	RL	yes	0.61	15.59	0.399
				A*	yes	0.57	10.52	0.218
				APF	yes	0.60	10.89	-

[†]in this case the *APF Path Planning algorithm* has been used [20]. The APF Path Planning algorithm used is based on the numerical research of the minimum APF value in the APF model matrix. The computational time of this kind of algorithm is extremely low (<0.1s per trajectory) since the APF matrix is pre-computed. Exact time values have not been measured and are not reported in the table.

Some interesting results can be observed in Table 1. The RL Path Planning agent is capable of always finding the trajectory, even when the original implementation of the APF path planning algorithm fails to find a suitable path to the goal (as in Tests 2 and 3). The minimum distance from obstacles is always the most conservative one, meaning that the agent can consistently find the goal while keeping a safe distance from obstacles. This behavior does not lead to sub-optimal trajectories, as can be observed in the column reporting the values of $\bar{\Delta}_{IO}$. The average curvature of the trajectory is always similar to the one of the path computed by A*. In some of the tests (e.g. in Test 1) the average angle difference is lower than the one of the other algorithms because the RL agent managed to find a more dynamically-efficient trajectory. This can be observed in the left image of Figure 12, where the three trajectories computed to solve Test 1 are displayed. The RL agent, whose trajectory is represented by the orange line, finds a completely different path from those of the A* and APF algorithms (represented respectively by the blue line and yellow line). The RL trajectory is optimized to minimize the turns, i.e., to optimize the energetic cost of traveling along the trajectory (or, a higher speed can be maintained along the trajectory). In Figure 12, on the right side, the data obtained measuring the computational cost to find each trajectory are plotted. It is worth noting that the computational cost of the RL agent grows linearly with the length of the trajectory, independently from the number of obstacles or the complexity of the environment. On the contrary, A* is far more sensitive to those parameters. The computational cost of A* grows exponentially with the trajectory length and is sensitive to the presence of many obstacles as can be observed in the results of Test 1 in Table 1 (the corresponding point on the plot can be observed approximately above $x = 20$).

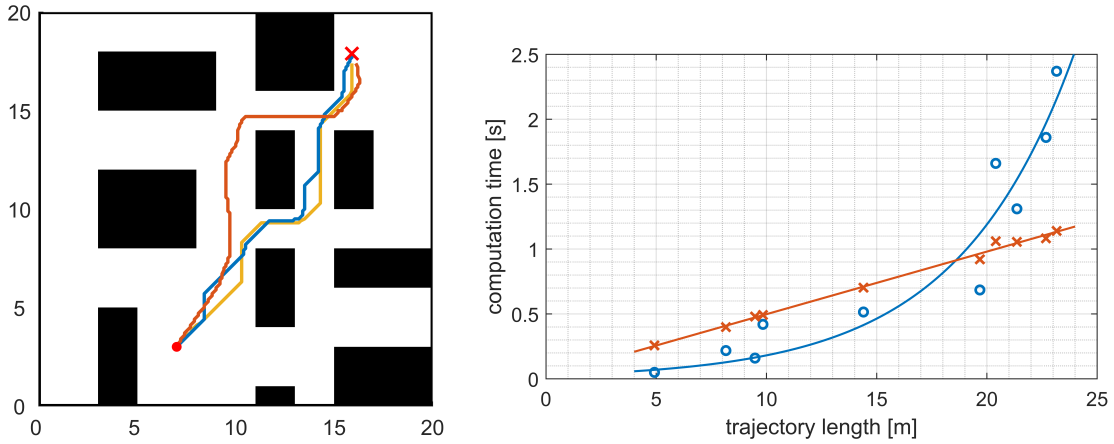

Fig. 12 On the left, the three trajectories computed by the Path Planning algorithms to solve Test 1 (orange is RL, blue is A*, yellow is APF). On the right, a comparison between the computational time of A* (blue) and RL (orange) plotted against the trajectory length.

B. Coverage results

An evaluation process similar to the one conducted on the Path Planning agent has been performed on the Coverage agent. First of all, some metrics have been defined to assess the performances of the Coverage agent. The values that have been taken into consideration are:

- $\bar{\lambda}$: the average distance between the UAVs composing the fleet;
- λ_{min}: the minimum measured distance between two UAVs. Is an indicator of the maximum packing of the UAVs.
- $A\%$: the size of the explored area, expressed as a percentage value with respect to the total map size. This metric is computed both as a collective value (i.e., the area explored by the whole fleet) and an individual value, i.e., $A\%_1$, $A\%_2$...;

Since the Coverage agent is still under development, the values extracted by the exploration process are useful to understand which aspects of the agent behavior need to be improved. The value of the three metrics over time (i.e., over the simulation steps) is represented in Figure 13. In the bottom image, the distance between each couple of UAVs is plotted; the average distance $\bar{\lambda}$ is indicated by the thick line, whereas the value of the minimum distance λ_{min} can be derived by looking at the lowest of the distance lines at each time step. In the top figure, the value of the percentage explored area $A\%$ over time is reported. Both the collective value and the individual contributes of the single UAVs to $A\%$ are represented. As already observed in Figures 10 and 11, after about 85% of the area has been explored the

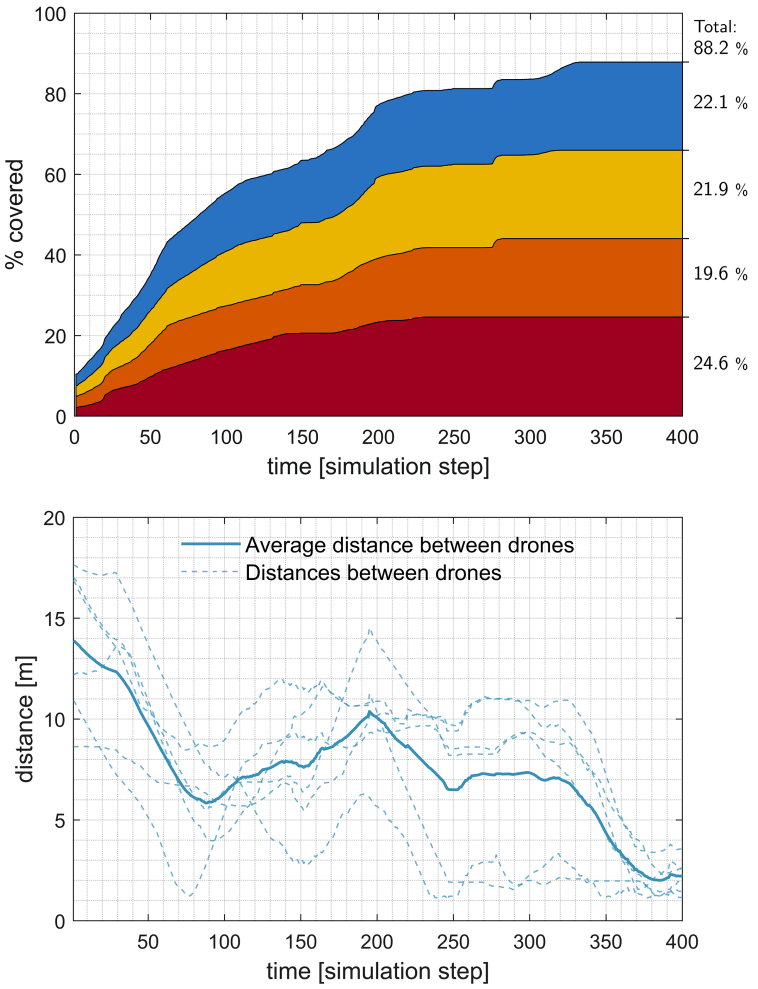

Fig. 13 Evaluation metrics of the exploration simulation of Figures 10 and 11.

Coverage agent is not able to produce new effective waypoints for the UAV, and the $A\%$ curve reaches a plateau. The average individual exploration value $A\%_i$ is around 22%. The exploration is well distributed among all the agents in the initial 350 steps when they are sufficiently distanced. Then, the UAVs group in the same area and the exploration rate drops. It is interesting to notice that, in the middle of the exploration (between step 150 and 200) the fleet manages to assume a more spread formation, as indicated by the growing average distance. This leads to a quick increase in the value of $A\%$, with contributions by all the UAVs. These plots confirm what already observed in the simulations of Figures 10 and 11. The agent has managed to learn an effective (even if not optimal) exploration behavior to cover the first $\sim 75\%$ of the exploration process, but still lacks the experience to properly manage the final steps. This is the topic toward which the future development of the Coverage agent will focus. The results just discussed are confirmed also by other simulations performed in the test environments. Table 2 collects the results obtained from the simulation of the exploration process in all the three validation maps.

Table 2 Numerical results obtained from the Coverage simulation in the test environments.

Test	Map	$\bar{\lambda}$ [m]	λ_{min} [m]	$A\%$	$t_{50\%}$ [step]	$t_{70\%}$ [step]	$t_{80\%}$ [step]
1	A	7.49	1.02	88.05	83	185	222
2	B	9.24	1.01	83.43	97	167	231
3	C	9.47	0.45	70.30	75	200	-

The data reported in Table 2 are defined in the following way:

- $\bar{\lambda}$: the average distance between the agents during the exploration process;
- $\lambda_{\min}$: the minimum measured distance between two agents during the exploration process;
- $A\%$: the percentage coverage reached by the Coverage agent after 400 simulation steps;
- $t_{N\%}$: the time (expressed in simulation steps) after which a certain exploration threshold $N\%$ has been reached.

As can be observed in the table, the Coverage agent efficiency decreases as the map complexity grows (see Figure 5 for details about each map complexity). The average distance between the UAVs is similar in the three cases, whereas in the third one the minimum distance between the agents is smaller than in the other two. The reason for this is that the UAV fleet is still not efficient at moving in the middle of an obstacle-rich environment and so the UAVs end up getting closer.

V. Conclusions and further development

An effective RL-based Path Planning agent has been designed and implemented. The results show that it is an efficient algorithm, capable of computing safe and dynamically-efficient trajectories also in obstacles-rich environments. The Coverage agent designed to manage the high-level exploration operations shows some positive early results but needs to be refined to become completely effective in the exploration of unknown environments. The main open point regards the further training of the Coverage agent. Further investigation of the reward function and NN structure design is programmed, as well as the tuning of the training hyperparameters. It is also possible that the overall architecture of the training process will be modified (mainly to implement some MARL-specific techniques) if it becomes evident that the current one is not suitable to obtain the desired behavior). A second open point regards the continuation of the Path Planning training. In this case, the main goal is to obtain a single agent capable of managing all the local minima while being able to compute optimal paths (i.e., a single agent equivalent to the union of the two Path Planning agents used for the simulations above). The addition of a second output node to the Path Planning agent NN, through which the UAV speed in each point of the trajectory can be computed, is also being considered.

Acknowledgments

We would like to thank HPC@POLITO for providing a significant part of the computational resources used in the training of the agents. HPC@POLITO is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://hpc.polito.it).

References

[1] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, 2016.

[2] Cabreira, T. M., Brisolara, L. B., and Ferreira Paulo, R., "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, Vol. 3, No. 1, 2019, pp. 1–38. https://doi.org/10.3390/drones3010004.

[3] Valavanis, K. P., and Vachtsevanos, G. J., *Handbook of Unmanned Aerial Vehicles*, Springer Netherlands, Dordrecht, 2015. https://doi.org/10.1007/978-90-481-9707-1.

[4] Barve, A., and Nene, M., "Survey of Flocking Algorithms in Multi-agent Systems," *International Journal of Computer . . .* , Vol. 10, No. 6, 2013, pp. 110–117.

[5] Brust, M. R., and Strimbu, B. M., "A networked swarm model for UAV deployment in the assessment of forest environments," *2015 IEEE 10th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2015*, 2015. https://doi.org/10.1109/ISSNIP.2015.7106967.

[6] Dai, F., Chen, M., Wei, X., and Wang, H., "Swarm Intelligence-Inspired Autonomous Flocking Control in UAV Networks," *IEEE Access*, Vol. 7, 2019, pp. 61786–61796. https://doi.org/10.1109/ACCESS.2019.2916004.

[7] Yasin, J. N., Mohamed, S. A., Haghbayan, M. H., Heikkonen, J., Tenhunen, H., and Plosila, J., "Navigation of Autonomous Swarm of Drones Using Translational Coordinates," *Lecture Notes in Computer Science*, Vol. 12092 LNAI, 2020, pp. 353–362. https://doi.org/10.1007/978-3-030-49778-1.

[8] Yasin, J. N., Sayed Mohamed, S. A., Haghbayan, M. H., Heikkonen, J., Tenhunen, H., Yasin, M. M., and Plosila, J., "Energy-efficient formation morphing for collision avoidance in a swarm of drones," *IEEE Access*, Vol. 8, 2020, pp. 170681–170695. https://doi.org/10.1109/ACCESS.2020.3024953.

[9] Innocente, M. S., and Grasso, P., "Self-organising swarms of firefighting drones: Harnessing the power of collective intelligence in decentralised multi-robot systems," *Journal of Computational Science*, Vol. 34, 2019, pp. 80–101. https://doi.org/10.1016/j.jocs.2019.04.009.

[10] Eberhart, R. C., Shi, Y., and Kennedy, J., *Swarm Intelligence*, 1st ed., Morgan Kaufmann, 2001.

[11] Parunak, H. V., Purcell, M., and O'Connell, R., "Digital Pheromones for Autonomous Coordination of Swarming UAV's," *1st UAV Conference*, American Institute of Aeronautics and Astronautics, Reston, Virigina, 2002, pp. 1–9. https://doi.org/10.2514/6.2002-3446.

[12] Alfeo, A. L., Cimino, M. G., De Francesco, N., Lazzeri, A., Lega, M., and Vaglini, G., "Swarm coordination of mini-UAVs for target search using imperfect sensors," *Intelligent Decision Technologies*, Vol. 12, No. 2, 2018, pp. 149–162. https://doi.org/10.3233/IDT-170317.

[13] Kuyucu, T., Tanev, I., and Shimohara, K., "Superadditive effect of multi-robot coordination in the exploration of unknown environments via stigmergy," *Neurocomputing*, Vol. 148, 2015, pp. 83–90. https://doi.org/10.1016/j.neucom.2012.07.062.

[14] Gronauer, S., and Diepold, K., "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, Vol. 2, No. 0123456789, 2021. https://doi.org/10.1007/s10462-021-09996-w.

[15] Buşoniu, L., Babuška, R., and De Schutter, B., "Multi-agent Reinforcement Learning: An Overview," *Technology*, Vol. 38, Springer, Berlin, Heidelberg, 2010, pp. 183–221. https://doi.org/10.1007/978-3-642-14435-6{_}7.

[16] Huang, Y., Wu, S., Mu, Z., Long, X., Chu, S., and Zhao, G., "A Multi-agent Reinforcement Learning Method for Swarm Robots in Space Collaborative Exploration," *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, IEEE, 2020, pp. 139–144. https://doi.org/10.1109/ICCAR49639.2020.9107997.

[17] Pham, H. X., La, H. M., Feil-Seifer, D., and Nefian, A., "Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage," *CoRR*, Vol. abs/1803.0, 2018.

[18] Adepegba, A. A., Miah, S., and Spinello, D., "Multi-agent area coverage control using reinforcement learning," *Proceedings of the 29th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016*, 2016, pp. 368–373.

[19] Xiao, J., Wang, G., Zhang, Y., and Cheng, L., "A Distributed Multi-Agent Dynamic Area Coverage Algorithm Based on Reinforcement Learning," *IEEE Access*, Vol. 8, 2020, pp. 33511–33521. https://doi.org/10.1109/ACCESS.2020.2967225.

[20] Khatib, O., "Real-time obstacle avoidance for manipulators and mobile robots," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, Vol. 2, Institute of Electrical and Electronics Engineers, 1985, pp. 500–505. https://doi.org/10.1109/ROBOT.1985.1087247.

[21] Yao, Q., Zheng, Z., Qi, L., Yuan, H., Guo, X., Zhao, M., Liu, Z., and Yang, T., "Path Planning Method with Improved Artificial Potential Field - A Reinforcement Learning Perspective," *IEEE Access*, Vol. 8, 2020, pp. 135513–135523. https://doi.org/10.1109/ACCESS.2020.3011211.

[22] Matoui, F., Boussaid, B., and Abdelkrim, M. N., "Local minimum solution for the potential field method in multiple robot motion planning task," *2015 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, IEEE, 2015, pp. 452–457. https://doi.org/10.1109/STA.2015.7505223.

[23] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.