

A Delay-Aware DRL-Based Environment for Cooperative Multi-UAV Systems in Multi-Purpose Scenarios

Damiano Brunori^a and Luca Iocchi^b

Department of Computer, Control and Management Engineering (DIAG), Sapienza University of Rome, Rome, 00185, Italy


Keywords: Deep Reinforcement Learning (DRL), Delay-Aware Environment, Multi-UAV Cooperation.


Abstract: We provide a customizable environment based on Deep Reinforcement Learning (DRL) strategies for handling cooperative multi-UAV (Unmanned Aerial Vehicles) scenarios when delays are involved in the decision-making process for tasks such as spotting, tracking, coverage and many others. Users can choose among various combinations of tasks and parameters and customize the scenarios by implementing new desired functionalities. This environment provides the opportunity to compare different approaches, taking into account either implicitly or explicitly the delays applied to actions and observations. The awareness of the delay, along with the possible usage of real-world-based external files, increases the reality level of the environment by possibly easing the knowledge transferability process of the learned policy from the simulated environment to the real one. Finally, we show that use cases could generate new benchmarking tools for collaborative multi-UAV scenarios where DRL solutions must consider delays.

1 INTRODUCTION

Many of the most recent studies involving emergent scenarios are progressively investigating the usage of Unmanned Aerial Vehicles (UAVs) to achieve different and relevant aims in different application fields. Indeed, UAVs are recently designed for supporting Internet of Thing (IoT) applications (Cheng et al., 2023), firefighting tasks (Peña et al., 2022), health-care items delivery (Scott and Scott, 2017), and many others. In order to complete the desired task with the most efficient strategy, cooperation among UAVs is undoubtedly crucial, especially for persistent and multi-target missions. Meanwhile, Deep Reinforcement Learning (DRL) techniques showed great results in handling different tasks by successfully playing ATARI games and solving the Rubik's cube, and hence, they are now being investigated to be applied to more and more dynamic and complex tasks such as those involving multi-UAV scenarios (Moon et al., 2021; Wang et al., 2019). Nevertheless, most of the works associated with DRL techniques are still based on a strong assumption which could affect real-world application deployment when complex tasks must be accomplished. Indeed, delays occurring when receiv-

ing observations or executing actions are seldomly taken into account during the learning process. In particular, to the best of our knowledge, obtained during the preparation of a survey on DRL techniques for multi-UAV applications (Frattolillo et al., 2023), delays are never taken into account when considering multi-agent DRL-based systems with agents represented by UAVs; hence the importance to consider the delay in this application field. Multi-UAV systems thought to be deployed in real-world scenarios are characterized by many hardware constraints mainly due to their design, which often reflects the nature of their application. In most cases, some real constraints are not taken into account in the considered virtual framework, neither in the scenario settings nor in the learning process. This approach can be successful in some cases but not in others. In a real-world scenario, not considering the delay can still be a valid assumption whenever the agents' actions duration perfectly fits each agent's environment sensing frequency (sampling capacity). Nonetheless, the latter condition is very unlikely to occur, both when dealing with single-agent systems where low-level control latency is crucial (Chen et al., 2021) and when multiple agents are involved in the learning process, resulting in a massive propagation of the actions execution time and hence of the observations. In order to properly consider the impact of delays in multi-

^a  <https://orcid.org/0000-0002-0384-3498>

^b  <https://orcid.org/0000-0001-9057-8946>

UAV applications and to assess the performance of solutions dealing with this problem, a framework to develop experiments and to compare the results of different approaches is needed to make efforts in the field more effective. To this end, the main contributions of this paper are given by the definition and implementation of a 3D customizable and user-friendly environment aimed at increasing the reality level of multi-UAV scenarios for coverage, spotting and tracking tasks by considering the delay on both the actions and the observations when DRL approaches are used as solution techniques. The environment is called Delay-Aware Multi-Aerial Navigation (DAMIAN)¹, and it is inspired and extends our previous work in multi-UAV reinforcement learning and path planning (Brunori et al., 2021b; Brunori et al., 2021a) and the one for air traffic control (Dalmau and Allard, 2020). DAMIAN is provided with two well-known DRL algorithms, i.e., Proximal Optimization Policy (PPO) (Schulman et al., 2017) and Soft-Actor Critic (SAC) (Haarnoja et al., 2018), and their delay-aware variations. Some use cases are provided and could be used as a benchmarking tool for specific application scenarios. Through external files following the Euro-control official format, we can reproduce real operative areas involving multi-UAV systems, hence helping the benchmark analysis for different DRL algorithms applied to a desired scenario. Finally, since the environment is designed to be easily extended and editable, it can also be used to reproduce other different multi-UAV applications.

2 RELATED WORKS

UAV applications using DRL techniques have been increasing in the last few years. Area coverage tasks are the most common, where UAVs can be used for different aims such as Mobile Crowd Sensing (MCS) operations (Liu et al., 2019) or in leader-follower situations (Mou et al., 2021): they mostly apply a learning paradigm based on Centralized Training and Decentralized Execution (CTDE), but using different DRL algorithms. Many other applications studying the combined usage of multi-UAV systems with DRL techniques are mainly represented by communication and computation offloading tasks, where communication support (Zhu et al., 2021) and allocation strategies for the available resources (Sacco et al., 2021) are investigated, respectively. The former uses a modified version of the Multi-Agent Deep Deterministic

¹<https://github.com/DamianoBrunori/DAMIAN-Delay-Aware-Multi-Aerial-Navigation-DRL-based-environment>

Gradient (MADDPG) algorithm in a 3D environment by using a CTDE learning paradigm, whilst the latter applies a decentralized paradigm involving a large number of agents (up to 200). Nevertheless, some strong assumptions are still made for multi-UAV systems relying on a DRL algorithm. Hence, we allow you to remove one of those main assumptions, namely that there are no delays in receiving observations (or rewards) or executing actions. More in detail, the implemented environment allows the application of a delay on both the actions and the observations, contrary to that described by Chen et al. (Chen et al., 2021), which only applies the action delay. The multi-agent case with delays is also handled (Chen et al., 2020), but the use cases considered are already set and it could be difficult to customize them by directly modifying the source code. Also, another work (Yuan et al., 2023) considers use cases obtained by code modification of existing environments.

Instead, the environment proposed in this paper allows us to configure new scenarios and learning features easily through user-friendly configuration files. Differently from other works dealing with delays (Agarwal and Aggarwal, 2021; Arjona-Medina et al., 2019), 3D environments can be tested as we are dealing with UAVs, and hence, the takeoff, flight and landing phases need to be taken into account when the action execution delay is applied to the considered scenario. The user can also set different clocks for the agents (which are supposed to be homogeneous), for the central node system communicating with them, and for the simulation step. Furthermore, none of the available delay-aware MDP frameworks handles external files containing information from a real scenario: DAMIAN also provides this new feature that allows learning a policy performing the simulated training phase directly on a real-world operative polygon based on latitude and longitude coordinates. Finally, hyperparameters tuning is also provided.

3 DELAY-AWARE MULTI-UAV ENVIRONMENT

In this section, after defining some necessary notions related to a delay-aware MDP, we describe the main structure of the proposed Environment.

3.1 Background Notions

Based on the single-agent MDP definition (Puterman, 1990), we can define a single-agent Delay-Aware MDP (DA-MDP) (Chen et al., 2021) as a tu-

ple $\langle \mathbf{X}, \mathbf{A}, \mathbf{T}, \mathbf{R} \rangle$ augmenting the single-agent MDP $\langle \mathbb{S}, \mathbb{A}, \mathcal{T}, \mathcal{R} \rangle$ such that:

- $\mathbf{X} = \mathbb{S} \times \mathbb{A}^d$ is a set of states $x_i \in \mathbb{X}$, where d indicates the maximum delay step that an agent can store;
- $\mathbf{A} = \mathbb{A}$ is the action space;
- $\mathbf{T}(x_{t+1}|x_t, a_{t+d}) = \mathbf{T}(s_{t+1}, a_{t+1}^{t+1}, \dots, a_{t+d}^{t+1}|s_t, a_t^t, \dots, a_{t+d-1}^t, a_{t+d})$ is the probability transition function. The action's subscript shows the action's completion time, whilst the action's superscript indicates when the action started (when the superscript is not specified, then it means that the corresponding action has been chosen at time t);
- $\mathbf{R}(x_t, a_{t+d}) = R(s_t, a_t, \dots, a_{t+d-1}, a_{t+d})$ is the reward function.

Note that for any possible delay D , if $D > d$, then some information is lost. \mathbb{S} and \mathbb{A} are the set of states (with $s_i \in \mathbb{S}$) and actions (with $a_i \in \mathbb{A}$), respectively, of the classical MDP; the mapping function associated with its transition function is expressed as $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$, whilst the function related to its reward is defined by $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. Thus, the state space of a DA-MDP is augmented by an actions' sequence that is executed in the next $d \in \mathbb{N}$ steps, representing the action delay: the action a_{t+d} starts at time t , but its execution ends at time $t+d$. The augmented MDP (Katsikopoulos and Engelbrecht, 2003) helps us to avoid getting arbitrary sub-optimal policies due to some hidden states (Singh et al., 1994). The concept of delay-aware MDP can be applied to the multi-agent case, i.e., to a Markov game (Littman, 1994), obtaining a Delay-aware Markov Game (DA-MG) (Chen et al., 2020) with a larger decrease in the system performance depending on the number of the agents N . The Delay-aware Markov Game is defined as a tuple $\langle \mathbf{P}, \mathbf{X}, \mathbf{A}, \mathbf{T}, \mathbf{R} \rangle$ augmenting the Markov Game $\langle \mathbb{P}, \mathbb{X}, \mathbb{A}, \mathcal{T}, \mathcal{R} \rangle$ such that:

- \mathbf{P} has the same meaning as \mathbb{P} in the classic Markov Game, i.e., the number of $p \in \mathcal{N}$ agents;
- $\mathbf{X} = \mathbb{S} \times \mathbb{A}_1^{d_1} \dots \times \mathbb{A}_N^{d_N}$ where d_i indicates the maximum delay step that can be stored by the agent i , while \mathbb{A}_i is the action space of the agent i described as in the DA-MDP;
- $\mathbf{A} = \mathbb{A}$ is the action space, where $\mathbb{A} = \{\mathbb{A}_1, \dots, \mathbb{A}_N\}$;
- $\mathbf{T}(x_{t+1}|x_t, a_{t+d}) = \mathbf{T}(s_{t+1}, a_{t+1}^{1,t+1}, \dots, a_{t+d_1}^{1,t+1}, \dots, a_{t+1}^{N,t+1}, \dots, a_{t+d_N}^{N,t+1}|s_t, a_t^{1,t}, \dots, a_{t+d_1-1}^{1,t}, \dots, a_{t+d_N-1}^{N,t}, a_{t+d})$ is the probability transition function, where the superscripts $i = 1, \dots, N$ indicate the agent i , and the rest of the notation is the same as in the DA-MDP case;

- $\mathbf{R}_i(x_t, a_{t+d}^i) = R_i(s_t, a_t^i, \dots, a_{t+d-1}^i, a_{t+d}^i)$ for the agent i ; similarly to the single-agent case, a_{t+d}^i means that the action of the agent i starts at t and its execution ends at $t+d$.

Now, we will see different types of delays with their consequences and how they could be handled. We start with a **direct action delay**, i.e., when the chosen action needs some time to be executed. It leads to: i) *indirect observation delay* as new observation will be generated only after executing a time-delayed action; *indirect reward delay*, indeed the reception of the reward is delayed as we cannot assign a reward until the effect of the action execution is not observable. The second possible delay is the **direct observation delay**, bringing with it: i) *incomplete observation* due to the agents that cannot access all the information at the current instant as some may not be available yet; ii) *obsolete information availability* as some of the available observations may not be updated yet, and hence, they refer to an old state; iii) *indirect reward delay* because of the inability to assign a reward until the observation is available; iv) *direct affection on action selection*, indeed actions are chosen based on what the agent can observe, and if the agent can only look at old observations at the current instant, then the action selection will be based on outdated information. Finally, there is a **direct reward delay**. Some additional rewards, either positive or negative, can occur according to some circumstances which do not strictly depend on the delays associated with actions or observations (e.g., progressive malfunction of onboard mechanical components). A time-delayed reward is characterized by a *direct affection on action selection*.

3.2 Environment Overview

In the proposed Delay-Aware multi-UAV environment, the agents, i.e., the UAVs, can complete different cooperative tasks for complexity and heterogeneity. The environment is compatible with the OpenAI Gym interface² for easy scalability, deployability, flexibility and extendability. The overall goal of the agents can be selected at wish by choosing one, some or all of the desired tasks to be completed, and hence modifying the scenario complexity accordingly. For instance, a quite complex scenario can involve a dynamic and discontinuous signal source that needs to be spotted and tracked by maximizing the area coverage while minimizing the battery consumption. It is known that an action delay leads to an observation delay, and hence, these two delays are mathe-

²<https://www.gymnasium.dev/index.html#>

matically equivalent (Katsikopoulos and Engelbrecht, 2003). Nevertheless, a crucial difference arises when the learning process takes place. Indeed, when an action delay occurs, the effect of the delayed action that has been executed could be observed only in the future. However, once the agents receive that observation, they will be able to access the present state of the environment at that future time. Conversely, when an observation delay occurs, the agents will be able to observe the past environment state and not how it looks when the agents are processing the observation. Thus, an action delay provides an (implicit and delayed) observation updated to the present time, whilst an observation delay provides information only about the past and not the present time. This relevant difference between these two delays can be crucial if not explicitly considered by a delay-aware (D)RL algorithm.

A straightforward overview of the workflow of the DAMIAN environment is shown in Figure 1, better explained in the rest of this section. From the environment structure, it is possible to notice that we chose to distribute a single policy with a CTDE learning paradigm in order to allow the system to be scalable to real-world applications, where the number of agents could differ from that used during the training phase. With the same aim, a server node is supposed to be used to receive information from all the agents, to process the information by making them independent from the number of the agents, and to share

the newly processed information with all the agents. The agents can exchange information bidirectionally with the server node by selecting an action only when landed: when considered, the agents' battery can be charged whenever they land as they are supposed to be provided with solar panels. In any case, this setting can be easily changed or re-implemented by limiting the possibility to charge only to specific areas where charging stations can be placed. In order to increase the motion reality level, the motion of the UAVs is modelled through polynomial trajectories (3rd, 5th, 7th degree) that can be chosen based on the constraints to be applied to the agents (e.g., maximum allowed velocity). The agents are supposed to travel a rest-to-rest trajectory between the takeoff and the landing point, stopping at their assigned flight level after the takeoff phase and immediately before starting the landing phase (which is typical for quadcopters' systems). A controller is assumed to be able to always interpolate all the points of a selected trajectory, which is defined as:

$$Traj_i = \left\{ \begin{array}{l} P_1(x_t, y_t, 0), P_2(x_t, y_t, FL_i), \\ P_3(x_l, y_l, FL_i), P_4(x_l, y_l, 0) \end{array} \right\} \quad (1)$$

where $i = 1, \dots, N$, with N indicating the number of agents, FL_i representing the flight level associated with the agent i , and (x_t, y_t) and (x_l, y_l) denoting the 2D coordinates related to the takeoff and landing, respectively. The signal source, which can possibly be set in spotting and tracking tasks and which moves

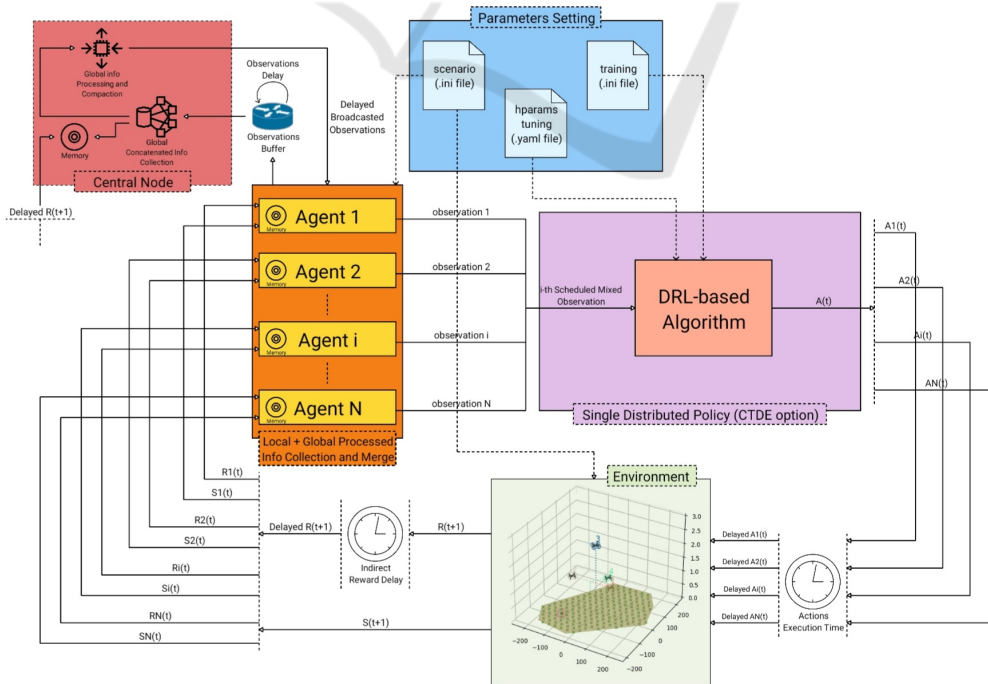


Figure 1: Structure overview of DAMIAN environment.

according to a trapezoidal speed profile, is modelled through the general *inverse-square law*. In particular, in the DAMIAN environment, we assume a spherical signal source at a specific distance r with the following intensity:

$$I = P/(4\pi r^2) \quad (2)$$

where P is the power of a source point. Now, based on the minimum intensity level I_{mic} that a UAV's microphone can detect, we can compute the radius R_f of the agents' sensing range (footprint) as follows:

$$R_f = \sqrt{P_{mic}/(4\pi I_{mic})} \quad (3)$$

where P_{mic} represents the power of the sound source detectable by the microphone that has been used. Finally, the area and the shape of the desired operative polygon can be either randomly generated or manually provided using an external file based on i) the official Eurocontrol format for the Flight Information Region (FIR)³ or on ii) a specific area of interest drawn from the map of a real-world zone through the *GeoJSON* online editor⁴. The Haversine equations allow us to produce a realistic response suitable for the latter cases based on real-world operative polygons.

3.3 Delayed Actions Handling

We assume that every perturbation which could affect the interpolation of the waypoints chosen by the UAV's policy is negligible. We do not need an action buffer (Chen et al., 2021) to execute the actions "instantaneously" after the delay associated with them is passed, as the agents' motion is already modelled in DAMIAN: we can keep track of the environment changes even during the execution of every UAVs' action. The single-agent solution proposed by Chen et al. (Chen et al., 2021) does not suit our multi-UAV system, which could suffer from scalability issues: they store in the state vector both the current observation and some of the last n actions executed by the agents, and hence, the more the number of the agents the larger would be the state space. However, we can adapt this solution in such a way as to avoid this issue by using a CTDE learning paradigm, where the past n actions are stored inside the actor network. Thus, the old actions stored in the observation space are considered only locally to avoid any dependence on the number of agents and the global system information. Including the last N actions in the state vector, we respect the *memoryless* Markov property associated with the MDP model of the environment: the features

³<https://www.eurocontrol.int/publication/rnd-data-archive-structure-and-sample>

⁴<https://geojson.io/#map=2/0/20>

of random variables associated with the future depend on observations about the current time and not related to the past. If the last N actions were not part of the state, then the reward would depend on changing and hidden variables from the agent.

3.4 Delayed Observations Handling

As already seen in Section 3.1, the observations' delays lead to a state space which could be either partial or obsolete. To mitigate this issue, we can introduce two memory buffers at global (server node memory) and local (UAV memory) levels, storing the observations and rewards history. This structure allows us to update (back in time) some possible delayed observations by assigning them to the proper time instant according to their Age of Information (AoI); afterwards, we can recompute the past rewards accordingly. The learning step is performed only after N steps in such a way as to let the *backupdating process* (Figure 2) be performed on the stored observations and rewards before they are fed as input into the learning procedure.

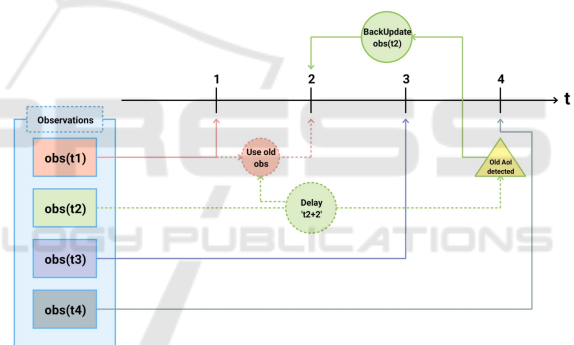


Figure 2: Observations' backupdating process.

This approach can be considered a *reward redistribution* technique, which is different from the *reward shaping* as the latter adds a new reward to an existing one, while the former completely replaces the old reward with a new one. In particular, in our case, we totally recompute the global observation (and reward accordingly) based on the proper Age of Information assigned to the delayed observation currently received (see Figure 2). The RUDDER algorithm (Arjona-Medina et al., 2019) instead does not *backupdate* the state space and the reward based on the AoI of the current observations. However, it applies supervised learning through past experience to estimate the future average cost by including it in the reward to achieve a zero value for the future expected reward. The reward used in DAMIAN can be considered as *non-Markovian*, i.e., similar to a situation representing a case in which the agent can receive the reward

only after having completed a series of previous tasks (Rens. et al., 2021). In our case, we do not have different tasks, but we have an implicit delayed reward due to either actions and/or observations delays; for both cases, the agents will obviously experience a delay in receiving the reward. The non-Markovian property deriving from the delayed reward is overcome by considering the past actions in the state vector and by the observation *backupdating* when the agents experience an action and an explicit observation delay, respectively. In particular, when dealing with an explicit observation delay, the selection of the learning time instant is crucial for smoothing the non-Markovian reward property as the learning step can exploit the *backupdating* either at best or partially or not at all. The possible delays that can be applied in our environment are shown in Figure 3, where only two agents are shown for clarity: violet colour refers to the first agent, whilst green indicates the second agent.

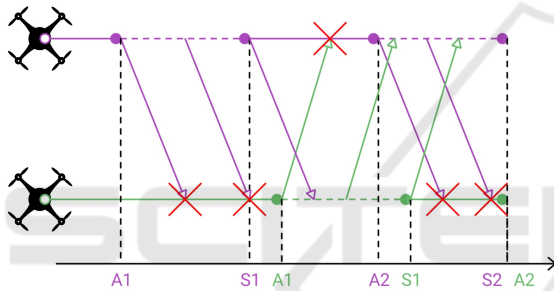


Figure 3: Possible environment delays.

In this Figure, A_i, S_i (with $i = 1, 2$ representing the sequential number of the agents' actions) are the ending time of movement actions (horizontal solid line) and sensing actions (horizontal dotted line), respectively. The latter depends on the time the agent is supposed to perceive the environment and then wait before taking the next action. Oblique lines indicate agents' exchanging process that can only occur when agents land and sense the environment.

3.5 Delayed Rewards Handling

We know that a proper reward can be shaped based on the reward function $r(t)$ associated with the desired task to be performed. Now, an over time-deteriorating reward function $R(t)$ can take into account the reward delay deriving from the time needed to execute the actions:

$$R(t) = \sum_{t=1}^T \gamma^{T-t} r(t), \quad \text{with } 0 < \gamma \leq 1 \quad (4)$$

where t is the current time instant, T is the maximum number of time instants the agent can look

back at, and γ is a discount coefficient composing the relevance term $\Gamma = \gamma^{T-t}$. The more recent the reward is, the more weight the relevance term will have on the total reward $R(t)$. Defining the reward variation between two subsequent time instants $\Delta_R = |R(t-1) - R(t)|$, we can also observe that the closer the discount factor γ is to 0, the larger Δ_R will be, and vice versa. A cumulative reward function, as in (4), seems to be an efficient way to consider the implicit delay that can occur at the reward level (Kim, 2022).

3.6 Clocks

DAMIAN environment is provided with a default sampling step corresponding to the simulation step, which is, in turn, the same as those associated with the server node and the agents. The user can arbitrarily set the simulation and the server (and agents) sampling steps; however, they must be multiples of each other. The system clock can also be set, allowing one to specify when collecting information for the learning procedure from both the server node and the agents. Thus, the environment can be sampled at different time-step levels (Figure 4): i) simulation; ii) system; iii) server node; iv) agents.

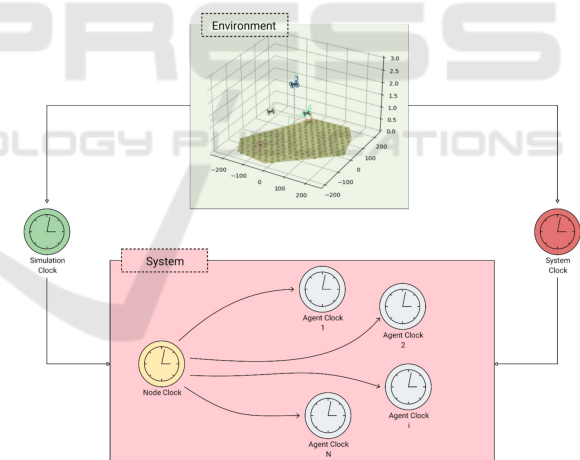


Figure 4: Sampling Features DAMIAN environment.

Indeed, four different clocks with different (or equal) sampling frequencies can be set. To the best of our knowledge, there is no such multi-UAV MDP framework allowing for different clock selections: this can help better exploit the backupdating process when observation delays are involved. The system clock frequency allows collecting the observations to be stored for the learning phase and can be a constant or variable.

4 CUSTOMIZABLE USE CASES

Three configuration files can be easily used to change and mix several scenarios and learning options (Table 1): a .yaml file also allows for hyperparameters tuning through the *W&B* platform ⁵.

Table 1: A non-exhaustive list of possible settings.

Scenario Settings	
Parameter definition	Unit Measure
Simulation step	$s \vee m \vee h$
Min/Max operative area	m
Number of agents	1
Min/Max flight level	m
Max agents velocity	m/s
Max agents acceleration	m/s^2
Power battery capacity	KWh
Battery charging power	KW
Min microphone threshold	dB
Max source velocity	m/s
Max source acceleration	m/s^2
Training Settings	
Parameter definition	Domain
Action space	$\{\bigcup_{i=1}^2 a_i : a_i \in \mathbb{N}, \mathbb{R}\}$
Observation space	$\{\bigcup_{i=1}^{12} o_i : o_i \in \mathbb{R}, \mathbb{N}, \mathbb{B}\}$
Learning rate	$\mathbb{R} \in [0, 1]$
Discount factor	$\mathbb{R} \in [0, 1]$
Number of epochs	\mathbb{N}

4.1 Scenario and Learning Settings

The complexity level of the operative scenario can be modified by simply adding, removing or changing some parameters in the corresponding configuration file. Other flag values (in addition to those shown in Table 1) can be set at wish to select: i) a static or dynamic motion of the source; ii) a constant or dynamic signal emission from the source; iii) a delay either during the action execution and/or in receiving the observations; iv) an explicit clock system or not. The learning phase is also customizable. Considering specific observations leads to achieving different goals according to different pre-built reward functions. Furthermore, other flag values can be used by enabling/disabling: i) the delay either during the action execution and/or in receiving the observations; ii) specific terminal conditions for ending the episodes (e.g., time failure, battery failure); iii) a CTDE learning paradigm; iv) an option to use or not a cumulative reward defined as in (4).

⁵<https://wandb.ai/site>

4.2 MDP Main Features

The main features of the MDP used in the DAMIAN environment are dealt with in more detail by describing the actions, observations and rewards available for the agents (Table 2 shows the notation used to define the rewards).

Table 2: List of the available observations for the system (global) and the UAVs (local): they can be modified, combined, reduced or increased at wish. *Min.* stands for Minimum, *dist.* is the distance, *CA* indicates the Closest Agent, *det.* is detection, and *avg* stands for average. Finally, *SS agents* are the Source-Spotting agents, whilst the *NS agents* are the source Not-Spotting agents.

Global Observations		
Name	Notation	UM
Min. agent-source dist.	d_s	m
CA-source bearing	b_s	deg
CA-source drift	θ_s	deg
Time since source det.	t_s	$s \vee m \vee h$
Agents distances	d_a	m
Agents coverage area	c_a	m^2
Avg dist. of all agents	d	m
Avg dist. of NS agents	d_{nss}	m
Avg dist. of SS agents	d_{ss}	m
N. of SS agents	N_{ss}	1
Local Observations		
Name	Notation	Domain
Battery	b	\mathbb{N}
Spotting the signal source	k	\mathbb{B}

The actions can be either discrete or continuous, picking a landing point for the agents through a two-choice selection associated with the distance to travel and the orientation angle of the agents. Both observations and actions are normalized in the interval $[0, 1]$. For what concerns the reward design, the whole system reward R^s , i.e., computed based on information coming from all the agents, is defined as follows:

$$R^s = \sum_{i=1}^N R_i^s \quad (5)$$

where R_i^s is the system reward i . Indicating with w_x the weight of the reward x , we define the reward for

the whole system as follows:

$$R_i^s = \begin{cases} \begin{cases} w_{d_s}(1-d_s) & l=0 \\ w_{d_s}(1-d_s)l^{-1} & l \neq 0 \end{cases} & i=1 \\ w_{t_s}(1-t_s) & i=2 \\ w_{b_s}(1-b_s) & i=3 \\ \begin{cases} w_{t_s}(2\theta_s) & \theta_s \in [0, 0.5] \\ w_{t_s}(1-\theta_s) & \theta_s \in (0.5, 1] \end{cases} & i=4 \\ w_{ss}[1 - (M_{ss} - \frac{N_{ss}}{N})] & i=5 \\ w_{avg}[1 - (D-d)] & i=6 \\ w_{nss}[1 - (D_{nss} - d_{nss})] & i=7 \\ w_{ss}[1 - (D_{ss} - d_{ss})] & i=8 \end{cases} \quad (6)$$

where D indicates a desired value associated with the considered reward, and l represents the number of motions (including holding the position) before spotting the source. The drift angle $\theta \in [-\pi, \pi]$ is computed as the difference between the bearing angle (between the signal source and the agent) and the agent orientation. M_{ss} is the desired percentage of agents spotting the signal source, whilst N is the total number of agents. The rewards R_i^s , with $i = 1, \dots, 3$, refer to rewards encouraging to reduce the distance, the bearing angle and the last detection time, respectively, w.r.t. the signal source. The reward R_4^s indicates a reward aiming at reducing the drift angle w.r.t the signal source, whilst R_5^s is a reward ensuring a desired number of agents spotting the signal source at the same time. The last system rewards available R_i^s (with $i = 6, \dots, 8$) enforce the distance values d, d_{nss}, d_{ss} to be consistent with D, D_{nss}, D_{ss} . The latter values represent the desired average agents' distances among all the agents in general, all non-source-spotting agents, and all the source-spotting agents, respectively. Concerning the rewards i related to the individual agent a , they are defined as follows:

$$R_i^a = \begin{cases} w_{c_a}c_a^i & i=1 \\ \begin{cases} w_k k & l=0 \\ w_k \frac{k}{l} & l \neq 0 \end{cases} & i=2 \\ w_b b & i=3 \end{cases} \quad (7)$$

The individual rewards R_1^a, R_2^a, R_3^a are related to the area coverage, signal source detection, and the battery consumption tasks, respectively; l has the same meaning as in (6). Each area coverage is computed by excluding possible overlaps between the current agent area i and both the operative polygon area and other agents' areas j ($j \neq i$); self-overlaps w.r.t. previous positions are also considered. Finally, based on (5), (7), the total reward for each agent i is as follows:

$$R_{tot}^i = w_a \sum_{i=1}^3 R_i^a + w_s R^s \quad (8)$$

4.3 Solution Approach

In order to explicitly introduce the delay in the learning process, we can apply the approaches described in Sections 3.3, 3.4, 3.5 to the standard DRL-based algorithms versions. An adaptive system clock would be desirable to make the most of delay-aware algorithms, mainly when explicit observation delays must be handled. Nevertheless, a manual (constant) or a variable selection can still allow profit from the already described backupdating process. By running different experiments, it is possible to derive the best clock system frequency that allows the agents to choose the actions based on information that is as up-to-date as possible. When using the deterministic clock selection, a trigger-based system clock will sample the environment with a deterministic updating rule that increases or decreases the sampling frequency based on whether a UAV has spotted the signal source or not, respectively. System performance variations need to be experimentally assessed in future work based on the clock selection.

5 USE-CASE SCENARIOS

The configuration's files allow us to perform different analyses comparing the agent(s) performance depending on the desired task(s). We show here two possible use cases that can be run using our environment: the scenario settings and the seed used are the same for all the tests done in the same use case to be able to reproduce the same results. A graphical comparison can be easily performed using the TensorFlow⁶ functionalities provided by our environment. PPO algorithm and its delay-aware variation associated with the scheme described in Section 3.4 by keeping fixed the following parameters values for all the use cases (no hyperparameters tuning): i) *learning rate* = 0.0003; ii) *discount factor* = 0.9; iii) *Generalized Advantage Estimator* = 0.85; iv) *Clipping* = 0.2; v) *Entropy* = 0.01. Both the clock frequency and the number of training epochs (500) are fixed. All the UAVs have the same initial location, their microphone threshold is set to 105 dB, and no energy constraint is considered; a static audio source with constant sound emission at 140 dB is supposed to be spotted by the agents. The robustness of the results that can be obtained (as in the following use cases) could be further evaluated by using different seeds, techniques (among those available) and parameters.

⁶<https://www.tensorflow.org/>

5.1 Use Case 1: Single-Agent with Delay

Only one agent is tested, and observation delay is applied. The goal is to find an audio source with the smallest number of possible movements. Observation space is made only by the local info k reported in Table 2, and the reward only includes the local agent credit R_2^a ($w_k = 1.0$) weighted at its maximum value $w_a = 1.0$. In case of a successful episode, i.e., the audio source has been spotted within 15 iterations, the agent is rewarded with an additional reward equal to 0.5. Otherwise the episode is simply stopped, and no negative reward is provided. Figure 5 shows that when the delay is applied to the considered scenario, the reward trend is quite similar for the delay-aware and the standard version of the PPO algorithm: even if the delay-aware version converges a bit slower, it overcomes the reward obtained with the standard algorithm after convergence. The plot for PPO without delay applied is shown as a reference.

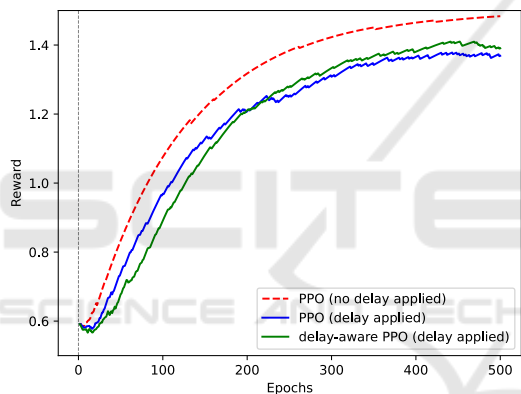


Figure 5: Use case 1 reward (smoothing factor $\alpha = 0.991$).

A proper evaluation metric (e.g., number of steps per episode) could be used to check whether the reward's performance is effectively reflected in the environment.

5.2 Use Case 2: Multi-Agent with Delay

Three agents are involved in this case, and observation delay is applied as in the previous case. We want to spot an audio source with all three agents in 15 iterations per episode at most, getting both local and global information: local ones are as before, whilst global ones are represented by d_s (see Table 2). The reward function here is made up of the local reward R_2^a , ($w_k = 0.05$), and the global one R_1^s ($w_{d_s} = 0.04$). The local and the global rewards are then weighted both at their maximum values, i.e., $w_s = w_a = 1.0$. The ending and the success of an episode are defined as in the first use case, but now we need all three

agents to spot the source at the same time: when successful, an extra sharp reward depending on the time elapsed since the beginning of the episode is provided to the agents. Thus, the primary task is still associated with the spotting action. Figure 6 shows that the delay-aware PPO achieves better performance with respect to the standard version, and it is close to the reference reward that can be obtained when delays are not introduced in the environment.

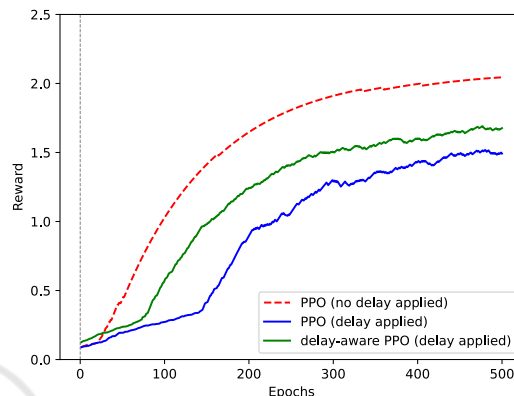


Figure 6: Use case 2 Total reward (smoothing factor $\alpha = 0.991$).

In this case, a proper evaluation metric to double-check the effectiveness of the algorithms could be using the same metric as in the previous case.

6 CONCLUSIONS

We propose DAMIAN, a 3D DRL-based environment allowing for testing customizable use cases at different complexity levels for cooperative multi-UAV scenarios. Our environment is provided with PPO and SAC algorithms with their delay-aware variations by allowing the agents' delay (in executing the actions and receiving the observations) to be considered. The user-friendly configuration eases the scalability, modularity and usability. Different clock selections for the agents and the whole system are available, and two use cases are also provided. Since external files can be used to reproduce a real scenario, our environment could be used by Eurocontrol to learn and test specific and desired agents' behaviours. Future work includes assessing the system's performance deterioration based on the considered task as the number of agents increases. Finally, learning clock agents should be explored further to develop more effective delay-aware DRL algorithms.

ACKNOWLEDGEMENTS

This research was partially funded by the ERC Advanced Grant WhiteMech (No. 834228), the PNRR MUR project PE000013-FAIR, and also supported by the BUBBLES Project (Grant No. 893206).

REFERENCES

- Agarwal, M. and Aggarwal, V. (2021). Blind decision making: Reinforcement learning with delayed observations. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31(1):2–6.
- Arjona-Medina, J. A., Gillhofer, M., Widrich, M., Unterthiner, T., Brandstetter, J., and Hochreiter, S. (2019). Rudder: Return decomposition for delayed rewards. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Brunori, D., Colonnese, S., Cuomo, F., Flore, G., and Iocchi, L. (2021a). Delivering resources for augmented reality by uavs: a reinforcement learning approach. *Frontiers in Communications and Networks*, 2.
- Brunori, D., Colonnese, S., Cuomo, F., and Iocchi, L. (2021b). A reinforcement learning environment for multi-service uav-enabled wireless systems. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 251–256.
- Chen, B., Xu, M., Li, L., and Zhao, D. (2021). Delay-aware model-based reinforcement learning for continuous control. *Neurocomputing*, 450:119–128.
- Chen, B., Xu, M., Liu, Z., Li, L., and Zhao, D. (2020). Delay-aware multi-agent reinforcement learning. *CoRR*, abs/2005.05441.
- Cheng, N., Wu, S., Wang, X., Yin, Z., Li, C., Chen, W., and Chen, F. (2023). Ai for uav-assisted iot applications: A comprehensive review. *IEEE Internet of Things Journal*, 10(16):14438–14461.
- Dalmau, R. and Allard, E. (2020). Air Traffic Control Using Message Passing Neural Networks and Multi-Agent Reinforcement Learning. In *10th SESAR Innovation Days (SID)*, Virtual Event.
- Frattolillo, F., Brunori, D., and Iocchi, L. (2023). Scalable and cooperative deep reinforcement learning approaches for multi-uav systems: A systematic review. *Drones*, 7(4).
- Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, abs/1801.01290.
- Katsikopoulos, K. and Engelbrecht, S. (2003). Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control*, 48(4):568–574.
- Kim, K. (2022). Multi-agent deep q network to enhance the reinforcement learning for delayed reward system. *Applied Sciences*, 12(7).
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*.
- Liu, C. H., Chen, Z., and Zhan, Y. (2019). Energy-efficient distributed mobile crowd sensing: A deep learning approach. *IEEE Journal on Selected Areas in Communications*, 37(6):1262–1276.
- Moon, J., Papaioannou, S., Laoudias, C., Kolios, P., and Kim, S. (2021). Deep reinforcement learning multi-uav trajectory control for target tracking. *IEEE Internet of Things Journal*, 8(20):15441–15455.
- Mou, Z., Zhang, Y., Gao, F., Wang, H., Zhang, T., and Han, Z. (2021). Three-dimensional area coverage with uav swarm based on deep reinforcement learning. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6.
- Peña, P. F., Ragab, A. R., Luna, M. A., Ale Isaac, M. S., and Campoy, P. (2022). Wild hopper: A heavy-duty uav for day and night firefighting operations. *Heliyon*, 8(6):e09588.
- Puterman, M. L. (1990). Chapter 8 markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 331–434. Elsevier.
- Rens, G., Raskin, J., Reynouard, R., and Marra, G. (2021). Online learning of non-markovian reward models. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pages 74–86. INSTICC, SciTePress.
- Sacco, A., Esposito, F., Marchetto, G., and Montuschi, P. (2021). Sustainable task offloading in uav networks via multi-agent reinforcement learning. *IEEE Transactions on Vehicular Technology*, 70(5):5003–5015.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.
- Scott, J. E. and Scott, C. H. (2017). Drone delivery models for healthcare. In *Hawaii International Conference on System Sciences*.
- Singh, S., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable markovian decision processes. In *International Conference on Machine Learning*.
- Wang, Q., Zhang, W., Liu, Y., and Liu, Y. (2019). Multi-uav dynamic wireless networking with deep reinforcement learning. *IEEE Communications Letters*, 23(12):2243–2246.
- Yuan, T., Chung, H.-M., Yuan, J., and Fu, X. (2023). Dacom: Learning delay-aware communication for multi-agent reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10):11763–11771.
- Zhu, Z., Xie, N., Zong, K., and Chen, L. (2021). Building a connected communication network for uav clusters using de-maddpg. *Symmetry*, 13(8).