Deep-Reinforcement-Learning-Based Collision Avoidance in UAV Environment

Sihem Ouahouah, Miloud Bagaa, *Senior Member, IEEE*, Jonathan Prados-Garzon, and Tarik Taleb

***Abstract*—Unmanned aerial vehicles (UAVs) have recently attracted both academia and industry representatives due to their utilization in tremendous emerging applications. Most UAV applications adopt visual line of sight (VLOS) due to ongoing regulations. There is a consensus between industry for extending UAVs' commercial operations to cover the urban and populated area-controlled airspace beyond VLOS (BVLOS). There is ongoing regulation for enabling BVLOS UAV management. Regrettably, this comes with unavoidable challenges related to UAVs' autonomy for detecting and avoiding static and mobile objects. An intelligent component should either be deployed onboard the UAV or at a multiaccess-edge computing (MEC) that can read the gathered data from different UAV's sensors, process them, and then make the right decision to detect and avoid the physical collision. The sensing data should be collected using various sensors but not limited to Lidar, depth camera, video, or ultrasonic. This article proposes probabilistic and deep-reinforcement-learning (DRL)-based algorithms for avoiding collisions while saving energy consumption. The proposed algorithms can be either run on top of the UAV or at the MEC according to the UAV capacity and the task overhead. We have designed and developed our algorithms to work for any environment without a need for any prior knowledge. The proposed solutions have been evaluated in a harsh environment that consists of many UAVs moving randomly in a small area without any correlation. The obtained results demonstrated the efficiency of these solutions for avoiding the collision while saving energy consumption in familiar and unfamiliar environments.**

***Index Terms*—Collision avoidance, deep reinforcement learning, machine learning, multiaccess-edge computing (MEC), unmanned aerial vehicles (UAVs).**

I. Introduction

UNMANNED aerial vehicles (UAVs), commonly recognized as drones, are small, fast, and mobile cyber–physical entities employed in different industrial verticals, including power supply inspection, parcel and package delivery, disaster management, and traffic monitoring [1]. The utilization of UAVs goes beyond industrial and academic purposes to daily personal use. A UAV operator must always be capable of maintaining the visual line of sight (VLOS) of its UAV that is piloting due to ongoing regulations, unaided by any technology other than prescription glasses or contact lenses. While UAVs are used mostly within VLOS, there is enthusiasm toward their utilization beyond VLOS (BVLOS) to enable new emerging applications. Therefore, there is a consensus between industry for attenuation of the regulation by extending UAVs' commercial operations to cover the urban and populated area controlled airspace BVLOS. The latter will be enabled by leveraging a cellular wireless network. 5G system and beyond considers the UAV management BVLOS as one of the essential demonstrators. On the other side, emerging networking paradigms, such as Edge Computing can substitute UAVs to handle high processing flight control applications. Furthermore, GPU vendors allow for realizing different microarchitectures (e.g., Fermi, Maxwell, and Pascal) that might enable real-time and high resourced applications for the UAV's flight control [2].

The UAVs' commercial revenue sees considerable growth by the near future [3].The expected increase in the number of UAVs involves new challenges related to their control and management. Efficient solutions for UAV's collision avoidance is one of the challenges that have been widely tackled in the literature in both ground vehicles [4], [5] context, as well as in the context of UAVs [3], [6]–[24]. Different sensors have been leveraged for scanning and detecting objects surroundings UAVs. Some solutions use cameras for detecting mobile and static obstacles around UAVs [6], [14]. Nevertheless, the information provided by video cameras requires intensive processing to be translated into useful information to control UAVs [25].

Several works [3], [11], [17], [18] have proposed path planning solutions where the UAVs are provided with their whole trajectories before starting their missions to overcome the limitation mentioned above. The path planning fits well in applications with invariable environment scenarios. However, mostly, UAVs fly in unsettled indoor, urban, and confined areas. Indeed, sensing and path planning approaches' success is highly related to the computational capacity of the UAV, the accuracy of the sensor, and the knowledge's degree on the environment. On another side, reinforcement-learning (RL)-based approaches got much success in emerging topics, including robotic prediction, vehicular ad hoc networks

Manuscript received February 1, 2021; revised April 23, 2021; accepted May 26, 2021. Date of publication October 8, 2021; date of current version March 7, 2022. This work was supported in part by the Spanish National Project TRUE-5G under Grant PID2019-108713RB-C53, and in part by the European Union's Horizon 2020 Research and Innovation Program through the 5G!Drones Project under Grant 857031. *(Corresponding author: Miloud Bagaa.)*

Sihem Ouahouah and Miloud Bagaa are with the Department of Communications and Networking, School of Electrical Engineering, Aalto University, 00076 Espoo, Finland (e-mail: sihem.ouahouah@aalto.fi; miloud.bagaa@aalto.fi).

Jonathan Prados-Garzon is with the Department of Signal Theory, Telematics and Communications, University of Granada, 18014 Granada, Spain (e-mail: jpg@ugr.es).

Tarik Taleb is with the Department of Communications and Networking, School of Electrical Engineering, Aalto University, 00076 Espoo, Finland, and also with the Department of Computer and Information Security, Sejong University, Seoul 05006, South Korea.

Digital Object Identifier 10.1109/JIOT.2021.3118949

(VANET), and UAVs. For instance, Xiao *et al.* [26] have provided a heuristic to enhance communication and prevent jamming attacks in VANET by leveraging UAV. An extended version of this work has been suggested in [27] to prevent the jamming attacks in VANET by leveraging both UAV and RL approaches. This success attracted the researchers to use RL to ensure a self-decision-making system for a safe UAV's autonomous flight.

RL consists in providing a kind of knowledge on the environment based on the previous UAV's experiences. Thus, RL builds the knowledge by interacting with the environment based on a Markov decision process (MDP) model and following one of the RL methods (e.g., *Q*_learning, deep *Q*-networks (DQN), Policy_gradient, or Actor_critic). The RL-based UAVs control solutions proposed so far [19]–[24] need large data sets that referrer to the abstraction level used to model the RL environment system (e.g., velocity, wind velocity, etc.). The data sets used in RL-based solutions might return the same limitations pointed in the classical approaches stated above [19]. To overcome these limitations, this article suggests two strategies for avoiding the collision in a UAV environment. The first solution, named a probability-distribution-based collision-avoidance framework (PICA), leverages the probabilistic model for avoiding collisions. In contrast, the second solution, dubbed-RL-based collision-avoidance framework (RELIANCE), leverages DQNs for avoiding collisions.

To deal with the disparate UAVs processing capacities and to ease the deployment of the proposed solutions, we suggest two deployment approaches: 1) the UAV's flight controller is deployed onboard or 2) at the multiaccess edge computing (MEC). The first approach convenes UAVs with the new GPU microarchitecture technology is where the agent, either of RELIANCE or PICA, can smoothly make decision and select the best actions. On the other hand, the second way assembles UAVs with limited computing capacities. In order to ensure close management, services running should be migrated among MECs using Follow Me Edge-Cloud concept. Bekkouche *et al.* [28] suggested a MEC architecture that ensures UAVs' resource provisioning. In case the UAV has a limited resource capacity, the same architecture as proposed in [28] can be adopted. In this case, the RELIANCE/PICA agent is responsible for making decisions at the MEC, and then sending the respective actions to the UAV for controlling its motion.

Both algorithms aim to enable autonomous decision making for a UAV while a safe and short flight is ensured to save energy consumption. To ensure a fast convergence of RELIANCE and PICA, we have used a detail-less and generalized state by focusing on the closest part of the environment to the agent. In RELIANCE states' design, we have leveraged partially observable MDP (POMDP) to ensure the generalization and fast convergence. We have used a partially observable state that focuses only on the UAV surrounding to avoid the collisions instead of the whole deployment area. The limitation of the observation at the UAV vicinity helps to reduce the state space by aggregating many observations to a single state. Thanks to this strategy, RELIANCE and PICA avoid overloading computation processing by ignoring useless knowledge. Moreover, this strategy helps the RELIANCE solution to converge quickly by treating many observations as the same state. Furthermore, to ensure the generalization and that both algorithms can work in unseen environments, we have used relative target positions. The benefits of this strategy are twofold: it facilitates and speeds up the convergence of the neural networks (NNs) and, most importantly, improves the generalization of the agent, which is agnostic to the scenario scale. We have evaluated and compared both algorithms in terms of collision avoidance and energy saving in familiar and unfamiliar environments. The obtained results demonstrate the ability of both algorithms in the generalization by performing well in unknown environments. Also, the simulation results clearly show the superiority of RELIANCE comparing to PICA.

The remainder of this article is organized as follows. Section II reviews the related works. Section III includes our system model and problem statement. In Section IV, the PICA solution is described. An overview on DQN and a RELIANCE solution are detailed in Section V. Section VI presents and discusses the simulation results of PICA and RELIANCE evaluations. Finally, the primary conclusion is drawn in Section VII.

II. Related Work

There is a vast literature to address the collision-avoidance problem in the context of both unmanned ground vehicles [4], [5] and UAVs [3], [6]–[24].

Most of the solutions rely on exact methods [3], [6]–[18], i.e., analytical modeling and optimization techniques, to tackle the UAVs collision-avoidance problem (UCAP). The existing works, based on exact methods, usually considers part of the UCAP aspects to handle its modeling and computational complexity. However, in order to provide a realistic and practical model of the UCAP, many issues have to be taken into account.

1) *Obstacle Detection:* In order to detect the static and mobile objects, the UAVs need to be equipped with onboard sensors. The number of these sensors and their precision might be affected and limited due to several external factors, e.g., specific scenario and UAVs' autonomy. For instance, GPS might not work for indoor scenarios like Industry. Other sensors like radars [8] might be too heavy, energy consuming, and expensive. Then, the concrete set of onboard sensors in UAVs depends on the application and scenario.
2) *Sensor Errors:* Onboard sensors to detect objects are not error-free. All of them have precision errors, which might be affected by external conditions. For instance, GPS error is affected by weather conditions and follows a Gaussian distribution [3], [29].
3) *Complex Control:* There are several variables to control the UAVs movement, e.g., direction, velocity, and acceleration. Furthermore, these variables strongly depend on external factors, such as wind velocity.
4) *Different Approaches:* There are two different approaches to solve UCAP, namely, path planning and sensing and avoiding [25] methods. Path planning solutions compute the trajectories of the UAVs offline, whereas sensing and avoiding (online) methods

determine the movement of the UAVs for small time steps depending on the environment conditions. Sensing and avoiding methods offer higher flexibility and are suitable for a wider range of scenarios. Path planning fits well only for scenarios where the environment remains relatively static during the whole UAVs' mission. The main drawback of the online methods is that typically, the UAV has to run the algorithm (e.g., due to latency constraints), which might exhibit high computational complexity and consume energy.

In the light of the above, the UCAP requires a high-domain knowledge and its modeling leads to complex or even intractable optimization programs. To overcome these problems, machine learning techniques are particularly attractive for addressing UCAP, so they have been recently received a lot of attention by the research community [19]–[24].

Choi and Cha [19] provided a comprehensive survey of ML-assisted solutions for autonomous flight. Specifically, they focus on object recognition and UAV's control strategy. The authors conclude that ML is a promising approach to enable stable flight under uncertain environments, though there are still some open issues that need to be carefully addressed. Among them, the existing works do not apply ML in all the UCAP's issues together for autonomous flight. Then, holistic solutions, which cover most of the real-world problems and are suitable for a wider spectrum of scenarios, are required. Furthermore, the existing solutions need large data sets for training. In this regard, they encourage new less data-hungry proposals with a more lightweight training. Finally, they motivate the need for real world tests for a stronger validation of the ML-based UAVs control strategy. Similarly, Fraga-Lamas *et al.* overview the latest advances on IoT UAV systems controlled by deep learning techniques. They analyze the object detection and collision-avoidance problems and present a survey on the state of the art of deep learning techniques to solve them. Also, they detail the most relevant existing data sets and UAVs' communication architectures. Finally, they identify the open challenges for UCAP. Interestingly, they extract some similar conclusions to the ones drawn in [19]. For instance, the necessity for large amount of data to generate robust models and the difficulty to produce those data.

In this article, we propose a simple, yet powerful deep RL (DRL) and probability-distribution-based solutions. Our proposals are suitable for many scenarios, while they need reduced data sets to converge and produce robust models.

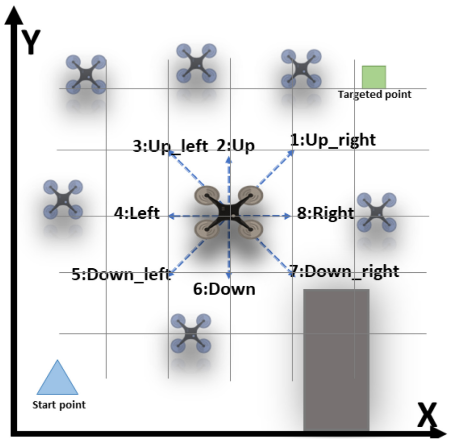


Fig. 1. System model of collision avoidance in the UAV environment.

TABLE I
SUMMARY NOTATIONS

Symbol	Description
UoI	The UAV of Interest.
$X \times Y$	The 2-dimensional geographical area.
P_S	The UoI's started position.
P_T	The UoI's targeted position.
$\mathcal{A}$	The set of directions controlling UoI motion.
$\mathcal{I}$	The set of mobile and static intruders.
$U(t)$	The position of UoI at time step (t).
$J(t)$	The position of intruder j at time step (t).
γ	The euclidean distance between the positions.
$\lvert A\rvert$	The cardinal of the set A .
$\mathcal{Z}$	The set of zones.
z_i	The zone surrounding the position i.
ρ	The radius of every z_i.
$\eta(i)$	The set of intruders neighboring the position i.
$\mathcal{P}(z_i)$	The probability of collision at the zone (z_i).
p_i^j	The probability of collision between the point i and its neighbor j.
θ	The priority factor rate.
Δ	The squared shaped area surrounds the UoI.
L_Δ	The size of Δ side.
α	The learning rate of DQN.
$\mathcal{L}$	The loss function.
ϵ	A threshold distance before two UAVs collide.
$batch\_size$	The size of each batch.
Q^π	The policy network.
Q^T	The target network.
$\mathcal{M}$	A number of episodes to update Q^T with Q^π.
ω	The wight and bias of neural network.
P_{rel}	The related address of P_T according to $U(t)$.
ξ	The decay parameter.

III. System Model and Problem Statement

In this article, we aim to control the movement of a UAV, hereinafter referred to as UoI (UAV of Interest), while avoiding the collisions with static and mobile objects. Table I summarizes the different notations used in the paper. UoI needs to move within a confined area of dimension $X \times Y$ while avoiding the collisions. The UoI motion begins from a predefined initial position $P_S = (x_S, y_S)$ and stops once achieves a predefined targeted destination $P_T = (x_T, y_T)$. Let $\mathcal{A}$ denote the possible action directions of UoI. As mentioned in [28], the possible movement of a UAV is limited and related to the environment that it works in. For the sake of simplicity and without loss of generality, we consider $\mathcal{A}$ has eight possible directions, $\mathcal{A} = \{$*Up*, *Down*, *Right*, *Left*, *Up_right*, *Up_left*, *Down_right*, *Down_left*$\}$, as depicted in Fig. 1. For simplicity, we assume a constant velocity and altitude for the UoI. Furthermore, we consider that UoI operates autonomously without either any remote ground control or predefined way-points plan. On another side, we consider that the 2-D flying area can include either static (e.g., buildings) and mobile (e.g., birds and other UAVs) obstacles. Therefore, the UoI has to be equipped with an accurate sensor (e.g., Lidar) to precisely detect the surrounding objects' positions. Other ultrasonic-based, video-based, and radio-based techniques for detecting obstacles and mobile objects have been investigated in [30] and [31]. Hereafter, we refer to the static and mobile objects as *static_intruders* and *mobile_intruders*, respectively. We define $\mathcal{I}$ as the set of intruders, where $\mathcal{I} = \{\{static\_intruder\} \bigcup \{mobile\_intruder\}\}$.

The UoI might collide with one of the mobile and static obstacles. We assume an arbitrary trajectory for the mobile intruders, which is unknown by the UoI. A collision occurs whenever the Euclidean distance between the UoI and any intruders is lower than a predefined threshold distance ϵ. The safety distance ϵ varies from few centimeters to few meters according to different parameters related to the environment and used sensors. The distance ϵ can vary according to the sensor technology used to measure the distances, such as Lidar, depth camera, video, or ultrasonic. Also, the accuracy of the same type of sensors can vary from a manufacturer to another. The UoI can detect this collision by harnessing its onboard sensors at any time t. Let $P_u(t) = (x_u(t), y_u(t))$ and $P_j(t) = (x_j(t), y_j(t))$ denote the position of the UoI and the position of the intruder j at a given instant t, respectively. Then, a collision instance is formally defined as follows:

$$Collision = \begin{cases} 1, & \text{if } \exists \quad j \in \mathcal{I} \quad \text{where} \quad \delta_u^j \le \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where

$$\delta_u^j = \sqrt{\left(x_u(t) - x_j(t)\right)^2 + \left(y_u(t) - y_j(t)\right)^2}. \tag{2}$$

In this article, we also take into consideration the limitation on the battery capacity of the UoI. To that end, the ultimate goal of the algorithm in charge of controlling the UoI movement is to achieve the target P_T following the shortest path while avoiding collisions with mobile and static objects. The shortest path minimizes the distance traveled by the UoI, thus contributing to energy saving.

IV. Probability-Distribution-Based Collision-Avoidance Framework

In this section, we propose a heuristic, dubbed PICA, to control the movement of the UoI to reach a target position while avoiding collisions. It considers the mission time, i.e., the time from the UoI starts its mission until it reaches its target P_T, is slotted. At each time step t, the UoI collects data from different sensors to sense the objects' presence in its vicinity. As depicted in Fig. 2, UoI is aware of the surrounding objects in a circular area z_i, whose extension is limited by the sensors' ranges. Specifically, the circular area z_i has a radius ρ. The state of the circular area z_i, i.e., the spatial distribution of the intruders within it, is updated at every time step after the UoI changes its position according to an action $a \in \mathcal{A}$ taken by PICA. Let $\mathcal{Z}$ denote the set of circular shaped areas z_i, where $\mathcal{Z} = \{z_i : \forall i \in \mathcal{A}\}$.

Inside every z_i it might exist intruders neighboring every z_i's center i. Let $\eta(i)$ denote the set of *static_intruders* and *mobile_intruders* inside the area z_i. We denote by δ_i^j the Euclidean distance between jth intruder belonging $\eta(i)$ and the position of i. The distance between each intruder and the center of z_i can be computed by PICA using the triangulation method.

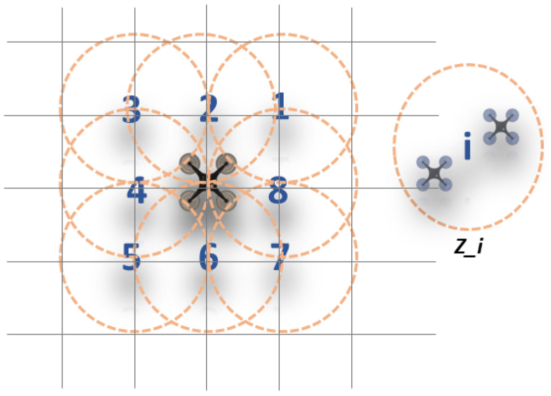


Fig. 2. PICA: zone concept for selecting directions.

The density distribution of the intruders inside z_i could refer to the likelihood of a collision if the UoI moves to position i. In other words, the denser z_i, the higher the probability that the UoI experiences a collision. Let us define $\mathcal{P}(z_i)$ as the probability of collision inside z_i, formally defined as follows.

$$\mathcal{P}(z_i) = \frac{\sum_{\forall j \in \eta(i)} p_i^j}{\sum_i \sum_{\forall j \in \eta(i)} p_i^j} \qquad \forall i \in \{1, 2, \ldots, |\mathcal{A}|\} \tag{3}$$

where

$$p_i^j = 1 - \frac{\delta_i^j}{\rho} \tag{4}$$

where p_i^j represents the likelihood that UoI collides with j if action $i \in \mathcal{A}$ is chosen. Formally, the closer j is to UoI, the higher the probability of collision. Note that the value of δ_i^j/ρ is within the interval [0, 1]. In order to avoid the collision, the position with the lower $\mathcal{P}(z_i)$ should be chosen.

In addition to the safety factor, PICA aims to go through the shortest path by seeking at each time step t the direction that brings the UoI closest to the target. To achieve this goal, PICA measures the remaining distance to the target from every i. Indeed, to decide the UoI's next direction, PICA ranks every Z_i's center, i, using the following equation:

$$R_i = \theta P(z_i) + (1 - \theta)\frac{\delta_i^{P_T}}{\sqrt{X^2 + Y^2}} \tag{5}$$

where $\theta \in [0, 1]$ is a parameter used to favor either safety or energy. $\delta_i^{P_T}$ denotes the distance between the current position and the target. The concept of application integrating the UoI has an immediate impact on selecting either θ or its complement $(1 - \theta)$. In our case, we give more priority to the safety of the UoI agent. For this reason, we have selected higher values of θ. Furthermore, $\delta_i^{P_T}$ refers to the Euclidean distance between the Z_i's center and the targeted point P_T. Since the unity of both the probabilities and the distances values are in different scales. To prevent distance domination, we have normalized the value of $\delta_i^{P_T}$ to be between 0 and 1 by diving it by the maximum possible distance $\sqrt{X^2 + Y^2}$. Indeed, the UoI will choose to move in the direction a that has the lowest rank using the following formula:

$$a = \arg\min_{i \in \mathcal{A}} \{R_i\}. \tag{6}$$

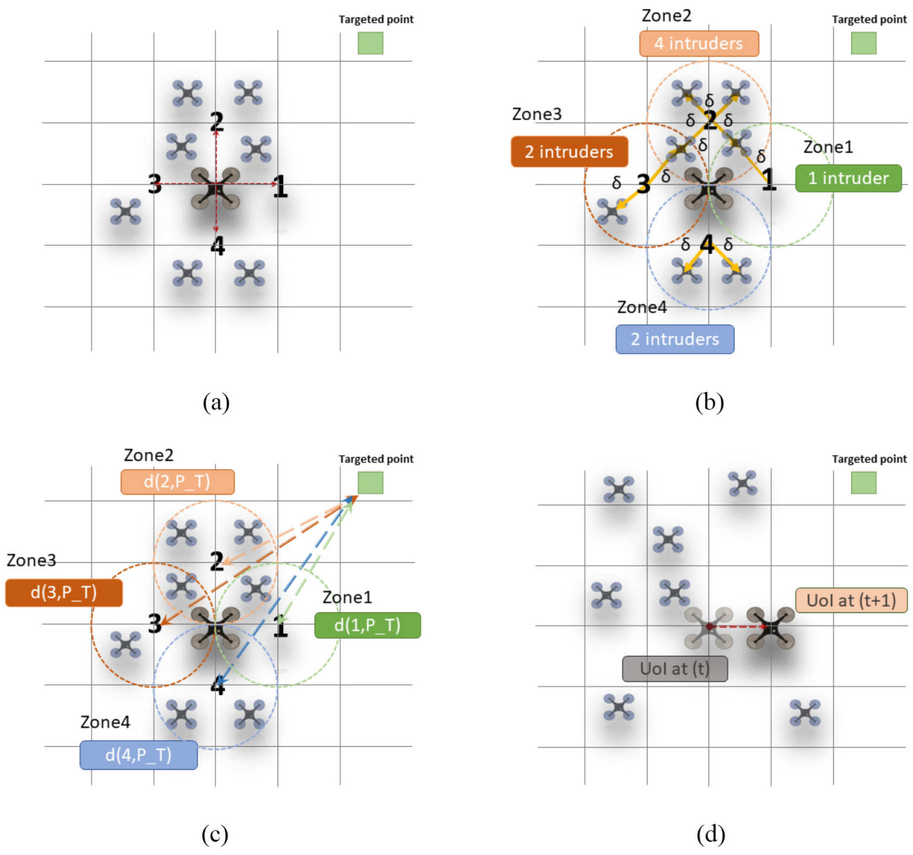


Fig. 3. PICA descriptive example. (a) UAV UoI at instant (t). (b) Density distribution of the intruders inside each zone. (c) Remaining distance to the target from the center of each zone. (d) UAV UoI at instant $(t+1)$.

Fig. 3 illustrates the PICA functionality, which is detailed in Algorithm 1. For simplicity, in this example, we consider that UoI can move only on four directions $\{$*Up*, *Down*, *Left*, and *Right*$\}$ corresponding to the positions $\{1, 2, 3, 4\}$, respectively. We also consider that all the distances δ_i^j are the same and equal to δ. Hereafter, based on the density of intruders in each zone, PICA computes the probability of collisions $\mathcal{P}(z_i)$ for every z_i (Algorithm 1: line 10) using (3) and (4). For example, the probability of collision $\mathcal{P}(z_2)$ at zone 2 is the summation of the probabilities that UoI collides at Z_i's center 2 with every $j \in \eta(2)$. Indeed, as shown in Fig. 3(b), $\mathcal{P}(z_2) = [(\sum_{\forall j \in \eta(2)} p_2^j)/(\sum_{\forall i \in \{1,\ldots,4\}} \sum_{\forall j \in \eta(i)} p_i^j)]$, where $p_2^1 = p_2^2 = p_2^3 = p_2^4 = (1-[\delta/\rho])$. In this case, the probability collision distribution of the positions 1, 2, 3, and 4 is $1/9$, $4/9$, $2/9$, and $2/9$, respectively. As depicted in Fig. 3(b), the zone 2 is the most dense in term of intruders compared to the other ones. In contrast, zone 1 is the less dense since it contains only one intruder with probability collision $1/9$. To rank the candidates' directions $i \in \{1, \ldots, 4\}$ of PICA, based on (5) and (6), it chooses the best action that has the lowest probability collision and the lowest remaining distance to the target (Algorithm 1: lines 12 and 15). Following the same example and as shown in Fig. 3(b) and (c), the candidate direction (1) has the smallest rank since it has the lowest probability and the lowest missing distance to the target (Algorithm 1: line 15). Finally, at the time step $(t+1)$, the UoI moves into the position (1) as depicted in Fig. 3(d).

V. RELIANCE: Reinforcement-Learning-Based Collision-Avoidance Solution

In this section, we provide an RL-based solution for avoiding the collision. In contrast to the model-based approach, MDP, which requires full knowledge about the environment (i.e., transition probabilities), RL does not require any prior knowledge. This makes RL a more suitable framework for dealing with unsuspected and uncorrelated mobility of objects around UoI. Thanks to sampling and bootstrapping in RL, RELIANCE can forecast the next movement of each mobile object and then avoid the collision. Fig. 4 depicts the main overview of the RELIANCE solution. In this section's balance, we will give first some background on RL, and, more precisely, DQN employed in this article. Then, we will give a detailed description of RELIANCE.

A. Background on RL and DQN

The RL technique has been widely used in the literature in various applications and services, such as robotics and industry 5.0. RL's ultimate goal is to endow vertical industry with the ability to learn, improve, and adapt according to the environment's changes. With the new trend toward the self-optimized and the cognitive network, industry, and academia shifted their attention to employing RL. An RL system mainly consists of five elements, as depicted in Fig. 4, which are: 1) environment $\mathcal{E}$; 2) states $\mathcal{S}$; 3) agent, in our case, is the motion controller

Algorithm 1: PICA

```
Input :
        $X$:The x_axis limit of the geographical area $X \times Y$.
        $Y$:The y_axis limit of the geographical area $X \times Y$.
        $\rho$: The radius.
        $\theta$: The priority rate.
        $\mathcal{A}$:The set of actions.
        $P_T$:The started point of UoI in the environment.
        $P_S$:The targeted point of UoI in the environment.
Output:
        done: The UoI reaches $P_T$ or collides with one of the intruders.
1  $done \leftarrow False$;
2  while ($done = False$) do
3      $Z \leftarrow \emptyset$;
4      $R \leftarrow \emptyset$;
5      foreach ($a \in A$) do
6          $z \leftarrow Circle(a, \rho)$;
7          $Z \leftarrow Z \cup z$;
8      end
9      foreach ($z \in Z$) do
10         Compute $P(z)$;
11         Compute $\delta_i^{P_T}$;
12         $r \leftarrow \theta P(z) + (1 - \dot{\theta}) \frac{\delta_i^{P_T}}{\sqrt{X^2+Y^2}}$;
13         $R \leftarrow R \cup r$;
14     end
15     $a \leftarrow \arg\min_{i \in A}\{R_i\}$;
16     UoI applies action $a$;
17     if ($U(t) = P_T$ or Collision) then
18         $done \leftarrow True$;
19     end
20 end
```

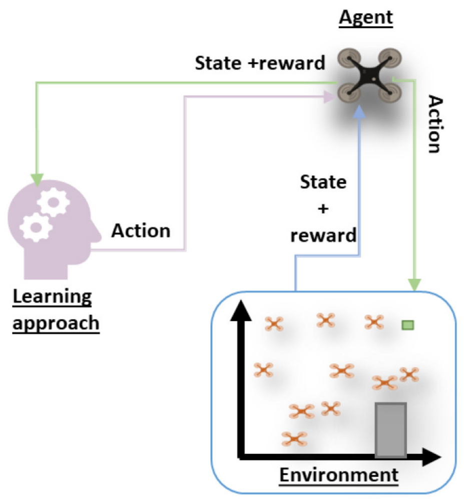


Fig. 4. RL-based collision-avoidance system overview.

of the UoI; 4) Actions $\mathcal{A}$; and 5) rewards r received after the execution of each action $a \in \mathcal{A}$.

While RL works either for episodic or continuous tasks, in this work, our environment is episodic. Each episode presents the UoI mission that starts at $P_S = (x_S, y_S)$ and ends either when UoI attends its destination $P_T = (x_T, y_T)$ or collides with a mobile or static obstacle. As depicted in Fig. 4, UoI discretely interacts with the environment by taking different actions, and then accordingly receiving an observation and rewards that reflect the action taken. The agent UoI keeps interacting with the environment $\mathcal{E}$ and receiving reward r_t on steps $t \in \{1, 2, \dots, T\}$. While $T = \infty$ for continuous tasks, it is finite in the case of an episodic task. The objective of the UoI agent is to increase cumulative reward G_t received after time step t until the end of the episode

$$G_t \doteq \sum_{k=0}^{T} \gamma^k r_{t+k+1} = r_{t+1} + \gamma G_{t+1} \tag{7}$$

such that $\gamma \in [0, 1]$ is the discount factor and r_t denotes the immediate reward received at the instant t.

Many RL techniques have been proposed in the literature, including policy-based (e.g., REINFORCE), actor–critic (e.g., A3C and DDPG), and value-based approaches (e.g., QN, DQN, and DDQN). While the two formal methods aim to provide the policy that estimates the state's action probabilities, the latter approach estimates the state–action value. Then, this value is used to deliver the optimal policy. Considering that the space of actions is discrete and limited, in this work, we opt for a value-based approach and, more precisely, DQN. Particularly, we have chosen DQN due to the size of the action–state space, as explained later.

The state–action value of state $s \in \mathcal{S}$ using action $a \in \mathcal{A}$ under the policy π is defined as follows [32]:

$$\begin{aligned} Q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &\leftarrow \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | S_t = s, A_t = a\right]. \end{aligned} \tag{8}$$

The optimal action–state value $Q^*(s, a)$ can be delivered from $Q_\pi(s, a)$ by choosing the optimal policy. Formally, $Q*(s, a) = \max_\pi Q_\pi(s, a)$ for $a \in \mathcal{A}$ and $s \in \mathcal{S}$. $Q^*(s, a)$ can be also delivered using the Bellman optimality equation using the following formula [32]:

$$Q_*(s, a) \leftarrow \mathbb{E}\left[r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_*(S_{t+1}, a') | S_t = s, A_t = a\right]. \tag{9}$$

The basic idea behind many value-based algorithms is sampling and bootstrapping. The sampling is leveraged for enabling the algorithm to learn by exploring the environment, thanks to the trial and error approach. Meanwhile, bootstrapping is a technique used to estimate the state–action value in order to speed up the algorithm convergence [32]. Q-Learning Algorithm is one of the widely used algorithm in the literature. The Q-learning algorithm leverages sampling and bootstrapping methods to converge to the optimal policy. During the learning step, the Q-learning algorithm updates the state value action using the following formula:

$$\begin{aligned} Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha \\ &\times \left[r_{t+1} + \gamma \times \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)\right] \end{aligned} \tag{10}$$

such that α is the learning rate.

Thanks to bootstrapping, Q-learning repeatedly updates $Q(s_t, a_t)$ by shifting it toward the optimal value using TD error $(r_{t+1} + \gamma \times \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t))$ and learning rate α. This approach enables to gradually increasing $Q(s_t, a_t)$ toward the optimal value. The optimal policy can be delivered from the optimal state action using the following formula:

$$\forall s \in S : \pi^*(s) \leftarrow \arg\max_a Q(s, a). \quad (11)$$

In the Q-learning algorithm, the state–action value Q is presented as a table, where the states are the lines and actions are the columns. Unfortunately, Q learning is unfeasible for large action–state spaces as ours. Fortunately, DQN has been suggested to overcome that limitation [33] by creating an estimator of the Q table by leveraging the NN. In fact, the Q table is approximated with an NN with parameter ω.

Unfortunately, the basic DQN algorithm suffers from overestimations of action value due to using the same NN in the update and estimation of the next Q value used to compute the TD error. This approach creates lots of noise and makes it hard to find the action with the maximum expected/estimated Q-value. To prevent this issue, Mnih *et al.* [34] have suggested DQN that uses two different Q NNs. The first one, called policy Q NN Q^π, is used for estimating the action. Meanwhile, the second one, called target Q network Q^T, is used to generate the target action values. To mitigate the noises in the update, while Q^π is updated at each iteration, Q^T is updated from Q^π only after a specific number of episodes.

In this case, the policy π during the exploitation, either during the training or inference modes, is generated from the approximation $Q^\pi(s_t, a_t, \omega)$ NN using the following equation:

$$\pi_{s_t} \leftarrow \arg\max_{a \in \mathcal{A}} Q^\pi(s_t, a, \omega). \quad (12)$$

Meanwhile, the parameter ω (bias and weights) of the estimator Q^π is updated periodically during the training step using the following formula:

$$\omega_{t+1} = \omega_t + \alpha \times \Big[r_{t+1} + \gamma \times \max_a Q^T(s_{t+1}, a; w') - Q^\pi(s_t, a_t; w)\Big] \times \nabla_w Q^\pi(s_t, a_t; w). \quad (13)$$

By leveraging different gradient descent methods (e.g., stochastic gradient descent, RMSprop, or ADAM), the DQN Algorithm keeps updating ω during the training step. ω is updated from replay memory ($\mathcal{B}$) that consists of transitions observed during the exploration or exploitation. Each transition $<s_t, a_t, r, s_{t+1}>$ consists of the current state s_t, the taken action a_t, the immediate received reward r and the next state s_{t+1}. To break the correlation between transitions and to allow a stable learning curve, a batch of transitions (i.e., batch_size) is randomly selected from the reply memory $\mathcal{B}$ [32].

To ensure a balance between the exploration and exploitation during the training to update ω, an epsilon greedy method is used. The algorithm keeps randomly switching between the exploration and exploitation modes. At the end of the training, the DQN algorithm should favor exploitation than exploration to assist its convergence. For this purpose, an epsilon decay strategy has been adopted by decreasing the epsilon decay ξ parameter during the training. ξ initially starts by 1, and it should converge to 0 at the end of the training. To switch between the exploration and exploitation, a random number (i.e., [0, 1]) is generated and compared to ξ. If the generated number is lower than ξ, the exploration procedure is executed. Otherwise, the exploitation procedure is considered.

B. RELIANCE Model Overview

The autonomous flight control system of the UoI is realized as an RL agent. To model the energy-aware collision-avoidance problem using the RL framework, as mentioned in the previous section, the following elements need to be formally defined: 1) environment; 2) state; 3) agent; 4) actions; and 5) reward.

1) *Agent:* The RL agent is instantiated and run within the UoI to control its trajectory in order to avoid collisions while minimizing the energy consumption by taking the shortest path until its destination.
2) *Environment:* Geographical are of dimensions $X \times Y$ that include a set of mobile (e.g., other UAVs) and static objects (e.g., walls). The agent moves within this 2-D confined area.
3) *Actions:* The action space comprises a set of eight directions, i.e., $\mathcal{A} = \{Up, Down, Right, Left, Up\_right, Up\_left, Down\_right, Down\_left\}$, as previously mentioned.
4) *Reward:* If the agent succeeds and reaches its targeted destination, it is positively rewarded with 100. If the UoI experiences a collision during its trajectory to the destination, the agent is penalized with a negative reward of -100. Finally, in order to encourage the agent to take the shortest path, there is a penalty of -0.1 for each step taken by the agent until reaching its destination.
5) *State:* The state (agent's observations) consists of two parts.
   a) The distance vector $P_{\text{rel}} = (x_{\text{rel}}, y_{\text{rel}})$ is defined as the vector from the current UoI's position (x_C, y_C) to the UoI's destination (x_T, y_T). That is

   $$x_{\text{rel}} = \frac{x_T - x_C}{X}$$
   $$y_{\text{rel}} = \frac{y_T - y_C}{Y}.$$

   Observe that $(x_{\text{rel}}, y_{\text{rel}}) = (0, 0)$ means the UoI is at the destination. Also, note that x_{rel} and y_{rel} have been normalized by X and Y (flight area dimensions), respectively. In this way, the agent is agnostic to the scenario scale, which makes the solution more general, i.e., the same trained model can be used in many environments.
   b) The number of mobile and static objects distribution across a grid square centered around the UoI. Specifically, a $\Delta = \|L_\Delta \times L_\Delta\|$ grid square is considered. The UoI dimensions give the size of each tile of the grid. The grid is formally described as a binary matrix. Each element of this matrix indicates whether there is any static or mobile object (intruder) within the respective cell (tile) (=1) or not (=0) (see Fig. 5). This approach enables us to consider the UoI vicinity, which is the most relevant to avoid collisions and reduce the state space by aggregating many observations to a single state. Thus, faster learning and convergence will be perceived. As depicted in Fig. 5, thanks to the

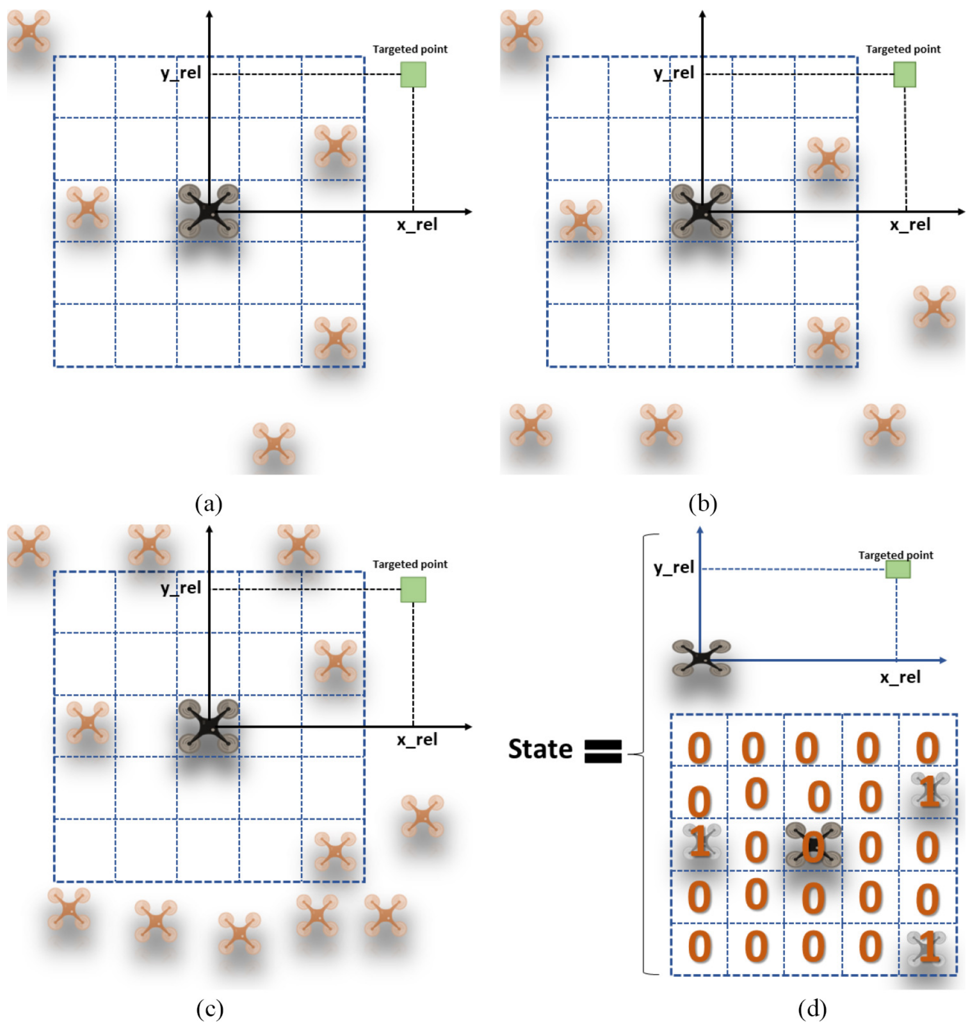


Fig. 5. State aggregation process adopted by RELIANCE. (a) Scenario1 with two intruders beyond the square. (b) Scenario2 with five intruders beyond the square. (c) Scenario3 with nine intruders beyond the square. (d) Same state refers to all the previous scenarios.

aggregation method adopted by RELIANCE, different observations depicted in Fig. 5(a)–(c) can be presented by the same state shown in Fig. 5(d).

It is remarkable that both components of the state considered are agnostic of the scenario, which makes the solution more general.

C. RELIANCE Example Description

As stated previously, we provide the agent with an RL-Algorithm that adopts one of the existing RL approaches. The principle role of this algorithm consists of giving the agent the ability to self-decide in which direction has to move following the defined goals. Before starting the flight, we provide the agent with the coordinates of its started and targeted points, P_S and P_T, respectively. Thus, at each step, the agent needs to do the following.

1) It receives the status of the area from the sensing equipment.
2) It traces the square-shaped area surrounding the agent where the current agent position centers the square.
3) It ignores the area beyond the square and uses the received sensing status to update every cell inside the square by (1) if it contains an intruder and (0) otherwise.
4) It computes the relative address P_{rel} of the targeted point P_T in proportion to the current position of the agent.
5) Normalize the value of the targeted point relative position.
6) It generates the current state S_t where $S_t = \{P_{\text{rel}} = (x_{\text{rel}}, y_{\text{rel}}), \Delta\}$.
7) It uses the prior learned knowledge to choose the best action (direction) that might allow the agent to not collide with any nearby intruder and get closer to its target.
8) Apply the selected action and observe the impact of the agent's dynamic on the environment by recording the new knowledge in terms of reward earned and new agent position.
9) The agent repeats the previous steps until reaching the targeted point P_T or an instance of collision occurs.

Indeed, as shown in Fig. 5, state S_t introduced at each step highly impacts the learning process and the action selection. Thus, the state's definition needs to be done based on clear and logical arguments. In what follows, we detail our logic behind the definition of state S_t.

First, instead of using the 2-D coordinate of the target, the use of the relative address allows the agent to move in the direction of the targeted position wherever its position is in the environment. Furthermore, the use of the relative address allows the agent to involve the remaining distance to the target in the learning process of the agent. Thus, we keep the agent seeking the shortest way to move on.

On the other side, the focus on the square surrounding the agent position instead of considering the whole environment area aims to aggregate many environment states to one state at the agent. Let Δ denote the surrounding area of the agent. The size of the surrounding area is a hyper-parameter that can be tuned during the training step. As shown in Fig. 5, three different environment states can be presented by the same state. However, they are the intruders' position beyond the square, and the state is the same since it considers only the intruders positioned inside the square and the relative targeted position. Indeed, the agent will know a limited number of states. Furthermore, the nonuse of related intruders features (e.g., intruders coordinates) aims to have a generalized view that might be used in similar status even with different intruders' positions. Finally, the use of such a state might help improve the learning convergence time of the agent. Moreover, this approach helps to train the agent on a limited number of intruders and can also be functional in an environment with a high number of intruders.

D. RELIANCE DQN Algorithm

Throughout this section, we detail the RL approach used by our based RL agent. Several RL approaches exist in the literature, such as the Q-learning, deep neuron network (DQN), policy gradient, and actor–critic. However, the elements state-space and the action-space from the RL modeled system are either deterministic or continuous, highly impacting the selection of the RL approach. In our case, eight deterministic actions compose the action space. On the other side, the state-space contains two deterministic elements: 1) the relative address and 2) the square-shaped area. The Q-learning approach requires that the agent is within a deterministic limited space. Indeed, the Q-learning approach seems to be the most suitable for our problem. However, the size of the square-shaped area might be considerable. Then, the agent could take a long time to converge, and consequently, the computation process will consume more resources. Furthermore, the learning process can be less efficient by getting a useless action.

To mitigate this problem, we opted to use the DRL (DQN) approach. Thus, more details about our based DQN autonomous UAV collision-avoidance and energy-aware agent are summarized in Algorithm 2. First, the agent starts by instantiating two different NNs, named the policy network Q^{π}, and target network Q^{T}. To ensure the fast convergence of the algorithm, we have used Xavier initialization to initialize the weights of both NNs Q^{π} and Q^{T}. The Xavier initialization helps to converge fast and prevent the exploding and vanishing gradients during the training process. To give the agent

Algorithm 2: RELIANCE: RL-Based Collision-Avoidance Solution

```
Input :
        $Q^{\pi}$ and $Q^{T}$: The initialized policy and target networks using Xavier-uniform.
        $\mathcal{B}$: The batch replay memory size to size $N$.
        $batch\_size$: The size of each batch.
        $\mathcal{M}$: A number of episodes to update $Q^{T}$ with $Q^{\pi}$.
        $\xi_0$: The initial value of epsilon greedy.
        $\mathcal{N}$: Number of episodes.
Output :
        done: The UoI reaches $P_T$ or collides with one of the intruders.
1  $episode = 1$;
2  while $episode \leq N$ do
3      $done \leftarrow False$;
4      $\xi \leftarrow \frac{\xi_0}{\xi_0 + episode}$;
5      $S_0 = \mathcal{E}.init()$;
6      while $done = False$ do
7          $S_t = \{P_{rel} = (x_{rel}, y_{rel}), \Delta\}$;
8          if $random() \leq \xi$ then
9              $a \leftarrow randint(\mathcal{A})$;
10         else
11             $a \leftarrow \arg\max_{a \in A} Q^{\pi}(S_t, a)$;
12         end
13         $S_{t+1}, reward, done \leftarrow \mathcal{E}(S_t, a)$ ;
14         $\mathcal{B} \leftarrow (S_t, a, reward, S_{t+1}, done)$;
15         $t \leftarrow t + 1$;
16         if $size(\mathcal{B}) \geq batch\_size$ then
17             $mini\_batch \leftarrow random(\mathcal{B}, batch\_size)$;
18             foreach $(S_i, a_i, reward_i, done_i, S'_i) \in mini\_batch$ do
19                 if $(done_i = True)$ then
20                     $y_i \leftarrow reward_i$;
21                 else
22                     $y_i \leftarrow reward_i + \gamma \max_{a' \in \mathcal{A}} Q^{T}(S'_i, a'_i)$;
23                 end
24             end
25             $\mathcal{L} = \frac{1}{N} \sum_{i=0}^{i=N-1} (Q^{\pi}(S_i, a_i), y_i)^2$;
26             Update $\omega$ of $Q^{\pi}$ using $\mathcal{L}$;
27         end
28     end
29     if $episode \% \mathcal{M} = 0$ then
30         $Q^{T} \leftarrow Q^{\pi}$ ;
31     end
32     $episode = episode + 1$;
33 end
```

more time to explore the behavior of the actions set, we opted to use the decayed ϵ-greedy strategy. The algorithm starts from the first episode and ends at the last episode N (Algorithm 2: lines 1 and 2). For each episode (Algorithm 2: lines 2–33), RELIANCE does the next steps. Initially, the episode sets to an undone state (Algorithm 2: line 3). Then, ξ is initialized to enable either the exploration or exploitation (Algorithm 2: line 4). Later, the environment is initialized by creating a new mission to train the agent (Algorithm 2: line 5).

While the episode is not completed (UoI achieves the target or collides), we do the following steps (Algorithm 2: lines $6-28$): first, the agent generates and normalize the current state (Algorithm 2: line 7). Then, according to the decayed value of ξ and a randomly generated number, we select either exploration or exploitation (Algorithm 2: lines 8–12). If the exploration is selected, a random action is issued from $\mathcal{A}$ (Algorithm 2: lines 8–10). Otherwise, the agent of the *UoI* chooses the action with the maximum reward previously earned using the policy network (Algorithm 2: lines 10–12). After the agent applies the selected action and saves the transition to $\mathcal{B}$ (Algorithm 2: lines 13 and 14), the agent moves to the new observed state (Algorithm 2: line 15). However, when the number of experiences exceeds the *batch_size*, the agent selects a random batch of transitions from $\mathcal{B}$ to update the Q^{π} following $TD(0)$ (Algorithm 2: lines 16–27). The agent keeps updating the Q^{T} using Q^{π} every $\mathcal{M}$ steps. The agent repeats the previous steps until the end of all the episodes in the training.

VI. Experimentation and Results

In this section, we evaluate the performances of our two solutions PICA and RELIANCE. In the balance of this section, we first present the simulation setup; then, we present the convergence of RELIANCE during the training mode. Last but not least, we conclude this section by evaluating the performances of RELIANCE in inference mode to PICA.

A. Simulation Setup

Existing UAV simulators, such as Air Sim and software in the loop (SITL), use telemetry data to control the motion of a single UAV in a closed and well-controlled environment. These simulators mainly focus on telemetry data to maintain a single UAV for landing and flying. Still, they did not consider mobile objects, which is a handicap facing their utilization to evaluate PICA and RELIANCE solutions. In order to overcome these limitations, we have developed an OpenAI Gym [35] compliant simulator[1] with graphical rendering capability using the Python language and OpenCV library. This simulator provides a customizable environment that considers both static (e.g., building) and dynamic (e.g., UAVs and birds) obstacles. The static obstacles can be included in a JSON format to the simulator. In the simulator, we have adopted a discrete-time implementation of the events (e.g., UAVs mobility). This strategy helps to reduce the simulation time significantly by considering only the counted events rather than using real execution time. To make the proposed framework orthogonal on agents (PICA, RELIANCE…) implementation, we have developed a complete framework that consists of an abstraction layer of the agent and environment. We have also designed RELIANCE and PICA to be transparent in the environment and easily adapted to other simulators or real experiments later. Thus, we believe the suggestion of this simulator will have an added value to the scientific community. While PICA has been implemented using Python and Numpy, the NN model of RELIANCE is implemented with Python and Pytorch library.

Besides the UoI, the environment also consists of a set of customizable number of *static_intruders* and *mobile_intruders*. As aforementioned, both UoI and *mobile_intruders* move in a 2-D plan using eight possible actions. The *mobile_intruders* move in the simulator using a random walk technique. In contrast to *mobile_intruders*, the

[1]https://youtu.be/7UcRxfaAREw

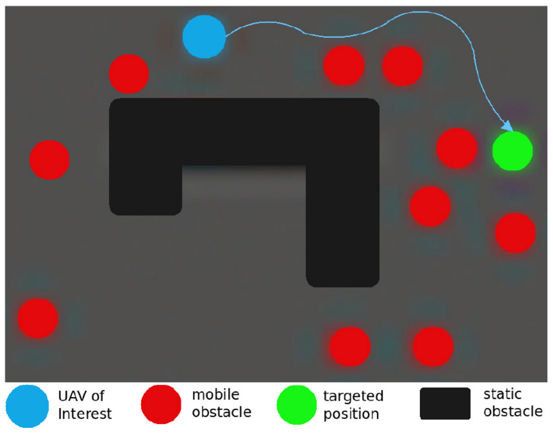


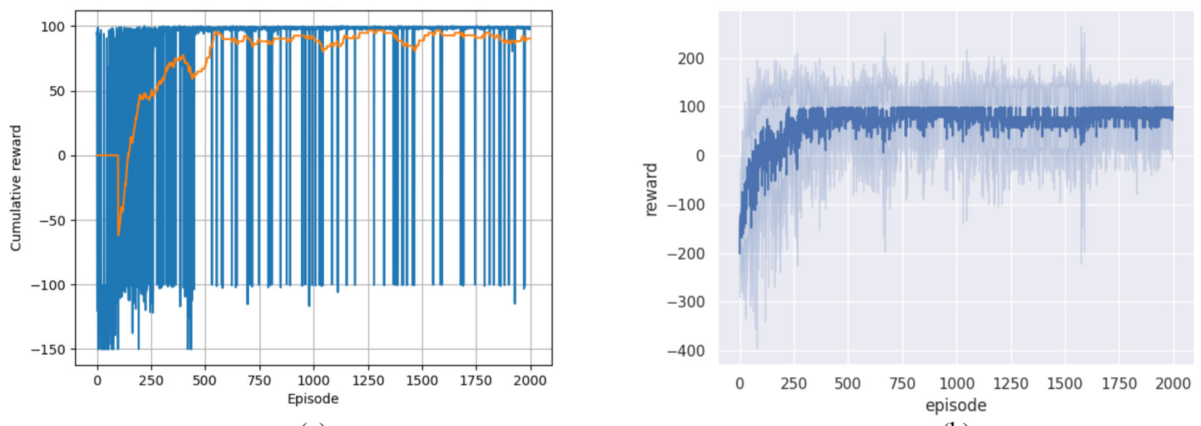
Fig. 6. OpenAI Gym compliant simulator.

UoI moves under the control of either PICA or RELIANCE agents. The rendering environment consists of a gray screen with black rectangles and blue, green, and red circles. As depicted in Fig. 6, the black rectangles and red circles refer to the static obstacles and the intruder(s), respectively. Meanwhile, the blue and green circles refer to the UoI and its targeted position, respectively. The simulation runs in episodes, such that each of which ends when UoI collides or reaches the target.

B. RELIANCE Training Mode

The training of the RELIANCE model happens using 14 Dual Intel Xeon E5-2680 v3 @ 2.5 GHz, with 117 GB of RAM, one Nvidia P100 GPU, and running CentOS 7. During the training process, we have fixed the size of the simulation area by 20×20 and considered ten *mobile_intruders* besides three static obstacles with different shapes and sizes. We have fixed the hyperparameter surrounding area Δ of the agent by 5×5 after performing a set of different tests. To ensure fast convergence without underfitting or overfitting, we have tuned the NN hyperparameters used by RELIANCE. We have performed many experimental tests before fixing the hyperparameters. We have fixed the discount factor γ by 0.95 and the learning rate α by 10^{-4}. We have also used two fully connected hidden layers in which the number of units (i.e., activation functions) is 40. We have also tested with 400 units in each layer. However, a similar convergence rate is perceived. We have also observed similar performances during the inference mode. We adopted the rectified linear unit (ReLU) activation function for both hidden and output layers. An Xavier initialization has been adopted to initialize the NN units in the model. This initialization helps to converge fast and prevent the exploding and vanishing gradients during the training process. During the training, we have used $batch\_size = 1024$, replay buffer size $= 500\,000$, and target update $= 8$ to update the weight of target network Q^{T} from the policy network Q^{π}.

As depicted in Fig. 7, we have conducted two sets of experiments. Initially, we have trained one RELIANCE agent as depicted in Fig. 7(a) for a period of 2000 episodes. In this figure, while the blue curve shows the cumulative reward gained



(a) (b)

Fig. 7. Convergence evaluation of RELIANCE during the training mode. (a) Immediate and average rewards. (b) Average rewards of 40 agents simultaneously.

at the end of each episode, the red one shows the average of the last 50 cumulative rewards. From this figure, we observe that the RELIANCE agent converges at 600 episodes. Starting from that point, the RELIANCE agent succeeds in most of the cases to achieve the target without any collision. A live video has been recorded that shows the convergence of RELIANCE.[2]

Meanwhile, in Fig. 7(b), we have evaluated RELIANCE agent's stability. The NN's bias and weights are randomly initialized in the RELIANCE agent, affecting the training convergence. Moreover, at each episode, the starting and target point of UoI, and the mobility of *mobile_intruders* are randomly generated. In Fig. 7(b), we have trained 40 RELIANCE agents, simultaneously. In this figure, we have evaluated both the average and the cumulative variance reward achieved. We observe that all the agents succeeded in converging by getting almost the total possible reward after only 400 episodes. Also, we observe that the variance between the trained agents is close to 0, which confirms the algorithm's convergence.

C. PICA and RELIANCE Performance Evaluation During the Inference Mode

In this section, we evaluate the performances of RELIANCE in the inference mode against the PICA solution. We simulate 10^3 episodes and compare the two solutions in terms of the following metrics.

1) *Percentage of Collision:* It is defined as the percentage of times that the UAV agent collides with *static_intruders* or *mobile_intruders*. This metric shows the percentage of time that the UAV agent fails to achieve its final destination.
2) *PDF of Extra Traveled Distance:* It shows the extra distance needed by a UAV to prevent collisions. This metric shows the probability of a distribution function (PDF) of the extra distance traveled to avoid collisions. In fact, the energy consumption in the UAVs is proportional to the traveled distance before attending the target location. Overall, the more the traveled distance, the higher energy consumption becomes.

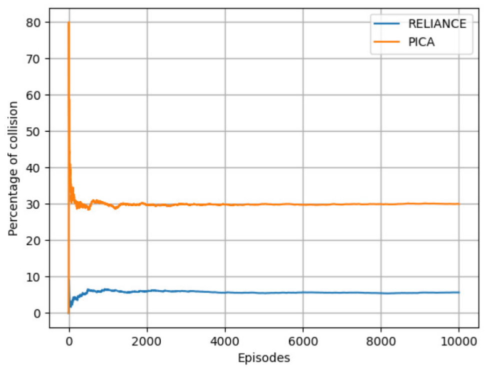
[2]https://youtu.be/5ULpSuMdRSE

3) *PDF of the Number of Success Before a Failure:* It shows the PDF of the number of successes arriving at the target before the failure, i.e., the UoI collides with any object. In other words, this metric shows the capability of each solution for traveling consecutive missions without any collision.

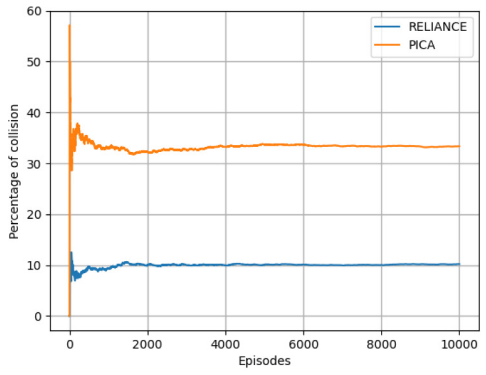
To assess the generalization capability of RELIANCE, we have considered three different scenarios during the inference mode. As aforementioned, we have trained RELIANCE agent against static obstacles and ten *mobile_intruders*. In contrast, during the inference mode, besides the three static obstacles, we have evaluated the performance of PICA and RELIANCE agents against 5, 10, and 20 *mobile_intruders*, respectively. The idea behinds these three scenarios is to show the capability of RELIANCE to outlive in unfamiliar environments by leveraging the effectiveness of surrounding area Δ and aggregated state.

1) Percentage of Collision: Fig. 8 shows the percentage of collisions as a function of the number of episodes. Both solutions have been evaluated in harsh conditions by including static obstacles and many *mobile_intruders* in a small area with dimensions 20×20. Moreover, the *mobile_intruders* move randomly without any correlation, which makes hard to predict their next movement.[1] The first observation that we can draw from this figure is that RELIANCE ensures the generalization by behaving well in unseen environments (e.g., 5 and 20 *mobile_intruders*). We also observe that whatever the scenarios (5, 10, or 20 *mobile_intruders*), the RELIANCE offers better performances than PICA. As expected, we also observe that the number of *mobile_intruders* has a negative impact on the number of collisions as shown in Fig. 8(a)–(c), respectively.

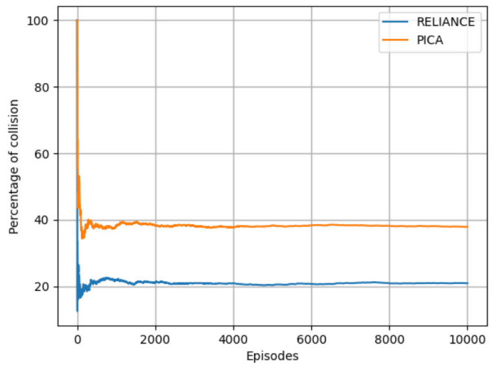
For five *mobile_intruders* as depicted in Fig. 8(a), regardless of the number of episodes, the PICA agent has arrived at the target without collision with 70% of success. Whereas, the RELIANCE agent has succeeded with 95% to reach the target while avoiding the collisions. Increasing the number of *mobile_intruders* to 10 hurts the collision percentage in the network as depicted in Fig. 8(b). We observe that the percentage of cases that the UAV agent arrives at the target without collisions drooped out from 70% and 95% to 65% and 90% for PICA and RELIANCE, respectively. Finally, as depicted



(a)



(b)



(c)

Fig. 8. Percentage of collision in PICA and RELIANCE solutions. (a) RELIANCE and PICA against five *mobile_intruders*. (b) RELIANCE and PICA against ten *mobile_intruders*. (c) RELIANCE and PICA against 20 *mobile_intruders*.

in Fig. 8(c), we observe that the increase of the number of *mobile_intruders* to 20 leads to reduce the percentage of success to arrive at the destination without collisions to 60% and 80% for PICA and RELIANCE, respectively.

The better performances achieved by RELIANCE compared to PICA can be explained as follows. In both solutions, the algorithm controlling the UoI makes the decisions relying on the snapshot from the environment to avoid collisions. The environment snapshot refers to the surrounding area of UoI that is defined by Δ and $\mathcal{Z}$ in RELIANCE and PICA, respectively. On the one hand, based on this snapshot, PICA takes the action that minimizes the likelihood of collisions in $\mathcal{Z}$. Nonetheless, PICA does not consider the dynamics of the mobile intruders within $\mathcal{Z}$. In contrast, RELIANCE using the DRL approach can learn the temporal correlation between different snapshots Δ and therefore make more effective decisions to avoid mobile intruders. This fact explains why PICA exhibits a higher number of collisions compared to RELIANCE.

2) PDF of Extra Traveled Distance: Fig. 9 shows the performances of PICA and RELIANCE related to energy saving. Unfortunately, avoiding the collision comes with an unavoidable overhead in terms of the extra distance traveled by the UoI. This figure shows this extra distance compared to the traveled distance in the straight travel, i.e., the Euclidean distance between the UoI starting and target points. We have estimated the PDF of the PICA and RELIANCE extra distances from the results from 10^3 episodes. To that end, we employed the KernelDensity function from sklearn.neighbors. We observe that regardless of the number of mobile intruders (5, 10, or 20), the percentage of extra distance does not exceed 60%, i.e., the UoI travels 1.6 times the distance of the optimal path.

Fig. 9(a) and (b) shows the PDF of the extra traveled distance for five mobile intruders. From Fig. 9(a), RELIANCE succeeded in almost 90% of cases to add only 35% of extra distance compared to the optimal one (i.e., 1.35 times). With more than 0.08 probability, RELIANCE succeeded in traveling the distance with less than 10% extra distance. We also observe from 9(b) that PICA succeeded in reaching the target in 70% of cases without adding any extra distance. Also, most of the extra distance of PICA does not exceed 40%. Observe that PICA offers shorter extra traveled distances than RELIANCE, which, overall, translates into energy saving. However, this reduction in the traveled extra distance offered by PICA is at the cost of a higher probability of collision, as discussed previously.

Fig. 9(c)–(f) shows the PDF of extra traveled distance for 10 and 20 mobile intruders, respectively. Similar to the case of five mobile intruders, we observe that the PICA algorithm succeeded in most of the cases without adding any extra distance. Interestingly, we observe that increasing the number of mobile intruders reduces the extra traveled distance offered by the PICA solution. This can be explained as follows, in the simulation, the extra distance of incomplete mission are filtered (not considered). At each episode, the starting and

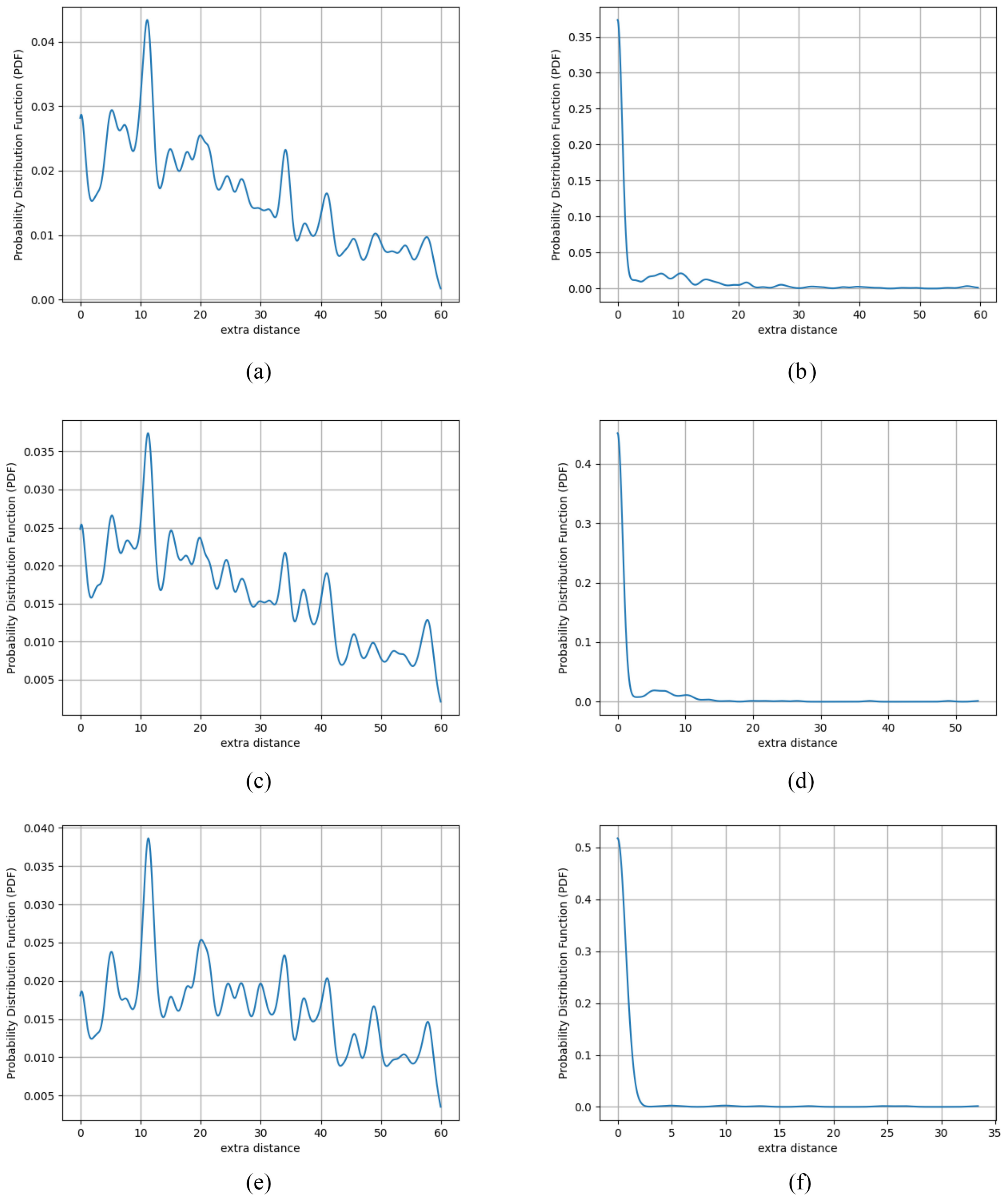


Fig. 9. PDF of extra traveled distance. (a) PDF of extra traveled distance in RELIANCE (five *mobile_intruders*). (b) PDF of extra traveled distance in PICA (five *mobile_intruders*). (c) PDF of extra traveled distance in RELIANCE (ten *mobile_intruders*). (d) PDF of extra traveled distance in PICA (ten *mobile_intruders*). (e) PDF of extra traveled distance in RELIANCE (20 *mobile_intruders*). (f) PDF of extra traveled distance in PICA (20 *mobile_intruders*).

target point (i.e., mission) of UoI are randomly generated. In fact, increasing the number of intruders will create more collisions on the long distance missions comparing to the short ones. Hence, more short distance mission will participate for generating the PDF of extra distance. Usually, the probability of adding extra distance in shorter mission is lower than the longer ones, which positively affects the PDF of extra distance metric. Meanwhile, from Fig. 9(a), (c), and (e), we observe that similar behavior in terms of extra traveled distance. The RELIANCE solution succeeded to save long distance mission, however with unavoidable extra distance.

The extra traveled distance and percentage of collision are two contradictory objectives. The lower percentage of collision is, the higher likelihood of extra traveled distance becomes. While the PICA solution leverages a probabilistic approach by considering only one snapshot of the environment, RELIANCE employs DRL to make the correlation between snapshots and then takes the decisions that consider the mobility of intruders. The safety level, i.e., low probability of collision with surrounding intruders, offered by RELIANCE is at the expense of traveling longer extra distances.

3) PDF of the Number of Success Before Failure: Fig. 10 depicts the PDF of the number of success before a failure happens. It shows the PDF of the number of hits arriving at the target before the collapse. This metric shows the capability of each solution for traveling consecutive missions without

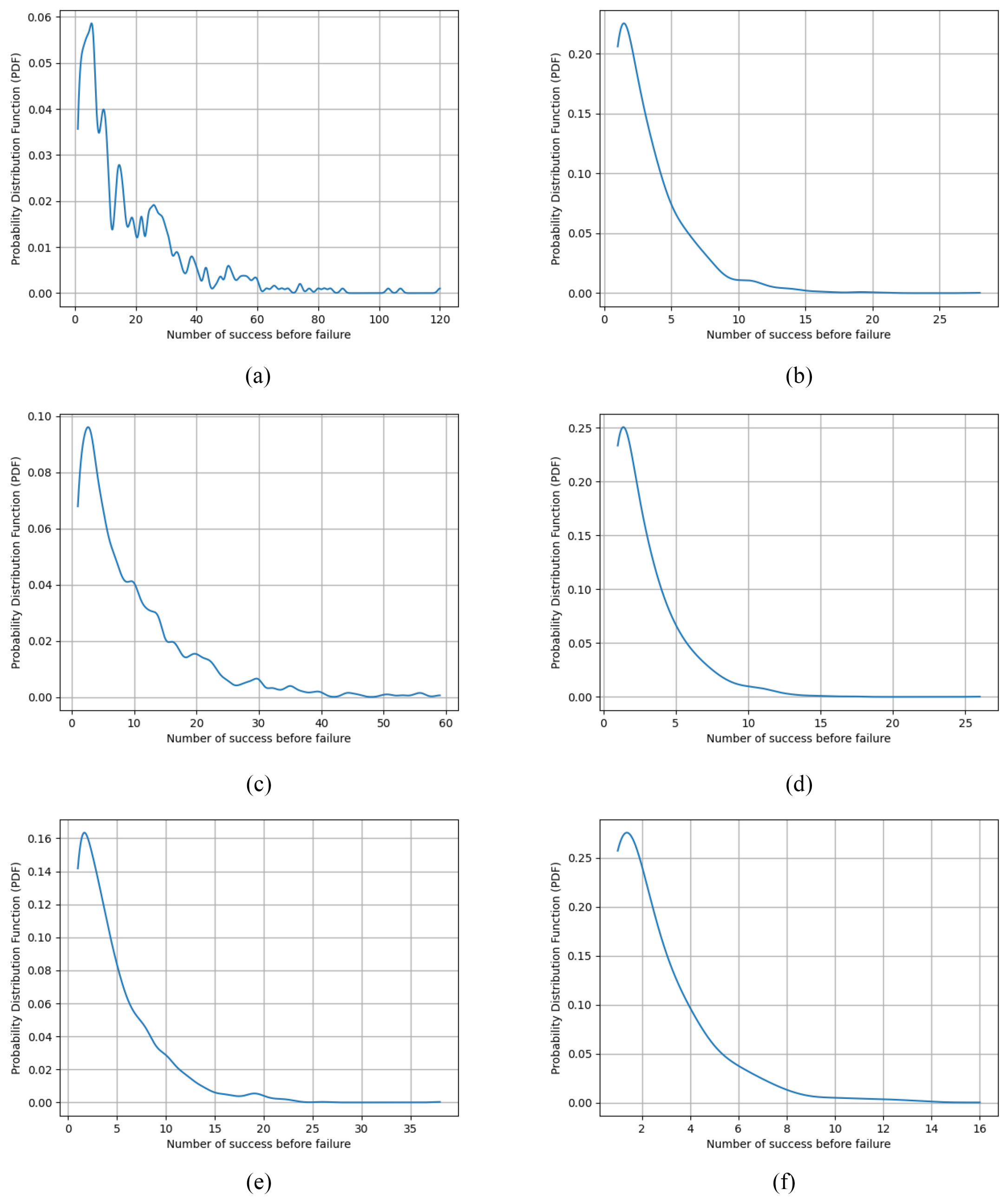


Fig. 10. PDF of extra traveled distance. (a) PDF of the number of success before a failure in RELIANCE (five intruders). (b) PDF of the number of success before a failure in PICA (five intruders). (c) PDF of the number of success before a failure in RELIANCE (ten intruders). (d) PDF of number of success before a failure in PICA (ten intruders). (e) PDF of the number of success before a failure in RELIANCE (20 intruders). (f) PDF of the number of success before a failure in PICA (20 intruders).

any collision. We have conducted three sets of experiments by varying the number of *mobile_intruders* from 5, 10, and 20, respectively. The first observation that we can draw from this figure is that the RELIANCE solution performs better than the PICA solution. Also, we observe that the number of *mobile_intruders* harms the number of successes before failure.

Fig. 10(a) and (b) shows the performances of PICA and RELIANCE when five *mobile_intruders* is considered. As depicted in these figures, while RELIANCE succeeded in getting 120 successful episodes achieving the target safely, PICA succeeded in achieving the target in 30 episodes without any single failure. We also observe that RELIANCE's probability of fewer than five times consecutively arriving at the target without interruption does not exceed 12%. Meanwhile, in PICA, UoI with a probability of almost 1 does not exceed the 15 episodes consecutively. Fig. 10(c) and (d) shows the impact of ten *mobile_intruders* on PICA and RELIANCE. We can observe that the increase in the number of *mobile_intruders* hurts the number of successes before failure. In RELIANCE, the number of successful episodes before a failure is drooped from 120 to 60. Also, the probability of five consecutive times arrive at the target without interruption does not exceed 20%. Finally, Fig. 10(e) and (f) shows the impact of ten *mobile_intruders* on the two solutions. We observe that in $1/3$ of cases RELIANCE, the number of success before a failure does not exceed the threshold 5. Meanwhile, for PICA, the agent with almost probability 1 does not succeed to exceed 15 episodes.

VII. Conclusion

The new enthusiasm for extending UAV commercial operations to cover the urban and populated area controlled airspace BVLOS comes with unavoidable challenges related to object detection and collision avoidance. In this article, we suggested two solutions named: 1) PICA framework and 2) RELIANCE. While the PICA solution leverages the probability density for avoiding collisions, RELIANCE uses the DQN technique to prevent collisions while saving energy consumption. We have also developed an OpenAI Gym [35] compliant environment[1] with graphical rendering capability using Python language and OpenCV library to evaluate these two solutions. We have developed a complete framework that includes an abstraction of the environment and agent. Our plan to make the platform's code source, including PICA and RELIANCE agents, public for the research community.

We have simulated the agent in the context of both PICA and RELIANCE under similar circumstances. The agent behaves successively following PICA or RELIANCE to prevent the collision and save energy consumption. We have evaluated both protocols in known and unknown environments to assist their generalization capability. The obtained results demonstrate their capacity for generalization. Also, they show the superiority of RELIANCE over PICA in terms of collision avoidance. Also, the simulation results demonstrated the convergence of RELIANCE during the training process[2].

As a future research direction, we plan to consider other RL Algorithms, including 1) the Policy gradient method, such as RELIANCE; Actor–Critic approach, including but not limited to A3C, deep deterministic policy gradient (DDPG), trust region policy optimization (TRPO), and proximal policy optimization (PPO). Also, we plan to consider more complex scenarios by considering the velocity and the acceleration of UAVs. A real deployment implementation is envisaged of RELIANCE by leveraging the UAVs available in our lab.

References

[1] N. H. Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based Internet of Things services: Comprehensive survey and future perspectives," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 899–922, Dec. 2016.

[2] D. Jaiswal and P. Kumar, "Real-time implementation of moving object detection in UAV videos using GPUs," *J. Real Time Image Process.*, vol. 17, no. 5, pp. 1301–1317, 2020.

[3] S. Ouahouah, J. Prados-Garzon, T. Taleb, and C. Benzaid, "Energy-aware collision avoidance stochastic optimizer for a UAVs set," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, 2020, pp. 1636–1641.

[4] T. Taleb, K. Ooi, and K. Hashimoto, "An efficient collision avoidance strategy in ITS systems," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2008, pp. 2212–2217.

[5] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Toward an effective risk-conscious and collaborative vehicular collision avoidance system," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2010.

[6] F. Giancarmine *et al.*, "Multi-sensor-based fully autonomous non-cooperative collision avoidance system for unmanned air vehicles," *J. Aerosp. Comput. Inf. Commun.*, vol. 5, no. 10, pp. 338–360, 2008.

[7] R. Sharma and D. Ghose, "Collision avoidance between UAV clusters using swarm intelligence techniques," *Int. J. Syst. Sci.*, vol. 40, no. 5, pp. 521–538, 2009.

[8] Y. K. Kwag and C. H. Chung, "UAV based collision avoidance radar sensor," in *Proc. IEEE Int. Geo-Sci. Remote Sens. Symp. (IGARSS)*, Jul. 2007, pp. 639–642.

[9] J. W. Park, H. D. Oh, and M. J. Tahk, "UAV collision avoidance based on geometric approach," in *Proc. SICE Annu. Conf.*, Aug. 2008, pp. 2122–2126.

[10] J. P. K. Kim and M. Tahk, "UAV collision avoidance using probabilistic method in 3-D," in *Proc. Int. Conf. Control Autom. Syst.*, Aug. 2007, pp. 826–829.

[11] M. Shanmugavel, A. Tsourdos, and B. A. White, "Collision avoidance and path planning of multiple UAVs using flyable paths in 3D," in *Proc. 15th Int. Conf. Methods Models Autom. Robot.*, Aug. 2010, pp. 218–222.

[12] Z. Chao, L. Ming, Z. Shaolei, and Z. Wenguang, "Collision-free UAV formation flight control based on nonlinear MPC," in *Proc. Int. Conf. Electron. Commun. Control (ICECC)*, Sep. 2011, pp. 1951–1956.

[13] J. G. Manathara and D. Ghose, "Reactive collision avoidance of multiple realistic UAVs," *Aircraft Eng. Aerosp. Technol.*, vol. 83, no. 6, pp. 388–396, 2011. [Online]. Available: https://doi.org/10.1108/00022661111173261

[14] M. C. P. Santos, C. D. Rosales, M. Sarcinelli-Filho, and R. Carelli, "A novel null-space-based UAV trajectory tracking controller with collision avoidance," *IEEE/ASME Trans. Mechatronics*, vol. 22, no. 6, pp. 2543–2553, Dec. 2017.

[15] L. A. Tony, D. Ghose, and A. Chakravarthy, "Avoidance maps: A new concept in UAV collision avoidance," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Jun. 2017, pp. 1483–1492.

[16] R. He, R. Wei, and Q. Zhang, "UAV autonomous collision avoidance approach," *Automatika*, vol. 58, no. 2, pp. 195–204, 2017. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00051144.2017.1388646

[17] Y. Lin and S. Saripalli, "Sampling-based path planning for UAV collision avoidance," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 11, pp. 3179–3192, Nov. 2017.

[18] S. Ouahouah, J. Prados-Garzon, T. Taleb, and C. Benzaid, "Energy and delay aware physical collision avoidance in unmanned aerial vehicles," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 1–7.

[19] S. Y. Choi and D. Cha, "Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art," *Adv. Robot.*, vol. 33, no. 6, pp. 265–277, 2019.

[20] A. Singla, S. Padakandla, and S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 1, pp. 107–118, Jan. 2021.

[21] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot," *Appl. Sci.*, vol. 9, no. 24, p. 5571, Dec. 2019. [Online]. Available: http://dx.doi.org/10.3390/app9245571

[22] P. Fraga-Lamas, L. Ramos, V. Mondéjar-Guerra, and T. M. Fernández-Caramés, "A review on IoT deep learning UAV systems for autonomous obstacle detection and collision avoidance," *Remote Sens.*, vol. 11, no. 18, p. 2144, 2019. [Online]. Available: http://dx.doi.org/10.3390/rs11182144

[23] D. Wang, T. Fan, T. Han, and J. Pan, "A two-stage reinforcement learning approach for multi-UAV collision avoidance under imperfect sensing," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 3098–3105, Apr. 2020.

[24] K. Wan, X. Gao, Z. Hu, and G. Wu, "Robust motion control for UAV in dynamic uncertain environments using deep reinforcement learning," *Remote Sens.*, vol. 12, no. 4, p. 640, 2020. [Online]. Available: http://dx.doi.org/10.3390/rs12040640

[25] B. M. Albaker and N. A. Rahim, "A survey of collision avoidance approaches for unmanned aerial vehicles," in *Proc. Int. Conf. Tech. Postgraduates (TECHPOS)*, Dec. 2009, pp. 1–7.

[26] L. Xiao, W. Zhuang, S. Zhou, and C. Chen, *UAV Relay in VANETs Against Smart Jamming With Reinforcement Learning*. Cham, Switzerland: Springer, 2019, pp. 105–129. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-01731-6_5

[27] L. Xiao, X. Lu, D. Xu, Y. Tang, L. Wang, and W. Zhuang, "UAV relay in VANETs against smart jamming with reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4087–4097, May 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8246580

[28] O. Bekkouche, T. Taleb, and M. Bagaa, “UAVs traffic control based on multi-access edge computing,” in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.
[29] T.-H. Yi, H.-N. Li, and M. Gu, “Experimental assessment of high-rate GPS receivers for deformation monitoring of bridge,” *Measurement*, vol. 46, no. 1, pp. 420–432, Aug. 2012.
[30] S. V. Amarasinghe, H. S. Hewawasam, W. B. D. K. Fernando, J. V. Wijayakulasooriya, G. M. R. I. Godaliyadda, and M. P. B. Ekanayake, “Vision based obstacle detection and map generation for reconnaissance,” in *Proc. 9th Int. Conf. Ind. Inf. Syst. (ICIIS)*, 2014, pp. 1–6.
[31] D. Kim and H. Ryu, “Obstacle recognition system using ultrasonic sensor and duplex radio-frequency camera for the visually impaired person,” in *Proc. 13th Int. Conf. Adv. Commun. Technol. (ICACT)*, 2011, pp. 326–329.
[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html
[33] V. Mnih *et al.*, “Playing Atari with deep reinforcement learning,” 2013. [Online]. Available: http://arxiv.org/abs/1312.5602.
[34] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236
[35] *Gym*. Accessed: May 26, 2021. [Online]. Available: https://gym.openai.com/envs/#classic_control

Sihem Ouahouah received the Engineering degree from the University of Science and Technology Houari Boumediene, Bab Ezzouar, Algeria, in 2005, and the master’s degree in computer science from Ecole Nationale Supérieure d’Informatique, Oued Smar, Algeria, in 2015. She is currently pursuing the Doctoral degree with Aalto University, Espoo, Finland.

Her research interests include unmanned aerial vehicles, machine learning, and the Internet of Things.

Miloud Bagaa (Senior Member, IEEE) received the engineer’s, master’s, and Ph.D. degrees from the University of Science and Technology Houari Boumediene, Bab Ezzouar, Algeria, in 2005, 2008, and 2014, respectively.

From 2009 to 2015, he was a Researcher with the Research Center on Scientific and Technical Information, Ben Aknoun, Algeria. From 2015 to 2016, he was granted a Postdoctoral Fellowship from the European Research Consortium for Informatics and Mathematics, and worked with the Norwegian University of Science and Technology, Trondheim, Norway. From 2016 to 2019, he was a Postdoctoral Researcher with Aalto University, Espoo, Finland, then a Senior Researcher from 2019 to October 2020, where he is currently a Senior Cloud Specialist with the IT Center For Science LTD. His research interests include machine learning, optimization, networking modeling, and network slicing.

Jonathan Prados-Garzon received the B.Sc., M.Sc., and Ph.D. degrees from the University of Granada, Granada, Spain, in 2011, 2012, and 2018, respectively.

From 2018 to 2020, he worked as a Postdoctoral Researcher with MOSA!C Lab, led by Prof. T. Taleb, and the Department of Communications and Networking, Aalto University, Espoo, Finland. He is currently a Postdoctoral Researcher with WiMuNet Lab, headed by Prof. J. M. L. Soler, and the Department of Signal Theory, Telematics and Communications, University of Granada. His research interests include mobile broadband networks, network softwarization, deterministic networking, and network performance modeling and optimization.

Tarik Taleb received the B.E. degree (with Distinction) in information engineering and the M.Sc. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 2001, 2003, and 2005, respectively.

He is currently a Professor with the School of Electrical Engineering, Aalto University, Espoo, Finland, where he is the Founder and Director of the MOSA!C Lab. He is also working as a Part Time Professor with the Center of Wireless Communications, University of Oulu, Oulu, Finland. Prior to his current academic position, he was working as a Senior Researcher and a 3GPP Standards Expert with NEC Europe Ltd., Heidelberg, Germany. He is also affiliated with Sejong University, Seoul, South Korea. He was then leading the NEC Europe Labs Team working on Research and Development projects on carrier cloud platforms, an important vision of 5G systems. Before joining NEC and till March 2009, he worked as an Assistant Professor with the Graduate School of Information Sciences, Tohoku University, Sendai, in a lab fully funded by KDDI. From October 2005 to March 2006, he worked as a Research Fellow with the Intelligent Cosmos Research Institute, Sendai. He has been also directly engaged in the development and standardization of the Evolved Packet System as a member of 3GPP’s System Architecture working group. His research interests lie in the field of architectural enhancements to mobile core networks (particularly, 3GPP’s), network softwarization and slicing, mobile cloud networking, network function virtualization, software defined networking, mobile multimedia streaming, intervehicular communications, and social media networking.

Prof. Taleb was a recipient of the 2017 IEEE ComSoc Communications Software Technical Achievement Award in December 2017, for his outstanding contributions to network softwarization. He is also the (co)-recipient of the 2017 IEEE Communications Society Fred W. Ellersick Prize in May 2017, the 2009 IEEE ComSoc Asia–Pacific Best Young Researcher Award in June 2009, the 2008 TELECOM System Technology Award from the Telecommunications Advancement Foundation in March 2008, the 2007 Funai Foundation Science Promotion Award in April 2007, the 2006 IEEE Computer Society Japan Chapter Young Author Award in December 2006, the Niwa Yasujirou Memorial Award in February 2005, and the Young Researcher’s Encouragement Award from the Japan chapter of the IEEE Vehicular Technology Society in October 2003. Some of his research works have been also awarded best paper awards at prestigious IEEE-flagged conferences. He is a member of the IEEE Communications Society Standardization Program Development Board. He is/was on the Editorial Board of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, *IEEE Wireless Communications Magazine*, IEEE JOURNAL ON INTERNET OF THINGS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, and a number of Wiley journals.