Autonomous Drone Swarm Deployment

Sariah Al Saati, Mehdi Benharrats, Swann Chelly, Tom Marty, Pierre Tessier

***Abstract*—This paper proposes a method for the coverage of a warzone with a swarm of UAV's in order to detect possible target of interest based on colaborative reinforcement learning. This study has been carried out for the 2020 challenge hosted by Ecole Polytechnique in partnership with the French General Direction of Armement (DGA). The method proposed includes field coverage and also target detection in a multi agent system.**

Introduction

For this 2020/2021 edition of the DGA challenge, the goal is to cover an area with a swarm of UAV's and a group of terrestrial robots in order to find individuals hiding in the area. This study is being motivated by the need to help troops to cover a battlefield or a rescue zone. In a battlefield, robots are exposed to destruction by hostile troops, while in a rescue operation, robots must civilians hidden under debris.

For such a goal, there are two main challenges that need to be overcome. The first one is to handle the detection of the individuals and the understanding of the environment. To do so, image analysis for each robot is primordial. The detection pipeline is divided in two steps, a processing step and a post-processing step . The first step consists in detecting points of interest such as humans, buildings or vehicles in 2D images generated by on-board cameras. This task is carried out by the well-known YOLO V5 algorithm that is able to detect the presence of objects of interest, along with their nature and their bounding boxes. Then, the post-processing step consists in inferring the 3D position of points of interest using clustering on directions of detection.

The second challenge that must be taken into account is the idea that robots must collaborate with each other in order to improve their understanding of the environment. Two kinds of approach can be used here. On the one hand we can go deep into the literature concerning deterministic algorithm that manage coverage of a field by a swarm of robots [1]. On the other hand, the last years saw an important spread of collaborative algorithms based on reinforcement learning. However, such method requires an important computational cost and the goal is to reduce it [10]. One way that was interesting to go in is allowing UAVs to communicate in order for them to exchange information. Creating a global reward for such swarm and using a 2D map has been developed in [9] and is the method we have chosen to develop for this challenge.

In section I, we introduce the problem statement and the way we tackle this project. In section II we describe the detection part of the project. Section III is dedicated to the behaviour of the swarm of robots and section IV to the collaborative Q-learning algorithm we used. Finally section V describes the results of our simulation and the improvement that still needs to be done in order to have a complete work.

I. Problem statement

Our goal for this 2020/2021 DGA's challenge is to cover an area in order to detect hidden people. To do so, DGA provided us a simulation environment that integrate the ability to control UAVs and terrestrial robots to make them discover a given area to detect our targets. The swarm of UAVs is divided in two types. Some are regular UAVs and the others are equipped with infrared sensors. The goal of the UAVs is to find the human targets in order to guide the terrestrial robots to these targets. A target will be considered as detected when a terrestrial robot is close enough to it. Since the application of such challenge is the guidance of troops in a war-zone, UAVs might be killed by hostile troops once they enter the battlefield.

A. Simulation Environment

We use Gazebo and ROS to simulate the physical behaviour of both terrestrial and flying robots interacting with their environment. The fleet is composed of 10 drones and 4 terrestrial robots. The simulation environment is a suburban area, where several people are located outside or inside the buildings. Some of them are colored in green, they represent our targets. The other people are considered as neutral civilians.

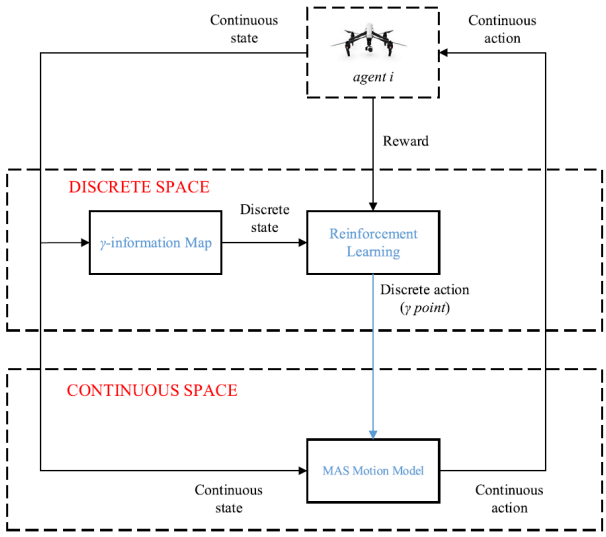

Fig. 1. Hybrid control system for one agent as a member of the swarm, as described in [9]. Information is thus parsed from continuous state to discrete state in order to process discrete actions and act in a continuous space.

B. Structure of the project

1) System description: There are many ways to address multi-agent reinforcement learning in a continuous space. However most of the methods require an important computation time due to the fact that with a large area and a large swarm, the action space is too big. In fact, even if terrestrial robots are moving in a 2D map, UAVs have the ability to interact with a 3D environment. In order to reduce the complexity of the algorithm, we decided to rely on a γ-information map that have been introduced in [9]. The main idea behind the use of the γ-information map is to work in a discrete environment to deploy a collaborative multi-agent *Q-learning*. Therefore we obtain the hybrid control system depicted in Figure 1.

Each of one agent's action will be chosen according to a reinforcement learning algorithm and will be performed in the continuous space, using a continuous motion model. To perform such movement we used the MAS motion model developed in [9] which uses both repulsive forces between UAVs and a PID control. After performing such action, the UAV will analyze the environment according to the image analysis described in section II. Then a reward will be computed according to the method described in [9] .

2) γ-information map: The γ-information map is a 2D grid of the environment composed of cells. Each cell is represented by a γ-agent, whose coordinates are the center of this cell. Therefore, an action taken by a UAV in this discrete state can be seen as a movement of this UAV from one γ-agent position to another. Unlike [9], we assume that each robot can communicate with the other robots. Therefore we define only a global γ-information map that will be shared by all the robots. Since we want to cover a predefined area, we have access to its size $n \times m$. In order to build such γ-information map, we have decided to divide the simulation environment in rectangles of size $k \times l$ chose centers are the previously mentioned γ-agents. This means that all UAVs will be flying at a same height z. Therefore according to the simulation environment, we have $r_s = z \tan(\frac{\pi}{4})$, where r_s is the radius of the area covered by the UAV at height z. Thus we have :

$$\begin{cases} k = \lceil \frac{n}{\sqrt{2} r_s} \rceil \\ l = \lceil \frac{m}{\sqrt{2} r_s} \rceil \end{cases} \quad (1)$$

For each $\gamma(x, y)$ coordinates of a γ-agent in the γ-information map M, we define a visit score as follows : $M(\gamma(x, y)) = 1$ if the γ-agent cell has already been visited and 0 otherwise.

3) Covering strategy: The simulation environment is designed to be as realistic as possible. That means that drones can hit walls or curbs. To tackle this additional problem, drones are equipped with embedded Lidar that allows them obstacle detection. Drones are trained to mainly go up an obstacle to avoid it. Thus, the γ-information map also contain a height-information map, that is completed while the environment is mapped. Using this trick, the drone will adjust its height to always keep the same distance to the floor. This allows us to reduce the dimension of the field of possible movement by one, and convert our 3D environment to 2D map.

Once the environment is mapped and once we have found our first target, terrestrial robots are sent on zone to confirm detection. To avoid walls, their path to the detection zone is computed using Dijkstra algorithm on the height-information map. The model we developed and provided does not take this improvment into account and it will be added in a next version.

II. Vision and Geometry

In this section, we present the detection pipeline, which is a process divided in two parts, the processing part and the post-processing part. The processing phase aims at predicting the presence of one or more objects and their local coordinates using images given by the UAV's camera. The post-processing phase aims at determining the 3D positions of a set of points of interest whose size is not known using the set of detections from all the UAVs over the entire data acquisition period.

A. YOLO algorithm

Our main idea to enable entity detection through the video streams of the drones was to use the *YOLO v3* algorithm. YOLO algorithms constitute a powerful algorithm family, with impressive detection results, while keeping a relatively high execution speed and needing reasonable computing resources. The last versions of *YOLO* are widely used in autonomous systems and thus it seemed a logical choice to us.

We successfully adapted the code of *YOLO v3* to our needs and obtained convincing results. However after a few tests, we were forced to admit that it was not worth investigating further. We were not able to use a proper GPU in parallel of our simulation environment, and thus we had poor computing performances on a CPU despite the efficient YOLO architecture. Moreover, in our simulation environment, given

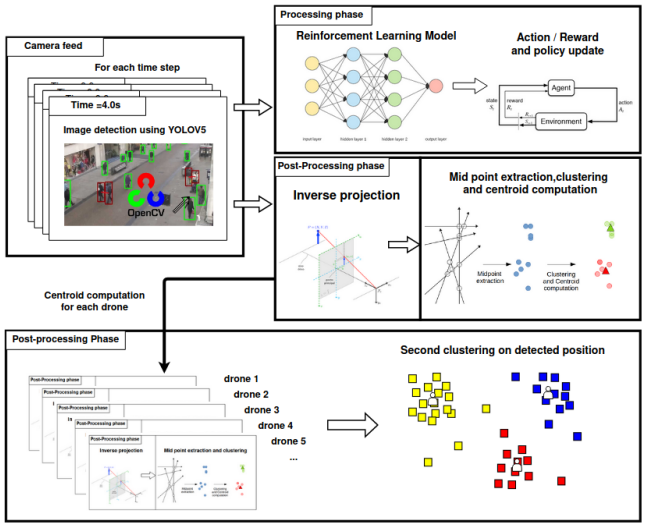

Fig. 2. Pipeline of the 3d detection of points of interest.

that our targets were colored in green, we were able to detect our main targets without using complex image processing, and thus *YOLO* was useful only to detect secondary objects which we considered not useful enough with our time constraints.

Fig. 3. Our moving drone -in black at the center of the simulation environment- is capturing the position of its targets, the green men. In the upper right corner is the mask obtained with the drone camera, on which we apply DBSCAN clustering ton find all the targets

B. Position identification

Once the position of a target is found in an image, we use the camera parameters to compute a ray starting from the drone position and pointing towards the detected entity. We then use the multiple frames captured by a single drone, and thus the multiple rays pointing toward the same target, to compute many points where rays are at the closest to each other. Finally, we use a clustering algorithm on those identified points to find all the observations related to the same target. It then makes it possible to compute an accurate estimator of the target position by averaging on all these observations.

Once we have done this work with each drone independently, we gather these results for the whole swarm and we once again use a simple clustering algorithm to filter targets that were identified simultaneously by multiples drones.

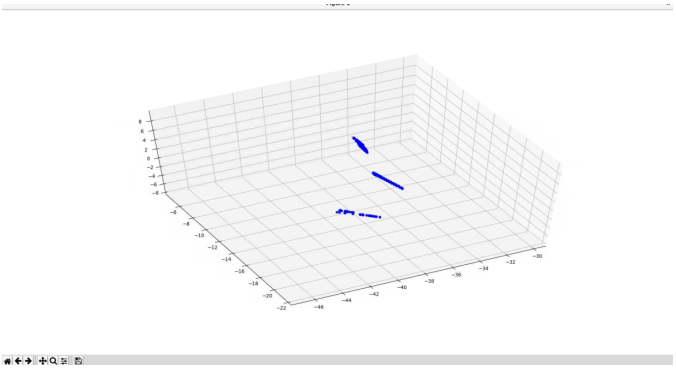

Fig. 4. Based on the rays computed from the pictures, we generate point clouds around our targets.

III. UAVs BEHAVIOUR

In this section, we present all the tools we implemented in order for the UAV to be able to move in the environment.

A. UAVs description

In the environment, one UAV have several tools in order to interact with the environment. These tools are mainly :

- a GPS
- an inertial measurement unit
- a sonar directed to the ground
- a lidar
- a 3 axis magnetometer
- an altimeter
- a front camera
- a camera which is facing the ground

These tools were the main tools used to implement the drone navigation programs, which are described in the following paragraphs.

B. Collision Detection

The first issue to address concerning drone navigation is the ability for the drone to detect obstacles and to avoid them. To do us, we mainly used the lidar with which each drone is equipped. It is important to mention that each drone are controlled so that it always faces the direction in which it goes. Thus, using the lidar, the drone is able to detect all the obstacles in front of him within an arbitrary angular range. The data of the lidar is then processed to detect if the drone can go around the obstacle by the sides, and it does so when it is possible. When the drone cannot go around the obstacle (typically when the drone is near a wall and does not see where the wall ends), it automatically goes up until it reaches the top of the obstacle and maintain a a fixed height relative to this obstacle. This height is controlled with the sonar, with which the drone is equipped. When the drone overcomes the obstacle, it goes down until it reaches a fixed height relative to the ground. During this maneuver, the position and the height information of the obstacle is stored in the γ-information map to allow the terrestrial robots to find their way to the potential targets by avoiding these obstacles.

C. Vision and Detection

Each drone is equipped with either a $240 * 320$ pixels standard camera, or with an infrared camera, each facing in the direction of movement. In order to provide relevant data to the RL model, each drone must be able to detect at each time step the presence of points of interests in the Image. This is achieved thanks to the Yolo V5 pre-trained model. Yolo V5 provides the coordinates in pixels of the bounding box of the detected object in the image 2D local basis.

D. Inverse Projection Transformation

Once we have all the local two-dimensional positions of the points of interest detected for all time steps. It is necessary to generate the three-dimensional coordinates of these points of interest. For that, we calculate the three-dimensional ray of detection knowing the position of the camera and the local coordinates of detection of the point of interest in the 2D image. A point P and an orientation vector $\bar{v}^T = \begin{bmatrix} X & Y & Z \end{bmatrix}$ are needed to define a 3D ray. One point P is given by the position of the camera P_c . We calculate the orientation vector with the help of the inverse projection matrix of our camera, thanks to the following formula :

$$\bar{x} = \begin{bmatrix} sx \\ sy \\ 1 \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \Pi X$$

x and y are the two-dimensional coordinates in the image, X, Y and Z are the world coordinates of the related orientation vector. Π is the combination of a extrinsic transition matrix and a intrinsic projection matrix, it is described as follow :

$$\Pi = \begin{bmatrix} f & 0 & c_x \\ 0 & \alpha f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0}_{3\times1} \end{bmatrix} \mathbf{R}_{4\times4} \, \mathbf{T}_{4\times4}$$

Where :

$$\mathbf{R}_{4\times4} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{1\times3} & 0 \end{bmatrix} \quad (2) \quad \mathbf{T}_{4\times4} = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{T}_{3\times1} \\ \mathbf{0}_{1\times3} & 0 \end{bmatrix} \quad (3)$$

The width of the image is noted $w_c = 320$ pixels and the height of the image is noted $h_c = 240$ pixels. The horizontal field of view is $\theta_c = \frac{\pi}{2}$.

K is the intrinsic matrix, describing the characteristics of the camera : focal length $f = \frac{w_c}{2} * \arctan(\frac{\theta_c}{2})$, principal point $(c_x = \frac{w_c}{2}, c_y = \frac{h_c}{2})$, pixel aspect size $\alpha = 1$. **R** and **T** are respectively the Translation matrix of the optical center in the reference frame and the Rotation matrix of the image plane.

Thus it is possible to compute $\bar{v}$ using the pseudo inverse of Π :

$$\begin{bmatrix} \bar{v} \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \Pi^{-1} \bar{x} \quad (4)$$

And finally we can define one 3D ray of detection :

$$D = \{P_i, \bar{v}\} \quad (5)$$

where P_i is the position of the Center of Projection of the camera.

E. First clustering on 3D-ray of detection

In the coverage process, each agent i generates many detection axes of points of interest :

$$\Omega_i = \left\{ (P_1^i, v_1^i), (P_2^i, v_2^i), ..., (P_n^i, v_n^i) \right\} \quad (6)$$

We would like to infer the three-dimensional position of these points of interest. To achieve that, a double clustering is applied to this data to determine the likely number and position of points of interest. First, we apply a local clustering on all the ray generated by a single drone. This part aims at inferring the number of detected points of interest by each drone.

It is not common to perform clustering on 3D ray and the result may even not be what we would expect. For this reason, we generate a set of points $(p_1, p_2, ..., p_{n^2})$ where each point is the mid point between every pair of ray $\left((P_a^i, v_a^i), (P_b^i, v_b^i) \right) \in \Omega \times \Omega$. In this way, we come back to a classic case of point clustering, as presented in [5].

For this task, there exist several clustering solutions, one of them is the well know K-means clustering. K-means works well most of the time, but one of the difficulty that remains is to define the number k of clusters (ie. of distinguishable points of interest). Indeed, we do not know in advance how many *distinguishable* points of interest had been detected by the drone at a given instant of the operation. Thus, to do so, one solution is to measure the average silhouette of our clustering. The silhouette value, presented in [7] is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The average silhouette is calculated using the Euclidean metric. The method to chose the optimal - or most likely - value of k is described in [7], and we choose to implement this method in this project.

In practice, we had some performance issues with this method as it requires to compute several K-means clustering. We finally decided to use a DBSCAN Clustering as explained in [6], one of the most cited clustering algorithm in bibliography. This algorithm is particularly useful to compute a clustering without knowing *a priori* the number of effective clusters in the data.

F. Second clustering on detected position

Once we computed all the detected position for each drone independently, it is necessary to gather all the data, simply because one distinguishable point of interest could have been detected by two different drones and thus could be given two times. Thus, we apply a second K-means clustering on the detected position, using the same optimisation of k' described above [7]. The likely position of distinguishable point of interest detected by the set of UAVs $\left(\hat{P}_1, \hat{P}_2, ..., \hat{P}_{k'} \right)$ is given

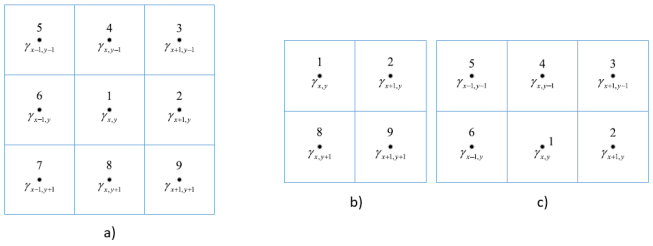

Fig. 5. Action space figure from [9]. a) No boundary, b) Upper Left corner, c) Lower boundary

by the coordinates of centroid of each cluster. By the law of large number $\lim_{t\to+\infty}\left(\hat{P}_1, \hat{P}_2, ..., \hat{P}_{k'}\right) = (P_1, P_2, ..., P_{k'})$, where t is this total acquisition time. This statistic method ensure that the inferred positions are close from the true positions.

IV. Reinforcement learning

Q-learning algorithm as presented in [4] aims to store in a Q-learning table the best strategy to choose, given an available action space A and a state S. In our case, the Q-learning table will give the optimal γ-agent to which each UAVs must go. We use the same nomenclature and Q-learning structure as in [9] : the current state, action and reward of an agent i are respectively s_i, a_i, r_i, and the next state and action according to the choice taken are respectively s'_i and a'_i.

A. Multi-Agent Reinforcement Learning

1) State and Action space: In natural Q-learning scenario with one agent, state s_i represents the agent's coordinates. However since we want the agent to communicate and share information according to the update of the global γ-information map, we must redefine the current state s_i as follows :

$$s_i = [M, p_1^\gamma, ..., p_N^\gamma] \tag{7}$$

where p_j^γ is the position of agent j in the γ-information map (therefore, the object p_j^γ is the coordinates of a given γ-agent).

In our study, we define the action space according to the neighborhood of each agent. From [9] we extract the following nomenclature :

$$A_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \tag{8}$$

that can be represented in Figure (5) with the boundary cases.

2) Reward function: The reward system is the one being used in [9] where we have for UAV i :

$$r_i(a_i, s_i) = \begin{cases} 0 & \text{if } M(\gamma'(x,y)) = 0 \\ & \text{and } a_i \in \{1,2,4,6,8\} \\ -0.3 & \text{if } M(\gamma'(x,y)) = 0 \\ & \text{and } a_i \in \{3,5,7,9\} \\ -0.2e^{c_r(k_r-1)} & \text{if } M(\gamma'(x,y)) \neq 0 \\ R(T) & \text{if } M(\gamma(x,y)) \neq 0 \\ & \forall \gamma(x,y) \end{cases} \tag{9}$$

In this formula, we define :

$$M(\gamma'(x,y)) = \begin{cases} 0 & \text{if the cell } \gamma'(x,y) \text{ had been visited} \\ 1 & \text{else} \end{cases} \tag{10}$$

We also define :

$$R(T) = \begin{cases} 0 & \text{if} \quad T > c_1^T T_{min} \\ \frac{r_{ref}}{2}(1+f(T)) & \text{if} \quad c_2^T T_{min} < T \leq c_1^T T_{min} \\ r_{ref} & \text{if} \quad T \leq c_1^T T_{min} \end{cases} \tag{11}$$

With :

$$f(T) = \cos\left(\frac{\pi(T - c_2^T T_{min}))}{(c_1^T T_{min} - c_2^T T_{min})}\right) \tag{12}$$

where $c_1^T > c_2^T > 1$ are constants. We have T_{min} the minimum traversal time of the area defined as follow :

$$T_{min} = \min\{\frac{(l-1)mk}{l|v_{max}|} + \frac{(k-1)n}{k|v_{max}|} + \frac{(k-1)nl}{k|v_{max}|} + \frac{(l-1)m}{l|v_{max}|}\} \tag{13}$$

The idea of such a reward system is to give negative rewards when the drone moves to an already visited cell if not all cells have been visited yet. If we visit all the cells in an enough short amount of time, then visiting a cell that has already been visited gives the UAV some rewards.

B. Collaborative Q-learning

The article [9] relies on the article [2] in order to do collaborative reinforcement learning. The update of the Q-value table of UAV i at iteration K is done as follows :

$$\xi_i^K(s_i, a_i) = Q_i^K(s_i, a_i) + \alpha(r_i^K + \lambda \max_{a'_i \in A_i} Q_i^K(s'_i, a'_i) - Q_i^K(s_i, a_i)) \tag{14}$$

$$Q_i^{K+1}(s_i, a_i) = w\xi_i^K(s_i, a_i) + (1-w)\sum_{j=1}^{N} \xi_j^K(s_i, a_i) \tag{15}$$

where α is the learning rate, λ the discount factor and $w \in [0, 1]$ the selfishness rate. If $w \to 0$ then the UAV is considered as selfish, while when $w \to 1$ he collaborates with the other UAVs.

The action space A_i of UAV i is being restricted to A'_i, the subset of A_i containing γ-agents that have not been visited yet.

$$A'_i = \{\gamma(x,y) \in A_i | M(\gamma(x,y)) = 0\} \tag{16}$$

We have therefore two cases :

1) $A'_i \neq \varnothing$, then UAV choose to reach a γ-agent that have never been visited before according to the principle of maximization of Q-value :

$$a'_i = \arg\max_{a_i \in A'_i} Q_i(s_i, a_i) \tag{17}$$

2) $A'_i = \varnothing$, then UAV choose to reach the closest γ-agent that maximises its reward :

$$a'_i = \arg\max_{\gamma(x,y) \in A_i} ||r_{x_1,y_1} - \gamma(x,y)||^2 \tag{18}$$

C. Q-Learning architecture

In order to increase the performance of our *Q-Learning* we used two technologies related to this deep learning method.

a) Double Q-Learning: The first one is called *Double Q-Learning*. It has been first proposed by [3] in order to reduce the bias link to the maximisation issue. In fact, when we consider equation 14 and especially its maximisation part $\max_{a'_i \in A_i} Q_i^K(s'_i, a'_i)$ we overestimate values at each operation introducing a bias in our model. In order to reduce such bias, we use two separate Q-value estimators, each of which is used to update the other.

b) Dueling Q-Learning: As we have several drones sharing information, we don't necessarily need to know each effect of each action for the environment. Therefore, we use a trick called "Dueling Q-Learning" that follows this idea. It has been introduced in [8].

V. Simulation and Evaluation

In this section, we present the simulation process, and define the evaluation metric for our algorithm. We also describe the work that need to be done and the improvment that will be important to do.

A. Simulation process

The simulation was divided in two parts : a training part and the evaluation. We did not have the time to do both of them but here are our results. During the training part, we launch the simulation several times in order for the drones to compute the Q table related to the gamma map. For the evaluation part, we wanted to store this map over the time and to compare the mean coverage at each iteration according to the method described in [9]. This must be done in order to improve our RL algorithm.

Our swarm of UAVs succeed in covering the field but we didn't had the time to do a very deep training process. Our architecture for the neural network is simple and it will be needed to improve it. Currently, the coverage takes a long time but we are convinced according to our algorithm that with a good training, we will be able to increase this performance.

During this challenge we had many issues that really slowed the project. One of them was the fact that we weren't able to display some information on the simulation environment which makes more complicated to do a good training. Most of the issues we faced were linked to the fact that we were given an already complex ros gazebo project, and we got to work on it without documentation, and without prior knowledge on how to work on a ros gazebo project. Learning the skills needed to handle such a project took us a lot of time (at least half of the time spent on this project). We regret having to spend so much time in a task that does not deserve the goal of this challenge, which was to work on autonomous drone swarn deployment, and not on how to move a drone. We cannot help thinking that we could have saved a precious amount of time if we were given a very simple documentation on the basics of ros gazebo. Even now, we are convinced that we lack some useful skills that would have allowed us to work on this project more appropriately.

B. What must be done now ?

As this project was done as a participation in a challenge we struggled to get all the work we wanted done. Therefore some fixes must be implemented :

1) Improve our collision detection model.
2) We need to connect the huskies tasks to the drone detection and implement the Djisktra algorithm.
3) We need to improve the logs concerning the detection part. For example, we may use a threshold on the variance of the position in order to give an accuracy limit to our detection. Moreover, we need to find the optimal parameters between the detection period and the speed of the UAVs.
4) We can still improve the accuracy of our detection model.
5) We didn't tested the kill zone for our model but according to [9], our model is strong concerning dead UAVs.

A last comment about our model is about its ability to adapt to orders. For example, once you have trained your model to cover a γ-information map, you can place UAVs wherever you want in the map. Since all UAVs have a Q-table trained, they can communicate with each other in order to find the perfect coverage strategy. A last, but important improvement will be to add a reward system that depends on time and on the field strategy. For example, if in your field you have some area of interest like buildings and whatever, you can positive rewards for these places that depend on time. Then, the optimal strategy for the swarm of UAVs will be a coverage where areas of interest will be visited faster.

References

[1] Pooyan Fazli et al. "Multi-Robot Area Coverage with Limited Visibility". In: (). DOI: 10.1145/1838206.1838451.

[2] R. Lim H. M. La and W. Sheng. "Multirobot cooperative learning for predator avoidance". In: *'IEEE Trans. Control Syst. Technol.* 23.1 (January 2015), pp. 55–63.

[3] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)* ().

[4] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. *Guided Deep Reinforcement Learning for Swarm Systems*. 2017. arXiv: 1709.06011 [cs.MA].

[5] Mahamed Omran, Andries Engelbrecht, and Ayed Salman. "An overview of clustering methods". In: *Intell. Data Anal.* 11 (Nov. 2007), pp. 583–605. DOI: 10.3233/IDA-2007-11602.

[6] Anant Ram et al. “A Density Based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases”. In: *International Journal of Computer Applications* 3 (June 2010). DOI: 10.5120/739-1038.

[7] Peter J. Rousseeuw. “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257.

[8] Ziyu Wang et al. “Dueling Network Architectures for Deep Reinforcement Learning”. In: *Google DeepMind, London, UK* ().

[9] JIAN XIAO et al. “A Distributed Multi-Agent Dynamic Area Coverage Algorithm Based on Reinforcement Learning”. In: *IEEE Acess* 8 (January 2020), pp. 33511–33521. IEEEAccess: 2169-3536.

[10] Matthew Zhu et al. “Reinforcement Learning for Multi-robot Field Coverage Based on Local Observation”. In: *IEEE* (). DOI: 10.1109/SoSE50414.2020.9130535.