Reinforcement Learning for Dynamic Area Coverage

Mehdi Benharrats, Ziyad Benomar, Celine Hajjar, Tom Marty, Icare Sakr, Pierre Tessier

***Abstract*—In this paper we propose a Reinforcement Learning (RL) algorithm to solve the problem of dynamic area coverage. Given a zone and multiple agents that control it, the goal is to achieve a complete coverage of the zone, while avoiding obstacles and gathering all targets. In the real world for example, the area could represent a disaster site where drones (the agents) are deployed to search for victims. In our work, we suppose that all the drones communicate with each other in the whole target area. The continuous movement of a drone is mapped to a discrete RL environment, where a deep Q-learning algorithm determines the best action, and that action is then converted back to the continuous space.**

I. Introduction

Drones have become an essential component of Search & Rescue operations, since they facilitate the localization of victims in difficult terrain conditions after a disaster. One problem that we aim to solve with the use of multiple drones in this context is maximizing the efficiency of the dynamic area coverage. What strategy should the drones adopt in order cover the whole target area as fast as possible while avoiding obstacles and ensuring robustness to failures of some agents?

We can distinguish two categories for multi-agent system (MAS) dynamic coverage algorithms: 1) non-self-organizing and 2) self-organizing. The main difference is that self-organizing algorithms are robust and flexible, what solves the problem where target areas cannot be covered completely due to agent failures. On the other hand, a good way to avoid non-optimal solutions (that involve visiting the same area multiple times), is to select the best target position for each agent considering the whole coverage process.

Several existing optimization methods including Ant Colony Optimization (ACO), Particle Swarm Optimization (PCO) and Model Predictive Control (MPC) are normally used to find the optimal motion path of every independent agent in the multi-agent system (MAS), without taking into account the motion state of its neighbors. Similarly, anti-flocking algorithms, as explained in [2], find the best control point for agents given the current state of coverage. However, in the case of dynamic area coverage, agents need to adapt their strategy to their environment as well as their past actions in order to learn an optimal overall strategy.

Reinforcement Learning (RL) algorithms can be adapted to this specific task, thus facilitating collaborative control in MAS. For example, Xiao et al. [6] deploy a distributed cooperative Q-learning algorithm that finds a global optimal coverage strategy with a coverage time that is close to the minimum time under ideal conditions, under the constraint that all drones are in continuous communication with each other.

In this paper, we propose another approach based on RL, where the best discrete strategies are learned by deep Q-learning, and are then converted to continuous movement.

Our main contributions are the following:

1) The implementation of a model that maps continuous action to a discrete space, using a γ information map.
2) The application of a deep Q-learning algorithm from scratch to learn the best collaborative behavior.
3) Implementing a neural network to compute the Q-table to overcome its enormous size.
4) Interpretation of the observed results.
5) The creation of a simulation environment using PyGame to visualize the behavior of the drones.

II. The Scenario

In order to test the efficiency of a map coverage by using reinforcement learning, we will simulate the behavior of drones in a 2D discrete universe, where each drone can move freely on two axes. This universe contains obstacles and goal cells.

The mission ends when all goal cells have been located. The aim is to locate these goal cells in a minimum of time.

One can discuss the relevance of this simulation and especially its simplicity compared to what could be encounter in a real world situation, but we wanted to specifically focus on the Reinforcement Learning part, and a simple scenario allows us to iterate faster and to more easily highlight the effect of some parameters on the individual or general behavior.

Eventually, the universe could be enhanced with new features, such as the possibility that drones be shut down, in order to get closer to reality. In the future, we can imagine a scenario where the environment and the obstacles change with time. For example, in the situation of a Search and Rescue operation after a landslide or an earthquake, the land/obstacles layout could change during the coverage.

III. The Environment

To simulate the behavior of our drones, we use AI Gym, a toolkit providing several tools and packages that facilitate the implementation of an environment and the comparison between different Reinforcement Learning algorithms. Below are the principal elements of the defined environment:

- *State* space: the *State* space is discrete. The gamma information-map presented in section V-B is a matrix of size $k \times l$ containing obstacles (ie. cells that can't

be reached by a drone in any case) and target cells, that drones need to discover. The goal for the fleet is to discover all the reward cells in the shortest time.

- *State* and *Action* space: when a drone is located in a cell (x, y), it can take 9 possible actions: stay on site or move to any of its 8 neighboring cells, under the condition that these cells are reachable (ie. neither an obstacle nor a boundary of the map or a drone is already on the cell). 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 γ-information map, we must redefine the current shared state s_i as follows:

$$s_i = [M, p_1^\gamma, ..., p_N^\gamma] \tag{1}$$

where:

  - p_j^γ is the position of the target position of agent j in the γ-information map (ie. the coordinates of γ-agents).
  - M is the state of the γ-map, meaning that $m(x, y) = 1$ if the (x, y) cell of the γ-map has been traversed, -1 if the (x, y) cell is an obstacle and 0 otherwise. The obstacle mapping is done as we go along the discovery of the environment.

In our study, we define the action space according to the neighborhood of each agent. From [6] we extract the following nomenclature:

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

that can be represented in Figure (1) with the boundary cases.

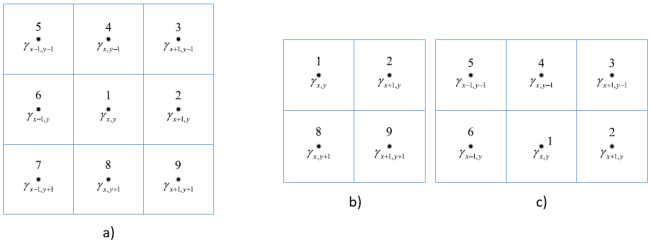

Fig. 1. Action space figure from [6]. a) No boundary, b) Upper Left corner, c) Lower boundary, ref [6]

We suppose that each drone has a field of view of 1 square unit around its position. Thus, a cell is marked (covered) if a drone has visited one of its neighbors.

- the *reward* function: all the drones share the same reward function, but every drone considers its own reward during the learning process. They only share a certain proportion of their Q-table to other agents, this proportion is handled by a selfishness coefficient w explicited in V .
- Obstacles and targets: the environment should also contain a map, describing where the obstacles and the targets are located. We will assume that we have a map of the same dimensions of the $\gamma-$map, where each cell can either be filled with an obstacle, contain a target, or be empty. Therefore, the obstacles cannot have free shapes, but they can only fill cells of the map, the same goes for targets. Note that the movement map and the $\gamma-$map have exactly the same cells, but they contain different information: the $\gamma-$map defines the action space, while the movement map defines the positions of obstacles and targets.
- Randomness: the environment we consider is generated randomly: the parameters needed for constructing it are its dimensions (width and height of the map), and two parameters p_{obs}, p_{tar} that are respectively the probabilities of a cell containing an obstacle and containing a target. Also, we have a given number of drones that are put randomly in the free cells of the the map.
  Once the environment is created, it becomes totally deterministic, drones are not affected by any exterior source and cannot be shot down.
- main difficulties: at the beginning, the environment is unknown, agents have no information about the location of target cells, obstacles or boundaries. Their field of view is also limited to the neighboring cells, making it impossible to forecast their trajectory in order to avoid potential future obstacles.

IV. The Agent

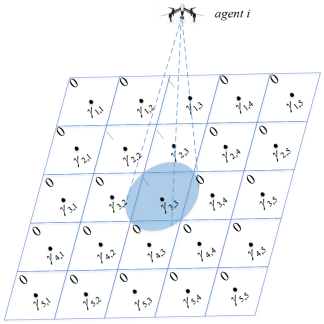

Fig. 2. Example of agent traversing γ-information map. The value in the corner of each small rectangular region is the value of $m_i(\gamma_{x,y})$, ref. [6]

Agents are drones that move in a pseudo-continuous space. The use of a continuous space allows drones to simulate real movement by moving smoothly between cells.

We use deep Q-learning to plan the γ points of agents. To simplify the model, we assume the following:

1) Agents are on the same horizontal space
2) Agents have the same detection range for coverage r_s.

A. Decision making

The decision space of the drone is a discrete space. In order to choose which action will be performed next, we use a classical Q-learning algorithm developed in the course [1].

To this individual Q-learning is added a term shared by all other drones as developed in part V-C.

B. Exploration and Randomness

At the beginning of the learning phase, the Q-table of each drone is not relevant because we are only at the beginning of the iterative calculation. In the first few learning periods, the UAV movements are not based on the Q-table but on a greedy algorithm which consists in always choosing the unexplored squares instead of the already explored ones and keeping the same decision as long as all neighboring cells have already been visited.

C. Movement of the drones

As stated before, the action space is discrete and the 9 possible actions correspond to the neighboring cells. However, we want our drones to have a continuous movement on the map, so we also define also continuous action space for each drone: this space is the set of all the continuous coordinates the drone can go to in a duration dt.
Basically, each drone first chooses a discrete action (so a neighboring cell in the $\gamma-$map), then makes several smaller "continuous" actions at each time unit dt to get to the targeted cell and accomplish the discrete action. The function *step* in our Gym environment corresponds to the continuous action. Further explanation is given in the section "Background and Related Work".

D. Communication

Naturally, we assume that all the drones share the same $\gamma-$map. To make our problem easier, we also assume that at each moment, all the drones share the same information: each drone knows the positions of all the other drones, and has information about all the cells visited by any drone in the fleet since the start of the covering process.

V. Background and Related Work

A. System description

There are many ways to tackle 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 large. In fact, even if terrestrial robots move 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 has been introduced in [6]. The main idea behind the use of the γ-information map is to work in a discrete environment to deploy a collaborative multi-agent *deep Q-learning* system. Therefore we obtain the hybrid control system depicted in Figure (3).

Every action will be chosen according to the reinforcement learning algorithm and will be performed in the continuous space, using a continuous motion model. To perform such a movement we used the MAS motion model developed in [6] which uses repulsive forces between UAVs and PID control. After performing such action, the UAV will analyze its environment. Then a reward will be computed according to the method described in [6].

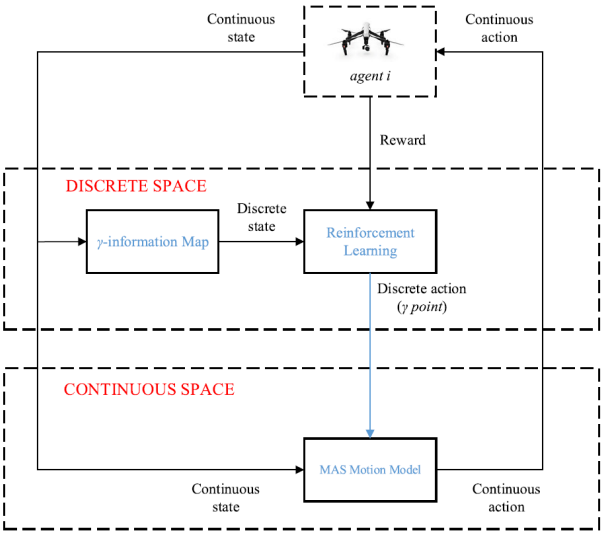

Fig. 3. Hybrid control system for one agent [6]

B. $\gamma-$information map

As hinted in the article [6], we use a discrete $\gamma-$information map for each agent i, that stores information about the state of a particular area; being either visited or not. Supposing the overall target area is a rectangular region of size $m \times n$ (in pixels for example), we divide it into $k \times l$ small rectangular regions (we define it like an occupancy grid), and the centers of each small rectangle is regarded as a γ point which is equivalent to $\gamma-$agent. The $\gamma-$information map for an agent i is the set of all the γ points ($\gamma_{x,y}$), and their corresponding traversal information $m_i(\gamma_{x,y})$ (binary variable indicating if a γ point has been visited or not), so $m_i(\gamma_{x,y}) = 1$ if $\gamma_{x,y}$ has been visited, $m_i(\gamma_{x,y}) = -1$ if $\gamma_{x,y}$ contains an obstacle, and 0 otherwise. (see figure 2 for reference).

C. Collaborative Q-learning

The article [6] relies on the article [3] in order to do collaborative reinforcement learning. The update of the Q-value table of UAV i at iteration K is done as follows:

$$\xi_i^K(s_i, a_i) = Q_i^K(s_i, a_i) + \alpha(r_i^K + \lambda \max_{a_i' \in A_i} Q_i^K(s_i', a_i') - Q_i^K(s_i, a_i)) \tag{3}$$

$$Q_i^{K+1}(s_i, a_i) = w\xi_i^K(s_i, a_i) + (1-w)\sum_{j=1}^{N} \xi_j^K(s_i, a_i) \tag{4}$$

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 $w \to 1$ the UAV 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.

$$A_i' = \{\gamma(x, y) \in A_i | M(\gamma(x, y)) = 0\} \tag{5}$$

We have therefore two cases:

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

$$a'_i = \underset{a_i \in A'_i}{\operatorname{argmax}} Q_i(s_i, a_i) \tag{6}$$

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

$$a'_i = \underset{\gamma(x,y) \in A_i}{\operatorname{argmax}} ||r_{x_1,y_1} - \gamma(x,y)||^2 \tag{7}$$

D. 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 [4] in order to reduce the bias link to the maximization issue. In fact, when we consider equation 3 and especially its maximization 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 [5].

VI. Implementation

The code has been entirely implemented from scratch, using the gym environment, and PyGame to visualize the simulation of drones coverage.

A. The PyGame interface

We first created a Pygame environment with two main classes *Drone* and *Grid*. The class $Drone$ contains information about the map, the covered zone, the position and the trajectory of the drone, its space of possible actions ..., and the class $Grid$ contains the previous information and also information on the fleet of drones. In addition to the methods updating the covered zone, positions of the drones ..., the class *Grid* has drawing methods, using the library PyGame to draw and animate the environment.

Figure 4 shows the graphical interface an environment of size 15×10, $p_{obs} = 0.15, p_{tar} = 0.15$. The map is composed of cells, each cell either contains an obstacle, a target, or is empty. The cells marked with red crosses are the ones not covered by the fleet yet, and the ones with a purple plus sign are the ones that have just been discovered after the last step. The drones are represented by black circles. The continuous action space of each drone is indicated with a blue rectangle around it, and its vision field by a red rectangle. As stated before, when a drone is in a given cell, it can see all the 8 neighboring cells, which correspond to its vision field, but also to its discrete action space in the γ-map.

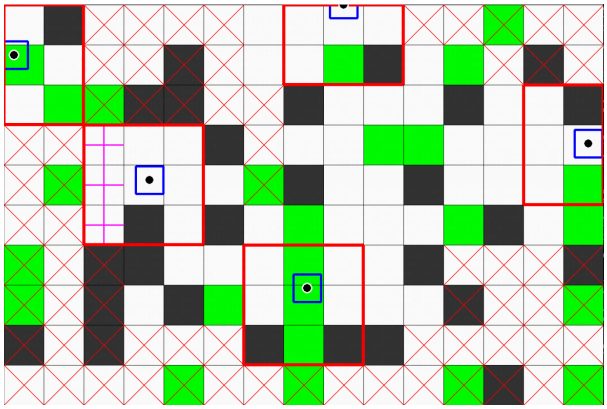

Fig. 4. The environment with a 15×10 γ map, and 5 drones: green cells are targets, black cells are obstacles, and the drones are represented by the black dots with the corresponding vision fields in red. Cells that have not been visited yet are marked with a red cross.

B. The RL algorithm

We adapt the reinforcement learning algorithm described in [6] to our use case, taking in account all the hypothesis we discussed earlier (all the drones are in the same horizontal plan, they all have the same information at any moment, ...), and we also adapt it to the Gym's environments template.
We first used the formula given in [6] to compute the $Q-table$, but we had memory problems doing so when considering an important number of drones. Recall that every drone has 9 possible actions, which means that if we have N drones, our Q-table will have a size 9^N. This makes it very difficult to deal with a large number of drones (¿10) because of the enormous size of the Q-table. The alternative we found for this problem was using a neural network to predict the Q-table at each step of the algorithm. Of course this method has a certain computation cost, but still we think it is much better than storing the Q-table at each iteration.

C. The code

A *README.md* file is provided with our code, recalling the explanations we gave earlier about the visualization, and explaining how to run the code and change the parameters of the environment (width, height, number of drones ...)

To download our code, please click on the link below:
https://github.com/Ziyad-Benomar/Drones_coverage

To run the simulation, simply run the Pyhton file "run.py" after having installed all requirements from "requirements.txt"

VII. Results and Discussion

In order to assess the performance of our model, we first created a benchmark model where the drones behave randomly. We then tested a greedy algorithm. For both of these algorithms, we do not use the γ map to take decisions, the drones make a new decision at each "continuous" step, that is the unit of time we use:

- The random walk: at each step, every drone chooses a random action in its continuous action space.
- The greedy algorithm: at the start of the simulation, each drone chooses an action (a movement (dx, dy)) in its

continuous action space, and for the next steps, as long as this action is possible, it takes it again with a probability $1 - \varepsilon$, and chooses a random action with probability ε. Once the drone hits an obstacle or meets the border of the map, i.e its previous action is no longer possible, it chooses a new random action and repeats the same process.

As expected, the random walk algorithm was not efficient, since every drone takes only random actions and therefore does not move a lot.
However, the greedy algorithm turned to be very efficient. In addition to the natural intuition behind the algorithm and its easy implementation, it covers all the map in a relatively short time.

For our experiment, we considered a 15×10 map with 5 drones as in the figure 4. We ran our reinforcement learning algorithm for 20000 episodes on the same map and with the same initial positions of the drones, where each episode stops when all the target cells (green cells) are covered or when the number of continuous steps exceeds 1000.
In the following figure, we present the runtime of each episode from 1 to 20000, that is, the number of steps needed to achieve perfect coverage if perfect coverage is reached in less than 1000 steps, and 1000 otherwise.

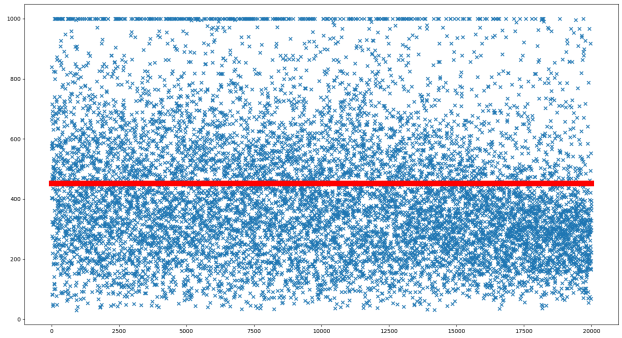

Fig. 5. The runtime of each episode from 1 to 20000, with a fixed map of size 15×10 and 5 drones

The red horizontal line corresponds to the value 452.24, which is the average runtime of the greedy algorithm on the same map with the same initial positions of the drones.
We can observe that the runtime of the RL algorithm decreases in average, but it decreases very slowly. During the first 10000 episodes, it seems that the RL does not improve, and that the runtime is almost random, but after that, it becomes more and more concentrated on smaller values. We could not go further than 20000 episodes because the running time was very long.

A limitation in our model is that all agents take action at the same time, as soon as the state changes, which is when an agent changes tiles on the information map. Intuitively, we can think of the problem we may encounter when two agents might take the action to go to the same tile; if they do consult each other before taking an action, this might cause some inefficiencies. To make it better, we could introduce a consulting phase between the agents. For example, by deciding before each phase of an order decision: drone 1 chooses his action first, then drone 2, etc. We could also experiment to find if keeping the same order for every action gives us the same results as choosing a random order. The fact that each agent is a separate deep Q network should also make each differently: having a fixed order of decision is therefore useful for each agent to "specialize" into a specific function among the "team" of drones.

On the other hand, the deep Q network, being an off-policy RL, is not a very suitable algorithm when we want to optimize the value of an agent that is exploring, which is the case in our project. If the combinatorial explosion of the state space had not been so immense, we should have tried an on-policy algorithm like SARSA, but since this is the case, we could try the "Deep network" version of SARSA, that is to say a policy gradient algorithm such as REINFORCE, but due to the time constraints of this project, we weren't able to do that.

VIII. Conclusion and Future Work

For future endeavors, an interesting thing to implement could be to add some environmental changes in the model to see how the agents would adapt. For example, in the situation of a Search & Rescue operation during a landslide or an earthquake the environment layout could change during the coverage thus making updates on the obstacle compulsory. As soon as the action space is considered discrete, the problem of coordinated piloting of drones becomes a tree exploration problem, for which a good adaptive technique such as D* could be implemented. D* would take into consideration those real-time changes in the planning of optimal coverage.

Another thing to test could be to make change in the number of agents, due to failures or introducing new drones for example, while the program is running. In the real world, preparing a drone for a mission could take some time and the drones that are already operational could have already started the area coverage.

Also, using a dynamic (non homogeneous) mesh to represent the terrain could be useful, as the terrain can be homogeneous in some regions (smooth) and more complex in other regions. So the dynamic grid would have large grid cells in smooth regions, and small cells in regions with high level of details. Similarly, when there is a sharp variation of height, a more detailed grid could be useful as it may allow the drone to choose a path to get around an obstacle without hitting it due to large cells coinciding with obstacles. This would allow us to refine our model, without adding too much complexity in terms of computing resources.

References

[1] INF581 Advanced Topics in Artificial. "Lecture 6 : Reinforcement Learning". In: (2020).

[2] Nuwan Ganganath, Chi-Tsun Cheng, and Chi Tse. "Distributed Anti-Flocking Algorithms for Dynamic Coverage of Mobile Sensor Networks". In: *IEEE Transactions on Industrial Informatics* 12 (Jan. 2016), pp. 1795–1805. DOI: 10.1109/TII.2016.2519913.

[3] 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.

[4] 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)* ().

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

[6] 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.