An Incremental Approach for Multi-Agent Deep Reinforcement Learning for Multicriteria Missions

Nicholas Scharan Cysne, Carlos Henrique Costa Ribeiro and Cinara Guellner Ghedini

***Abstract*—Mobile robots or drones can act as agents in a wide range of scenarios in which sending human personnel is of great risk or difficulty, such as exploration and reconnaissance missions in dangerous areas or of extreme conditions. These missions can employ multiple agents forming a decentralized network that requires communication and cooperation among its members. And, in addition to task completion *per se*, a successful operation requires a robust network topology capable of maintaining characteristics such as connectivity between all agents and resilience to failures. Such scenarios characterize a multicriteria mission, where several distinct activities must be performed simultaneously by a single set of agents. Multicriteria missions are natural candidates for Machine Learning techniques due to a large number of possible configurations and lack of an explicit model that determines an optimal solution. We propose herein the use of Deep Reinforcement Learning and Multi-Agent Reinforcement Learning (MARL) techniques, namely Proximal Policy Optimization (PPO) and Centralized Training with Decentralized Execution (CTDE), to train a set of 20 agents forming a network topology able to incrementally learn 4 distinct tasks: Move to Target Location, Connectivity Maintenance, Area Coverage, and Robustness to Failures. The resulting model showed good results concerning the tasks of Move to Target Location, Connectivity Maintenance and Robustness to Failures, but underperformed in Area Coverage. Considering that there is still room for improvement and exploration of other techniques, the findings suggest that MARL has the potential to achieve good performance when incrementally training a set of agents for multicriteria missions.**

***Index Terms*—Decentralized Networks, Cooperative Autonomous Systems, Fault Tolerant Systems, Deep Reinforcement Learning, Multicriteria Missions**

I. Introduction

Multi-agent systems design is one of the main challenges in Robotics for the next decade [1], and an important prerequisite to its success is the existence of a resilient network topology for its agents. Yang *et al* [1] define resilient systems as "(...) systems that are self-aware and self-regulating, and that can recover from large-scale disruptions". A resilient network topology has all its members connected, and provides robustness to failures, meaning that even if a node fails, the network will continues to be able to perform its task.

This work deals with multicriteria missions, characterized by a decentralized network of agents that must perform several distinct tasks, sometimes with opposing objectives, composing complex scenarios in which network resilience and cooperation are essential for mission success. An example of such missions are coverage missions, in which a network of agents cooperates among themselves to explore a region and avoid nearby obstacles, while maintaining connectivity with each other and communicating the discovery of possible points of interest. The success of such coverage missions in most cases depends on area coverage, connectivity maintenance, and robustness to failures.

While area coverage is more closely related to the mission purpose itself as spreading the nodes of the network increases the search space, connectivity and robustness mechanisms are more related to safeguarding the conditions of the network throughout the mission. Independently of the purpose of the mission, the agents should always arrange themselves in a resilient topology that enables them to perform the required mission even when suffering failures of agents that play an important role in the network topology, *i.e.*, when their failures can lead to network fragmentation.

In such context, multicriteria missions define complex scenarios suited for ML techniques, such as Deep Reinforcement Learning [2], due to the large number of possible configurations and lack of an explicit model that determines an optimal solution. In this direction, this work aims to use state-of-the-art Multi-Agent Deep Reinforcement Learning techniques to train an agent that cooperates with its peers and creates a resilient network topology able to perform multicriteria missions. In our scenario, we set different tasks to simulate such missions where each agent takes the role of a mobile entity (*e.g.*, a drone) in the network with the goal to move from a start to a target location while performing: Connectivity Maintenance, and Area Coverage and Robustness to Failures improvement. Agent performance will be evaluated in terms of connectivity and robustness to failures in scenarios with and without the presence of failures of central agents.

II. Related Work

Ghedini *et al* [3] addresses the problem of network topologies for multicriteria missions with a combination of control laws that takes into account the objectives of connectivity maintenance, collision avoidance, robustness to failures, and area coverage. The resulting model demonstrated that it is possible to determine a set of weights for each control law that achieves a resilient network topology. On top of this approach, Minelli *et al.* [4] proposed an online optimization of weights for the control laws which outperforms the scenario of static gains in some cases.

The application of Deep Reinforcement Learning to network topologies of drones has a few interesting results such as the ones reported by Nallanthigal and Gupta (2021) [5], showing that a set of drones can be trained to act in simple tasks with relative ease. In this sense, our proposal is a Multi-Agent Deep Reinforcement Learning framework for training

a decentralized network of agents that cooperate among themselves to create a resilient network topology. Our main contribution is a strategy to train a set of agents, constrained by limited communication range, capable of cooperation in the execution of obstacle avoidance, connectivity maintenance, and robustness to failures in multicriteria missions.

III. Proposed Model

A. Problem Modeling

We model the network as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of up to 20 nodes. In this scenario, each node $v_i \in \mathcal{V}$ represents an agent of our network, and every edge $e_{ij} \in \mathcal{E}$ represents an available interaction (over a predefined communication range) between agents i and j.

The network must traverse a field of dimensions $50\ m \times 500\ m$, where there are 100 randomly distributed obstacles. Also, we have 200 possible initial scenarios as devised in [3], defined by the initial positions of the agents in the field and the positions of all 100 obstacles.

Each agent is a simplified model of a drone of mass $M = 10$ kg treated as a point in space. The observable radius (communication range) of each drone is set as $R = 16\ m$, and to avoid collisions between drones or obstacles, we set a minimum distance $d_{min} = 3\ m$ between the drone and its surroundings. Each drone has a speed limit $v_{max} = 0.6\ m/s$, needing about 900 seconds to traverse the field while avoiding obstacles, giving us an episode length of 15 seconds when running at a $60\ Hz$ sampling rate.

Our agents have a local-ranged communication restriction where each agent can only access information from drones within a distance of 2 hops from it. Thus, each drone is capable of observing the positions of all drones and obstacles within its observation radius, plus the ones that its neighbors can observe.

In order to avoid collision between drones and obstacles or between the drones themselves, we designed a high-level controller implemented as a potential field that creates a repulsion force $\vec{f}_{rep}$ between drones and obstacles, given by

$$\vec{f}_{rep} = \begin{cases} -k_{rep}\frac{1}{d-d_{min}}\vec{r} & \text{if } d < R, \\ \vec{0} & \text{otherwise,} \end{cases} \quad (1)$$

where R is the observation radius, d_{min} the minimum distance to maintain from obstacles and other drones, and $\vec{r}$ a unitary vector giving the force direction.

The agent velocity is composed by two parts, $\vec{v} = \vec{v}_a + \vec{v}_p$, where $\vec{v}_a$ is the output of the agent neural network (control output from the RL process to be detailed), and $\vec{v}_p = \vec{a}_p * \Delta t = \frac{\vec{F}_p}{M} * \Delta t$ is derived from the acceleration produced by the resulting potential field, $\vec{F}_p = \vec{F}_n + \vec{F}_o$ being the sum of the resulting potential field due to neighbors, $\vec{F}_n$, and due to obstacles, $\vec{F}_o$.

If the potential field is not enough to make the agent avoid stepping into an obstacle, we use a controller to check if the next action is valid. This way we can control the agent not to hit obstacles and to maintain itself within the field boundaries.

For the Deep RL implementation, the problem was modeled as a Markov Decision Process (MDP) with multiple agents interacting with the environment. In what follows, we define each component of the MDP tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$.

1) State Space $\mathcal{S}$: Due to the varying number of neighbors and obstacles in the agent observation radius at each time step, it needs an alternative method of how to interpret these objects in order to feed the neural network a fixed input. For this, instead of passing the original positions of neighbors and obstacles, we devised an abstraction that matches our purposes with the resulting potential field vectors of neighbors and obstacles. By using the resulting potential field components due to neighbors, the agent can have an estimation of its position relative to the network, in terms of direction and absolute distance. We devised a similar metric for obstacles, but in this case considering only the closest obstacle to the agent, so that the agent do not fall in local minima generated by the joint action of multiple obstacles.

Alongside these two metrics, we also feed the agent its position in the field, its own normalized betweenness centrality [6] — so it can identify if it is a prominent node or not, and the network algebraic connectivity [6], so that it identifies if the network is about to disconnect or not.

Using the Centralized Training and Decentralized Execution (CTDE) method, we train the network with each agent having access to a global state. In this case, the resulting potential field due to neighbors takes into account all drones in the network and the metrics of betweenness centrality and algebraic connectivity, computed using all nodes in the network. In evaluation, we restrict the access of the agent to a local state, which complies with the real execution of the model, so it can only access information of drones within a 2-hop distance from it and obstacles that are seen by these drones. Given this restriction, we change how we calculate the potential fields, taking into account only drones within the 2-hops distance, and how we calculate the betweenness centrality of a node and the network's algebraic connectivity. These two metrics are replaced by local estimates calculated for each node based on the information it possesses.

2) Action Space $\mathcal{A}$: Consists of the velocity of the agent, $\vec{v}_a = [v_{a,x}, v_{a,y}]$, determined by the neural network output.

3) State Transition Matrix $\mathcal{P}$: As we are working with a Deep RL model, the state transition probability matrix $\mathcal{P}$ is encoded in the simulation environment and is unknown to the agent.

4) Reward Function $\mathcal{R}$: The rewards functions were tailored individually for each training task.

5) Discount Factor γ: We set the discount factor as $\gamma = 0.99$.

B. Network Failures

Our model focus on the failures of prominent nodes through network attacks, such that every episode ends when only 30% of the initial network remains. An attack is simulated by removing the node with the highest betweenness centrality at the time. For experimental analysis, we only enabled attacks on the network with 20 agents and after

reaching the last task, Robustness to Failures. Thus, only 6 agents will remain in the network by the end of each episode. Since every episode is bounded by a 15 second time limit, with 14 attacks to occur, we have 1 attack/sec.

C. Multi-Agent Reinforcement Learning Model

We implement the Naïve Parameters Sharing method [7] with the Multi-Agent Deep Stochastic Policy Gradient (MASDPG) [8] model for up to 20 agents. All agents share the same network parameters and every agent equally contributes to parameter updates during training.

IV. Methodology

We relied on several tools and frameworks, such as PyGame for visualization and evaluation of episodes, and the Stable Baselines 3 platform for the implementation of the Proximal Policy Optimization algorithm. Alongside, we created an environment that can interact with several agents simultaneously using the PettingZoo framework, which is also used to work with several environments in parallel with multi-threading, in order to enhance training experience.

A. Policy Configuration

We implemented our agent policy following the Actor-Critic method [2] with PPO [9]. For this, we instantiated two separate actor and critic networks with the same architecture and with the same learning rate α. Table I presents the values of hyperparameters used in PettingZoo's Actor-Critic policy and Stable Baselines 3 PPO's implementation.

TABLE I
Actor-Critic and PPO's Hyperparameters.

Hyperparameter	Description	Default value
Learning Rate (α)	Rate of updates to the policy's parameters	$(3/N)10^{-4}$
Epochs	Number of epochs when optimizing the surrogate loss	10
Batch Size	Minibatch size	60
Timesteps per update (T_{update})	Number of steps to run for each environment per update	900
PPO Clip (ϵ)	Policy ratio clipping parameter	0.2
Entropy Coefficient (ϵ_{coeff})	Entropy coefficient for the loss calculation	0.01
Discount Factor (γ)	Discount factor	0.99
General Advantage Estimation (λ)	Factor for trade-off of bias vs variance for GAE	0.95
ADAM (ϵ)	Numerical stability	10^{-5}
Activation Function ($\phi(.)$)	Non-linear activation function	tanh

The learning rate α is divided by the number of agents N in the network, varying from task to task.

B. Actor-Critic Network Architecture

Table II shows the layer parameters for the actor and critic networks.

The network architecture was defined after running the first training task with 15 different network configurations, varying the number of hidden layers and number of neurons per layer, and was selected by comparing the time spent to converge and the overall performance of the model once trained.

TABLE II
Actor-Critic Model Network Architecture.

Layer	# Neurons	Activation
Input Layer	8	tanh
Hidden Layer 1	32	tanh
Hidden Layer 2	32	tanh
Hidden Layer 3	16	tanh
Output Layer	2	tanh

C. Training

We defined incremental objectives to achieve our main goal. First, we implemented a potential field algorithm that enables our agent to avoid collisions with other drones and obstacles. Then, we started training using CTDE with the objective of moving a small set of agents from start to end positions in the field. Then, we begin gradually increasing the number of agents in each training task as we begin to tackle other behaviors, namely Connectivity Maintenance, Area Coverage and Robustness to Failures.

Initially, we trained the first task only with the Move to Target Location Reward and a set of 2 agents. Once this behavior was learned, we added the Connectivity Maintenance Reward and trained until the network reached 10 agents. Later, we added the Area Coverage Reward and trained until the network reached 20 agents. Finally, we started training Robustness to Failures and added disturbances in the network by promoting attacks on prominent nodes. Algorithm 1 presents the combined reward function with all its terms.

Algorithm 1 Combined Reward Function

```
# Move to Target Location Reward
if position in (LEFT_BORDER, UPPER_BORDER, LOWER_BORDER) then
    reward ← −1.0/N
end if
# Connectivity Maintenance Reward
if network is connected then
    reward ← reward + 2.0/N
end if
# Area Coverage Reward
reward ← reward + (||position - CM|| - R )/ R
# Robustness Controller Reward
reward ← reward + (robustness level − 0.1)/N
```

Table III presents the training schedule used to each task, showing the number of episodes, the number of agents N, and the learning rate, α. For each training session we took advantage of Transfer Learning to utilize the learned behavior from the last session.

1) Task 1 – Move to Target Location: As discussed in previous sections, we implement a simple potential field for each drone and each obstacle to avoid collision, being the

TABLE III
NUMBER OF TRAINING EPISODES FOR EACH TRAINING SESSION.

Task	N	Episodes	α
Move to Target Location	2 Agents	2,000	$1.5 \cdot 10^{-4}$
Connectivity Maintenance	3 Agents	1,000	$1.0 \cdot 10^{-4}$
	4 Agents	2,000	$7.5 \cdot 10^{-5}$
	5 Agents	3,000	$6.0 \cdot 10^{-5}$
	10 Agents	7,000	$3.0 \cdot 10^{-5}$
Area Coverage	10 Agents	7,000	$3.0 \cdot 10^{-5}$
	15 Agents	10,000	$2.0 \cdot 10^{-5}$
	20 Agents	10,000	$1.5 \cdot 10^{-5}$
Robustness to Failures	20 Agents	15,000	$1.5 \cdot 10^{-5}$

only constraints existing in the environment. The following repulsion functions were used for drones and obstacles, respectively

$$\vec{f}_{rep,drones} = \begin{cases} \frac{-1}{d-3}\vec{r} & \text{if } d < 16, \\ \vec{0} & \text{otherwise,} \end{cases} \tag{2}$$

$$\vec{f}_{rep,obstacles} = \begin{cases} \frac{-2}{d-3}\vec{r} & \text{if } d < 16, \\ \vec{0} & \text{otherwise.} \end{cases} \tag{3}$$

With the physics of the environment implemented, we were able to start training Task 1 with a set of 2 agents and for 2,000 episodes. Given that the potential field created by the obstacles on the right of the simulation area diminishes the chances that the agent network will ever reach the target location by random exploration, we implemented a reward function that gives a penalty for staying of the left side of the field instead of an reward for reaching the left end side, being the first term of Algorithm 1.

2) Task 2 – Connectivity Maintenance: Once Task 1 was accomplished, we started training a set of drones to move to the target location while maintaining all agents connected. We followed a schedule of 4 training sessions, increasing the number of agents from 3 to 10, with a reward function given by the first 2 terms of Algorithm 1. In Task 2, the agent was rewarded by establishing a connected network.

3) Task 3 – Area Coverage: With the completion of Task 2, we continued to train the network of 10 drones with an additional reward for Area Coverage. For the first training session, the added a term for Area Coverage is given by

$$\text{Area Coverage Reward} = \frac{\text{Area Coverage}}{\text{Total Coverage}} \tag{4}$$

which calculates the current area being observed by all 10 drones in the network divided by the theoretical maximum $10\pi R^2$ that could be covered.

For the last 2 training sessions, this ratio was replaced by each drone relative position to the network center of mass, shifted and scaled by the observable radius, *i.e.*,

$$\text{Area Coverage Reward} = \frac{\|\text{position} - CM\| - R}{R}. \tag{5}$$

4) Task 4 – Robustness to Failures: Before training Task 4, we implemented a failure policy on the network considering attacks on the highest betweenness centrality node, and analyzed the performance of the agents under the influence of failures. Then, we added the Robustness to Failures Reward term. At this stage, we tested 2 reward functions, one just adding an extra final term to address Robustness to Failure, and another by also removing the Connectivity Maintenance Reward, to test if the similarity between tasks 3 and 4 could be used to simplify the reward function. In both cases, the robustness term was calculated by the Robustness Level of the network [6], *i.e.* the number of drones needed to be removed to fragment the network divided by the total number of drones, minus a threshold devised to teach the network to maintain a minimum value of Robustness Level.

For this task, we trained a network of 20 drones along 15,000 episodes in two different scenarios — with and without attacks, to check if turning on attacks during training time helps enhance results.

D. Evaluation Metrics

To evaluate the model when learning each task, besides the visual aid of the simulator, we used 3 main metrics: Algebraic Connectivity λ, Robustness Level $\Theta(\mathcal{G})$, and Area Coverage Percentage shown in (4). To summarize the evaluation of the model over all 200 scenarios, we calculate the Mean Algebraic Connectivity $\overline{\lambda}$, the Mean Robustness Level $\overline{\Theta}(\mathcal{G})$, and the Mean Area Coverage Percentage, along all episodes.

Also, for each training session we analysed the Accumulated Reward (AR) plot, which consists of all the cumulative sums of rewards generated throughout training. The accumulated reward plotted for an episode is the resulting average from all 8 parallel environments.

V. RESULTS AND DISCUSSION

This section presents the main training results for each task, including the evolution of accumulated reward per episode and the evaluation based on Algebraic Connectivity, Area Coverage and Robustness Level. We also detail the descriptions of the behavior learned in each task, and specific aspects observed in the network[1].

A. Task 1 – Move to Target Location

In this first task, we trained a set of 2 agents with the penalty of staying at the left side of the field. The network learned to move to target after 120 episodes, following a simple policy of moving right and leaving the obstacle avoidance behavior to the potential field controller[2].

B. Task 2 – Connectivity Maintenance

For Task 2, we took advantage of Transfer Learning to continue the training of the agent created in Task 1. In this new scenario, we started training with 3 agents and an extra reward in which the network received a unitary reward every time step it was connected.

[1]A YouTube playlist was created with stored videos recording the learned behaviors for each task: https://tinyurl.com/you2be0drl-fmas-imm
[2]Learned behavior: https://www.youtube.com/watch?v=YlQ7jOcb7MI

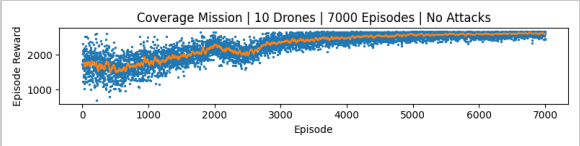


Fig. 1. Task 2 – Accumulated Reward per Episode, AR, for a network of 10 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

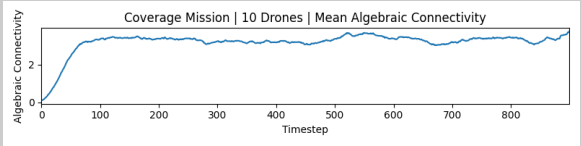


Fig. 2. Task 2 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 10 drones.

(a) Network of 10 drones at $t_1 = 100$.

(b) Network of 10 drones at $t_2 = 300$.

(c) Network of 10 drones at $t_3 = 600$.

(d) Network of 10 drones at $t_4 = 800$.

Fig. 3. Task 2 – Behavior learned for a network of 10 drones.

By analyzing the learned behavior[3], we noticed that the agents learned to keep high proximity among them, increasing the chances of maintaining connectivity even when avoiding obstacles. As in the previous task, the movement of the network was still mostly guided by the potential field controller, making them avoid obstacles and other drones without any complex decisions. We believe that a simpler strategy was learned because there was not enough complex scenarios to learn from. In later stages of training, with more drones in the network, we begin to see a remarkable change regarding this aspect.

To evaluate how well this network performed Connectivity Maintenance, we calculated the mean algebraic connectivity $\overline{\lambda}$ throughout an episode over all 200 scenarios. Analyzing $\overline{\lambda}$, we perceived a significant increase in its value in the initial time steps in which the agents start at random positions and soon rearrange themselves to form a more connected topology. The mean algebraic connectivity $\overline{\lambda}$ remains stable for most of the episode, with few disruptions.

Following these results, we trained networks of 4 agents[4], 5 agents[5], and finally 10 agents[6]. Figure 1 presents the AR per episode for the last training session of Task 2.

In these training sessions our objective was to analyze if the increased number of agents affected the training time, overall performance and if the previously learned behaviors were kept. We noticed that the network continued to move to target without complications. In a few occasions, one or two drones distanced themselves from the network to avoid obstacles, but over the training sessions they generated a regroup policy to quickly return to their original position once the obstacle was surpassed.

Analyzing the AR plot (Figure 1), it is noticeable that the training of the network took way more time to converge due to the increase in variance caused by the addition of agents. There was also an increasing of a few hundred episodes to convergence each time a new agent was added.

Figure 2 shows the mean algebraic connectivity $\overline{\lambda}$ throughout an episode for this network over 200 scenarios. Although the stable behavior remains, its value is slightly lower throughout the episode compared to the previous network.

Analyzing Figure 3, we can observe how the network maintains and improves the behavior of close proximity between drones, leading to increasing the value of λ. This behavior became widely observed in almost all scenarios. After training Connectivity Maintenance for 13,000 episodes throughout all network sizes, the agents learned that keeping a tight topology helps avoid disconnecting. One potential reason for this is that being all close to each other all of them suffers almost the same resulting potential field from the environment, so the network does not need to learn more complex behaviors.

From these results, we observe that the agents have successfully learned to maintain a network topology that performs Connectivity Maintenance while moving to target and avoiding obstacles.

C. Task 3 – Area Coverage

The task of training the network to expand its observable area was built on top of the behaviors learned of Moving to Target Location and Connectivity Maintenance.

To train the network to expand its topology we first tried to feed the network a reward for expanding its observable area based on the total area currently covered divided by the total possible area the drones can cover, Equation (4). Unfortunately, this term did not help the network to learn how to expand its domain. At the evaluation step any major change in behavior was also noticed, so the evaluation snapshots are omitted here. This issue may have risen for a number of reasons. We hypothesized that being the ratio always positive, the reward gives little to no information to the agent if its actions benefits or not the Area Coverage.

To address the problem found in the network with 10 drones, the Area Coverage Reward was reformulated, as presented in Equation (5), in which we reward drones that are more distant from the center of mass of the network. In

[3]Learned behavior: https://www.youtube.com/watch?v=EXRNTNIGZ2U
[4]Learned behavior: https://www.youtube.com/watch?v=1PZGYOuUgM8
[5]Learned behavior: https://www.youtube.com/watch?v=UrwlK4xIiTc
[6]Learned behavior: https://www.youtube.com/watch?v=yD98inuUeMo

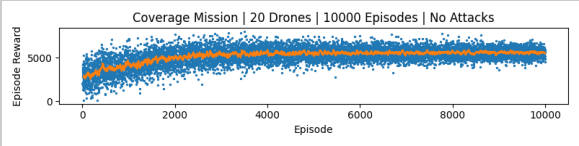


Fig. 4. Task 3 – Accumulated Reward per Episode, AR, for a network of 20 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

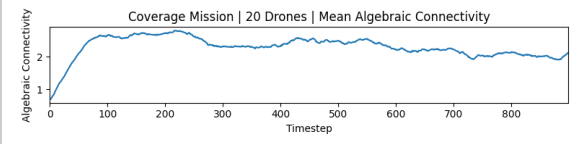


Fig. 5. Task 3 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 20 drones.

this function, the Observable Radius is subtracted from the calculated relative position to also penalize those inner agents that are concentrated on the center of mass. Training with the new reward term yielded convergence of the reward function but with high variance. Observing the network behavior[7], it is possible to notice a large concentration of drones around a single point, evidencing the predominance of Connectivity Maintenance over Area Coverage.

To overcome that, a few variations of the reward function were tried but no improvement in model performance was achieved. Thus, we decided to continue training with this reward function and followed to train a network of 20 agents.

Analyzing the resulting AR per episode (see Figure 4), it is possible to notice that after another 10,000 episodes the variance decreases and the result seems to be more stable. However, $\overline{\lambda}$ dropped significantly in later timesteps (see Figure 5), suggesting that the network started to trade some of its connectivity with area coverage.

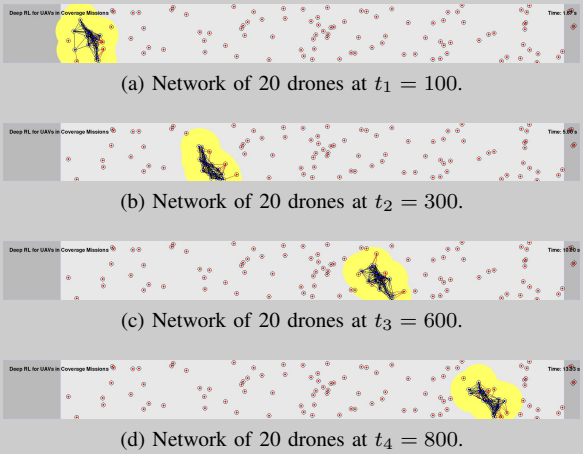
Figure 6 presents snapshots of the network traversing the field and the behavior learned, consolidating the result presented in Figure 5: the network seems more spread out, exchanging connectivity with area coverage[8].

This result may come because of the number of drones in the network, generating more potential fields, causing a agent to distance itself more from its neighbors, or by the additional time of training with this particular reward. But, the result is still beneath the expectations for the task, once the network was expected to assume a topology similar to a Voronoi Tesselation, as in the network generated by Ghedini *et al.* [3].

Figure 7 presents the Mean Area Coverage Percentage for a network of 20 drones. Its value stays stable, with approximately a 7% relative variation.

With these results, we conclude that the network learned a sub-optimal solution for Area Coverage, performing the task below expectations. Despite that, the agents now can perform 3 out of the 4 tasks proposed: Move to Target Location, Connectivity Maintenance, and Area Coverage. In the following Section we show the results of training the network for its final task: Robustness to Failures.

[7]Learned behavior: https://www.youtube.com/watch?v=hlOaMlbRM4g
[8]Learned behavior: https://www.youtube.com/watch?v=jpb9PPhz3BY



(a) Network of 20 drones at $t_1 = 100$.

(b) Network of 20 drones at $t_2 = 300$.

(c) Network of 20 drones at $t_3 = 600$.

(d) Network of 20 drones at $t_4 = 800$.

Fig. 6. Task 3 – Behavior learned for a network of 20 drones.

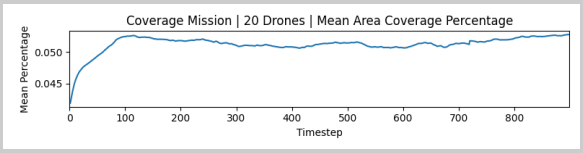


Fig. 7. Task 3 – Mean Area Coverage Percentage throughout an episode for a network of 20 drones.

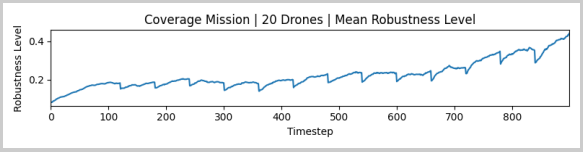


Fig. 8. Task 4 – Mean Robustness Level $\overline{\Theta}(\mathcal{G})$ throughout an episode for a network of 20 drones.

D. Task 4 – Robustness to Failures

Starting directly with a network of 20 drones building on top of the behaviors learned from Move to Target Location, Connectivity Maintenance, and Area Coverage, the goal of the network training is to establish a topology that can carry on the mission even under attacks. The failure policy was implemented as described in Section III-B.

For this task, the current network performance is measured in terms of Mean Robustness Level $\overline{\Theta}(\mathcal{G})$ during an episode. Figure 8 presents the results computed over 200 scenarios.

The rhythmic behavior of $\overline{\Theta}(\mathcal{G})$ shows the attacks occurring in the network: every timestep a drone is removed from the network, its connectivity and robustness are affected. Despite that, the connected behavior learned in the previous training sessions allowed the network to maintain $\Theta(\mathcal{G})$ above 0.1 at all times, meaning that, on average, it would take 2 simultaneous attacks to disconnect the network. Furthermore, after each attack, the network rearranges itself to increase the robustness level, thus trying to maintain a resilient topology.

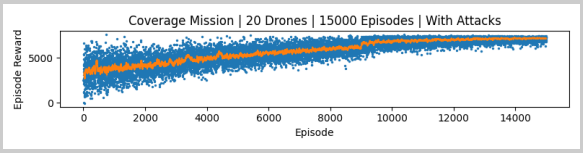


Fig. 9. Task 4 – Accumulated Reward per Episode, AR, for a network of 20 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

(a) Network of 20 drones at $t_1 = 100$, 19 drones remaining.

(b) Network of 20 drones at $t_2 = 300$, 15 drones remaining.

(c) Network of 20 drones at $t_2 = 600$, 10 drones remaining.

(d) Network of 20 drones at $t_4 = 800$, 7 drones remaining.

Fig. 10. Task 4 – Behavior learned for a network of 20 drones.

Using the reward function given by Algorithm 1, we trained the network for a total of 15,000 episodes, this time enabling failures during training. Figure 9 presents the resulting AR per episode. The first aspect to notice is how much longer the network took to converge. This can be explained by the decreasing number of drones in the network until the end of the episode caused by the attacks, generating less and less experiences towards the end.

Figure 10 presents the learned behavior[9] with the number of agents decreasing due to attacks policy but still connected to each other until the end, enduring the attacks.

Evaluating different scenarios, the network showed consistent results w.r.t. Connectivity Maintenance and Robustness to Failure by exchanging performance in Area Coverage.

To evaluate the Robustness Level of the network during an episode, we plot the Mean Robustness Level $\overline{\Theta}(\mathcal{G})$, Figure 11. Note that after the training, the value of $\overline{\Theta}(\mathcal{G})$ had a slight increase in the beginning of episodes until 400 timesteps, maintaining its value above 0.2 at almost all times. In addition, the rhythmic behavior continues to appear, showing that the network tries to recompose itself in a resilient manner every time after an attack.

The Mean Algebraic Connectivity $\overline{\lambda}$ and the Mean Area Coverage Percentage, Figures 12 and 13, demonstrate that the network under attack rearranges itself to keep the value of λ. This may be caused by the similarity of the two tasks.

Interesting to see, $\overline{\lambda}$ seems to be never go below 2.0, showing that even though the network have space for a trade-off from Connectivity to Area Coverage, it chose to specialize in Connectivity Maintenance and Robustness to Failures. This

[9]Learned behavior: https://www.youtube.com/watch?v=Ga7SBLxuw04

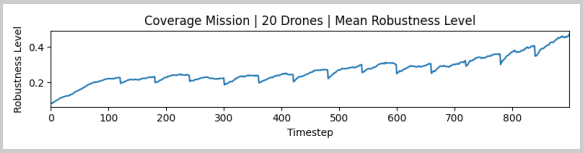


Fig. 11. Task 4 – Mean Robustness Level $\overline{\Theta}(\mathcal{G})$ throughout an episode for a network of 20 drones.

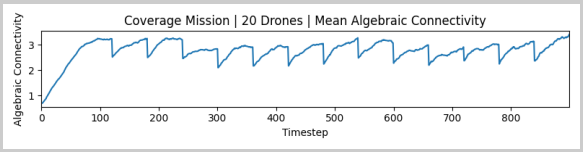


Fig. 12. Task 4 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 20 drones.

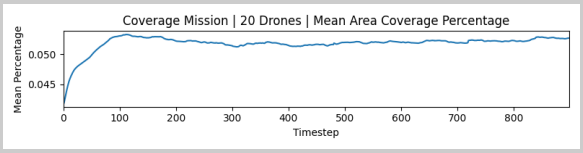


Fig. 13. Task 4 – Mean Area Coverage Percentage throughout an episode for a network of 20 drones.

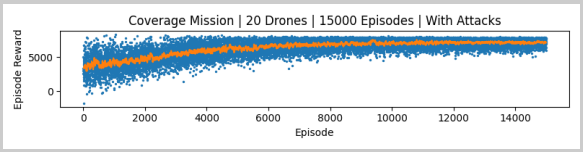


Fig. 14. Task 4 – Accumulated Reward per Episode, AR, with the simplified Reward Function for a network of 20 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

suggests that there is still opportunity for continue learning Task 3 afterwards.

After training the network with an additional reward to tackle robustness, we devised a second reward function, keeping the additional term but removing the reward regarding connectivity of the network. This was done after realizing that the previous reward function just enhanced the behavior already learned for Task 3. The training occurred in the same standards as before, 15,000 episodes and on top of what the agent had already learned from the 3 previous tasks.

Figure 14 presents the Accumulated Reward per episode for this task. By the end of it, when comparing the behavior learned, the same characteristics of maintaining a resilient topology throughout the episode and rearrangement after attacks were noticed in the retrained network as well, with no major change detected.

This result suggests that when the Robustness to Failures is introduced in the model, the Connectivity Maintenance starts to lose importance, since despite having different objectives both try to enhance the network connectivity in some way.

The last analysis is to evaluate whether enabling attacks during training is beneficial or not to the agent, considering that attacks decrease the number of agents (and experience generated) in the network and there is no explicit penalization

for network fragmentation. We re-ran the training of Task 4 once again with the simplified reward function. When comparing the AR plots generated by the two training sessions, the agent converged faster to a solution by disabling attacks during training. In terms of behavior learned, there was not a clear difference between both networks, with Robustness Levels being similar throughout the episodes and no disparity in any other learned behavior.

VI. Conclusion

In terms of the behaviors that the agent learned in the context of multicriteria missions, Task 1 - Move to Target Location was straightforward to learn with the implemented reward, allowing the agents to travel through the obstacles and reach the target location. This task was accomplished without any issues.

For Task 2 - Connectivity Maintenance, the Mean Algebraic Connectivity measured showed that the network learned to be well connected from the start, surpassing the determined threshold. During this task, a few generated behaviors were noted, as following to target with a slight bias to the lower border and a regrouping policy.

In Task 3 - Area Coverage, the agent did not achieve the expected results. We would expect that the agent would reach a coverage ratio of 25%, following the results generated by Ghedini *et al.* [3] when using a Voronoi Tesselation to adjust the topology, but it reached a maximum value of only 7%. One possible reason is that the Connectivity Maintenance behavior learned previously overcame the learning of Area Coverage, causing too much overlap between the agents' observable areas. Despite this task was not perfectly accomplished, there are still some feasible paths to address it. They will be subject to further study.

In our final task, Task 4 - Robustness to Failures, given the previous adequacy from Task 2, the network showed good results even before training. When our attack policy was enabled the network endured the attacks and kept a Robustness Level over 10% during the entirety of the episodes, matching the threshold added in the reward function. After training, the Robustness Level had a slight increase in the beginning of episodes, with over 20% Robustness Level at the end of episodes. Task 4 was thus accomplished, and all tasks were completed. We can observe that the order in which each task was introduced to the network affected which it learned better.

For the overall purpose of a coverage mission, the network performed well in terms of connectivity and robustness to failures, but there is still room for improvement in area coverage. For a multicriteria mission, the agent presented good overall results in learning several different behaviors in a extremely complex scenario. It is therefore possible to use Multi-Agent Deep Reinforcement Learning techniques to incrementally train agents that create a resilient network topology able of performing in multicriteria missions.

We propose a few ideas of possible research paths and future works based on this research, as:

- Continue the training of Task 3 – Area Coverage on top of the final model, focusing in exchanging algebraic connectivity for area coverage with a more suited reward function.
- Train a Reinforcement Learning Model to optimize the weights of the controller devised by Ghedini *et al.* [3].
- Introduce a battery parameter to simulate the drone's decline of usability through time, and identify the changes that the topology suffers to adequate itself for a drone that is close to shutdown.
- Change the input vector of the network for an image of the agent surroundings, removing all complexity of calculating the potential field of nearest neighbors and obstacles, leaving for a Convolutional Neural Network to identify these aspects.
- Introduce moving obstacles and enemies in the field so the agent has to take into account their future positions for path planning.

References

[1] YANG, G. Z.; BELLINGHAM, J.; DUPONT., P. E. The grand challenges of science robotics. Science Robotics, vol. 3, n. 14, 2018.
[2] SUTTON, R. S.; BARTO, A. G. Reinforcement Learning: An Introduction. MIT Press, 2018. http://incompleteideas.net/book/the-book-2nd.html.
[3] GHEDINI, C.; H.C.RIBEIRO, C.; SABATTINI, L. Toward efficient adaptive ad-hoc multi-robot network topologies. Ad Hoc Networks, vol. 74, p. 57–70, 2018. Available from Internet: ¡https://doi.org/10.1016/j.adhoc.2018.03.012¿. Cited: 03 jul. 2022.
[4] MINELLI, M.; KAUFMANN, M.; PANERATI, J.; GHEDINI, C.; BELTRAME, G.; SABATTINI, L. Stop, think, and roll: Online gain optimization for resilient multi-robot topologies. Distributed Autonomous Robotic Systems, p. 357–370, 2018
[5] GUPTA, A.; NALLANTHIGHAL, R. Decentralized multi-agent formation control via deep reinforcement learning. 2021. Available from Internet: https://www.scitepress.org/Papers/2021/102413/102413.pdf. Cited: 04 jul. 2022.
[6] OWEN-SMITH, J. Network Theory: The Basics. 2017. https://www.oecd.org/sti/inno/41858618.pdf.
[7] CHRISTIANOS, F.; PAPOUDAKIS, G.; RAHMAN, A.; ALBRECHT, S. V. Scaling multi-agent reinforcement learning with selective parameter sharing. 2021.
[8] KASSAB, R.; DESTOUNIS, A.; TSILIMANTOS, D.; DEBBAH, M. Multi-agent deep stochastic policy gradient for event based dynamic spectrum access. 2020.
[9] SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. 2017.