Optimizing Multi-Agent Coverage Path Planning for Maritime Search and Rescue Operations using Proximal Policy Optimization

Ricardo R. Rodrigues[1], Renato L. Falcão[1], Joras C. C. Oliveira[1], Pedro H. B. A. Andrade[1], José F. B. Brancalion[3], and Fabrício J. Barth[2]

[1]Insper, São Paulo, Brazil
Email: {ricardorr7, renatolf1, jorascco, pedroa3}@al.insper.edu.br

[2]Insper, São Paulo, Brazil
Email: fabriciojb@insper.edu.br

[3]Embraer, São José dos Campos, Brazil
Email: jose.brancalion@embraer.com.br

***Abstract*—This paper explores the application of multi-agent reinforcement learning (MARL) to enhance the coverage path planning (CPP) of maritime search and rescue (SAR) operations for persons in water (PIW). A key objective of maritime SAR is to maximize the cumulative probability of success (POS) within a limited time frame using available resources. Traditional CPP algorithms often struggle with the dynamic and complex nature of the maritime environment. This research investigates the potential of MARL, specifically a Proximal Policy Optimization (PPO) algorithm, to enable autonomous coordination among multiple SAR units, optimizing their search paths to maximize POS. The experiments were conducted using a simulated SAR scenario. The results demonstrate that PPO agents learned efficient behaviors and completed tasks. Those results highlight that reinforcement learning algorithms can train agents to coordinate and efficiently cover a search area autonomously.**

***Index Terms*—Multi-Agent Systems, Reinforcement Learning, Simulation**

I. Introduction

The issue of missing persons-in-water (PIW) is as old as humankind itself, and given the chaotic nature of the ocean, search and rescue (SAR) operations have never been optimal, with limitations on human ability ranging from creating proper search paths to visibility and recognition of PIW.

The usage of drones allows for continuous search over extended periods and distances, while the capabilities of artificial intelligence (AI) with reinforcement learning (RL) for problem-solving are still in their infancy, it is theorized that the introduction of AI for SAR applications may significantly improve the efficacy of search operations and reduce the time needed to find and save PIW [1]–[3]. RL is believed to enable the development of new, more efficient search patterns tailored to specific applications. This is based on the hypothesis that reward maximization is sufficient to foster generalization abilities, thereby creating powerful agents [4]. Such advancements could potentially lead to the saving of more lives.

Ai [5] and Wu [6] explore RL to improve maritime SAR coverage path planning. Both acknowledge the limitations of traditional, fixed search patterns and propose the use of RL to learn models that will be used by agents (i.e., drones) in maritime SAR missions. These models prioritize high-probability areas based on drift predictions and maximize the cumulative probability of success (POS) while minimizing search time. The authors offer distinct approaches, with [5] focusing on a multi-objective reward function and non-linear action selection, and [6] proposing a hierarchical probability map for multi-agent coordination. Both research teams validate their approaches through simulations and comparison with existing algorithms, highlighting improvements in the efficiency of search path planning.

One commonly employed method in the traditional path planning approaches is the Parallel Line Scanning (PLS) algorithm, which ensures complete coverage of the search area but does not account for variations in the probability of target presence. This limitation results in lower cumulative POS growth rates, as PLS follows a systematic, non-adaptive search pattern. The Sequential Parallel Line Scanning (SPLS) algorithm offers a refinement by initiating the search from the grid with the highest probability. Despite its faster initial POS growth, SPLS's reliance on fixed patterns leads to inefficiencies due to overlapping paths [7], [8], [9].

Recent advancements in path planning have introduced more sophisticated algorithms. The A^* search (A^*) is well-suited for coverage path planning in SAR, efficiently prioritizing high-probability areas while ensuring complete coverage. Its adaptability enhances real-time decision-making, making it ideal for uncertain environments [7].

Reinforcement learning-based algorithms, such as Q-learning [10], have demonstrated significant improvements by achieving full coverage without path overlaps and exhibiting rapid cumulative POS growth. However, Q-learning may still

traverse areas with zero probability, impacting overall search efficiency. A novel Deep Q-Network (DQN) based approach [11] has been introduced to address these challenges. The DQN algorithm consistently outperforms traditional methods in terms of efficiency, coverage, and prioritization of high-probability areas. It achieves complete coverage without path overlaps and demonstrates the highest cumulative POS growth within a shorter timeframe by effectively balancing exploitation and exploration [6].

The DQN model's ability to prioritize high-probability areas and balance exploitation with exploration is crucial for navigating complex and dynamic search environments. Its scalability for handling high-dimensional state and action spaces makes it particularly well-suited for complex maritime SAR scenarios.

However, the algorithms proposed in [5], [6] are constrained by the assumption of a static search environment, implying that the search space remains unchanged during the path-planning process. An additional limitation identified in [5] is the use of region partitioning to enable the deployment of multiple agents (drones) in rescue operations. Similarly, [6] recognizes the drawback of their single-agent system and emphasizes the necessity for multi-agent collaboration in real-world SAR operations.

This study explores the effectiveness of reinforcement learning in solving coverage path planning problems within dynamic, multi-agent environments. The primary research question is: can reinforcement learning enable agents to autonomously coordinate and efficiently cover a search area, increasing the cumulative Probability of Success (POS) and reducing search time?

The paper is organized as follows: Section II reviews common algorithms used in coverage path planning. Section III details the dynamic environment utilized in our simulations. Section IV discusses the reinforcement learning algorithm implemented in this research. Section V presents the outcomes of our experiments, and finally, Section VI summarizes our findings and conclusions.

II. Coverage Path Planning Algorithms

According to [6], searching for PIWs includes three main tasks: (1) accurately and quickly predicting the drift trajectory of PIWs; (2) determining the optimal search area to ensure full coverage of the possible distribution range, and; (3) planning the search path for the SAR units and maximizing the cumulative probability of success (POS) of the entire search process. POS relies mainly on probability of containment (POC) and probability of detection (POD) [1]. The first task is usually solved by using drift prediction models, while the second and third tasks are solved by coverage path planning (CPP) algorithms.

The usual algorithms applied to coverage path planning activities are parallel line scanning (PLS) and expanding square search (ESS). Some works also consider algorithms like Dijkstra algorithm, A^* algorithm, and D* algorithm for this task. The reinforcement learning (RL) method is an important approach in machine learning. In contrast with other intelligent algorithms for machine learning, RL focuses on the acquisition of system mapping from the environment to the behavior. It does not rely on labeled interactions as seen in supervised learning; instead, it learns from its own experiences. The objective of RL is not to discover hidden structures but rather to maximize rewards.

In this section we will describe: the parallel line scanning algorithm (PLS); the expanding square search (ESS), and; A^* algorithm. Those algorithms will be compared with our reinforcement learning implementation.

A. Parallel Line Scanning

The Parallel Line Scanning, also known as the Parallel Sweep Search, is an algorithm designed to efficiently cover extensive search areas, particularly when the target's location is highly uncertain. This method is most effective over flat terrains or water surfaces, where the search area can be systematically divided into rectangular sub-areas for comprehensive coverage. As illustrated in the figure 1, the search begins at the Commence Search Point (CSP) located at one corner of the sub-area. From the CSP, search legs are executed parallel to the longer sides of the rectangle, maintaining equal spacing between successive tracks to ensure uniform coverage across the entire area. This systematic approach is especially advantageous when multiple search units collaborate to cover subdivided regions.

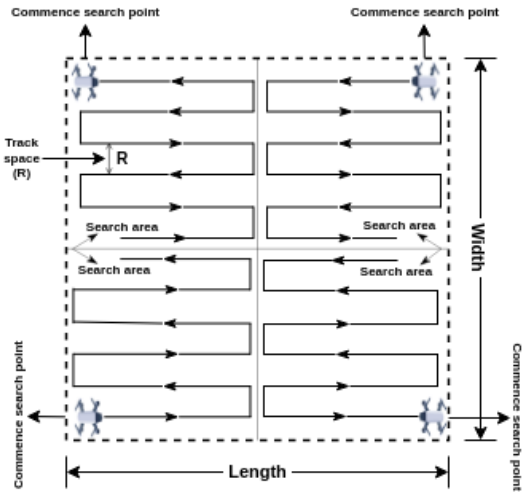

Fig. 1. Parallel line search for maritime search and rescue.

Each search leg is separated by a defined Track Space (R), which is determined based on the maximum distance at which the searching agent can effectively detect the search object. This distance is influenced by various factors, including meteorological conditions, the detection equipment employed, and the overall search environment. By appropriately adjusting the track spacing based on these factors, with smaller spacing when conditions are less favorable and larger spacing when

[1] $POS = POC \times POD$

conditions improve, the algorithm maximizes the probability of detection while ensuring efficient utilization of search resources.

B. Expanding Square Search

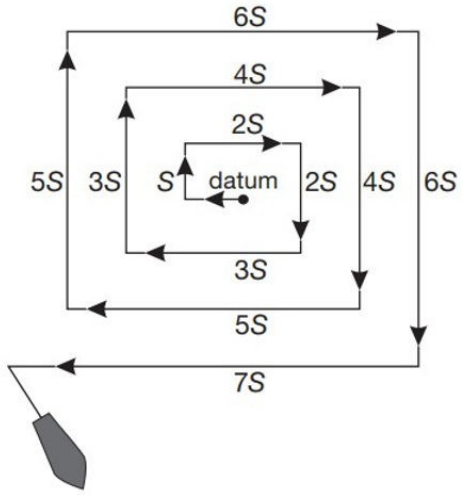

Fig. 2. Expanding square search for maritime search and rescue [7].

The Expanding Square Search (ESS) algorithm is a systematic search pattern that begins from a central point, known as the datum, and expands outward in an ever-widening square spiral, as shown in figure 2. It is especially useful when the approximate location of the search object is known but not precisely determined. The search commences at the datum, also referred to as the Commence Search Point (CSP), and progresses outward, with each leg incrementally increasing in length to ensure comprehensive coverage.

In the ESS pattern, the first two legs match the track spacing (S), while each subsequent pair of legs increases by an additional track spacing, creating a spiral that expands uniformly. This systematic approach allows for thorough coverage as the search extends outward from the center. It is particularly advantageous for relatively small, well-contained areas.

To minimize navigational errors, the initial leg is typically aligned with environmental factors such as wind or current direction. If the search must be extended, the pattern can be repeated, with each new iteration rotated 45 degrees to close any gaps and enhance overall effectiveness.

The ESS pattern excels in maritime operations, such as locating objects or individuals in water, where the target is expected to remain near the starting point and environmental factors such as water currents may influence its position.

C. A^* Algorithm

The A^* algorithm is a heuristic-driven search method adapted for coverage path planning, designed to minimize travel cost while efficiently navigating an environment. In this context, it focuses on high-probability areas in a partially known space, avoiding redundant coverage and balancing efficiency with thorough exploration. Unlike traditional exhaustive search patterns, A^* continuously updates its path based on real-time data, making it particularly effective in large-scale and uncertain settings.

A^* operates by evaluating potential moves through a cost function that incentivizes traveling to cells with higher probability of detection, while penalizing longer routes and revisiting already covered regions. This approach allows the search agent to prioritize the most promising locations without neglecting the broader search area.

By representing the search environment as a graph, A^* can efficiently travel over complex spaces, avoid obstacles, and minimize unnecessary travel. However, careful tuning of heuristics and optimization strategies is essential to manage redundant paths and maintain an optimal balance between comprehensive coverage and navigation efficiency.

In the next section, we describe the environment used to validate these coverage path planning algorithms and train the agents through a reinforcement learning approach.

III. Coverage Environment

For this work, the Drone Swarm Search Environment (DSSE) framework, developed by the authors and available as a Python package [12], was used to study the viability of using multi-agent reinforcement learning in searching for shipwrecked people. To facilitate the development and properly differentiate intended uses, two distinct environments were established: a *search environment* and a *coverage environment*.

The Coverage environment was designed based on state-of-the-art research in maritime coverage search path planning [5], [6]. This complex field blends coverage path planning with maritime SAR operations, akin to a variant of the traveling salesman problem [5]. Trumme [13] has demonstrated that finding an optimal search path, where the agent must search all sub-areas using the shortest possible path, is NP-complete. Considering this complexity, the environment is specifically tailored to reward agents for maximizing area coverage without redundant searching, while prioritizing areas with a greater Probability of Containment (POC).

The environment is a 2D grid (figure 3) with a matrix representing the probability of containing a PIW. This environment includes exclusively variables relevant to the coverage objective and uses a probability matrix, which integrates a Lagrangian particle model [14] with maritime and wind satellite data to achieve greater precision. Implemented by OpenDrift [14], this model treats the PIW as particles, and applies current and wind data, predicting the final position of a PIW based on the initial position and the time of drift. The result of this simulation is a list of latitude and longitude points for each of the particles, with this it is possible to use some mathematical transformations to create a probability matrix.

The first transformation implemented is a minimum bounding rectangle (MBR) method. This method processes a list of points to identify the points with minimum and maximum latitude and longitude. With these points, the haversine formula

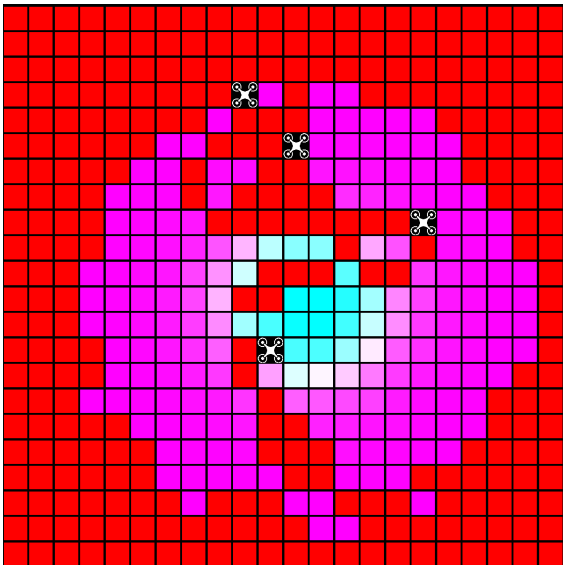

Fig. 3. Simulation environment showcasing the algorithm's execution.

- expressed in (1), where R is the radius of the earth, λ is the longitude, and φ calculates the distance between geographical coordinates in meters on the Earth's surface. This calculation helps determine the search grid size. The grid cell size, a configurable parameter, defaults to 130m. With this, the grid's width and height can be calculated, and the measurements converted into grid cells.

$$d = 2R \times \arcsin\left(\sqrt{\sin^2\frac{\Delta\varphi}{2} + \cos\varphi_1 \cos\varphi_2 \sin^2\frac{\Delta\lambda}{2}}\right) \tag{1}$$

Once the map size is calculated, the latitude and longitude coordinates must be converted to x and y coordinates on the simulation grid. This is achieved through an equirectangular projection, expressed in the following equation:

$$\begin{aligned} x &= R \cdot (\lambda - \lambda_0) \cdot cos(\varphi_1) \\ y &= R \cdot (\varphi - \varphi_0) \end{aligned} \tag{2}$$

where λ is the longitude to project, λ_0 is the central meridian of the plane, φ_1 is a latitude close to the center of the plane, φ is the latitude to project and φ_0 is the central parallel of the plane. This formula helps translate the points on the spherical surface of the Earth to the simulation's Cartesian plane. Having each particle's (x, y) position established on the grid, the POC is calculated via a counting method: the POC is derived from the ratio of particles within a grid cell to the total number of particles. After implementing these transformations, the probability matrix is generated, integrating both simulation and statistical data.

The environment states are represented as a tuple of two boxes: one for the drone's position (x, y) and another for the probability matrix. This allows the agents to use the probabilities and their own positions to decide the best action, facilitating the orchestration of complex search patterns to cover the region. The action space of the environment is discrete, consisting of 8 actions: moving in the cardinal and inter cardinal directions. The probabilities of cells that have already been searched are set to 0.

This environment incorporates multi-objective optimization, expecting agents to learn to minimize coverage time while simultaneously prioritizing cells with greater POC. The reward structure reflects how agents learn to optimize these tasks. If one task yields a greater discounted sum, agents tend to prioritize that task and treat the other as a sub-goal. This poses a challenge in developing an effective reward function. To address this a reward function that tries to balance the objectives was created as described in the following equations:

$$R(Ts) = \begin{cases} R_{done}, & \text{if } S_{t+1} = S_{done} \\ 1 + R_{poc}, & \text{if } S_{t+1} \neq S_{done} \ \ \& \ \ S_{t+1} \notin V \\ -0.2, & \text{otherwise} \end{cases} \tag{3}$$

where S_{done} is the state where all the cells with POC greater than zero have been searched, V is the set that holds all the non-searched cells, R_{done} is defined as:

$$R_{done} = 2 \cdot n_{cells} - n_{cells} \cdot \left(\frac{Ts}{Ts_{limit}}\right) \tag{4}$$

and R_{poc} is defined as:

$$R_{poc} = POC \cdot n_{cells} \cdot \left(1 - \frac{Ts}{Ts_{limit}}\right) \tag{5}$$

where POC is the probability of containment in the searched cell, Ts is the search's time step, Ts_{limit} is the limit of the time steps for the simulation and n_{cells} is the number of cells with POC greater than zero. The fraction on R_{done} and R_{poc} is used to stimulate the agents to conclude the search as quickly as possible, giving out greater rewards for faster searches, while the usage of the number of cells in the formula is used to keep the discounted sum of the rewards in the same scale, so that all tasks are made relevant to the agents.

IV. Proximal Policy Optimization Implementation

The agents' behaviors are modeled using an implementation of Proximal Policy Optimization (PPO) [15], a state-of-the-art reinforcement learning algorithm designed to stabilize and streamline the learning process.

Earlier methods of policy optimization faced two key issues. If parameter updates were too gradual, the agent learned very slowly, showing little improvement. On the other hand, overly abrupt updates risked leading the agent into undesirable policies, even if those updates were moving toward an optimal solution. To address this dilemma, Trust Region Policy Optimization (TRPO) introduced a mechanism to constrain how far an updated policy could deviate from its previous version, thereby preventing performance degradation. However, TRPO's reliance on second-order derivatives made it computationally expensive for large-scale problems. Consequently,

PPO was developed as a more efficient successor to TRPO, striking a balance between policy stability and computational cost.

The core idea behind PPO is to maintain stable updates by comparing how likely an action is under the new policy versus the old one at time step t—a ratio denoted by $r_t(\theta)$—and "clipping" it within $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter. Alongside this ratio, PPO uses an advantage function $\hat{A}_t$, which indicates how much better or worse an action performed compared to what the agent expected at state s_t. Concretely, $\hat{A}_t$ is usually computed by subtracting the expected value (from a baseline value function $\hat{V}(s_t)$) from the actual return (or partial return) following action a_t. A positive advantage ($\hat{A}_t > 0$) means the action outperformed the baseline estimate and should be reinforced; a negative advantage discourages repeating that action. The final parameter update is then the minimum of $r_t(\theta)\hat{A}_t$ and the clipped ratio multiplied by $\hat{A}_t$, preventing large, destabilizing jumps and preserving stable learning progress.

The agent uses a PPO algorithm with a Multilayer Perceptron (MLP), a fully connected neural network with two layers (512 and 256 nodes), and a `tanh` activation function. Implemented with the Ray RLlib [16] library, this setup enables efficient training and scalable policy evaluation.

V. Experiment and Results

The experiment was carried out on a 9x9 grid with 58 cells having a POC greater than 0, generated by a 2-hour drifting simulation from default initial points near Guarujá, São Paulo, Brazil. This experiment aimed to extend previous research by increasing complexity and using a multi-agent approach. We used 58 valid search grid cells, compared to 19 in [6] and 30 in [5], to test whether agents could autonomously learn and organize to parallelize tasks effectively.

This experiment aims to compare Parallel Line Scanning, Expanding Square Search, A^*, and a PPO implementation. To compare, we will use the following tree indicators: accumulated POS, coverage rate, and repeated coverage. The accumulated probability of success (POS) of the SAR mission serves to quantify the chance of finding all SAR targets within a mission. The coverage rate is the percentage of cells the drone has covered with a probability greater than zero. Repetition coverage is the percentage of the grid covered more than once, indicating overlap in search areas.

All agents trained using the PPO algorithm were implemented using the same neural network architecture, a multilayer perceptron network (MLP), as described in Section IV. Considering configuration with 2 and 4 agents, the agents were trained with 40 million steps. Each episode has a time step limit equal to 200. The episode will end when the episode reaches the 200 steps or when the agents cover all positions where the POS is greater than zero. We ran the training process five times for each configuration. This way, we can present a more accurate learning curve for each agent.

In Figure 4, we can see the learning curve and the size of each episode of both scenarios. In this graphic, it is clear that the agents can learn how to create coverage path planning because the size of each episode decreases according to the number of steps. The agents could cover all positions where the POS is greater than zero. The learning curve for each agent does not have a high variance.

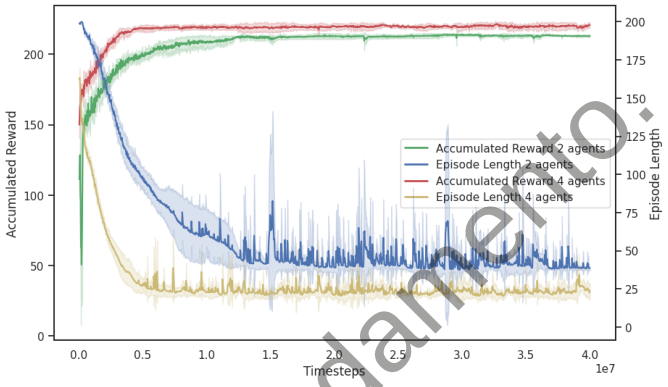

Fig. 4. Learning curve and episode length for PPO implementation.

The PPO implementations, once trained, were executed in an environment with the exact dimensions of the training setup. The other algorithms were also executed in the same environment, considering one scenario with two agents and another with four. Each combination of algorithm and scenario was executed 100 times, considering different positions for each agent. The results are discussed below.

TABLE I
Repeated coverage values for environments with two and four agents

Algorithm	Two agents	Four agents
A^*	0.27 ± 0.07	0.44 ± 0.22
ESS	inf	1.24 ± 0.49
PLS	inf	1.35 ± 0.05
PPO	0.42 ± 0.25	0.84 ± 0.24

The parallel line scanning (PLS) algorithm performs worst in the scenario with two agents because it fails to cover all positions with POS greater than zero. The PLS algorithm cannot find in all tests all relevant cells. In other words, this algorithm sometimes did not reach a coverage rate equal to one (figure 6). The algorithm Expanding Square Search (ESS) was not able to find all relevant cells in many tests. For this reason, both algorithms have the value inf for the repeated coverage value in the scenario with two agents in the table I.

The A^* algorithm and the agents trained with PPO exhibit very similar performance across all indicators. The PPO and A^* agents can cover on average all the significant regions in less than 40 steps (figure 5, 6) considering the scenario with two agents.

Considering a scenario with four agents, the worst agents are the agents implemented using the Parallel Line Scanning algorithm. The other three algorithms have similar behavior regarding cumulative POS (figure 7), coverage rate (figure 8), and repeated coverage (table I).

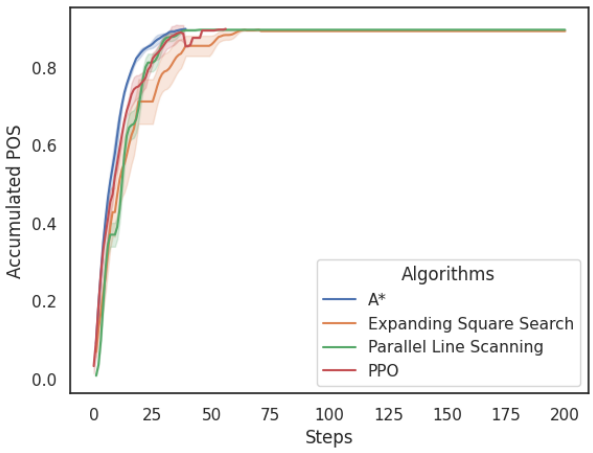

Fig. 5. Comparison of cumulative POS with two agents.

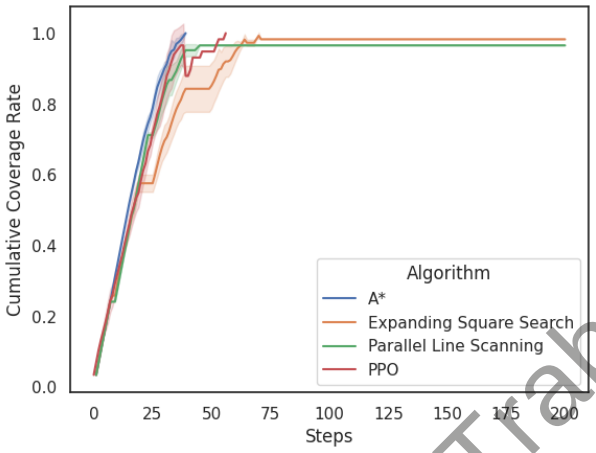

Fig. 6. Coverage rate for two agents.

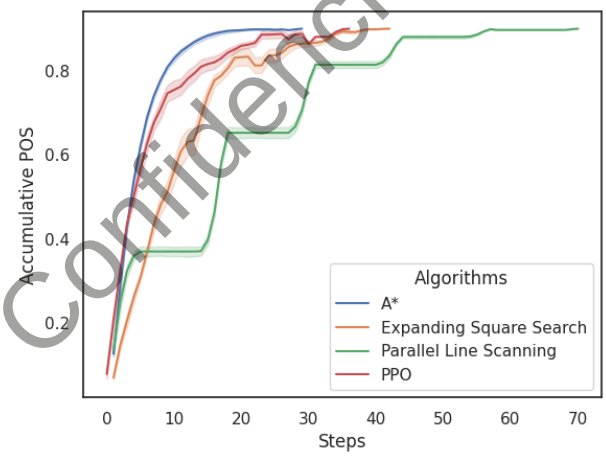

Fig. 7. Comparison of cumulative POS with four agents.

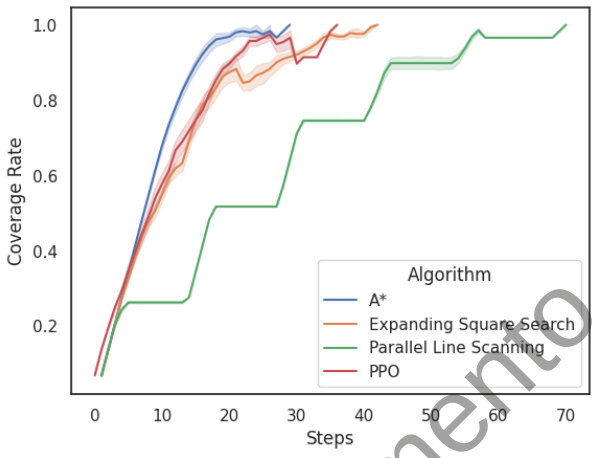

Fig. 8. Coverage rate for four agents.

The results indicate that PPO agents learned efficient behaviors and completed tasks equal to a group of agents built using the A^* algorithm. Those results highlight that reinforcement learning algorithms can train agents to autonomously coordinate and efficiently cover a search area, increasing the cumulative Probability of Success (POS) and reducing search time. The experiments' results provided sufficient evidence to suggest that the PPO algorithm learned a policy similar to the heuristic applied in the A^* implementation.

The values of repeated coverage of A^* implementation are better than that of PPO agents. The heuristic used in the A^* implementation does not consider any agent collaboration.

VI. Conclusions

This study explored the effectiveness of reinforcement learning in solving coverage path planning problems within multi-agent environments for maritime search and rescue operations. The primary research question was whether reinforcement learning could enable agents to coordinate autonomously and efficiently cover a search area, increasing the cumulative Probability of Success (POS) and reducing search time.

To answer this question, we trained two models, one considering a scenario with two agents and another considering four agents, using a publicly available environment. We used an environment with higher dimensions than other studies [5], [6], leading us to evaluate the scalability and adaptability of the reinforcement learning approach in more complex settings.

We compare the behavior of agents trained using PPO against classical algorithms used in maritime search and rescue operations, like parallel line scanning (PLS), expanding square search (ESS), and A^*. To analyze the behavior of all agents, we used the same environment available to train agents using reinforcement learning.

The PLS and ESS algorithms had the worst results in our experiments, and the A^* and PPO agents had better results. By comparing the performance of PPO agents with those using the A^* algorithm, we assessed their ability to learn efficient

behaviors and complete tasks effectively. The results demonstrated that PPO agents could cover the search area, achieving comparable performance to heuristic-based methods. These findings provide further evidence that reinforcement learning can be a viable alternative for multi-agent coverage path planning, particularly in scenarios where predefined heuristics may not be optimal.

The heuristic implemented in the A^* algorithm prioritizes cells with high POC without accounting for the positions or actions of other agents, resulting in a lack of explicit collaboration. In other words, the agents operate independently without demonstrating cooperative behavior during area coverage. The similarity between the behaviors of PPO agents and A^* agents suggests that the PPO agents failed to learn effective collaboration strategies. However, at the same time, it is not clear if the environment characteristics allow us to check if any cooperative behavior exists.

Future work should focus on enhancing the collaborative capabilities of reinforcement learning agents in multi-agent coverage path planning. One possible direction is incorporating communication mechanisms or shared reward structures to encourage agent coordination. Further research could also investigate the impact of environmental factors, such as dynamic obstacles or changing search conditions, to evaluate the adaptability of trained agents in real-world maritime search and rescue scenarios.

REFERENCES

[1] J. C. C. de Oliveira, P. H. B. A. Andrade, R. L. Falcão, R. R. Rodrigues, J. F. Brancalion, and F. Barth, "Reinforcement learning applied to train autonomous maritime search and rescue drones," in *21st National Meeting on Artificial and Computational Intelligence*, 2024.

[2] P. H. B. A. Andrade, R. R. Rodrigues, R. L. Falcão, J. C. C. de Oliveira, J. F. Brancalion, and F. Barth, "Using deep reinforcement learning to coordinate autonomous maritime search and rescue drones," in *The Latin American Workshop on Information Fusion*, 2024.

[3] L. D. M. Abreu, L. F. S. Carrete, M. Castanares, E. F. Damiani, J. F. Brancalion, and F. J. Barth, "Exploration and rescue of shipwreck survivors using reinforcement learning-empowered drone swarms," in *XXV Simpósio de Aplicações Operacionais em Áreas de Defesa*, 2023, pp. 64–69.

[4] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, vol. 299, p. 103535, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370221000862

[5] B. Ai, M. Jia, H. Xu, J. Xu, Z. Wen, B. Li, and D. Zhang, "Coverage path planning for maritime search and rescue using reinforcement learning," *Ocean Engineering*, vol. 241, p. 110098, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801821014220

[6] J. Wu, L. Cheng, S. Chu, and Y. Song, "An autonomous coverage path planning algorithm for maritime search and rescue of persons-in-water based on deep reinforcement learning," *Ocean Engineering*, vol. 291, p. 116403, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801823027877

[7] *International Aeronautical and Maritime Search and Rescue Manual*, International Maritime Organization and International Civil Aviation Organization, 2022. [Online]. Available: https://store.icao.int/

[8] D. W. Schuldt and J. A. Kurucar, "Maritime search and rescue via multiple coordinated uas," Defense Technical Information Center, Tech. Rep., 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:177273

[9] E. T. Alotaibi, S. S. Alqefari, and A. Koubaa, "Lsar: Multi-uav collaboration for search and rescue missions," *IEEE Access*, vol. 7, pp. 55 817–55 832, 2019.

[10] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[11] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015, cite arxiv:1509.06461Comment: AAAI 2016. [Online]. Available: http://arxiv.org/abs/1509.06461

[12] R. L. Falcão, J. C. C. de Oliveira, P. H. B. A. Andrade, R. R. Rodrigues, F. J. Barth, and J. F. B. Brancalion, "Dsse: An environment for simulation of reinforcement learning-empowered drone swarm maritime search and rescue missions," *Journal of Open Source Software*, vol. 9, no. 99, p. 6746, 2024. [Online]. Available: https://doi.org/10.21105/joss.06746

[13] K. Trummel and J. Weisinger, "The complexity of the optimal searcher path problem," *Operations Research*, vol. 34, no. 2, pp. 324–327, 1986.

[14] K.-F. Dagestad, J. Röhrs, Ø. Breivik, and B. Ådlandsvik, "Opendrift v1.0: a generic framework for trajectory modelling," *Geoscientific Model Development*, vol. 11, no. 4, pp. 1405–1420, 2018. [Online]. Available: https://gmd.copernicus.org/articles/11/1405/2018/

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[16] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning." International Conference on Machine Learning, 2018.