Predator-Prey Q-Learning Based Collaborative Coverage Path Planning for Swarm Robotics*

Michael Watson[1], Hanchi Ren[2], Farshad Arvin[1], and Junyan Hu[1]

[1] Department of Computer Science, Durham University, UK
{michael.watson, farshad.arvin, junyan.hu}@durham.ac.uk
[2] Department of Computer Science, Swansea University, UK
hanchi.ren@swansea.ac.uk

Abstract. Coverage Path Planning (CPP) is an effective approach to let intelligent robots cover an area by finding feasible paths through the environment. In this paper, we focus on using reinforcement learning to learn about a given environment and find the most efficient path that explores all target points. To overcome the limitations caused by standard Q-learning based CPP that often fall into a local optimum and may be in-efficient in large-scale environments, two methods of improvement are considered, i.e., the use of a robot swarm working towards the same goal and the augmenting of the Q-learning algorithm to include a predator-prey based reward system. Existing predator-prey based reward systems provide rewards the further away an agent is from its predator, the paper adapts this concept to work within a robot swarm by simulating each agent of the swarm as both predator and prey. Simulation case studies and comparisons with the standard Q-learning show that the proposed method has a superior coverage performance in complicated environments.

Keywords: Coverage Path Planning · Reinforcement Learning · Swarm Robotics.

1 Introduction

The autonomous coverage of unknown environments is a very hot and important research field in robotics. This task aims to completely cover the entirety of an environment which means the mobile agents must visit each location safely and efficiently [1]. Many coverage path planning (CPP) algorithms have been designed for mobile robots during the past decade, and they have already been applied to various real-world applications, such as precision agriculture [2, 3], autonomous vehicle navigation [4], automated harvesting [5,6], search and rescue [7,8]. Furthermore, CPP also plays an important role in some emerging research fields, e.g, a multi-arm manipulator could be used to interact with bee hives and follow the trajectory of the queen honey bee [9].

* This work was supported by EU H2020-FET-OPEN RoboRoyale project [grant number 964492].

There exist several different methods to achieve autonomous coverage, but this paper aims to improve upon one of the most known reinforcement learning algorithms, Q-learning [10, 11]. Q-learning is a long-standing algorithm and is popular due to its simple implementation allowing it to be easily adapted to several situations. However, this simpleness also leads to several limitations such as easily falling into a local optimum or struggling in more complex environments. These are the main limitations the solution aims to overcome; it is important to overcome these issues as in more complex use cases such as exploring dangerous terrain and environments it is vital that the agents can find optimal paths to save on resources. A shorter path means that more of an environment can be explored, both in total and within a given period. Furthermore, the algorithm must be good in more complex environments where there may exist obstacles that the agents must avoid as it is extremely unlikely that in real-world use cases, the environment will be unobstructed. Another key technology used throughout this paper is swarm robotics [12], swarms are a collection of multiple robots all working together towards the same goal. They are highly effective in exploration tasks as they multiply how much of the environment can be explored at once. Therefore, this paper focuses on implementing the upgraded algorithm into a swarm so that they can be used together to provide a superior method of exploitation.

The paper aims to answer the question: "Can you adapt Q-learning to be more effective in Coverage Path Planning problems?" To attempt to answer this question the paper provides a completed algorithm and explains the method used to obtain the results displayed at the end of the paper. It will explain the Q-learning method used as well as the adaptions made to improve it for the necessary purpose. A complete program was made to test the algorithms against each other simulating custom environments, as well as easily allowing differing swarm sizes. An implementation of a standard algorithm was also implemented to be compared. The adapted solution aims to improve upon the existing method in several ways, it aims to improve performance in complex scenarios in which obstacles exist increasing the difficulty of exploration. Another vital result is to ensure that full coverage always occurs within the given movement budget, it also aims to improve the efficiency of the algorithm by reducing the average number of steps needed to fully cover. Finally, the solution aims to be consistent so that it produces reliable results.

2 Related Work

There are multiple solutions to the CPP problem which can be categorized into offline and online algorithms [13]. Offline algorithms perform coverage in a static and known environment, whereas online algorithms work in dynamic environments. The online algorithm is preferred as most use cases will involve dynamic environments in which the agent will have to adapt [14]. However, even online algorithms have their limitations one being they require agents with sensors to learn about their surroundings. Another is that they lack the same efficiency

as offline algorithms which can find an optimal path and then stick to it. CPP algorithms can also take many different approaches [15] such as machine learning [16], greedy search [17], graph search [18], frontier-based exploration [19] as well as many others. Each method has its strengths and weaknesses and the best method to use often depends on the use case. For example, the greedy algorithm (Dijkstra's algorithm) is simple, easy to implement and generally fast, but it does not guarantee a global optimum. It is also an offline algorithm which means it cannot be used in dynamic environments.

Rerinforcement learning is the method which this paper focuses on. Piardi et al. [20] present a Q-learning algorithm which uses a grid-based map to optimize the CPP trajectory, this implementation has some limitations. Its simple nature means it doesn't work in environments with unknown obstacles. However, it does provide a good basis for improvement by adapting this algorithm with further methods. Zhou et al. [21] provide numerous improvements over a standard Q-learning algorithm to create an optimized algorithm, optimized Q-learning. It includes improvements such as novel Q-table initialization, a new action-selection policy, and a new reward function. The limitations of this paper however are that its reward function is still simplistic, and it also lacks integration for multiple robots. Puente-Castro et al. [22] make use of a swarm of agents to accomplish the task as a group. This method effectively demonstrates the usefulness of a swarm over an individual agent, it shows how using a swarm reduces the individual load of each agent whilst maintaining the overall result of the solution. The limitation of this method however is the reward structure, it is very simple and therefore leaves room for improvement. One method to improve the reward structure is through the use of a predator-prey based reward structure [23]. This reward structure combines three different reward functions to produce a reward for the agent. These rewards given are the predator-prey reward, a boundary reward, and a smoothness reward, in combination this provides the agent with a much more responsive representation of its environment. The limitation of this method comes from its lack of swarm implementation, it focuses on only one agent and with multiple agents, collisions can reduce the effectiveness of the rewards.

3 Methodology

This solution aims to create an efficient reinforcement learning algorithm for use in a coverage path planning scenario. Robot agents are used to complete the coverage. The multiple agents mean more of the environment can be explored concurrently; it also provides redundancies in the scenario that an agent is lost. However, there are some drawbacks in the way that the swarm interacts such as collisions as well as issues in how the swarm explores. The solution to this is to adapt a reinforcement learning algorithm around the swarm to overcome these issues. The reinforcement learning method Q-learning was selected as it is an easy-to-implement method which can be used easily in a swarm as if needed the agents can act individually.

3.1 Foundation Work

An implementation of Watkins Q-learning [10] was used for this solution. Q-learning is a simple reinforcement learning algorithm which makes use of a data structure called a Q-table and a function known as the Q-function. Q-learning works by initializing the agent's Q-table which holds a predicted reward for each state the agent can be in, and each action it can take from this state. The estimated rewards are then continually improved using a specialized Q-function (1) to produce rewards which represent how beneficial each action is.

The function takes the observed reward and the estimated future value and updates the value in the table according to the function. The function is formulated as follows:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max Q(s') - Q(s,a)], \tag{1}$$

where $Q(s,a)$ represents the current Q-value for the given state and action, s' refers to the next state (the state the current action will lead to), α=Learning Rate, r=reward and γ=Discount Factor. The function can be altered through the use of the parameters α, γ. These values can range from 0 to 1. The learning rate determines how much each learning increment will affect the Q-value, the greater the learning rate the faster the Q-values will converge. It is therefore important to have a learning rate small enough that it does not converge too early but also ensure that it is not so small it takes too long to converge. The discount factor determines how much importance is placed on the future reward. The greater the discount factor the more important the function finds the future reward; it is also important to balance this parameter so that the correct amount of weighting is placed on the potential future.

Another significant strategy implemented was the Epsilon-Greedy method. This is a method of introducing some randomness into the agent's selection of an action, which allows it to explore and find better paths and actions it can use. It works by using a value denoted as epsilon, the agent will then choose a random action with probability ϵ and the best action with probability $1 - \epsilon$. The value of epsilon then decays and reduces after each cycle so fewer random actions are taken. Performance can be heavily impacted by both the starting value and the decay rate of epsilon. If epsilon remains too high the algorithm will take too many random actions and never be able to converge on a suitable answer. However, if epsilon starts to low, no random actions are taken, and the agent gets stuck in a local optima never branching off to explore potentially better alternatives. A combination of these methods can be seen in Algorithm 1.

The basic reward structure used for standard Q-learning is simple, a positive reward is given the first time a location is seen, and a negative reward is given each time the agent moves. This negative reward acts as a movement penalty, it is given so that the agent will aim to cover the environment in a minimal number of steps as the more it moves the smaller the reward it gains.

$$r = \begin{cases} 1, & \textit{Undiscovered Location} \\ -0.05, & \textit{Movement Penalty.} \end{cases}$$

Algorithm 1 Q-learning

```
Initialize Q(s, a) as 0 ;
for each episode do
    Initialize s;
    for each step of the episode do
        Random p;
        if p < ϵ then
            Any action a;
        else
            a = max Q(s);
        end if
        Take action a, gather r, s′;
        Q(s, a) = Q(s, a) + α[r + γ max Q(s′) − Q(s, a)];
        s = s′;
    end for
    Until s is terminal.
end for
```

Backtracking is another method implemented into the solution to assist the agent in what to do when it sees no unvisited neighbours. If the agent reaches a position where it has no unvisited neighbours, it will begin to retrace its steps until it sees an unvisited location and it will then return to normal. The use of backtracking is to overcome situations where the agent becomes stuck in a loop moving between locations it has already seen until it terminates. Although backtracking is not efficient it is effective at ensuring the environment will always be completely covered and the agent will not get stuck. Coverage is guaranteed as anywhere the agent can reach it must have passed at some point and this method of backtracking ensures it is seen again and can be explored this time. This does create some limitations however as this form of backtracking is not very efficient because it retraces its steps. This occurs each time the agent starts backtracking. It could be improved by implementing a better method which remembers the location where the agent stopped backtracking and returned to exploration, this location could then be returned if the agent needs to backtrack again.

3.2 Swarm Structure

This solution makes use of multiple agents working together to create a collaborative swarm. The implementation of the swarm allows for the use of a changing number of agents, the swarm can contain any number of agents and still function correctly. Using more agents will allow for faster exploration, but it requires greater computational resources. The agents of the swarm all use the same algorithm however they use individual Q-tables so they can determine their best paths individually. This is necessary to take full advantage of the swarm as with a shared Q-table all agents would attempt to follow the same path which would negate the advantages of using a swarm in the first place. The use of one Q-table

would also lead to agents disrupting other agents' Q-values leading to worse results than using a single agent. There is also a central controller of the swarm which holds information about the swarm. This includes information about the locations of all swarm members, which allows any member to know the position of the other members. This can be used for both avoiding collisions as well as in the reward function (2). The controller also holds how much coverage the swarm has completed so it can monitor progress and tell the members to stop once coverage is complete.

There exist some potentially problematic interactions with agents in the swarm, one of which is how the agents avoid crashing. In the simulation, a crash occurs when two agents move into the same location. This is overcome by restricting the agent's movement to not allow actions which will result in the agents overlapping. This can reduce performance sometimes as agents may have to wait for other agents to move first before they can, but it is vital to ensure all agents remain safe.

Another problem is the fact that with multiple agents exploring different locations, the agents cannot know when they have collectively covered the entirety of the environment. Without some shared information of what each agent has covered the agents will run until they have covered the entire environment individually. This is overcome using a simulated server that holds the amount of the environment covered, this allows any agent to access the collective memory to share what they have covered and know when to stop. The server also holds the specific locations that have been explored, this avoids duplicate counting as well as allowing the correct rewards to be given to the agent for unexplored and explored locations.

3.3 Predator-Prey Improvement

Predator-Prey is a method which simulates the animal relationship between predator and prey. Specifically, it mimics the behaviour of the prey, in which they aim to get as far from the predator as quickly as possible. This method was adapted into a reinforcement learning scenario by Zhang et al. [23]. The method implemented in their paper focuses on using one agent to fully explore an environment. This is done by simulating a predator at one end of the environment so that the agent will move quickly away from it, the method is effective at getting the agent to move in straight lines as it is its optimal pathing for maximum distance. This behaviour is especially useful in a coverage scenario as moving in straight lines avoids any small pockets of unvisited locations being missed and needing to be found later. Once a boundary is reached the predator can then be moved to encourage the agent to move in a direction of choice, by doing this repeatedly the agent can sweep the entirety of the environment very effectively. The solution is built upon this method; however, it is altered to work in a swarm. The key difference is that instead of simulating a predator for the agent to run from all agents in the swarm simulate both predator and prey. An individual agent aims to move as far as possible from the other agents in the swarm, simulating the prey with every other member of the swarm acting as its

predator. This method is effective when the entire swarm begins in the same region of the environment as it means the swarm spreads out which allows for different regions of the environment to be explored faster. A reward is given depending on how far the agent is from its closest predator (2). The reward for moving to neighbour x is formulated as follows:

$$r_p(x) = \frac{D(x) - D_{min}(x_n)}{D_{max}(x_n) - D_{min}(X_n)}, \tag{2}$$

where $D(x)$ gives the distance from x to its nearest predator. $D_{max}(x_n)$ gives the maximum distance between one of x's neighbours and any predator, and $D_{min}(x_n)$ gives the minimum distance between one of x's neighbours and any predator.

Two further functions are also used which give rewards based on the smoothness of the path travelled and the boundary of the next state. The smoothness reward gives a reward based on the direction of the agent's travel (3), if it moves in the same direction multiple times it receives a greater reward. This is done to entice the agent into moving in straight lines, as well as discourage moving back along its path when unnecessary. The reward for moving to neighbour x is formulated as follows:

$$r_s(s) = \begin{cases} \angle x_{k-1}x_k x = 0°, & reward = 1 \\ \angle x_{k-1}x_k x = 90°, & reward = 0.5 \\ \angle x_{k-1}x_k x = 180°, & reward = 0 \end{cases} \tag{3}$$

where x_{k-1} and x_k are the positions covered at $k-1$ and k, respectively. The reward is given based on the angle between the vectors $(x_{k-1} - x_k)$ and $(x_k - x)$.

The boundary reward gives a reward based on the number of unvisited neighbours a given location has (4). The fewer the neighbours the greater the reward. This helps the agent to move into locations which may otherwise get cut off and left behind. The reward for moving to neighbour x is formulated as follows:

$$r_b(x) = \frac{N_{max} - x_N}{N_{max}}, \tag{4}$$

where N_{max} is the total number of neighbours and x_N is the number of unvisited neighbours x has.

The rewards are then combined to give a complete reward. They are combined with different weights to manipulate the importance of each individual reward. It is formulated as follows:

$$r(x) = w_p(r_p(x)) + w_s(r_s(x)) + w_b(r_b(x)), \tag{5}$$

where w_p, w_s and w_b represent the weights for the predation reward, smoothness reward and boundary reward, respectively. $r(x)$ is the final reward which is returned to the agent. The weights can be changed to differing values to change how rewards are given to the agent. A larger weight means more emphasis is placed on that reward; this allows for the algorithm to work in different ways

such as focusing on spreading out as quickly as possible or trying to move in only straight lines. Using powerful values for the weights which are much higher than the others can lead to a worse performance as it overpowers the other reward functions and only one is truly considered.

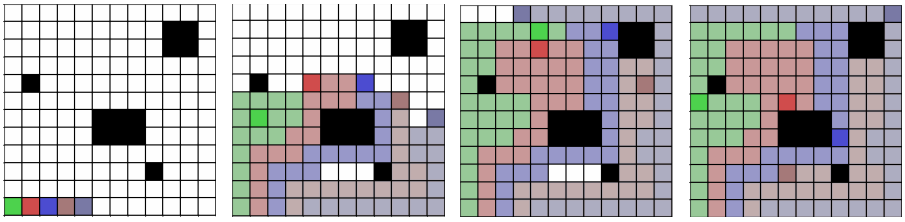
The pseudocode for the complete algorithm can be seen in Algorithm 2. This combines all improvements including the use of a swarm, the use of backtracking and the use of the adapted reward function.

Algorithm 2 Improved Predator-Prey Swarm Q-learning

```
Initialize Swarm;
for each agent in Swarm do
    Initialize Q(s, a) as 0;
    Initialize Exploration Count;
end for
for each episode do
    for each agent in Swarm do
        Initialize s;
    end for
    for each step of the episode do
        for each agent in Swarm do
            Random p;
            if p < ε then
                Any action a;
            else
                if there is an unvisited location then
                    a = max Q(s)
                else
                    a = Previous action
                end if
            end if
            Take action a, gather s', gathering r using (5);
            Q(s, a) = Q(s, a) + α[r + γ max Q(s') − Q(s, a)];
            if s' is unvisited then
                Exploration Count += 1;
            end if
            s = s';
        end for
        Until s is terminal.
    end for
end for
```

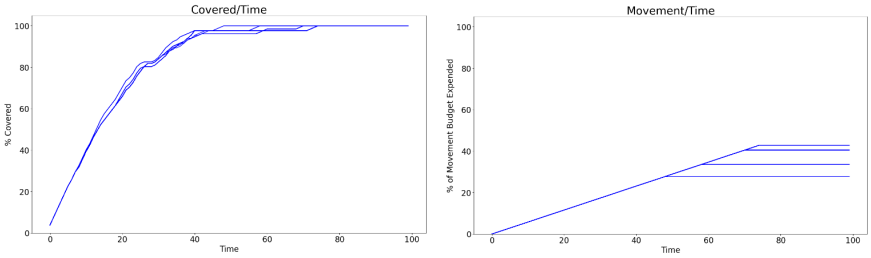
4 Results and Evaluation

The results are simulated in a custom-made environment, which is a square grid that can vary in size. Throughout the testing, the weights used in (5) are $w_p = 2$,



(a) (b) (c) (d)

Fig. 1. The trajectories of 5 agents using the proposed method on an environment size 12 with random placed obstacles at (a) 0 seconds, (b) 14 seconds, (c) 28 seconds, and (d) 42 seconds.



(a) (b)

Fig. 2. The (a) Coverage/Time and (b) Movement/Time of the proposed method with swarm size 5 on an environment size 12 with random placed obstacles.

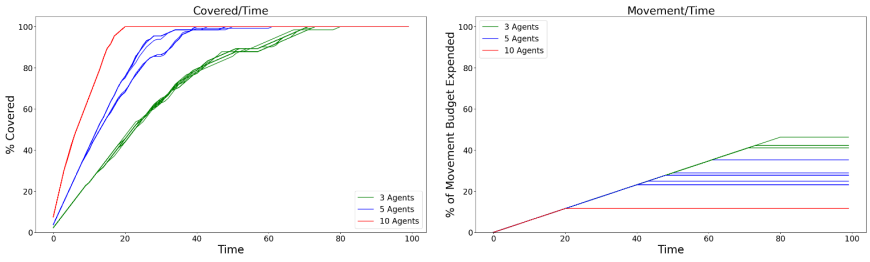
$w_s = 1$, $w_b = 1.5$. The learning rate was 0.001, the discount factor was 0.9, the starting epsilon value was 0.2 and there were 1000 learning episodes carried out. There was a movement budget set based on the environment size, this was the maximum number of steps allowed for each coverage attempt.

4.1 Coverage Performance in Cluttered Environments

The following plots illustrate an example of coverage in an environment of size 12 with a swarm size of 5 agents. Some obstacles are randomly placed in the environment. These results will show how the agents navigate in more complex environments.

In Fig. 1, the trajectories of the agents can be seen. It demonstrates how each agent explores their section of the environment and also demonstrates the predator-prey methodology with the agents moving apart to explore different edges of the environment.

Figures 2(a) and 2(b) show that even when obstacles are added the algorithm maintains its consistency with tight groupings of the simulations. It also shows that even with obstacles the algorithm always ensures that the environment is covered.



(a) (b)

Fig. 3. The (a) Coverage/Time and (b) Movement/Time of the proposed method with different swarm sizes.

4.2 Coverage with Different Swarm Sizes

To test the performance of our strategy when using different number of robots, the next plots show the coverage in an environment of size 12 with differing swarm sizes, i.e., 3, 5, and 10 agents all using the same predator-prey learning.

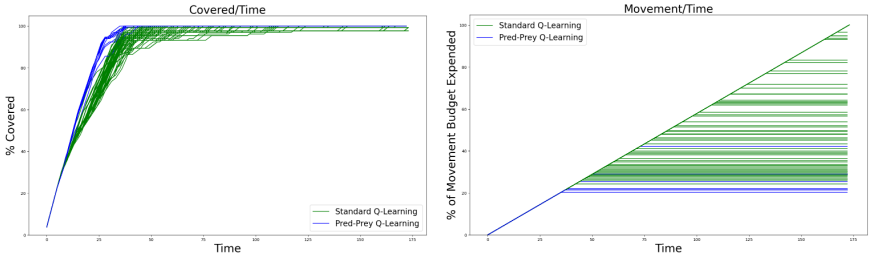
Figures 3(a) and 3(b) show the comparisons of different swarm sizes. This provides evidence of the effectiveness of the swarm as by adding additional agents the speed of the solution is drastically improved. These plots show that reliable results can be provided by any size swarm however the larger the swarm the better the results. This is useful to know as depending on the use case there may be different restrictions on the number of agents a swarm can have.

4.3 Comparison with Standard Q-Learning

The following graphs demonstrate the comparison between the standard Q-learning [10] and the version created in this paper. Each algorithm was run for 50 loops and then the average results of the simulations were plotted so they could be compared. A swarm of 5 agents in the same environment was used for both algorithms to ensure a fair comparison.

It can be seen in Figure 4(a) and Figure 4(b) that the predator-prey algorithm performs very well ensuring that the environment is always covered. Whereas with the standard Q-learning, there are some cases in which coverage fails and the entire movement budget is expended. This can be seen in Figure 4(a), not every line (simulation) reaches 100% before the end of the time. This was an important criterion to meet for the solution ensuring that the environment can always be completely covered in a small number of steps, ensuring that the algorithm was efficient. This could potentially be further improved by improving the backtracking part of the algorithm.

Figure 4(a) and Figure 4(b) also show the improved algorithm is much more consistent providing similar results for all simulation loops, whereas the standard algorithm results vary drastically in its results. This was also important to ensure that the swarm would act predictably so that the user could know how the swarm



(a) (b)

Fig. 4. Comparison between Predator-Prey and Standard algorithms with (a) Coverage/Time and (b) Movement/Time.

would perform before using it, this is a very important factor in some potential use cases.

5 Conclusion

This paper utilizes a robot swarm using an adapted Q-learning algorithm to learn about their environment and plan an efficient path to explore it. The solution managed to successfully extend a standard predator-prey Q-learning to allow agents of the swarm to simulate both prey and predator, this led to a more effective coverage as agents aimed to stay apart and therefore covered different sections of the environment. The solution was much more efficient than a standard Q-learning swarm, ensuring that the environment was fully covered in fewer steps. The efficiency and effectiveness were then validated by various simulation case studies.

References

1. Cao, Z.L., Huang, Y., Hall, E.L.: Region filling operations with random obstacle avoidance for mobile robots. Journal of Robotic systems **5**(2), 87–102 (1988)
2. Höffmann, M., Clemens, J., Stronzek-Pfeifer, D., Simonelli, R., Serov, A., Schettino, S., Runge, M., Schill, K., Büskens, C.: Coverage path planning and precise localization for autonomous lawn mowers. In: 2022 Sixth IEEE International Conference on Robotic Computing (IRC), pp. 238–242. IEEE (2022)
3. Huang, K.C., Lian, F.L., Chen, C.T., Wu, C.H., Chen, C.C.: A novel solution with rapid voronoi-based coverage path planning in irregular environment for robotic mowing systems. International Journal of Intelligent Robotics and Applications **5**(4), 558–575 (2021)
4. Xie, S., Hu, J., Ding, Z., Arvin, F.: Cooperative adaptive cruise control for connected autonomous vehicles using spring damping energy model. IEEE Transactions on Vehicular Technology **72**(3), 2974–2987 (2023)
5. Hameed, I.A.: Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain. Journal of Intelligent & Robotic Systems **74**(3), 965–983 (2014)

6. Wang, L., Wang, Z., Liu, M., Ying, Z., Xu, N., Meng, Q.: Full coverage path planning methods of harvesting robot with multi-objective constraints. Journal of Intelligent & Robotic Systems **106**(1), 17 (2022)
7. Rekabi-Bana, F., Hu, J., Krajník, T., Arvin, F.: Unified robust path planning and optimal trajectory generation for efficient 3d area coverage of quadrotor uavs. IEEE Transactions on Intelligent Transportation Systems **25**(3), 2492–2507 (2024)
8. Wu, K., Hu, J., Li, Z., Ding, Z., Arvin, F.: Distributed collision-free bearing coordination of multi-uav systems with actuator faults and time delays. IEEE Transactions on Intelligent Transportation Systems (2024)
9. Rekabi-Bana, F., Stefanec, M., Ulrich, J., et al.: Mechatronic design for multi robots-insect swarms interactions. In: 2023 IEEE International Conference on Mechatronics (ICM), pp. 1–6. IEEE (2023)
10. Watkins, C.J.C.H.: Learning from delayed rewards (1989)
11. Clifton, J., Laber, E.: Q-learning: Theory and applications. Annual Review of Statistics and Its Application **7**, 279–301 (2020)
12. Navarro, I., Matía, F.: An introduction to swarm robotics. Isrn robotics **2013**, 1–10 (2013)
13. Galceran, E., Carreras, M.: A survey on coverage path planning for robotics. Robotics and Autonomous systems **61**(12), 1258–1276 (2013)
14. Champagnie, K., Arvin, F., Hu, J.: Decentralized multi-agent coverage path planning with greedy entropy maximization. In: 2024 IEEE International Conference on Industrial Technology, pp. 1–6 (2024)
15. Tan, C.S., Mohd-Mokhtar, R., Arshad, M.R.: A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms. IEEE Access **9**, 119,310–119,342 (2021)
16. Xing, B., Wang, X., Yang, L., Liu, Z., Wu, Q.: An algorithm of complete coverage path planning for unmanned surface vehicle based on reinforcement learning. Journal of Marine Science and Engineering **11**(3), 645 (2023)
17. Jia, Y., Zhou, S., Zeng, Q., Li, C., Chen, D., Zhang, K., Liu, L., Chen, Z.: The uav path coverage algorithm based on the greedy strategy and ant colony optimization. Electronics **11**(17), 2667 (2022)
18. Nasirian, B., Mehrandezh, M., Janabi-Sharifi, F.: Efficient coverage path planning for mobile disinfecting robots using graph-based representation of environment. Frontiers in Robotics and AI **8**, 624,333 (2021)
19. Zhou, B., Zhang, Y., Chen, X., Shen, S.: Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning. IEEE Robotics and Automation Letters **6**(2), 779–786 (2021)
20. Piardi, L., Lima, J., Pereira, A.I., Costa, P.: Coverage path planning optimization based on q-learning algorithm. In: AIP Conference Proceedings, vol. 2116. AIP Publishing (2019)
21. Zhou, Q., Lian, Y., Wu, J., Zhu, M., Wang, H., Cao, J.: An optimized q-learning algorithm for mobile robot local path planning. Knowledge-Based Systems **286**, 111,400 (2024)
22. Puente-Castro, A., Rivero, D., Pazos, A., Fernandez-Blanco, E.: Uav swarm path planning with reinforcement learning for field prospecting. Applied Intelligence **52**(12), 14,101–14,118 (2022)
23. Zhang, M., Cai, W., Pang, L.: Predator-prey reward based q-learning coverage path planning for mobile robot. IEEE Access **11**, 29,673–29,683 (2023)