Distributed Multi Unmanned Aerial Vehicles Dynamic Area Coverage Algorithms Based on Reinforcement Learning

Candidate Number: ZLIAO83
MSc in Robotics and Computation

Supervisor: Yuanchang Liu

Department of Computer Science

University College London

September 23, 2020

Abstract

System of multi unmanned aerial vehicles (UAVs) [1] has been widely deployed and utilized for various types of missions such as maintenance, inspection involving detection [2], monitoring [3] and searching [4], [5] without human intervention. Making improvements on the efficiency of covering the entire area of interest without collision is one of the main focuses for multi UAVs dynamic area coverage. In this paper, we propose two distributed dynamic area coverage algorithms based on reinforcement learning (RL) and a γ-information map with different communication distance according to [6]. The γ-information map is obtained by decomposing the free target area into a grid map of discrete γ points to discretize the continuous coverage process for the use of RL. For the first proposed modified cooperative Q-Learning (MC-Q) algorithm, each UAV learns using the restricted state-action values and back-propagation of rewards cooperatively to find the close to optimal coverage strategy. For the second proposed modified independent Q-Learning (MI-Q) algorithm, every UAV learns the close to optimal coverage strategy independently using the whole traversal reward from the best experiences. Through simulations both algorithms ensure complete coverage and the required time of MI-Q is closer to the optimal but with smaller learning effect than MC-Q.

Contents

Chapter 1

Introduction

1.1 Background

Dynamic area coverage has been widely used in military and civil fields to perform a wide variety of tasks in different types of environments. Some of these environments are unknown or dynamically varying where there possibly exists potential danger for direct human interaction and the difficulty of task execution is increased due to its dynamic feature compared to the static environment. To achieve the process of dynamic area coverage, a mobile sensor network (MSN) is established by a team of agents with wireless communication. These agents can be UAVs, unmanned ground vehicles (UGVs) [7] or a combination of both UAVs and UGVs. For solving the problem of dynamic area coverage, MSN demonstrates better flexibility because of its mobility than the static sensor network (SSN) where each sensor as an agent is stationary after deployment [8], [9]. Furthermore, the number of required agents in MSN is less than the one of the SSN for the dynamic area coverage of large target area. The team of agents in MSN work in a distributed and cooperative manner with communication and information sharing to maximize the efficiency of dynamic area coverage. The cooperation in the team significantly improves the performance of all agents compared to the purely independent control of each agent.

1.2 Related Work

The problem of dynamic area coverage is similar and related to coverage path planning (CPP) problem where the objective is to find a path or paths that an agent or agents can follow to achieve no-hole coverage for the whole target area. Some state-of-the-art existing researches present algorithms for single agent CPP. Theile et al. [10] proposed an algorithm based on deep RL [11] which enables a single UAV to fully cover the area of interest at the same time balancing limited power budget. It utilized 2 layers of convolutional neural networks (CNNs) [12] with zero and one padding to extract spatial relationship from the state input represented as a 5-Channel grid map containing information of positions, coverage and different types of zones within the target area. Then a double deep Q-Network (DDQN) using an experience replay buffer [13] to stablize the training process and trick of DDQN to address the problem of overestimation of Q values [14] is trained to form control policy for the UAV to achieve the goal of CPP efficiently. With limited power budget and the proper design of reward function, the UAV is trained to complete the mission as efficient as possible with close to minimum repeated traversal, in which the coverage process is similar to the one of dynamic area coverage. Another similar algorithm for single agent to complete CPP problem (CCPP) for Tetromino based cleaning and maintenance robot proposed by Anirudh et al. [15] is developed using Actor Critic Experience Replay (ACER) [16] with Long Short Term Memory (LSTM) [17]. It uses rgb images as state input and one of the policy gradient RL methods (ACER) with the actor network designed with CNN, fully connected layer and LSTM to achieve the CCPP. Instead of using deep RL methods to solve CPP, Luis et al. [18] proposed an algorithm to solve CPP in the environment with known obstacles based on Q-learning for single agent. It enables the agent to learn by updating a Q-table with the rule of Q-learning update where learning rate is equal to 1 that the agent forgets the previous experience completely. Compared to previously mentioned methods using deep RL, the dimension of state space from this method is significantly smaller but it could possibly take longer time to train since all entries of Q-table are required to learn and experiences are not used efficiently as by deep RL methods where neural networks could recognize the pattern of relationship between state-action pair and the corresponding value efficiently. However, since positions of obstacles are known and only entries in Q-table corresponding to possible actions available at the current state are updated, the convergence of Q-learning is accelerated significantly. All of the methods mentioned above do not consider the kinematic motion of UAV in real scenario where for the UAV to reach from one cell to another adjacent cell along a trajectory it is required to control the acceleration and moving direction of the UAV as continuous states. In addition, if we simply extend any of the single agent algorithm to multi agent algorithm using independent Q-learning (IQL) introduced in [19], then the issue of instability of environment will be raised since each agent's policy can be treated as part of the environment for the case of multi-agent setting and it is constantly changing during training and it will potentially cause the instability of learning and the divergence of the method. Therefore there is a need of discretizing continuous states of the UAV into discrete states to be used in RL and find a way to stabilize the environment for the multi-agent setting to solve the dynamic area coverage problem in this paper.

In terms of applying multi-agent RL in the problem which is field coverage using drones similar to CPP, Huy et al. [20] proposed a cooperative and distributed RL algorithm making use of the team's joint state, action and team's global reward to address the issue of instability of the environment and speed up the convergence of the learning process. This method can be in a way considered as transforming the multi-agent problem into single agent problem since every agent in the team has the same state and global reward and the reward received by each agent will be determined by their joint action. To choose the joint action, the Correlated equilibrium (CE) [21] is solved with the help of linear programming (LP) [22]. And the huge state space is reduced by using efficient function approximation, which is similar to using neural network in deep RL where mapping between state-action pair and value is parameterized by coefficients of the network.

At present, there exists some multi-agent methods that are not based on RL for solving CPP problems and result in trajectories for different agents that are non overlapping, which can be used for the dynamic area coverage with trajectory planning and motion model of agents. Xudong Wang and Vassilis L. Syrmos [23] proposed a multi-agent CPP algorithm for the automated inspection of an unknown

2D environment. Under the multi-agent setting, the 2D environment is divided into several sub-areas of the same or close to the same area based on the number of agents. Each sub-area is decomposed into non overlapping triangle simplices using the principle of Delaunay triangulation. Then the circumcenter of each simplice is calculated and a path is planned using the Lin-Kernighan (LK) heuristic approach [24]. This method generates non-optimal paths within each sub-areas since based on results there are usually some repeated traversals along the path within each region and the edges along the path are not aligned as straight lines but with angles. Another similar method proposed by Dario et al. [25] also follows similar structure of the work flow while there are known obstacles. It first decomposes the target area using Morse decomposition [26] and plans parallel and perpendicular trajectories with respect to every edge of subregion for each subregion following back-and-forth motion. Then flock management is used for division of the coverage tasks among the agents. This method also results in suboptimal solution to the problem since there are overlapping trajectories among different agents and exists some path that are neither horizontal nor vertical between subregions in the target area. Another similar method aiming to determine the global optimum time required to cover an entire environment is proposed by Adiyabaatar et al. [27]. In this method, the decomposition of target area is performed by boustrophedon decomposition [28] with a vertical line referred to as slice and cells (subregions) are formed via a sequence of open and close events. Then a flow network implemented using modern adjacency lists is used to indicate the adjacency of the cells. For each cell, a trajectory is planned following back-and-forth motion with main path along the long axis of the cell to reduce the number of turns. A path between two adjacent cells is determined by searching the closest couple points between one of the four corners of the current cell and one of the four corners of the next cell. Using this method trajectories within each cell consist of horizontal and vertical paths with reduced number of turns. However, it does not explicitly handle the cell allocation to different agents and by following the trajectories connecting cells allocated to each agent there seems to be inevitably repeated traversals of agents for the coverage of the entire environment and hence this method results in suboptimal solution to the complete coverage problem.

For dynamic area coverage problem to be addressed in this paper, several algorithms based on the Flocking algorithm [29], Anti-Flocking algorithm [30] and Semi-Flocking algorithm [31] based on the Flocking algorithm are proposed. Wanmai et al. [32] proposed a fully distributed semi-flocking algorithm with each agent using the available information from the operating environment by information exchanges among nearby agents. Each agent maintains a local information map recording the latest visited time of each cell with information exchange with neighbours and information list for multiple targets tracking. The semi-flocking algorithm is developed using Anti-Flocking algorithm for searching mode and Flocking algorithm for tracking mode. Nuwan et al. [33] proposed a distributed anti-flocking algorithm to maximize the coverage of the network of agents. For this method, each agent also maintains a local information map with information exchange with its neighbours having different update rules using the visited time information. Similarly, each agent selects its target cell based on a benefit function using the information from the local information map and other information from its neighbours. The difference between this method and the previous one is that for this one anti-flocking algorithm is used for both searching and tracking to maintain high area coverage. Compared to the previously stated algorithms for multi-agent CPP, both of these two algorithms make use of the updated information from the entire target area as the coverage process progresses and each agent exchanges information among neighbours. With the global information sharing it can significantly facilitate finding the optimal solution to the dynamic area coverage problem. However, even both of algorithms results in the complete area coverage, based on the experimental results both of them gives non-optimal solution where there exists lots of repeated traversal in the resulting trajectories and for the one with semi-flocking algorithm using flocking algorithm for tracking there tends to be more of that.

To take advantage of both Flocking algorithm and multi-agent RL, in this paper we proposed two distributed multi-UAVs dynamic area coverage algorithms based on [6] with improvement in terms of both performance and speed of convergence. Both of proposed algorithms ensures no-hole coverage while one of them demonstrates stronger learning effect and the other one results in solution closer to the optimal one.

1.3 Contributions

Algorithms described in this paper are proposed by modifying the one from [6]. The main contributions to solve the dynamic area coverage problem with modifications are summarized below:

The first contribution is for the multi-agents system motion model, the update of the position, speed in the particle motion model of each agent is implemented by integration in a discrete fashion.

The second contribution is the proposed action selection of each agent in the special cases when two agents are moving towards the same target either knowing or temporarily without knowing each other's target.

The third contribution is the first proposed MC-Q algorithm with modified update rule using restricted Q-table, back-propagation of the reward for the whole traversal process and the modification for cooperative Q-learning. This algorithm demonstrates strong learning effect since after training the agents using this algorithm the average coverage time decreases more compared to the second proposed MI-Q algorithm in this paper while the performance of the second one is better and closer to the optimal. It is interesting to notice that the algorithm proposed in the referred paper does not enable the agents to learn at all and a discussion on this is included in section 5.1.

The final contribution is the second proposed algorithm with prioritized actions for action selection and based on the framework of independent Q-learning. The performance of the agents after training with this algorithm is better than the one in the referred paper and close to the optimal. The speed of convergence is significantly faster than the one from the referred paper.

Chapter 2

Reinforcement Learning

Reinforcement Learning (RL) is one of the machine learning paradigms concerned with how agents could figure out a way of making decisions in an environment such that they could maximize the expected sum of rewards. It enables agents to find out how to behave in the environment in an optimal way by having agents interact with the environment and receiving observations and reward indicating the quality (how good or bad) of taking each particular action from the environment.Figure 2.1 demonstrates the process of how an agent interacts with an environment at time step t.

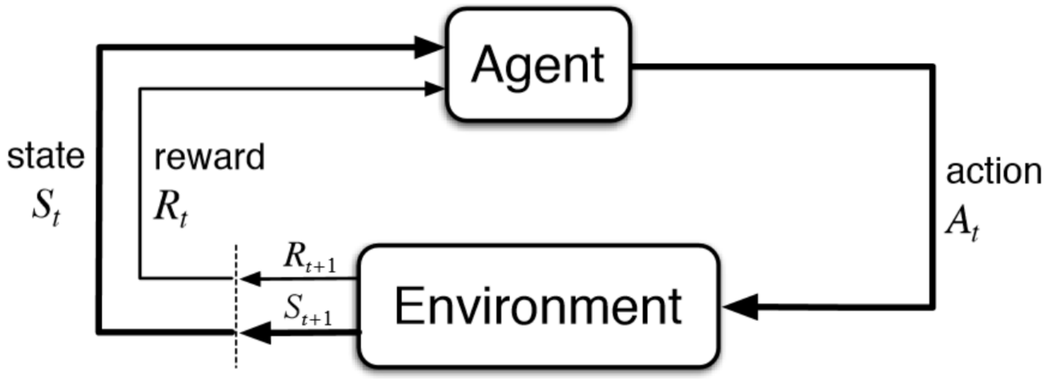


Figure 2.1: Illustration of RL process from [34]

In the above figure, an agent in the state S_t takes an action A_t in the environment and then it receives a reward R_{t+1} and the next state S_{t+1} from the environment. The state is a representation of what the agent observes from the environment at a particular time step. The environment can be partially observable where only part of the environment information is available to the agent or fully observable where the entire environment can be observed by the agent. The next state then becomes the current state at the next time step.

In this chapter, the model of RL is introduced and an model-free RL algorithm is described. These two sections illustrate the model (environment) and the learning algorithm that our proposed algorithms are based on.

2.1 Markov Decision Process

A Markov decision process (MDP) [35] is a discrete-time stochastic control process that is used to describe an environment for reinforcement learning, where the environment is fully observable. It provides a framework for modelling agents' learning of the optimal solution via interaction with the environment. More specifically, an MDP is Markov Reward Process with actions describing an environment where all states are Markov that the current state contains all the useful information from what it has experienced so far since the initial state and the previous states as histories can be ignored. The Markov property of states in an MDP can be described mathematically as follow:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, S_2, ...S_t] \tag{2.1}$$

In (2.1) $P[...|...]$ stands for conditional probability and it can be interpreted as based on the current or previous state(s) what the probability of the next state is. In other words, given the current state the future state is independent of the previous state(s).

An MDP is described by the tuple (S, A, P, R, γ), where S is the state space consisting of all possible states that can be observed from the environment, A is the action space with a finite set of possible actions taken by the agent(s), P is the state transition probability function that given the current state S_t and the action taken A_t at time step t, the probability of landing in the next state S_{t+1}:

$$P_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \tag{2.2}$$

R is the reward function giving the expected one step reward at time step t:

$$R_{ss'}^{a} = \mathbb{E}[S_{t+1} = R_{t+1} | S_t = s, A_t = a] \tag{2.3}$$

γ is a discount factor discounting the future rewards from the current state for the computation of the cumulative reward where $\gamma \in [0,1]$.

As stated in the first paragraph in this section, an MDP is a framing concerned with the learning of agents by interaction with environment to obtain the optimal strategy(actions) aiming to maximize the expected cumulative reward. In order to achieve the goal, Bellman equation [36] is used for this case. First, a function which is called the value function $v(s)$ used to represent the expected cumulative reward from the current state S_t is defined as follow:

$$v(s) = \mathbb{E}[G_t | S_t = s] \tag{2.4}$$

where G_t is the cumulative reward from time step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1} \tag{2.5}$$

With Bellman equation, the value function will be decomposed into an immediate reward R_{t+1} which is the immediate reward received by taking action at the current state and the discounted cumulative reward starting from the next state:

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \tag{2.6}$$

Hence, using Bellman equation with all the elements in the tuple describing an MDP, the value function of the state s is

$$v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s') \tag{2.7}$$

where $P_{ss'}$ is the state transition probability without action:

$$P_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s] \tag{2.8}$$

The Bellman function above can be solved by Dynamic Programming [37] [38], Monte-Carlo evaluation [39], and Temporal-Difference learning [40]. Then given a policy π which is a distribution over actions given states, which defines what the action will be chosen by the agent based on its current state:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{2.9}$$

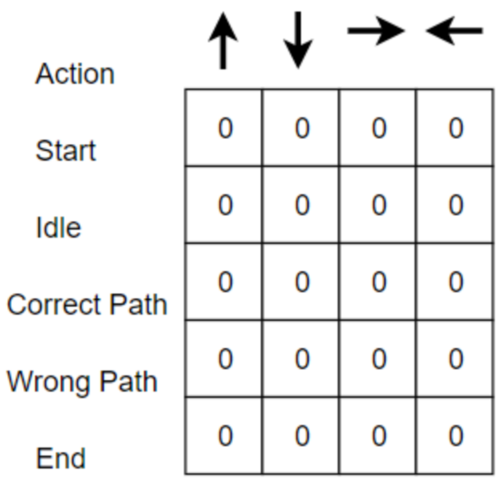
the state value function under policy π for MDP is defined as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{k+t+1}|S_t = s] \tag{2.10}$$

2.2 Q-Learning

Q-learning is a model-free [41] and value-based RL algorithm which enables an agent to learn the optimal policy from experiences collected via interaction with an environment by updating the mappings between state-action pair and the expected cumulative discounted reward using Bellman equation. For a model-free RL algorithm, the state transition probability and the reward function associated with the MDP as described in the previous section 2.1 are not known and used. Using this type of algorithm, the agent learns the best action to take in different states by exploration and exploitation (trial-and-error) in the environment. Value-based RL algorithms update the state-value function or action-value function for the case of Q-learning using Bellman equation. In addition, Q-learning is an off-policy learner which allows the agent to learn from other agents' policies other than the one carried by the agent itself. With Q-learning, the agent usually interacts with the environment using both a certain type of random policy and its own policy to learn. For deep Q-learning, the agent learns from the previous experiences stored in an experience replay buffer collected using the behaviour (old) policy and the new experiences obtained using the updated policy. In this case, the behaviour policy can be considered as other agents' policies.

In terms of the Q-learning process, it iteratively updates the Q-function which is a mapping between state-action pair and the expected cumulative discounted reward inside a Q-table where rows of the table are a set of possible states of the agent and the columns are a set of possible actions for the agent. Figure 2.2 shows an example of Q-table for a path following problem:



Action	↑	↓	→	←
Start	0	0	0	0
Idle	0	0	0	0
Correct Path	0	0	0	0
Wrong Path	0	0	0	0
End	0	0	0	0

Figure 2.2: Initial Q-table from [42]

More specifically, the Q-function is updated using Bellman equation and it takes the state and the action as inputs:

$$Q_\pi(s,a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{k+t+1} | S_t = s, A_t = a] \tag{2.11}$$

And the update rule of Q-learning using Temporal Differences(TD) with one-step reward to stabilize the learning process is:

$$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \max Q(s',a') - Q(s,a)] \tag{2.12}$$

where $Q(s,a)$ is the current Q-function value, α is the learning rate which determines how much the agent forgets from what it has learned previously, $R(s,a)$ is the one-step reward received by taking an action from a state, $\mathrm{maxQ(s',a')}$ is the maximum expected cumulative discounted reward from the next state among all possible actions, γ is the discount factor.

The reward is collected by having the agent choose and perform using a mixture of random policy and its updated policy with greedy strategy. A typical strategy is the Epsilon greedy strategy [43] which enables the agent to explore the environment and exploit what it has learned before which is the Q-function values stored in the Q-table. The entire Q-learning process can be summarized in figure 2.3:

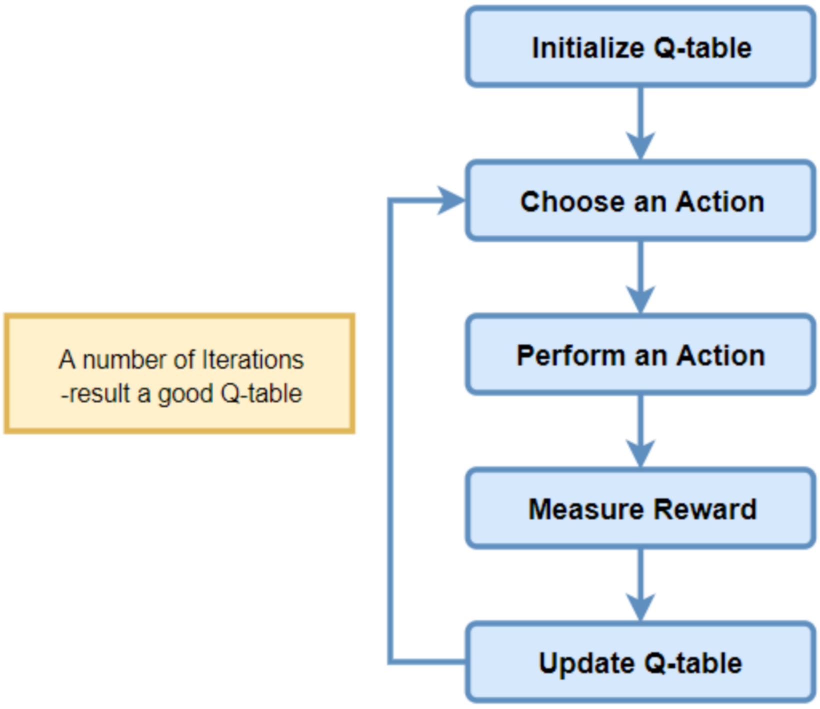


Figure 2.3: Q-learning Process from [42]

Chapter 3

Problem Formulation

In this chapter, the framework of multi-agent system for dynamic area coverage is described, a multi-agent motion model built based on the flocking algorithm is introduced and the definition of the γ-information map is given.

3.1 System Framework

In this paper, a multi-agent system (MAS) for dynamic area coverage is built as a control system consisting of both continuous and discrete state and action. The system framework is shown in figure 3.1:

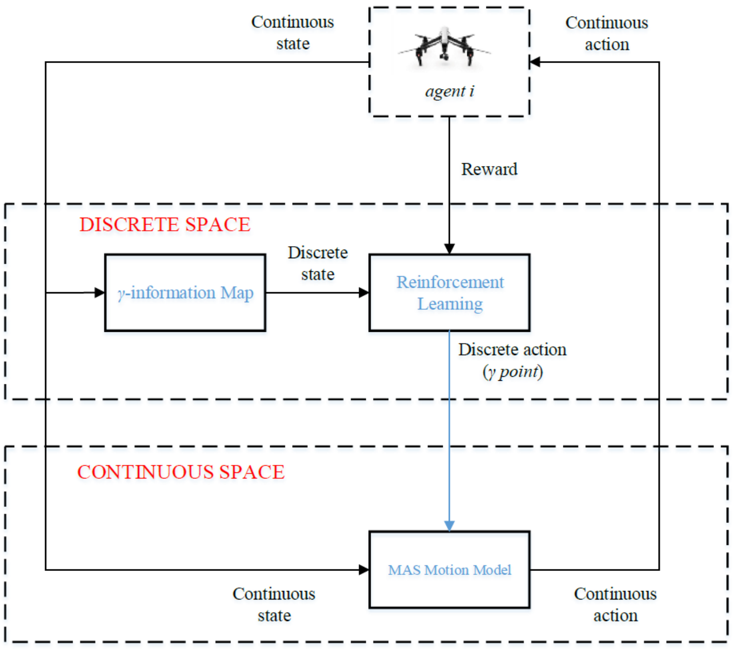


Figure 3.1: MAS for dynamic area coverage from [6]

Based on figure 3.1, part of the continuous state of each agent is transformed into discrete state represented using γ-information map where the 2D coordinates of the current target position of each agent is described by the 2D γ-point and then transformed into an index as part of the discrete state. The local γ-information map of each agent is part of the discrete state. The agent chooses the best action and the next target γ-point based on the discrete state which is either the global or local observation from the target area depending on the communication distance with neighbours using proposed algorithms modified based on RL. The 2D acceleration, speed and position of each agent are updated according to their values from the last time step and the next target γ-point and the continuous action outputted from the MAS motion model is the acceleration for the current time step to drive the agent towards the target without colliding with other agents. The reward signal is received by the agent for the agent to learn when the next target γ-point is reached or the entire target area is fully covered.

3.2 Multi-agent Systems Motion Model

In this paper, the multi-agent systems (MAS) motion model introduced in [6] is used with integration implemented additionally for updating the position, velocity and acceleration which is the control input as the continuous state for each agent where p_i, v_i and $u_i \in R^2$ represent each of those elements respectively for the ith agent. Each agent follows the following equation of motion:

$$\begin{cases} \dot{p}_i = v_i, \\ \dot{v}_i = u_i, \quad i = 1, 2, \cdots, N \end{cases} \tag{3.1}$$

where N is the number of agents (UAVs) in the MAS, and the set of agents is denoted as $V = \{1, 2, \cdots, N\}$.

The update of position and velocity elements of the continuous state for agent i is performed using the integration in a discrete fashion. It assumes constant acceleration during the time period between two consecutive time steps. The following equation for position and velocity update are satisfied by each agent:

$$\begin{cases} p_i^c = p_i^p + \frac{1}{2}(v_i^p + v_i^c)\Delta t, \\ v_i^c = v_i^p + u_i \Delta t, \qquad i = 1,2,\cdots,N \end{cases} \tag{3.2}$$

where p_i^c, p_i^p, v_i^c, v_i^p and Δt are current position, previous position, current velocity, previous velocity and the time step period respectively.

The set of neighbours of agent i is defined based on the communication distance r_c within which agents can share information with each other as follows:

$$N_i = \{j \in V : \|p_j - p_i\| \leq r_c, j \neq i\} \tag{3.3}$$

During the dynamic area coverage process, it is required for each agent to reach its planned target position without colliding with other agents. As stated in [6], the agent and the target position are taken as α-agent and γ-agent respectively. There is no need for agents to maintain a certain typology of flock as described in [29] since one can imagine flocking will significantly reduce the efficiency of the coverage process. Based on that, only a repulsive force between agents is required to avoid collision between them and the formula for computing the force between agent i and agent j is defined as follows from [6]:

$$f_i^\alpha = \phi(\|p_j - p_i\|_\sigma)\sigma(p_j - p_i) \tag{3.4}$$

where $\phi(x)$, $\|x\|_\sigma$ and $\sigma(x)$ are computed as follows:

$$\phi(x) = \rho_h(x/d)(\sigma(x - d_\sigma) - 1) \tag{3.5}$$

$$\|x\|_\sigma = \sqrt{1 + \|x\|^2} - 1 \tag{3.6}$$

$$\sigma(x) = \frac{x}{\sqrt{1 + \|x\|^2}} \tag{3.7}$$

d in (3.5) is the avoidance distance controlling the distance between agent i and j, which has a significant impact on the coverage time. If d is set to be too large which is by over a certain amount larger than the maximum distance between the adjacent target positions of agent i and j for the environment, agents will get stuck and not be able to fully cover the target area. d_σ is calculated as

$$d_\sigma = \sqrt{1+d^2} - 1 \tag{3.8}$$

$\rho(x)$ in (3.5) is the bump function from [6]

$$\rho_h(x) \begin{cases} 1, & 0 \leq x < h \\ \frac{1}{2}[1+\cos(\pi\frac{x-h}{1-h})], & h \leq x < 1 \\ 0, & otherwise \end{cases} \tag{3.9}$$

where h is a constant and $h \in (0,1)$. $\rho_h(x)$ smoothly varies between 0 and 1 and maps x to that range smoothly.

In consideration of all the neighbours of agent i, the control input for collision avoidance is a sum of the repulsive force between agent i and each of its neighbour multiplying by a positive constant c^α

$$u_i^\alpha = c^\alpha \sum_{j \in N_i} \phi(\|p_j - p_i\|_\sigma)\sigma(p_j - p_i) \tag{3.10}$$

The control input for the agent to reach γ-agent (target position) is calculated based on the PID control algorithm as stated in [6] using the position difference between the position of agent i and the γ-agent p_r and the velocity of agent i as follows:

$$u_i^\gamma = -c_1^\gamma(p_i - p_r) - c_2^\gamma v_i \tag{3.11}$$

where c_1^γ and c_2^γ are the proportional and differential coefficients in the PID algorithm respectively.

The complete control input used for the coverage process of agent i is computed as a sum of the control input for collision avoidance and moving towards the target position as follows:

$$u_i = u_i^\alpha + u_i^\gamma \tag{3.12}$$

The parameters and notations used in this paper are included in Table 3.1.

Table 3.1: Summary of parameters and notations

Parameters	Notations
N	The Number of agents
N_T	The maximum number of training episodes
m	The length of the target area
n	The width of the target area
k	The number of rows of the γ-information map
l	The number of columns of the γ-information map
α	Learning rate
λ	Discounting factor
w	Weight of the cooperative learning
h	Parameters of the bump function
r_{ref}	Maximum reward value for the whole traversal process
T_{min}	Minimum traversal time in the ideal condition
r_s	perceived radius
r_c	Communication distance
ε	Permissible position error
c_r, c_1^T and c_2^T	Parameters of the reward function
c^α, c_1^γ and c_2^γ	Control Parameters of the MAS motion model
P_{stop}	Minimum exploration probability
P_{start}	Maximum exploration probability
λ_d	Decay rate of exploration probability

3.3 γ-information Map

To discretize the continuous traversal process into the discrete traversal process for the proposed algorithms based on RL according to the identical perceived radius of each agent, a γ-information map is designed for each agent maintaining the coverage information as described in [6]. The γ-information map for agent i is defined as follows:

γ-information map: For the problem to be addressed in this paper, the target area is considered as a rectangular region of $m \times n$ where m and n are the horizontal and vertical side length of the region of interest. The region is decomposed into $k \times l$ small rectangular regions and the center of each small rectangle is the γ point. A γ point set of agent i consists of all γ points within the target area, $M_i(\gamma) = \{\gamma_{x,y}\}, x = 0, 1, \cdots k-1, y = 0, 1, \cdots l-1$. The ranges of x and y are different from the one in [6] since the indexing of array in our implementation starts from 0. The coverage information of each γ point in $M_i(\gamma)$ is described by $m_i(\gamma_{x,y})$. The value of $m_i(\gamma_{x,y})$ is 1 if $\gamma_{x,y}$ is covered otherwise it is 0. The γ-information map of agent i is comprised of all the $m_i(\gamma_{x,y})$, $\gamma_{x,y} \in M_i(\gamma)$. In terms of the implementation, the γ-information map of each agent is a matrix or 2D array of size k rows by l columns with all entries initialized to 0 before the traversal process starts.

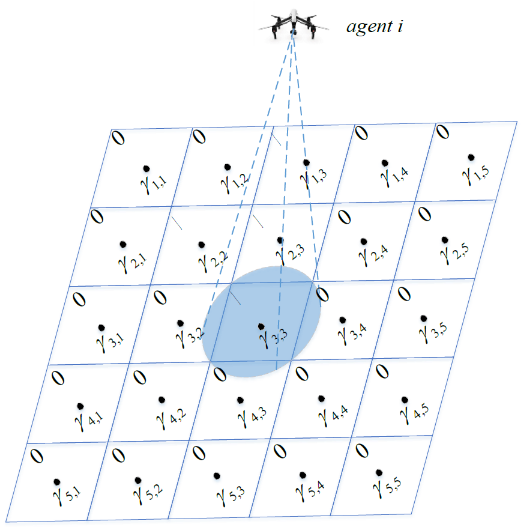


Figure 3.2: γ-information map during the traversal of an agent where the value in the top left corner of each small rectangle is the value of $m_i(\gamma_{x,y})$ from [6]

k and l that determine the number of rectangular regions can be calculated based on the perceived radius r_s of each agent as follows:

$$\begin{cases} k = \lceil \frac{n}{\sqrt{2} r_s} \rceil, \\ l = \lceil \frac{m}{\sqrt{2} r_s} \rceil, \end{cases} \tag{3.13}$$

$\lceil x \rceil$ is the ceiling operation for x. Equation (3.13) makes sure that when agent i reaches the center of the rectangular region $\gamma_{x,y}$ that region is fully covered by the agent as shown in figure 3.2.

When agents are located within the communication distance, the agent is able to extract the information from its neighbours and fuse its γ-information map with its neighbours' γ-information map using the method by the following equation which is different from the one in [6]:

$$m_i(\gamma_{x,y}) = \mathbb{1}_{\mathbb{R}^+}\left(\sum_{k \in (N_i \cup i)} m_k(\gamma_{x,y}) \right) \tag{3.14}$$

$\mathbb{1}_{\mathbb{R}^+}(x)$ is an indicator function as follows:

$$\mathbb{1}_{\mathbb{R}^+}(x) = \begin{cases} 1 & if x \in \mathbb{R}^+ \\ 0 & if x \notin \mathbb{R}^+ \end{cases} \tag{3.15}$$

Using equation (3.15) enables the agent i to not only know if $\gamma_{x,y}$ has been already covered by itself before but also by any of its neighbours, which improves the efficiency of the traversal process.

Chapter 4

Elements of Reinforcement Learning

In this chapter, we described elements of RL which the two proposed algorithms are based on. Both of the algorithms have the same assumptions, state space and action space. The differences between them are in terms of the reward function.

4.1 Assumptions

To apply the proposed algorithms to the dynamic area coverage problem in this paper, the following assumptions are made:

(1) All agents in MAS motion are on the same altitude and follow the MAS motion model described in section 4.

(2) Each agent has the same perceived radius $r_s > 0$. The communication distance r_c between agents for every agent are the same. Agents can share their own information including γ-information map, learning experiences (Q-tables), experiences (tuples representing MDP) and so on with its neighbours when they are located within r_c.

In this paper, assumption (3) in [6] is not made since it does not improve the learning efficiency for the proposed algorithms. The initial position of agents that does not enable the communication at the beginning demonstrates no negative impact on the training of the agents, which shows our proposed algorithms are more robust to different initial conditions.

In order to illustrate the proposed algorithms for the dynamic area coverage using the γ-information map, the symbols used are given here. s_i, a_i and r_i are the current state, action and reward of agent i respectively. The next state and action of agent i are s_i' and a_i' respectively.

4.2 State Space

The current state of agent i consists of the γ-information map and the γ point of each agent. During the process of coverage, agent i fuses its γ-information map with its neighbours' using the method described in section 3 and obtains the current γ point of each of its neighbours by communication with neighbours. Hence, the state of agent i is defined as in [6] as follows:

$$s_i = [M_i, p_1^\gamma, p_2^\gamma, \cdots, p_i^\gamma, \cdots, p_N^\gamma] \tag{4.1}$$

where $p_i^\gamma = (x, y)$ is the index which is row and column on the γ-information map of the current γ point of agent i and it is obtained as follows:

$$x = (n - y_{\gamma_{x,y}})/(n/k) - 0.5 \tag{4.2}$$

$$y = x_{\gamma_{x,y}}/(m/l) - 0.5 \tag{4.3}$$

where $\gamma_{x,y} = (x_{\gamma_{x,y}}, y_{\gamma_{x,y}})$. n/k and m/l are the horizontal and vertical side length of the small rectangular region. Equation (4.2) and (4.3) are used to obtain the index of agent i and the neighbours of agent i. For the case where agent j is not within the interaction range of agent i defined by r_c, $p_j^\gamma = (-1, -1)$.

While the difference between the index of the current and the next γ point of agent i and which one is used in the current state of agent i are not given explicitly, in this paper we define them explicitly for our proposed algorithms:

- The index of current γ point: This refers to the index of the small rectangular region which the agent is getting away from in the way to its next γ point. This is the index of the γ point closest to the agent when it is selecting the next γ point. When the agent finishes the coverage of the region of the next γ point, the next γ point will be updated as the current γ point and the new next γ point will be selected based on the index of the current γ point. It is used as part of the current state of agent *i*.

- The index of next γ point: This refers to the index of the small rectangular region which the agent is moving towards. It is obtained by applying the current action to the index of the current γ point where more detail is stated in the next section. This index is not part of the current state of agent *i*.

4.3 Action Space

The action used in the proposed algorithms based on Q-learning determines the next target position (γ point) of an agent. There are in total at most nine γ points for agent i to choose by taking the corresponding action. The optional γ points include the current γ point $\gamma_{x,y}$ where the agent is leaving determined by p_i^γ and eight surrounding γ points. $1-9$ are used to represent the nine γ points as shown in figure 4.1 from [6]. Hence, the action space of agent i is defined as nine possible actions resulting in nine different γ points as follows:

$$A_i = \{1,2,3,4,5,6,7,8,9\} \tag{4.4}$$

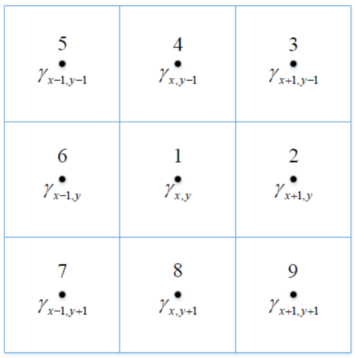


Figure 4.1: Nine possible actions where each number corresponds to the next target position from [6]

The action space will be reduced as a subset of the complete action space described by (4.4) when the agent is at the boundary of the γ-information map. Reducing the action space by considering only the possible actions based on the current state of the agent can reduce the number of entries in Q-table to be updated and hence accelerate the learning process. The reduced action space is illustrated by using two cases and the other cases can be inferred using the same idea. The first case is when the agent is located at the top left corner of the γ-information map and the reduced action space $A_i = \{1,2,8,9\}$; the second case is when the agent is located at the bottom boundary (not at either the bottom left or right corner) of the γ-information map and the reduced action space $A_i = \{1,2,3,4,5,6\}$. Figure 4.2 illustrates those two cases.

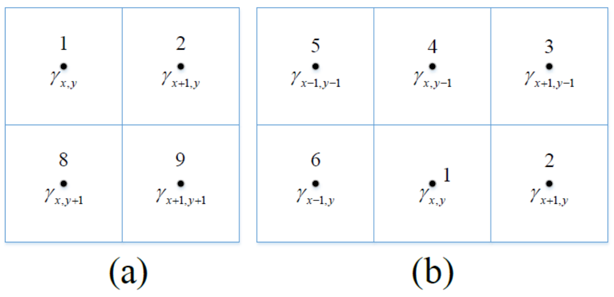


Figure 4.2: Reduced action space from two different cases. (a)Agent at the top left corner of the γ-information map. (b)Agent at the bottom boundary of the γ-information map from [6]

As our proposed algorithms uses the γ-information map to transform the continuous coverage process into the discrete coverage process, it is required for agent i to update its action when it reaches the next target γ point $\gamma_{x,y}$. Whether agent i has reached $\gamma_{x,y}$ is determined by the following:

$$|p_i - \gamma_{x,y}| < \varepsilon \tag{4.5}$$

where ε is the permissible error. Another situation for agent i to update its action is when agent i and agent j have the same γ point. This type of situation is classified into two cases as follows:

(1) The first case is when the communication is established between agent i and agent j. In this case, based on the control quality of avoidance described in section 3, if two agents are moving towards the same target position, they will never be able to reach that position due to the repulsive force imposed by the control quality of avoidance where the avoidance distance ensures the

distance between two agents will be maintained. Hence in our proposed algorithms we restrict the action space of each agent further by not having the same next γ point as its neighbours determined by the p_j^γ and the current selected action of its neighbours after sharing the current state information via communication, which is described in detail in next chapter.

(2) The second case is when the communication distance is too small and agent j is out of the interaction range of agent i. In this case, if they have the same γ point any of them will never be able to reach it and get stuck at some point due to the avoidance distance. To handle this case, when either agent i and agent j is able to communicate with the other as they get sufficiently close to each other (assuming the communication distance is larger than avoidance distance so that this scenario could happen), if the other agent being communicated with has the same γ point, the agent obtaining this information will update its action based on the current action space according to the current state (not the current position) excluding the same γ point as the other agent as described in (1).

4.4 Reward Function

The value of reward for the chosen action received by each agent is estimated based on the coverage state of the γ-information map as described in [6]. The reward function is defined as follows:

$$r_i(s_i, a_i) = \begin{cases} 0, & m_i(\gamma'_{x,y}) = 0, a_i \in \{1,2,4,6,8\}, \\ -0.3, & m_i(\gamma'_{x,y}) = 0, a_i \in \{3,5,7,9\}, \\ -0.2e^{c_r(k_r-1)}, & m_i(\gamma'_{x,y}) = 1, \\ R(T), & m_i(\gamma_{x,y}) = 1, \gamma_{x,y} \in M_i(\gamma), \end{cases} \tag{4.6}$$

where $\gamma'_{x,y}$ is the next γ point obtained by performing the current action a_i from the current p_i^γ in the current state s_i. In terms of the implementation for the purpose of determining $r_i(s_i, a_i)$, a global γ-information map is obtained by fusing the current γ-information map of all agents and the information value $m_i(\gamma'_{x,y})$ of $\gamma'_{x,y}$ is retrieved from the global map. The global γ-information map is also used to determine if the entire target area is fully covered by summing the information value of all $\gamma_{x,y}$ of the map and checking if the result is equal to the total number of $\gamma_{x,y}$ in the map. c_r is a constant and $0 < c_r < 1$, k_r is how many times the agent covers the same $\gamma_{x,y}$, T is the time taken by agents during the process of dynamic area coverage and $R(T)$ is defined as follows:

$$R(T) = \begin{cases} 0, & T > c_1^T T_{min}, \\ \frac{r_{ref}}{2}[1 + cos(\frac{\pi(T - c_2^T T_{min})}{(c_1^T T_{min} - c_2^T T_{min})})], & c_2^T T_{min} < T \leq c_1^T T_{min}, \\ r_{ref}, & T \leq c_1^T T_{min}, \end{cases} \tag{4.7}$$

where c_1^T and c_2^T are constants and $c_1^T > c_2^T > 1$; r_{ref} is a maximum reward value for the entire process of traversal. T_{min}^s is the minimum time for a single agent to take to fully cover the target area in the ideal situation which the computation of minimum time for all agents to complete the coverage process T_{min} is based on and is computed using the following equation:

$$T_{min}^s = \min\{\frac{(l-1)mk}{l|v_{max}|} + \frac{(k-1)n}{k|v_{max}|}, \frac{(k-1)nl}{k|v_{max}|} + \frac{(l-1)m}{l|v_{max}|}\} \tag{4.8}$$

where $|v_{max}|$ is the magnitude of the maximum velocity of the agent. The two terms used to compute T_{min}^s in (4.8) correspond to two ideal situations where trajectories follow back-and-forth motion with horizontal and vertical main path respectively. For the MAS dynamic area coverage problem in this paper, T_{min} used in (4.7) is calculated as follows:

$$T_{min} = \frac{T_{min}^s}{N} \tag{4.9}$$

Since T_{min} is computed without considering the change of velocity and the initial position of the agent during the traversal process, we have $T > T_{min}$.

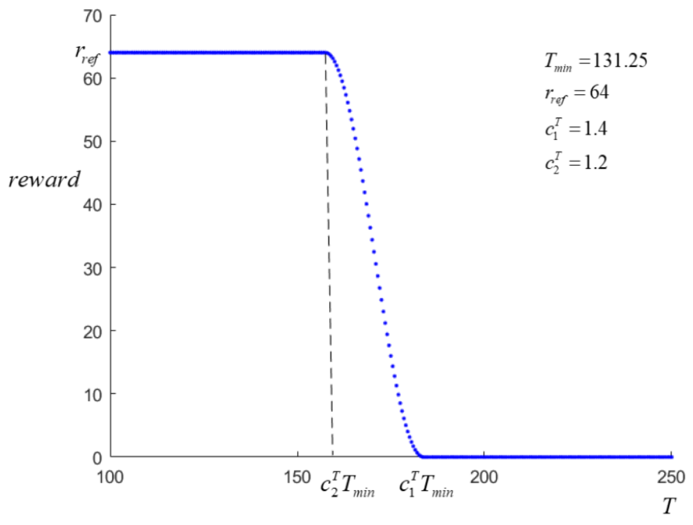


Figure 4.3: Curve of $R(T)$ for $T_{min} = 131.25, r_{ref} = 64, c_2^T = 1.2$ and $c_1^T = 1.4$ from [6]

Based on equation (4.6), the horizontal and vertical action is preferred to the diagonal one since when $\gamma_{x,y} = 0$, the value of reward of $a_i \in \{1,2,4,6,8\}$ is higher than the one of $a_i \in \{3,5,7,9\}$. This is because the time ratio for the time required to take of executing two different types of actions is $1 : \sqrt{2}$ under the assumption that the agent has constant velocity and each small rectangle on the γ-information map is a square. A negative reward is given to the agent when it traverses the same γ point for more than once and the more times it traverses the same γ point the larger the negative reward. There is one thing to notice is that the design of the reward function for repeated traversal does not necessarily result in better trajectories and higher efficiency of the coverage process since selecting another action at the same state as before to avoid the previously encountered repeated traversal could possibly give rise to other different repeated traversals for the MAS. But it provides a way for searching and finding the optimal strategies of each agent.

Once the whole traversal process of agents is accomplished, $R(T)$ is used to reward the entire process and the value of $R(T)$ will be received by all agents (not only the agent finishing the traversal of the last rectangular region) as a way similar to constructing a global reward for the team to help in the learning process. In this paper, for the first proposed MC-Q algorithm the complete equation (4.7) is used while for the second proposed MI-Q algorithm only $R(T)$ given by (4.7) is used.

Chapter 5

Q-value Update and Action Selection

In this chapter, we proposed two different methods used in two different proposed algorithms for the update of the Q-table and the action selection of each agent.

5.1 Modified Cooperative Q-Learning

The first proposed MC-Q algorithm is based on cooperative Q-learning algorithm proposed in [6] and [44]. The update of Q-table is performed as follows:

$$\xi_i^\kappa(s_i,a_i) = Q_i^\kappa(s_i,a_i) + \alpha[r_i^\kappa + \gamma \max_{a_i' \in A_i^p} Q_i^\kappa(s_i',a_i') - Q_i^\kappa(s_i,a_i)] \tag{5.1}$$

$$Q_i^{\kappa+1}(s_i,a_i) = w\xi_i^\kappa(s_i,a_i) + w\sum_{j=1}^{|N_i|} \xi_j^\kappa(s_i,a_i) \tag{5.2}$$

where α is the learning rate, γ is a discounting factor, $Q_i^\kappa(s_i,a_i)$ is the Q-value under s_i and a_i at iteration κ, w is the weight and $0 \leq w \leq 1$, A_i^p is the possible actions at the state dependent on various cases, which is discussed later in the following. Using (5.2) enables each agent to update its Q-table with the aggregated Q-value information of its neighbours in order to speed up the learning process. The idea behind this is that each agent can learn from its neighbours' previous experience and hence learn faster than learn only by itself without making use of its neighbours' Q-value. Given state s_i and action a_i of agent i, (5.2) utilizes the updated Q value at s_i (the state of agent i) instead of at s_j (the state of the neighbour of agent i) of its neighbours (in other words, $\xi_i^\kappa(s_i,a_i)$ is used instead of $\xi_i^\kappa(s_j,a_j)$) to update $Q_i(s_i,a_i)$ since as described in section 4.2 the state of each agent consists of the fused γ-information map and the index of γ point of all agents, which can be considered as a global state of MAS. In this case, the same experience described by the agent's state and action depends on the global state of all agents. In other words, if both agent i and j experience the same scenario, the entries of their Q-table to be updated will be the same.

There are two issues of using this type of cooperative Q-learning to update agents' Q-table for the problem in this paper, which are described and (or) handled as follows:

(1) The first issue is when in the same scenario described by s_i and a_i, the location of agent i's neighbour is different and the possible actions that can be selected by its neighbour could possibly be different from agent i's. In this case, the next state of neighbour agent j may not exist if a_i is not in the action space which consists of all possible actions that agent j can take at s_i. Then based on (5.2) without the next state of the agent the Q-value can not be obtained. Moreover, the value of the cumulative reward $Q_j(s_i,a_i)$ given by performing a_i by its neighbour will possibly not be able to be used as an accurate aggregating information for the update of $Q_i(s_i,a_i)$ due to agent j being located at a different position to agent i and executing the same action as agent i. Hence updating the Q-table in this manner may not give rise to the most accurate $Q_i(s_i,a_i)$. But by doing this $Q_i(s_i,a_i)$ can be considered as becoming a global Q-value and the multi-agent scenario is transformed into a single agent one, which can stabilize the environment and hence stabilize the learning process.

(2) The second issue is agent's neighbours can be either less or more experienced than agent itself for the same scenario. The agent's neighbours with less experience could potentially slow down the learning process of the agent due to its lack of experience and the inaccurate Q-values. For example, if one of the agent i's neighbours has never experienced the scenario (under s_i and a_i) that agent i has experienced for many times, using the Q-value from that neighbour's Q-table to update the corresponding Q-value of agent i could increase

the bias and make it less accurate. This situation will occur for the distributed decision making and learning setting since not all agents will reach their next γ points at the same time step because of different distances between the current position of the agent and the target γ point. Hence in our first proposed MC-Q algorithm, only the Q-value of the agent's neighbour which has experienced the same scenario will be used to update the Q-value of the agent in order to speed up the learning process of the agent. The level of experience the neighbour has for the same scenario is not taken into account that we do not eliminate the neighbour having experienced the same but experienced with less amount of times since the learning process of all agents is performed using their own updated policy without applying random exploration (which will be discussed in more detail later) and the difference between the level of experience (expertness) of agents are small for our problem.

In order to further speed up the convergence of the proposed MC-Q algorithm, the action space of the agent is restricted based on the coverage information of the fused γ-information map and the index of the next γ points of its neighbours:

$$A'_i = \{\gamma_{x,y} \in A_i | m_i(\gamma_{x,y}) = 0, f_{index}(\gamma_{x,y}) \neq p^{\gamma}_j, j \in N_i\} \tag{5.3}$$

where A'_i is a subset of A_i ($A'_i \subseteq A_i$). Based on (5.3) an agent prefers cover the surrounding unvisited γ points, which demonstrates that even without training the agents the proposed algorithm can enable the agents to cover the entire target area.

It is apparent that when the γ points of the A_i of agent i are all covered or being the next γ points of its neighbours or in the case where some of them are traversed and the remaining are being the next γ points of its neighbours, A'_i will become an empty set. In the final stage of the coverage process, the number of untraversed γ points can possibly be less than the number of agents. For agent i, the set of all untraversed γ points in the γ-information map is denoted as S^{γ}_i which is defined as follows:

$$S^{\gamma}_i = \{\gamma_{x,y} \in M_i(\gamma) | m_i(\gamma_{x,y}) = 0, f_{index}(\gamma_{x,y}) \neq p^{\gamma}_j, j \in N_i\} \tag{5.4}$$

For the situation where there is no untraversed γ points, the agents without the need of traversing untraversed γ points will stay at the same location without affecting other agents moving towards the untraversed γ points. Hence the action selection process of agent i is classified into three cases according to whether A_i' is empty and whether there remains untraversed and unallocated γ points for the agent to visit as follows:

Case 1 ($A_i' \neq \varnothing$): In this case, the next action a_i' is chosen based on the greedy policy which exploits the learned experience and selects the action with maximum Q-value as follows:

$$a_i' = \underset{a_i \in A_i'}{\operatorname{argmax}} Q_i(s_i, a_i) \tag{5.5}$$

To accelerate the training, the exploration is not achieved by applying any kind of random policy since the joint state space of all agents is huge and there are enormous amount of possible combination of trajectories of agents. Instead when A_i' is not empty, the greedy policy described by (5.5) is used. Initially all entries of the Q-table are initialized to 0 and the action is selected based on the initialization values of the Q-table. After that the Q-table will be updated with negative or positive reward described in section 4.4 if there is any diagonal movement, repeated traversal or the time taken for the whole traversal process is within a certain range. The agent will change its sequence of actions based on the updated Q-table to explore the environment and search for the optimal strategy in our problem. This process will continue until the convergence is achieved. This proposed method reduces the space of searching for the optimal (or close to optimal) solution and significantly speed up the convergence. For the testing after the training, the greedy policy is applied to make use of the learned values of Q-table without updating the Q-table.

Case 2 ($A_i' = \varnothing$): In this case, first the closest γ point that is unvisited and not being the next γ points of any of agent i's neighbours is found using the state of agent i (s_i) with the fused γ-information map and the shared next γ points of its neighbours as follows:

$$\gamma_{x_1,y_1} = \underset{\gamma_{x,y}\in S_i^{\gamma}}{\operatorname{argmin}} \|p_i^r - \gamma_{x,y}\| \tag{5.6}$$

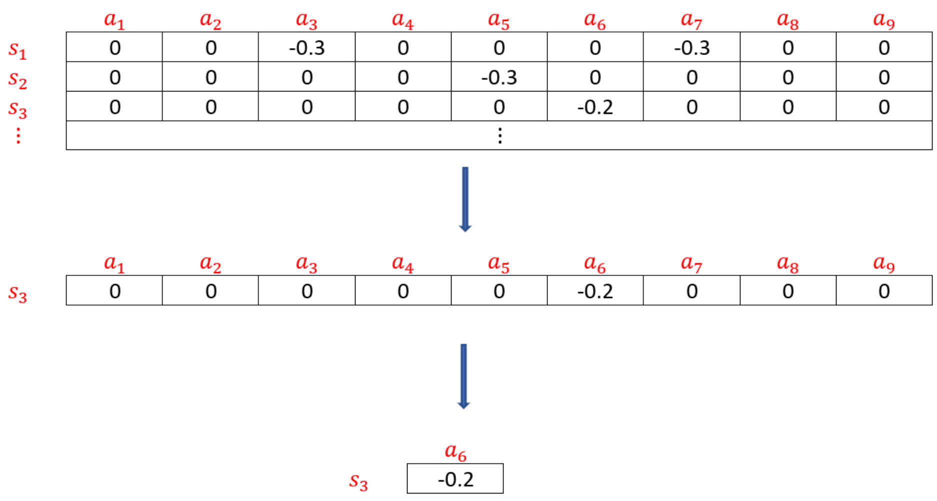
p_i^r is the position of the current position of agent i. The next action a_i' is chosen according to the closest surrounding γ point to γ_{x_1,y_1} in order to minimize the time cost of traversal as follows:

$$a_i' = \underset{\gamma_{x,y}\in A_i^{\gamma}}{\operatorname{argmin}}(\|\gamma_{x_1,y_1} - \gamma_{x,y}\|^2) \tag{5.7}$$

where A_i^{γ} is defined as follows:

$$A_i^{\gamma} = \{\gamma_{x,y} \in A_i | f_{index}(\gamma_{x,y}) \neq p_j^{\gamma}, j \in N_i\} \tag{5.8}$$

In case 2 the agent will have to select an action to visit the traversed γ point, hence an immediate negative reward whose value is based on the number of repeated traversal will be given to the agent. In this situation regardless of whether the Q-table of the agent is updated with the value of the immediate negative reward, it will always choose the same action given the same current state or the same previous state (the state given at one time step before the current state) using the greedy policy because of the update of Q-table using (5.1) and initialization values of Q-table being 0. In other words, the negative reward will not propagate back to the Q-value of the previous state by updating the Q-table with the same experience and the agent will not learn to deviate from the current strategy to avoid the previously experienced repeated traversal. The proposed action selection process for the case similar to case 2 in [6] has this issue. To handle this issue during the learning process, we restricted the row of Q-table of a particular state based on the possible actions the agent can take at that state. For example, if agent i is in case 2 then it is considered there is only one possible action for the agent to take and the row of Q-table of the corresponding state denoted as s_i^2 is restricted to contain only one entry. When updating the Q-value of the previous state one time step before s_i^2, the maximum Q-value of s_i^2 is the maximum value from the corresponding restricted row of Q-table for the next state s_i' in (5.1), in the given example there will be only one value in the row and that value will be the maximum Q-value of the next state. In (5.1), A_i^p is the set of all possible actions at s_i' in either case 1 or case 2. In case 3 there will be no next state and A_i^p does not exist. In case 1, $A_i^p = A_i'$; in case 2, A_i^p contains only one action selected using (5.6) and (5.7). With the discounted immediate negative reward propagated back to the Q-value of the previous state in case 2, the agent will learn to select a different action to avoid the repeated traversal and hence explore the environment to find a better strategy in order to eventually obtain the optimal one. Figure 5.1 illustrates an example of the reduced Q-table for case 2 where $a_1, a_2, \cdots, a_9$ are nine actions as described in section 4.3, s_2 and s_3 are the current and next states of the agent, a_6 is the action selected by the agent in case 2 resulting in the repeated traversal.



	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
s_1	0	0	-0.3	0	0	0	-0.3	0	0
s_2	0	0	0	0	-0.3	0	0	0	0
s_3	0	0	0	0	0	-0.2	0	0	0
⋮					⋮				

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
s_3	0	0	0	0	0	-0.2	0	0	0

	a_6
s_3	-0.2

Figure 5.1: Illustration of reduced Q-table in case 2

Case 3 ($S_i^\gamma = \varnothing$): In this case, there remains no untraversed and unallocated γ points in the γ-information map for agent i to traverse and hence agent i will stop traversing and stay in the same position. The next action $a_i^{'}$ will be:

$$a_i^{'} = 1 \tag{5.9}$$

To further speed up the learning process of the agents, the reward for the whole traversal process $R(T)$ obtained at their last state in an episode is used to update the Q-table of each agent. An episode is the whole traversal process of the target area of all agents with an initial condition (in this paper they refer to initial positions of agents). It can be observed that for an agent if the update of Q-learning is performed from the last (terminal) state to the first (initial) state instead of from the first to the last state, the episodic reward $R(T)$ received at the terminal state can propagate back to all states that Q-values of all states from the episode will get updated using $R(T)$ by going through and updating the states of the episode one time in this reversed manner instead of K times going through and updating the states of the episode from the start to the end of the episode where K is the total number of states in the episode. Hence this type of Q-table update performed in a reverse manner for an episode is adopted in the proposed MC-Q algorithm described in this section 5.1 with various steps back from the terminal state. In the following, propagating back or back-propagation refers to the stated manner of Q-table update. Moreover, the more steps the episodic reward propagates back from the last state, the less exploration agents will have, potentially the worse strategies of agents will be learned but the faster convergence of the learning process will be achieved. When the communication distance r_c is sufficiently large, the back-propagation with $R(T)$ is not used in order to obtain better performance with more exploration since in this case most of the time agents are able to interact with each other and the difficulty of dynamic area coverage is small so that the convergence of the algorithm is easy to obtain; when the communication distance r_c is small, it is more difficult for agents to find the optimal strategies and the convergence is harder to achieve, in this case both negative rewards due to either the diagonal action or the repeated traversal and $R(T)$ are propagated multiple steps back of an episode until the state where the size of the action space of that state is greater than 1 to speed up the convergence of the learning process.

The training procedure for the proposed MC-Q algorithm is shown in Algorithm 1. The action selection algorithm is shown in Algorithm 2. The back-propagation of the episodic reward is shown in Algorithm 3 and 4. For each episode, experience of each time step κ of agent i is recorded as a tuple of $(s_i^\kappa, a_i^\kappa, r_i^\kappa, s_i^{\kappa\prime})$ and stored in an variable called memory. The mapping mpg_i between state-action and the mapping of next state and A_i^p is stored in a data structure of dictionary to determine whether agent i's neighbour has experienced the same scenario. The initial γ point of each agent is set to be the γ point closest to the initial position of the agent. The initial action is set to be 1 for all agents in order to accelerate the training process and obtain the best performance since random initialization of action at the beginning will lead to much larger search space of solutions and initially moving towards the closest γ point is the optimal choice. $p_i^{\gamma\prime}$ is the index of the next γ point obtained by executing action a_i at state s_i of agent i. The operation of fusion of γ-information map and Q-table updates are performed in a discrete manner (not at every time step of the update of the continuous state but periodically when the next γ point is reached) to improve the efficiency of the implementation.

Algorithm 1: Modified Cooperative Q-Learning

```
Initialization of all parameters, Q-tables and memories of all agents.
for each episode do
    Initialize the position and velocity of each agent.
    Initialize state $s$ and action $a$ for agents based on their initial positions.
    Set the maximum number of training episodes $N_T$.
    while not meet episode end condition do
        for each agent $i$ do
            if $S_i^\gamma \neq \varnothing$ then
                Calculate the value of initial state-action
                $r(s_o, a_o)$ using (4.6).
                if $p_i^{\gamma'} = p_j^{\gamma'}, j \in N_i$ then
                    Update the $\gamma$-information map using (3.14).
                    Calculate $A_i'$ using (5.3).
                    Perform Algorithm 2.
                end
                Calculate $u_i$, $v_i$, $p_i$ through the MAS motion model.
                if $|p_i - \gamma_{x,y}| < \varepsilon$ then
                    Update the $\gamma$-information map using (3.14).
                    Calculate state $s_i'$ using (4.1).
                    Store tuple $(s_i, a_i, r_i, s_i')$ into memory.
                    Store mapping $mpg_i$ into dictionary.
                    if not meet episode end condition then
                        Perform Algorithm 2.
                        Update the Q-value using (5.1) and (5.2)
                        or perform Algorithm 3 based on $r_c$.
                        if $S_i^\gamma \neq \varnothing$ then
                            Update the state and action: $s_i = s_i'$, $a_i = a_i'$.
                        end
                    end
                    Calculate the value of next state-action
                    $r(s_i', a_i')$ using (4.6).
                end
            end
        end
    end
    Perform Algorithm 4.
end
```

Algorithm 2: Action Selection Algorithm

```
if $A'_i \neq \varnothing$ then
    Select the next action $a'_i$ using (5.5).
else if $A'_i = \varnothing, S^{\gamma}_i \neq \varnothing$ then
    Select the next action $a'_i$ using (5.7).
else
    The next action $a'_i = 1$.
end
```

Algorithm 3: Back-propagation Algorithm

```
for each experience tuple at $\kappa$, $\kappa = K, K-1, \cdots, 0$ do
    if $|A^p_i| \leq 1$ then
        Update the Q-value with the tuple using (20) and (21).
    else
        break.
    end
end
```

Algorithm 4: Episodic Reward Back-propagation Algorithm

```
for each agent $i$ do
    if $r_c$ is sufficiently large then
        Update the Q-value with the tuple using (20) and (21).
    else
        Perform Algorithm 3.
    end
end
```

5.2 Modified Independent Q-Learning

The second proposed algorithm is based on independent Q-learning where the Q-table of each agent is updated independently without utilizing the previous experiences of its neighbours. The instability of environment caused by independent learning of each agent due to the not coordinated change of policy of each agent (where each agent can not keep track of the change of other agents' policy and adapt to their newest policy) is addressed by using only the same episodic reward $R(T)$ for each agent introduced in (4.7) to update each agent's Q-table as a way of constructing a global reward for the whole team of agents. The main idea of back-propagation of the episodic reward discussed in section 5.1 is applied using a different update rule for the Q-table update to significantly speed up the learning process. In this case, the update is not required to be performed in a reverse manner. The traditional update of Q-value using one step reward and the maximum Q-value of the next state is changed into assigning the episodic reward directly with comparison to the previous values of experiences from all episodes to the entries in Q-table. Taking the dynamic area coverage problem in this paper as a problem of searching for the optimal trajectories (strategies) to minimize the time taken to traverse and cover the entire target area, we maintain the best trajectories and the ones closest to the best ones that the agents have ever experienced from their previous episodes in their Q-table and enable the agents to explore randomly while exploiting the good experiences to facilitate the process of searching for the optimal. Since the state space is huge and there is enormous amount of combinations of trajectories of the agents, the process of finding the optimal is performed initially as a sparse search and gradually as a dense search by exploration and exploitation of the environment. To achieve this, the update of Q-value of each state-action pair within an episode of agent i is updated based on the following:

$$Q_i^{\kappa+1}(s_i^t, a_i^t) = \max(Q_i^{\kappa}(s_i^t, a_i^t), R^{\kappa}(T)), \forall t \in \{1, 2, \cdots, N_{ep}\} \tag{5.10}$$

where κ is the number of an episode, t is the time step of the episode, N_{ep} is the total number of time steps of the episode. (5.10) shows that the maximum $R(T)$ agent i has ever received from all the previous experiences is maintained in the corresponding entries in its Q-table. When the Q-value of new maximum $R(T)$ received by agent i already exists in the row of Q-table corresponding to the Q-values at s_i, the value of the entry of the Q-table containing the previous same maximum $R(T)$ is clear and set to 0 and the entry of the Q-table correspond to the current s_i and a_i is updated with the new maximum $R(T)$. By renewing the row of the Q-table with the same Q-value in this way, the required memory is reduced and the newest best trajectories from all experienced episodes of all agents are kept in the Q-table of each agent and the old ones are removed to eliminate the mismatch of strategies of different agents so that we can restore the dynamic area coverage process with the best strategies found from the entire learning process of the agents.

In terms of the action selection process, in order to significantly improve the efficiency of finding the optimal solution for the problem in this paper, we prioritize the horizontal and vertical actions over the diagonal ones in the restricted action space used in case 1 described in section 5.1. When A_i' is not an empty set, the next action a_i' is first selected from those horizontal and vertical actions otherwise from the diagonal ones based on (5.5). Restricting the action space further in this way can speed up the learning process of each agent since each of them already knows to choose the horizontal and vertical actions to reduce time cost of the traversal without having to learn it via interaction with environment and update of Q-values using the associated negative reward. While there is an alternative that restricts the action space with only horizontal and vertical actions remained, which seems more tempting and better to enable the agents to find the optimal solution. However, this alternative reduces flexibility of the traversal of the agent and increases the time cost for reaching the next γ point along the diagonal direction since in this case it takes one step horizontal and another step vertical movement, in total two steps for an agent to reach the diagonal γ point where the time ratio of the case of applying alternative and the case of moving diagonally is $2 : \sqrt{2}$. Another advantage of using the scheme of prioritization of horizontal and vertical movements is that to achieve the objective of the proposed algorithm which is to find trajectories with least number of diagonal movements and repeated traversal for the complete coverage of the target area, not only is it able to result in the type of trajectories we are after but also at the same time it can produce more suboptimal solutions during the learning process to improve the efficiency of the entire process of finding the optimal (or close to optimal) solution based on the idea of dense search (exploitation of previous good experiences) stated above. The exploration of the environment is achieved by using Epsilon-Greedy strategy with exponentially decaying rate of exploration.

The training procedure for the proposed MI-Q algorithm is shown in Algorithm 5. Mainly apart from the Q-table update and the action selection process, it remains similar as the proposed MC-Q algorithm.

Algorithm 5: Modified Independent Q-Learning

```
Initialization of all parameters, Q-tables and memories of all agents.
for each episode do
    Initialize the position and velocity of each agent.
    Initialize state $s$ and action $a$ for agents based on their initial positions.
    Set the maximum number of training episodes $N_T$.
    while not meet episode end condition do
        for each agent $i$ do
            if $S_i^\gamma \neq \varnothing$ then
                if $p_i^{\gamma'} = p_j^{\gamma'}, j \in N_i$ then
                    Update the $\gamma$-information map using (3.14).
                    Calculate $A_i'$ using (5.3).
                    Perform Algorithm 6.
                end
                Calculate $u_i$, $v_i$, $p_i$ through the MAS motion model.
                if $|p_i - \gamma_{x,y}| < \varepsilon$ then
                    Update the $\gamma$-information map using (3.14).
                    Calculate state $s_i'$ using (4.1).
                    Store tuple $(s_i, a_i, r_i, s_i')$ into memory.
                    if not meet episode end condition then
                        Perform Algorithm 6.
                        if $S_i^\gamma \neq \varnothing$ then
                            Update the state and action: $s_i = s_i'$, $a_i = a_i'$.
                        end
                    end
                end
            end
        end
    end
    Perform Algorithm 7.
end
```

The action selection process using action prioritization and Epsilon-Greedy strategy is shown in Algorithm 6. The update of Q-table after each episode is shown in Algorithm 7.

Algorithm 6: Action Selection Algorithm

```
if $A'_i \neq \varnothing$ then
    Obtain a subset $A_i^{hv}$ of $A'_i$ with horizontal and vertical actions.
    Obtain a subset $A_i^{dg}$ of $A'_i$ with diagonal actions.
    if $A_i^{hv} \neq \varnothing$ then
        $A'_i = A_i^{hv}$
    else
        $A'_i = A_i^{dg}$
    end
    Randomize a number $P_r \in [0,1]$.
    Calculate exploration probability $P_e$ using (5.11).
    if $P_r < P_e$ then
        Select the next action $a'_i$ randomly from $A'_i$
    else
        Select the next action $a'_i$ using (5.5).
    end
else if $A'_i = \varnothing, S_i^{\gamma} \neq \varnothing$ then
    Select the next action $a'_i$ using (5.7).
else
    The next action $a'_i = 1$.
end
```

Algorithm 7: Episodic Q-update Algorithm

```
for each agent $i$ do
    for each experience tuple at $\kappa$, $\kappa = 0, 1, \cdots, K$ do
        Update the Q-value with the tuple and $R(T)$ using (5.10).
    end
end
```

The exploration probability is calculated as follows:

$$P_e = P_{stop} + (P_{start} - P_{stop})e^{\lambda_d t} \tag{5.11}$$

where P_{stop} is the minimum exploration probability, P_{start} is the maximum exploration probability, λ_d is the decay rate.

Chapter 6

Simulation

6.1 Simulation and Evaluation Process

In this section, parameter settings and the termination conditions for the entire training process and the episode for the two different proposed algorithms (MC-Q and MI-Q) are presented. The same evaluation criterion for the performance of the two different proposed algorithms is defined.

6.1.1 Simulation Process

For the environment settings in the experiment of simulation for both two algorithms, we set $N = 3$, $r_s = 4.5$, $m = n = 50$m, the time step of simulation to be 0.5s and the time taken to initialize the MAS to be 0.3s. Based on (3.13), we have $k = l = 8$. In [6], d is set to be 8 and the maximum distance between the adjacent target positions is 6.65 based on the parameters of agents and learning and size of the target area. In this situation for most of different initial conditions, agents cannot fully cover the target area since d is larger than the maximum distance between the positions of center of adjacent target rectangular regions by over a certain amount which is a bit too large. Hence in this paper, we reduce the value of d and set it to be 7, which is by less amount larger than 6.65 and close to it to ensure the complete coverage of the target area. The MAS motion parameters used for the simulation of both algorithms are the same and shown in Table 6.1. The learning and associated parameters of the two different proposed algorithms are shown in Table 6.2 and 6.3 respectively.

Table 6.1: MAS motion parameters

Parameters	$\lvert v_{max}\rvert$	d	h	c^{α}	c_1^{γ}	c_2^{γ}
Value	1.0	7	0.1	0.5	0.5	0.8

Table 6.2: Modified cooperative Q-learning parameters

Parameters	α	γ	w	r_{ref}	ε	c_r	N_T
Value	1.0	0.8	0.5	64	0.2	1.0	8×10^3

Table 6.3: Modified independent Q-learning parameters

Parameters	r_{ref}	ε	P_{stop}	P_{start}	γ_d	N_T
Value	64	0.2	0.05	1.0	3×10^{-6}	4×10^3

It is required to consider r_c when we are setting c_1^T and c_2^T. Since m and n are set to 8 for the target area, when $r_c \geq 40$, during the process of dynamic area traversal the communication between agents can be established for most of the time. When $r_c < 40$, the chance of establishing communication between agents become lower as r_c gets smaller, in this case it becomes harder for the first proposed MC-Q algorithm to converge due to the increased difficulty of finding the optimal strategy for the traversal. Moreover, when $r_c = 10$ the situation where agents move towards the same γ point and only know that via communication until they are sufficiently close to each other will occur for many times, which significantly decrease the quality of the found traversal strategy. To ensure the convergence of the MC-Q algorithm for the case when $r_c = 10$, the values of $c_1^T - c_1^T$ should be increase a lot. For the case when $r_c < 40$, the back-propagation of the episodic reward is applied as described in section 5.1 while the values of $c_1^T - c_1^T$ are the same as the ones used in the case when $r_c \geq 40$. For the second proposed MI-Q algorithm, even the communication between agents is affected by r_c, for all cases of r_c the same values of $c_1^T - c_1^T$ can be applied to ensure the convergence and performance of the algorithm.

The termination conditions for the entire training process of two different proposed algorithms are set in the same way for the simulations:

(1) When the number of training episodes is larger than N_T, the training process will be terminated.

(2) When the time taken for the entire traversal process of an episode is unchanged for 10 episodes, in other words the algorithm has converged, the training process will be terminated.

The stop criterion for each episode of training for two different proposed algorithms is identical and contains the following two cases:

(1) $T^{\kappa} > NT_{min}$, $T_{min} = 131.25$s, which can be computed from (4.8) and (4.9).

(2) $m(\gamma_{x,y}) \neq 0, \gamma_{x,y} \in M(\gamma)$ and $m(\gamma_{x,y})$ is obtained by fusing the γ-information maps of all agents, which can be obtained using the following formula:

$$m(\gamma_{x,y}) = \mathbb{1}_{\mathbb{R}^+}\Big(\sum_{k \in \{1,2,3\}} m_k(\gamma_{x,y})\Big) \tag{6.1}$$

where the indicator function $\mathbb{1}_{\mathbb{R}^+}(x)$ is defined as (3.15). For the first case, agents are not able to complete the full coverage of the target area in time NT_{min}; for the second case, agents are able to finish the full coverage process of the target area within time NT_{min}.

6.1.2 Evaluation Criterion

As the objective of this paper is to propose algorithms that maximize the efficiency of the traversal process of the target area, with the set motion parameters of agents the time T taken for the entire traversal process of an episode can be used to evaluate the performance of the proposed algorithms. The indicators of performance of the algorithms are the mean traversal time $\overline{T}$ and variance s^2 of the traversal time.

$$\overline{T} = \frac{1}{100} \sum_{\kappa=1}^{100} T^{\kappa} \tag{6.2}$$

$$s^2 = \frac{1}{100} \sum_{\kappa=1}^{100} (T^{\kappa} - \overline{T})^2 \tag{6.3}$$

$\overline{T}$ shows the average coverage efficiency of the proposed algorithms and s^2 shows how stable the performances of the proposed algorithms are given various initial positions of agents.

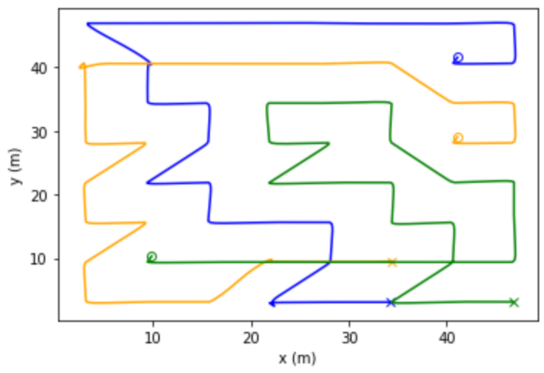
In the simulation process, the performance of the proposed algorithms is compared in the target area where there is no obstacle using $r_c = 60$m, 50m, 40m, 30m, 20m and 10m.

6.2 Simulation Results and Analysis

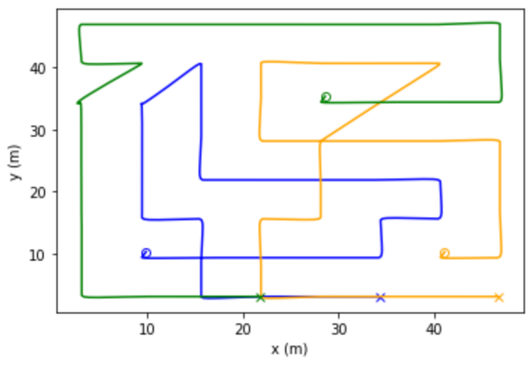
In this section, the trajectories of agents as diagram of the coverage effect demonstrating the dynamic area coverage process for the MC-Q and MI-Q algorithms are first presented. Then the test results for the $\overline{T}$ and s^2 of the MC-Q and MI-Q algorithms with different r_c are given.

6.2.1 Coverage Effect

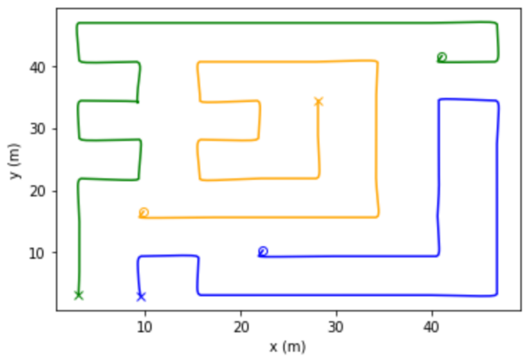
In the simulation, we set $r_c = 50$m, $c_1^T = 1.30$ and $c_2^T = 1.07$ for both of two proposed algorithms. For the MC-Q algorithm, c_r is set to be 1.0. Using these parameters, the trajectories of agents for the untrained and trained MC-Q and MI-Q algorithms are obtained and shown in figure 6.1a, 6.1b, 6.1c and 6.1d respectively.



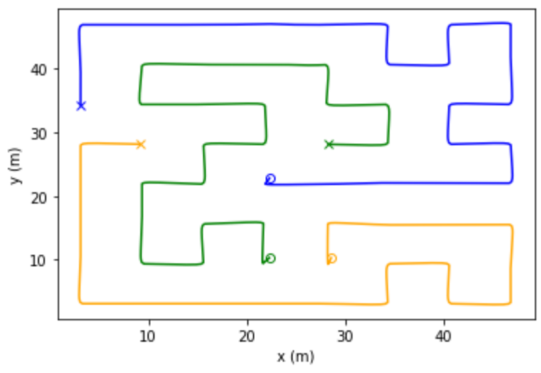
(a)



(b)



(c)



(d)

Figure 6.1: Map of trajectories of coverage effect. Different colors represent different agents. Circles and crosses denote the initial and final positions of agents. (a) Trajectory map for the untrained MC-Q algorithm. (b) Trajectory map for the untrained MI-Q algorithm. (c) Trajectory map for the trained MC-Q algorithm. (d) Trajectory map for the trained MI-Q algorithm.

Figure 6.1 illustrates that both untrained and trained of two proposed algorithms can complete covering the target area within a given time from (4.8) and (4.9). Figure 6.1a and 6.1b shows that the performance of two untrained proposed algorithms is similar since both of the time taken for the coverage process is around 170s and there are diagonal movements and overlapping trajectories between agents which significantly reduces the efficiency of the coverage process. Figure 6.1c and 6.1d shows that training either using the MC-Q algorithm or the MI-Q algorithm for some episodes enables agents to cover the target area efficiently and optimally where all movements are either horizontal and vertical without diagonal and there does not exist any repeated traversal either within an agent's trajectory or between agents' trajectories. The initial movement represented by the circle and the first γ-point is sometimes not completely horizontal and vertical as each agent is controlled to move towards the closest γ-point at the beginning, which is a optimal strategy. Both of the time taken for agents trained by either the the MC-Q algorithm or the MI-Q algorithm to complete the coverage process shown by Figure 6.1c and 6.1d is around 142s.

To compare and analyse the rate of coverage of untrained and trained proposed algorithms, the cumulative area at different times is recorded by discretizing the target area into very small squares and update the coverage map consisting of these squares which keeps tracks of the global coverage information using the perceived radius of the agents at each time step. The cumulative area coverage versus times is shown in Figure 6.2. The coverage rate of untrained and trained methods is the slope of the curve. It can be seen from Figure 6.2 before around 65s the coverage rates are similar and close for all untrained and trained algorithms. However, after that the coverage rates of untrained MC-Q algorithm are significantly lower than the other three and when the time is greater than 87s the ones of untrained MI-Q algorithm gradually get smaller than the other three and around 107s become similar to the untrained MC-Q algorithm. Both of the trained MC-Q and MI-Q algorithms are greatly more efficient than the corresponding untrained ones since the agents learn the closed to optimal strategies to avoid the repeated traversal of the target area where in the late stage of the process it is easier for agents to traverse repeatedly across the same area as the number of the remaining unvisited cells is much smaller than at the initial stage and the agents perform significantly better with trained proposed algorithms applied in the late stage.

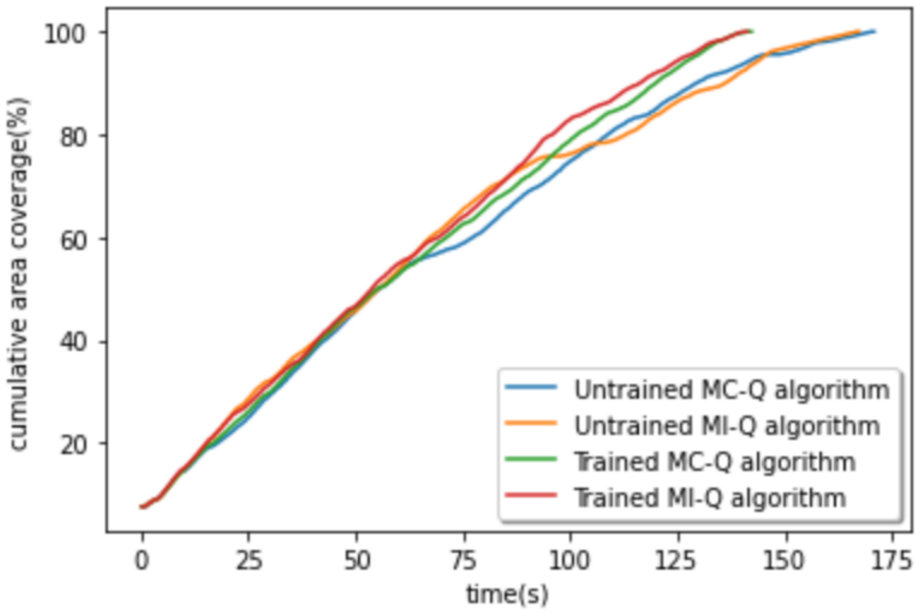


Figure 6.2: Cumulative area coverage of the MAS for two different untrained and trained algorithms

6.2.2 Performance at Different Communication Distances

In the simulation, we test the performance of both untrained and trained MC-Q and MI-Q algorithms by setting r_c to be 60m, 50m, 40m, 30m, 20m and 10m. Based on different values of r_c, the values of c_1^T and c_2^T are set accordingly. For the simulation of MC-Q algorithm, when $r_c = 10$, c_1^T is set to be 1.60. For the remaining cases of r_c of the simulation of both algorithms, c_1^T and c_2^T are set to be 1.30 and 1.07 respectively.

The curves of the performance indicator $\overline{T}$ for the untrained and trained MC-Q and MI-Q algorithms with $r_c = $ 10m, 20m, 30m, 40m, 50m and 60m are shown in figure 6.3.

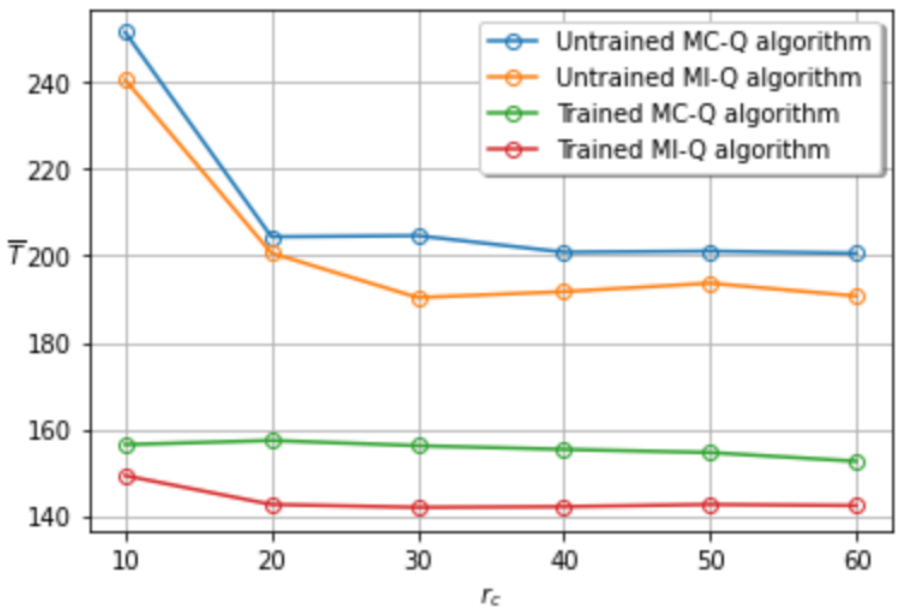


Figure 6.3: $\overline{T}$ of the MAS for the four different methods with different r_c

Based on figure 6.3, when $r_c < 20$ the $\overline{T}$ of both the untrained MC-Q and MI-Q algorithms is significantly larger than the one when $r_c \geq 20$. The reason behind this is because when $r_c < 20$ the communication distance is too small for agents to interact with each other for most of the time during the traversal process of the target area where the coverage information contained in the γ-information map of each agent cannot be shared. In this case, sometimes agents possibly move towards the same γ point without noticing until they get sufficiently close to each other and then one of them has to change the target γ point midway through going to the current γ point, which results in significant decrease in efficiency of the dynamic area coverage process. When $r_c \geq 20$, the $\overline{T}$ of both the untrained MC-Q and MI-Q algorithms is similar where the one of MI-Q algorithm is slightly smaller than the one of MC-Q algorithm, but the one of both the trained MC-Q and MI-Q algorithms is much better where the trained MI-Q algorithm performs better than the trained MC-Q algorithm and yields close to optimal strategy of agents. Since with MC-Q algorithm agents have no prior knowledge about how to traverse the target area efficiently while with MI-Q algorithm agents know to prioritize the horizontal and vertical over the diagonal actions (movements), both the untrained and trained MI-Q algorithms give rise to smaller $\overline{T}$ than the untrained and trained MC-Q algorithms. However, the trained MC-Q algorithm demonstrates stronger learning effect of agents as on average the decrease in terms of $\overline{T}$ from the untrained to trained MC-Q algorithm is larger than the one from the untrained to trained MI-Q algorithm. When $r_c \geq 20$, the $\overline{T}$ of both the untrained and trained MC-Q and MI-Q algorithms does not decrease significantly but become relatively stable as r_c decreases.

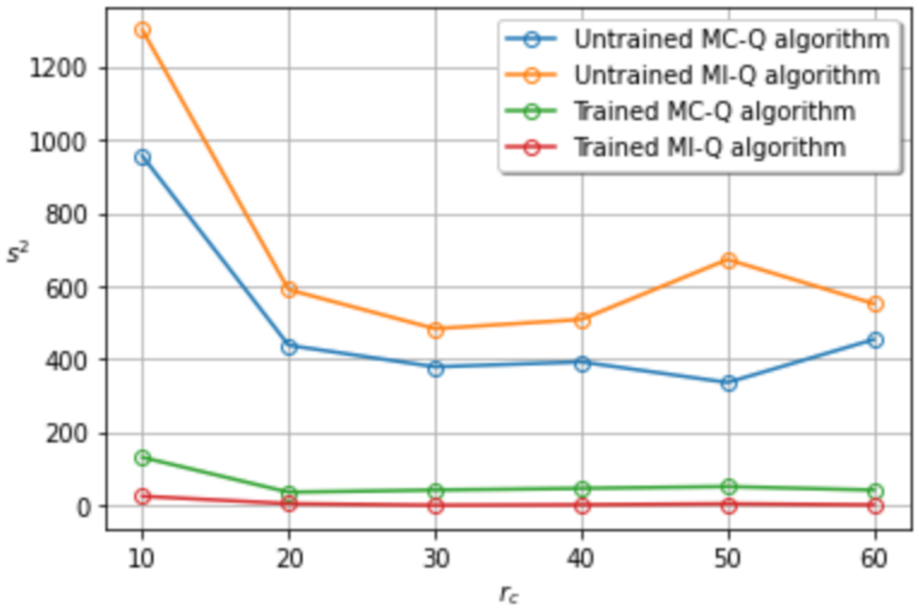
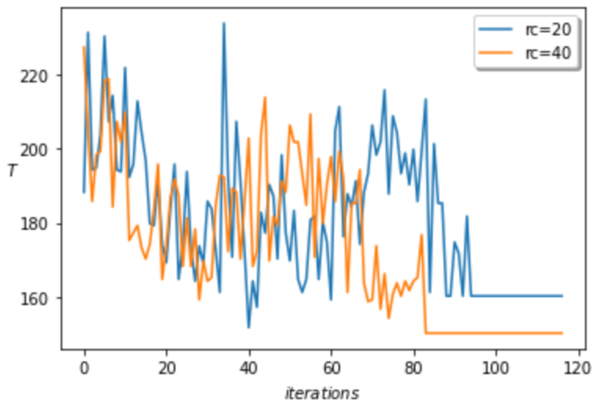


Figure 6.4: s^2 of the MAS for the four different methods with different r_c

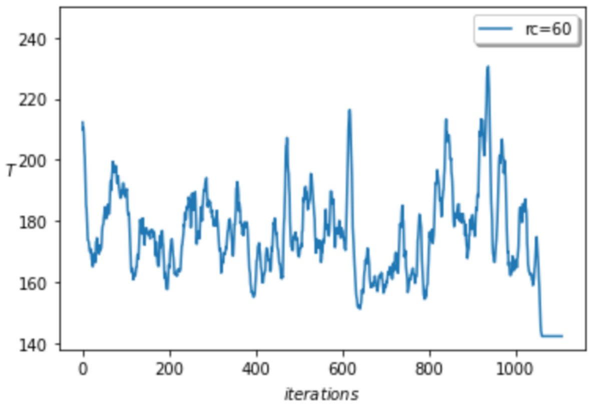
The curves of s^2 for the untrained and trained MC-Q and MI-Q algorithms with different values of r_c are shown in figure 6.4. Similar to the curves of $\overline{T}$ shown in figure 6.3, when $r_c < 20$ the value of s^2 for the untrained MC-Q and MI-Q algorithms is significantly larger. The trained MC-Q and MI-Q algorithms are much more stable than the untrained MC-Q and MI-Q algorithms as the s^2 is much smaller for all different values of r_c. The trained MI-Q algorithm is the most stable method under different initial conditions and the influence given by various initial conditions is almost completely eliminated since the values of s^2 of the trained MI-Q algorithm are close to 0 with different r_c.

6.2.3 Convergence of MC-Q and MI-Q Algorithms

In this experiment, the convergence of the trained MC-Q and MI-Q algorithms with $r_c = 20$m, 40m and 60m is tested. The initial positions of the three agents is set to be $(21.875, 9.375)$, $(9.375, 15.625)$ and $(40.625, 40.625)$ where the communication between agents is not established for all pair of agents initially for $r_c = 20$m or 40m since the distance between $(9.375, 15.625)$ and $(40.625, 40.625)$ is larger than 40m. The parameters used in this experiment for the trained MC-Q and MI-Q algorithms are the same as the ones described in section 6.2.2. The simulation results plotted as time taken for the coverage process T versus iterations are shown in figure 6.4 and 6.5.



(a)



(b)

Figure 6.4: Plot of convergence of T at different r_c for the trained MC-Q Algorithm

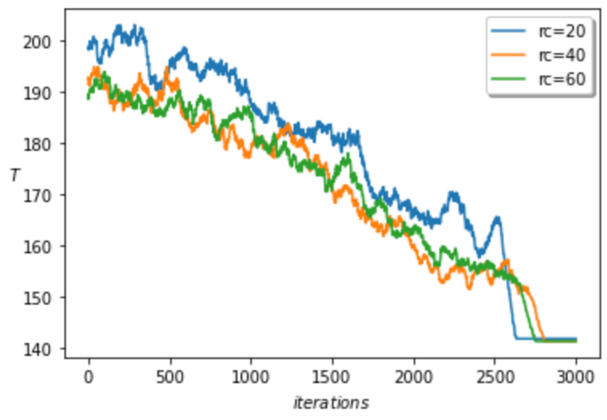


Figure 6.5: Plot of convergence of T at different r_c for the trained MI-Q Algorithm

Based on figure 6.4 and 6.5, the convergence can be achieved for both the trained MC-Q and MI-Q algorithms at different communication distance. Figure 6.4 shows that the number of iterations taken for MC-Q algorithm to achieve convergence at $r_c = 20$m or 40m are significantly smaller than the one at $r_c = 60$m since when $r_c = 20$m or 40m agents cannot communicate with each other for some cases and back-propagation of negative and episodic rewards as described in section 5.1 is used to speed up the convergence. In this case, there is less exploration of agents during the training process and as can be seen from figure 6.4 after convergence agents trained by MC-Q algorithm at $r_c = 60$m are able to find better strategies which are closer to the optimal solution. The trend of the change of T in figure 6.4 shows that during the training process agents explore the environment and keeps searching for better strategies in different part of the searching space (space of all combination of trajectories of agents). It seems that when the best strategies that can be found by agents are found for a part of the searching space, agents will try to explore a different part of the searching space and then find the best strategies they can find in this part. After exploring different parts of the searching space, the MC-Q algorithm eventually converges at the best strategies obtained from the part agents have explored.

Figure 6.5 shows that regardless of different r_c the trained MI-Q algorithm enables agents to find the close to optimal strategies after convergence. As the Epsilon-Greedy strategy is applied for the exploration and exploitation of environment by agents unlike the case for the trained MC-Q algorithm where the greedy strategy is utilized, the time taken for the coverage process T keeps decreasing overall shown by the trend in figure 6.5. On average T obtained at $r_c = 20$m is larger than the ones obtained at $r_c = 40$m or 60m due to the lack of communication between agents and the ones of $r_c = 40$m and 60m are similar and close to each other.

Chapter 7

Conclusion

In this paper, we first design the supplementary component for the MAS motion model and then propose two different algorithms with modified action selection process in order to maximize the efficiency of the dynamic area coverage based on [6]. First of all the update of the continuous state of an agent of the MAS including its velocity and position is achieved by integration of its acceleration and velocity over time respectively in a discrete manner assuming constant acceleration within the time period of update. Then we redesign and supplement the process of action selection of each agent at a given state during the traversal process for both cases of sufficient and insufficient communication distances between agents by restricting the action space further upon consideration of the next γ points of agent neighbours and adjusting the current action being executed by the agent. In the end we propose the MC-Q and MI-Q algorithms for dynamic area coverage of free area based on the γ-information map and Q-learning algorithm. In terms of the MC-Q algorithm, we modify the cooperative Q-learning algorithm by considering the level of experience and the possible actions of the neighbour of the agent to speed up the convergence. The reduced Q-values of the Q-table at a particular state is used during the learning process to enable agents to learn to avoid as many diagonal actions and repeated traversals as possible. The back propagation of the reward is utilized when the communication distance is insufficient to accelerate the convergence of the algorithm resulting in less exploration. For the MI-Q algorithm, we enable agents to prioritize the horizontal and vertical actions over the diagonal ones to improve the efficiency of the traversal. We use full back propagation of the episodic reward and enable each agent to learn independently with the same episodic reward to stabilize the learning process. The best experiences of agents are used for the update of the Q-table of each agent so that it can explore and exploit the environment (look for the optimal strategies) intelligently with specific focus on the previous experienced good trajectories.

Based on results of the simulation, the entire target area can be fully covered by the MAS using one of the four methods that are the untrained and trained MC-Q and MI-Q algorithms. The coverage effect, efficiency and stability of both the untrained MC-Q and MI-Q (the untrained MC-Q compared to the untrained MI-Q) or both the trained MC-Q and MI-Q (the trained MC-Q compared to the trained MI-Q) algorithms are similar while the trained ones demonstrate significant better performance than the untrained ones in terms of $\overline{T}$ and s^2. Regardless of different communication distance between agents, both trained MC-Q and MI-Q algorithms result in similar performance and in particular MI-Q algorithm performs better and enables the agents to learn the closer to optimal strategy even for some cases where the communication distance is not sufficiently large and the interaction cannot be established between agents for most of the time during the process of dynamic area coverage. The MC-Q algorithm demonstrates stronger learning effect than the MI-Q algorithm that agents with no prior knowledge can learn to cover the target area efficiently and the improvement on efficiency of the coverage process they make by learning using the MC-Q algorithm is larger since the reduction on average time cost $\overline{T}$ from agents without to with training is larger than the one of MI-Q algorithm that agents have some prior knowledge about strategies of efficiently covering the target area. The variance s^2 of the time taken for the entire coverage process due to different initial conditions of agents can be reduced using the trained MC-Q algorithm and almost eliminated using the trained MI-Q algorithm.

In the future work, we intend to extend our current work to enable agents to be able to dynamically cover the non-free area where there are static and dynamic no-fly zones. We will consider redesigning the state space of the agent by adding extra maps to record the detected no-fly zones. In this case where the size of the state space will become significantly larger, it might be beneficial to investigate the deep RL techniques such as DDQN instead of using Q-table to make use of the experiences more efficiently to approximate the Q-values faster. We will also redesign the reward function for the dynamic non-free area coverage so that agents can learn to avoid traversing the no-fly zones and at the same time cover the target area efficiently.

Bibliography

[1] Wikipedia contributors, "Unmanned aerial vehicle — Wikipedia, the free encyclopedia," 2005, [Online; accessed 31-August-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle#:~:text=An%20unmanned%20aerial%20vehicle%20(UAV,of%20communications%20between%20the%20two.

[2] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated coverage and connectivity configuration for energy conservation in sensor networks," *ACM Trans. on Sens. Netw.*, vol. 1, no. 1, pp. 36–72, Aug. 2005.

[3] X. Cao, J. Chen, Y. Zhang, and Y. Sun, "Development of an integrated wireless sensor network micro-environment monitoring system," *ISA Trans.*, vol. 47, no. 3, pp. 247–255, Jul. 2008.

[4] F. Yang, X. Ji, C. Yang, J. Li, and B. Li, "Cooperative search of uav swarm based on improved ant colony algorithm in uncertain environment," in *IEEE ICUS*, Beijing, China, Oct. 2017, pp. 231–236.

[5] W. Meng, Z. He, R. Su, P. K. Yadav, R. Teo, and L. Xie, "Decentralized multi-uav flight autonomy for moving convoys search and track," *IEEE Trans. on Control Systems Technology*, vol. 25, no. 4, pp. 1480–1487, Jul. 2017.

[6] J. Xiao, G. Wang, Y. Zhang, and L. Cheng, "A distributed multi-agent dynamic area coverage algorithm based on reinforcement learning," *IEEE Access*, vol. 8, pp. 33 511–33 521, Jan. 2020.

[7] Wikipedia contributors, “Unmanned ground vehicle — Wikipedia, the free encyclopedia,” 2008, [Online; accessed 31-August-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Unmanned_ground_vehicle

[8] M. Thiene, Z. S. Khodaei, and M. H. Aliabadi, “Optimal sensor placement for maximum area coverage (mac) for damage localization in composite structures,” *Smart Mater. Struct.*, vol. 25, no. 9, p. 095037, Aug. 2016.

[9] V. Mallardo, M. H. Aliabadi, and Z. S. Khodaei, “Optimal sensor positioning for impact localization in smart composite panels,” *J. Intell. Mater. Syst. Struct.*, vol. 24, no. 5, pp. 559–573, Mar. 2013.

[10] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning,” *arXiv e-prints*, p. arXiv:2003.02609, Mar. 2020.

[11] D. Silver, “Deep reinforcement learning,” https://deepmind.com/blog/article/deep-reinforcement-learning, accessed: 2020-09-01.

[12] S. Saha, “A comprehensive guide to convolutional neural networks — the eli5 way,” https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, accessed: 2020-09-01.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[14] H. V. Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Thirtieth AAAI conference on artificial intelligence*, 2016, pp. 2094–2100.

[15] K. L. Anirudh, E. M. Rajesh, R. Balakrishnan, V. L. Anh, V. Prabahar *et al.*, “Complete coverage path planning using reinforcement learning for tetromino

based cleaning and maintenance robot," *Automation in construction*, vol. 112, p. 103078, Apr. 2020.

[16] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample Efficient Actor-Critic with Experience Replay," *arXiv e-prints*, p. arXiv:1611.01224, Nov. 2016.

[17] C. Olah, "Understanding lstm networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed: 2020-09-01.

[18] L. Piardi, J. Lima, A. I. Pereira, and P. J. C. G. d. Costa, "Coverage path planning optimization based on q-learning algorithm," in *International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2018)*, vol. 2116 (1), Rhodes, Greece, Jul. 2019, pp. 220 002–1–220 002–4.

[19] X. D. Gong, B. Ding, J. Xu, H. M. Wang, X. Zhou, and D. w. Feng, "Parallelized synchronous multi-agent deep reinforcement learning with experience replay memory," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, San Francisco East Bay, CA, USA, USA, Apr. 2019, pp. 325–330.

[20] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage," *arXiv e-prints*, p. arXiv:1803.07250, Mar. 2018.

[21] A. Greenwald, K. Hall, and R. Serrano, "Correlated q-learning," in *ICML*, vol. 3, 2003, pp. 242–249.

[22] C. H. Papadimitriou and T. Roughgarden, "Computing correlated equilibria in multi-player games," *Journal of the ACM (JACM)*, vol. 55, no. 3, p. 14, 2008.

[23] X. D. Wang and V. L. Syrmos, "Coverage path planning for multiple robotic agent-based inspection of an unknown 2d environment," in *2009 17th Mediterranean Conference on Control and Automation*, Thessaloniki, Greece, Jun. 2009, pp. 1295–1300.

[24] S. Lin and B. W. Kernighan, “An effective heuristic algorithm for the traveling salesman problem,” *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[25] D. C. Guastella, L. Cantelli, G. Giammello, C. D. Melita, G. Spatino, and G. Muscato, “Complete coverage path planning for aerial vehicle flocks deployed in outdoor environments,” *Computers & electrical engineering*, vol. 75, pp. 189–201, 2019.

[26] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, “Morse decompositions for coverage tasks,” *The International Journal of Robotics Research*, vol. 21, pp. 331–344, Apr. 2002.

[27] B. K. W. G. L. A. Janchiv, D. Batsaikhan and S.-G. Lee, “Time-efficient and complete coverage path planning based on flow networks for multi-robots,” *IJCAS*, vol. 11, pp. 369–376, 2013.

[28] S. M. LaValle, “Boustrophedon decomposition,” http://planning.cs.uiuc.edu/node352.html, accessed: 2020-09-02.

[29] R. Olfati-Saber, “Flocking for multi-agent dynamic systems: algorithms and theory,” *IEEE Trans. on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006.

[30] Y. Miao, A. Khamis, and M. S. Kamel, “Applying anti-flocking model in mobile surveillance systems,” in *22010 IEEE International Conference on Autonomous and Intelligent Systems (AIS)*, Povoa de Varzim, 2010, pp. 1–6.

[31] S. H. Semnani and O. A. Basir, “Semi-flocking algorithm for motion control of mobile sensors in large-scale surveillance systems,” *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 129–137, Jul. 2014.

[32] W. M. Yuan, N. W. Ganganath, C.-T. Cheng, G. Qing, and F. C. M. Lau, “Semi-flocking-controlled mobile sensor networks for dynamic area coverage and multiple target tracking,” *IEEE sensors journal*, vol. 18, no. 21, pp. 8883–8892, Nov. 2018.

[33] N. W. Ganganath, C.-T. Cheng, and C. K. Tse, “Distributed anti-flocking control for mobile surveillance systems,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, Portugal, May 2015, pp. 1726–1729.

[34] M. Ashraf, “Reinforcement learning demystified: A gentle introduction,” https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14, accessed: 2020-09-02.

[35] Wikipedia contributors, “Markov decision process — Wikipedia, the free encyclopedia,” 2006, [Online; accessed 02-September-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Markov_decision_process

[36] ——, “Bellman equation — Wikipedia, the free encyclopedia,” 2007, [Online; accessed 02-September-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Bellman_equation

[37] ——, “Dynamic programming — Wikipedia, the free encyclopedia,” 2004, [Online; accessed 02-September-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Dynamic_programming

[38] S. Tanwar, “Bellman equation and dynamic programming,” https://medium.com/analytics-vidhya/bellman-equation-and-dynamic-programming-773ce67fc6a7, accessed: 2020-09-02.

[39] A. Choudhary, “Reinforcement learning: Introduction to monte carlo learning using the openai gym toolkit,” https://www.analyticsvidhya.com/blog/2018/11/reinforcement-learning-introduction-monte-carlo-learning-openai-gym/, accessed: 2020-09-02.

[40] Wikipedia contributors, “Temporal difference learning — Wikipedia, the free encyclopedia,” 2007, [Online; accessed 02-September-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Temporal_difference_learnin

[41] ——, “Model-free (reinforcement learning) — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 02-September-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Model-free_(reinforcement_learning)

[42] C. Shyalika, “A beginners guide to q-learning,” https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c, accessed: 2020-09-02.

[43] Samishawl, “Epsilon-greedy algorithm in reinforcement learning,” https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/#:~:text=Epsilon%2DGreedy%20is%20a%20simple,a%20small%20chance%20of%20exploring., accessed: 2020-09-02.

[44] H. M. La, R. Lim, and W. Sheng, “Multirobot cooperative learning for predator avoidance,” *IEEE Trans. on Control Syst. Technol.*, vol. 23, no. 1, pp. 52–63, Jan. 2015.