

Multi-Agent Reinforcement Learning for Non-uniform Area Coverage

1st Jainik Mehta
Aerospace Engineering
IIT, Bombay
Mumbai, India
jainikjmehta@gmail.com

2nd Arpita Sinha
System and Control Engineering
IIT, Bombay
Mumbai, India
arpita.sinha@iitb.ac.in

3rd Animesh Chakravarthy
Mechanical and Aerospace Engineering
University of Texas
Arlington, USA
animesh.chakravarthy@uta.edu

Abstract—In this paper, we introduce a method for generating close to optimal trajectories for multiple autonomous agents so that they can cooperatively carry out non-uniform area coverage missions with shortest travel length without colliding with any static obstacles in their path. We have used ResNet architecture as Q-value function approximator to learn centralized action specific multi-agent policies through independent learning method. A unique reward model is proposed to solve problem of overshooting trajectories and to enforce collaboration among agents. Constraints on the agent movement were introduced to avoid oscillations and with aim to use MPC controller in future to guide the UAV along the trajectory generated by our algorithm. Our algorithm is capable of generating trajectories for large area coverage missions and supports arbitrary number of agents with minimal training effort. The agent paths are generated offline, thus, there is no need for communication between the agents. The details regarding selecting particular network architecture are discussed along with the analysis and simulation results to demonstrate the applicability of the proposed algorithm.

Index Terms—Multi-Agent Deep Reinforcement Learning, Non-uniform Coverage Path Planning, Priority Patrolling, Multi-UAV System

I. INTRODUCTION

Coverage path planning is a task that consists of computing collision-free paths that pass through all defined points of an area or volume of interest [1]. Non-uniform coverage path planning is an extension of the coverage path planning problem, which involves adjusting the frequency of coverage for different areas based on specific needs. This is particularly relevant in real-world scenarios where different parts of the environment require varying levels of attention. For example, during a heatwave, certain areas of an agricultural field might be more susceptible to damage and may require more spraying. Non-uniform coverage allows for increased focus on these critical zones. Another example where non-uniform area coverage is required is during patrolling an area with high-priority zones requiring more frequent visits [2] [3]. This allows robots or security personnel to prioritize these areas while still ensuring adequate coverage of the entire patrol area. An even further extension of coverage path planning is moving from a single-agent case to multiple agents. Non-uniform area coverage with multiple agents will obviously be faster than coverage using a single agent when coverage area is too big.

For example, let's again consider the scenario of an ongoing heatwave where the entire field needs to be sprayed quickly to protect the crops. In such cases, it is much more time-efficient to use multiple coordinated UAVs for a simultaneous spraying operation rather than a single UAV [4]. Additionally, for large-sized fields, the amount of spray required can be substantial, and if the entire spray is carried by a single UAV, it would necessitate a large and expensive UAV. As an important alternative, one could split the quantity of the spray so that it is carried by multiple smaller coordinated UAVs. Furthermore, multiple agents improve the robustness of the mission in case any agent fails in the middle of the coverage operation.

In its simplest form the multiagent non-uniform coverage task can be modeled as the multi-agent multi-target (MAMT) path planning problem, where multiple agents must plan optimal paths to reach several stationary targets. An agent may have more than one target, but once a target is reached, it does not need to be visited again. The simplest case of a multi-agent non-uniform area coverage task is shown in Figure 1, where two agents are spawned at predefined positions and orientation and are required to visit the cells marked in gray once. MAMT path planning problems are NP-hard [5]. If we consider that each agent has multiple targets that need to be visited more than once, finding the optimal path becomes even more challenging. This will be the focus of this research. Here, we define the optimal path as the one that minimizes the combined path length traversed by all agents to cover the area non-uniformly. In order to find the optimal path, the agents must collaborate with each other and intelligently allocate tasks [6]. The problem shown in Figure 1 can also be modeled as a generalized Multi-Depot Multiple Traveling Salesmen Problem (MDMTSP) where salesmen are initially based at different depots and are tasked to visit several location one after another without requiring them to return to their bases. This can be solved using Linear Programming method with bipartite network flow-based model of the problem [7]. This method also accounts for the fixed cost that can be associated with activation of given salesman if there is no requirement that every salesman be activated. This is then included in the cost-minimization objective of the problem, along with (or in lieu of) the usual total inter-site travel costs. A problem similar to the one shown in Figure 1 involves the motion planning of

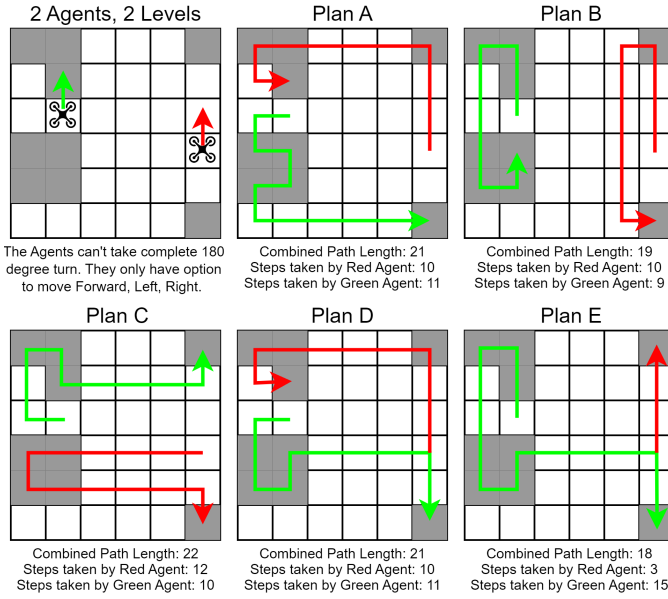


Fig. 1. The figure illustrates possible trajectories for two agents tasked with visiting each gray cell once. Plan E achieves the shortest combined path length. However, in this plan, one agent appears to prioritize shorter travel after covering only one cell, leaving more work for the green agent. Conversely, Plan A, Plan B and Plan D demonstrates equal effort from both agents in traveling and covering cells, but it results in a longer path.

m Dubins' vehicles through n points in a plane was addressed using combination of modified Lagrangian heuristics and Dynamic Programming scheme [8]. The goal of this problem was to select $q(\leq m)$ Dubins' vehicles and determine their motion plans so that all points are visited, minimizing the total cost of the tours among all possible vehicle choices and their respective routes. There are several other methods for solving MDMTSP or MAMT type problems such as meta heuristic based ant colony optimization [9] or heuristic based on a biased random-key genetic algorithm [10]. A combination of actor-critic based reinforcement learning and a dynamic programming (DP) algorithm was also explored [11] for problem of similar type. This approach first used DP-based method to find an exact, analytical optimal solution. Then using the high-quality training data from the DP method a two-stage reinforcement learning method with ELM initialization of actor and critic parameters was trained which significantly improved learning performance. However, when extending the problem shown in Figure 1 to a large number of agents and a vast coverage area where each cell requires multiple visits, previously mentioned algorithms become impractical. This is because such problems cannot be accurately described or formulated within any existing algorithmic frameworks, and even if they could, they would likely require exponentially large computation times. Another approach to solving the problem where each cell requires multiple visits is to model the non-uniformity in the coverage area as a potential function and then compute the gradient of this potential field to guide agent movement [12] [13]. However, this method does not explicitly encourage cooperation between agents, leading to

suboptimal solutions. Moreover, this approach is problematic when derivatives are not available or when there are multiple local optima. Therefore, instead of searching a high-dimensional space or performing computationally expensive gradient calculations on complex potential fields to find the optimal solution, a more efficient approach is to learn how to generate trajectories by observing the global system state. This state includes all information regarding completed and remaining coverage, as well as the position and orientation of the agents along with their past actions. This allows an agent to determine which area to cover (task allocation) and which action to take next by observing the global system state. This problem can then be translated into a Reinforcement Learning problem, where the actions chosen by individual agents and the rewards they receive for those actions are factored into the planning process and used to update the policies. In this paper, we pursue this approach by encoding specific information about the global system state in an image-like fashion, with multiple channels representing different information. We then use a convolutional neural network as a Q-value function approximator to learn multi-agent policies [14]. We show that the learned policies can be generalized to new coverage shapes and sizes, as well as to previously unseen numbers of agents. The contribution of this paper is threefold. First, we developed and presented a unique method to represent the global system state of any non-uniform coverage problem while maintaining the Markovian property. Second, we created a custom DQN-based learning algorithm that uses this global system state as input, guided by a uniquely crafted reward function to achieve fast convergence. Third, we devised a series of test simulations to evaluate the efficiency of our algorithm and compared it with industry-standard algorithm. The rest of the paper is organized as follows. Section II clearly explains the problem, outlines the constraints on agents, and defines the optimization objective. In Section III, we introduce Q-learning-based deep reinforcement learning and extend it to the multi-agent case. We explain how using convolution neural networks (CNNs) along with an image-like representation of state preserves the agent's entire history while maintaining the Markovian property. We also discuss the algorithm used for training the networks and the training settings. In Section IV, we present numerical results demonstrating the efficiency of our proposed algorithms in several test cases. Finally, Section V concludes the paper and suggests possible future directions for expanding the present research work.

II. OPTIMIZATION SETUP

Our optimization problem focuses on the non-uniform spraying of agricultural fields by a fleet of N_a UAVs as mentioned in Section I of this paper. Since information about spraying amounts typically comes from hyperspectral cameras, which capture data in continuous form, preprocessing the captured images is crucial for our method's application. This involves removing unnecessary information from the images and discretizing them into finite spray levels. An example of this image preprocessing is shown in Figure 3. Once we

have the spray levels associated with each grid cell as well as the initial positions of each UAV, we can start implementing our method. However, to solve the problem of a continuous domain in a discretized domain setting, we need to make several assumptions about the problem, which are stated below:

- We assume that a field of any shape and size can be discretized and represented within a square grid, where each grid cell represents the spraying amount for that area. Thus, our current implementation focuses on generating trajectories for square fields but can be easily extended to generate trajectories for fields of any shape and size.
- We assume that the number of UAVs is less than the number of grid cells in a row of the matrix. This assumption is necessary to prevent the algorithm from halting if all UAVs align along the same border of the square field. This situation is shown in Figure 4
- We assume that all UAVs travel at similar speed and that all UAVs takes equal time when moving from one grid cell to another grid cell.
- We assume that the minimum turning radius of a UAV should be less than or equal to half the length of the square grid cell used for discretization. Additionally, the nozzle spray footprint of the UAV should fit within the square grid cell. While these are not strict requirements for executing our method, considering them when choosing the grid cell size will help minimize spray errors.

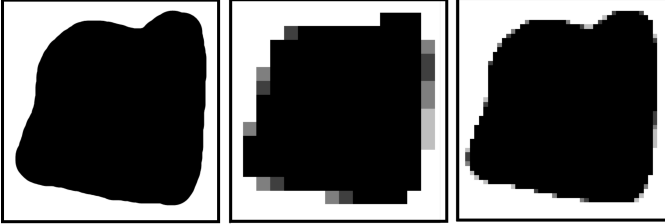


Fig. 2. The image shows a field of arbitrary shape that that field of any shape and size can be represented in discretized square grid. The finer the discretization lesser will be the error in spray.

III. MULTI AGENT REINFORCEMENT LEARNING

To train our agents to perform non-uniform area coverage task we used independent learning method to train Deep Q-Networks with experience replay method [15]. The basic idea behind DQN is to estimate the action value function, by using the Bellman equation as an iterative update. To train Deep Q-Networks (DQN) using experience replay, we need to collect and store tuples of state s , action a , next state s' , and reward r . At each step, the current transition is added to the replay buffer, and some transitions are randomly sampled from the buffer to train the network. The following equations illustrate

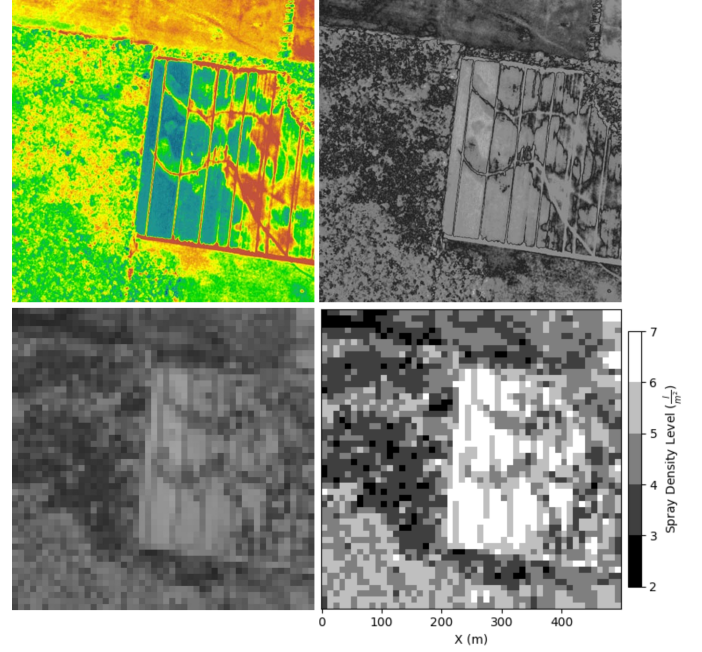


Fig. 3. The hyperspectral image is first converted into a black and white image. Then, it is discretized using a specific grid size. During discretization, the average value of all pixels within each grid cell is considered to fill that cell. Finally, all grid cell values in this discretized image are rounded to the closest spray level based on the spraying rate of the UAV.

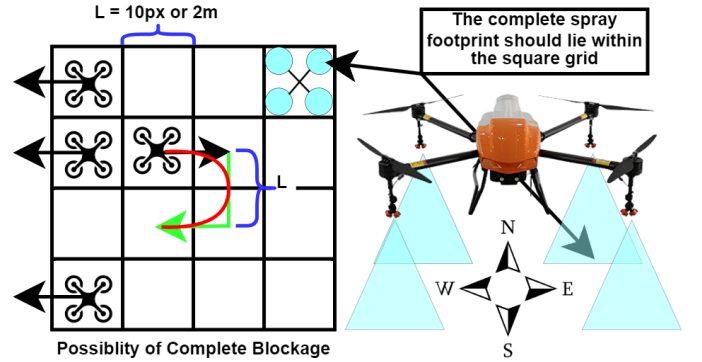


Fig. 4. Figure demonstrates that an agent heading East can only choose actions Forward (East), Left (North), or Right (South) but is prohibited from selecting actions opposite its heading. This constraint is imposed to prevent oscillations and ensure efficient coverage of designated cells. Figure also illustrates the possibility that all agents may align in a manner where no valid actions remain, particularly if number of grid cell along the side of the field matches or is smaller than the number of agents. Furthermore the grid size used to discretize the image should accommodate the nozzle spray footprint, and the UAV's minimum turning radius should be half the size of the square grid's side.

the process of updating the network parameters:

$$\delta = r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \quad (1)$$

$$\lambda = \frac{1}{2} \delta^2 \quad (2)$$

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \lambda \quad (3)$$

Here, δ in Equation 1 represents the temporal difference error, which is the difference between the target Q-value and the current Q-value. The loss function λ in Equation 2 is the mean squared error of the temporal difference error. Finally, the

parameters θ of the Q-network are updated using the temporal difference error and the gradient of the loss function with respect to the network parameters as shown in Equation 3.

Extending Deep Q-Networks (DQN) to multi-agent scenarios requires carefully crafting the state s to encompass all decision-making information while maintaining a Markovian property from a single agent's perspective, ensuring environmental stationarity in independent learning methods [16] [17]. We achieve this by combining spray maps (representing achieved and remaining coverage), agent positions, orientations, and past trajectories into a single matrix-like image representation. This approach allows us to represent any number of agents without altering the ResNet architecture used to approximate the Q-value function. The ResNet architecture is chosen because the image representation of the state allows us to leverage Convolutional Neural Networks (CNNs), which are highly effective for image classification tasks [18]. In our case, CNNs can be extended to select a particular action from a discrete set of actions. At each step, the global system state s is input into a specific residual network chosen from a set of 14 different residual networks based on the feasible action set $A \subseteq \{N, S, E, W, Stay\}$ to determine the action a of a particular agent. Using 14 different ResNets instead of one speeds up the training process and improves network efficiency by eliminating the need for the network to learn which actions are invalid given the current global system state.

Upon selecting and executing the action a with the largest Q-value, we receive a reward r , and the environment's state may change. We then update all matrices, resulting in the next image-like state s' . It is important to note that this formulation assumes homogeneous systems, where all agents share the same action space if they are in the same state. In other words, agent's action space is dependent on agents' positions and orientations rather than on the agents themselves. The matrix like image tensor representing global system state at particular step is of size $8 \times W \times H$, where W and H are the height and width of our two dimensional coverage map and eight is the number of channels in the image. Each channel encodes a different set of information from the global system state as its pixel values (shown in Figure 5). The channels can be broken down in the following way:

- **Active agent position and orientation channels:** contains information about the position (denoted by pixel value of -2) and orientation (denoted by 1 if heading is north) of the agent whose actions are being determined. The active agent information is represented differently allowing the network to interpret and decide the importance of this input channel.
- **Inactive agents position and orientation channels:** contains information about the position (denoted by pixel value of -1) and orientation of all inactive agents whose actions are not being determined as well as all the obstacles (denoted by pixel value of -5) in the grid map.
- **Achieved and remaining spray channels:** contains the information about the total spray level achieved so far by all agents and remaining amount of spray required

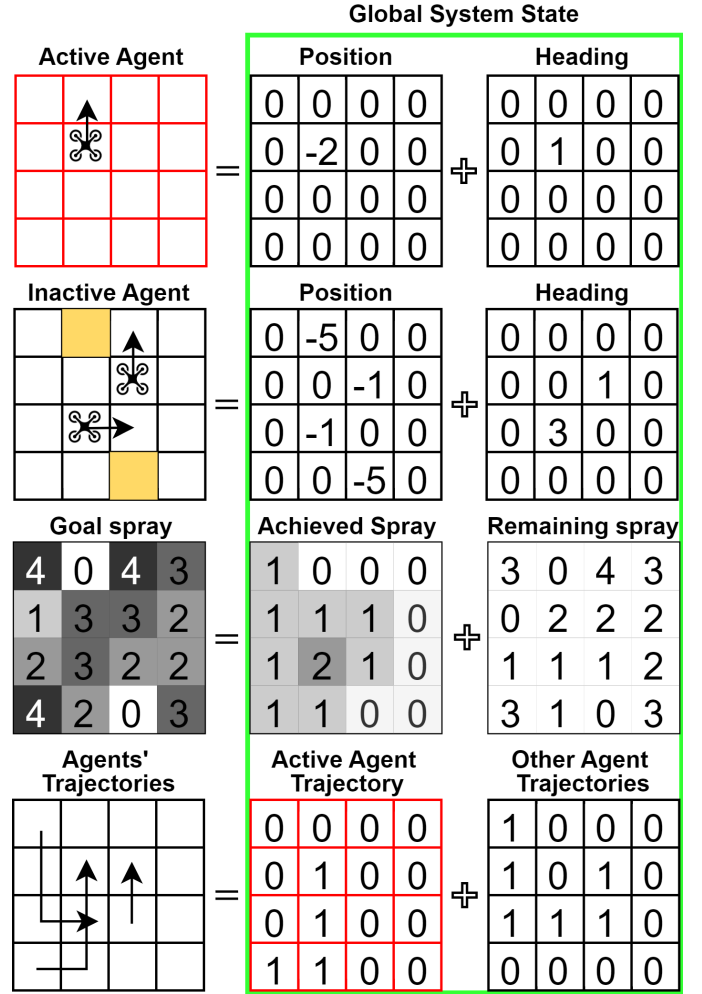


Fig. 5. The figure shows how global system state can be represented in image like fashion containing various channels. These eight matrices (inside the green box) forms the global system state at particular time step.

at each pixel position. The value of each pixel in the achieved spray channel can be calculated based on the spray rate of the UAV.

- **Active and inactive agents trajectories channels:** contains information about the past trajectories of the active agent and all other agents. This channel tracks each agent locations: cells visited are filled with 1, and each additional visit increases the value by 1.

A. Algorithm

We employ Deep Q-learning to train 14 separate action set specific Q-value function approximators based on the ResNet architecture as shown in Figure 6 (a). At each time step the number of valid actions of an active agent can only vary between 1 and 4. A particular ResNet is chosen based on feasible actions of active agents. The input to the ResNet consists of an $W \times H \times 8$ image produced by preprocessing function ϕ (defined later in Table I). This image is passed through several convolutional layers within the ResNet architecture, each containing multiple residual blocks. While experimenting, we found that convolution parameters like strides and padding had

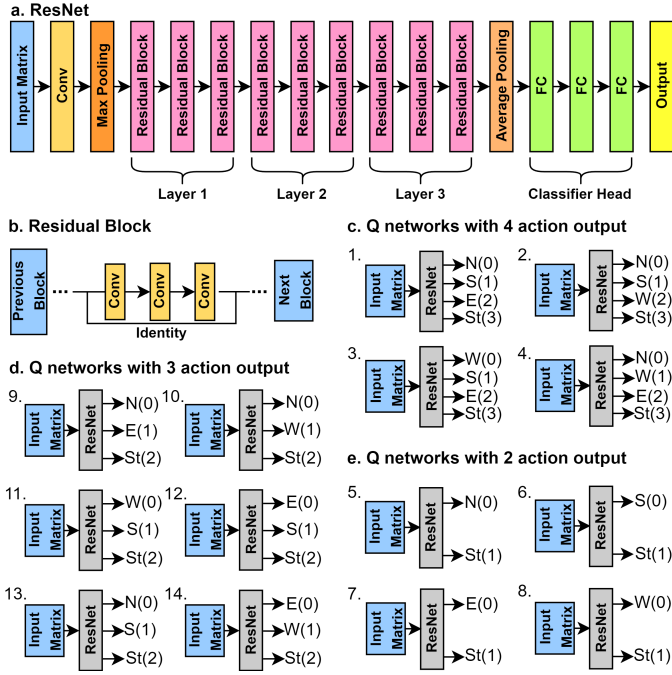


Fig. 6. Figure (a) illustrates a block diagram of a typical ResNet architecture used as a Q-value function approximator. The architecture consists of three layers and a classifier head. Figure (b) of the diagram depicts the number of convolution steps within each block. Figure (c), (d), and (e) showcase variations of the last layer of ResNet architecture, with different numbers of actions as output (four, three, and two, respectively) collectively they form 14 action set specific multiagent policies.

minimal impact on the network's output and were therefore set to 1. Similarly, the kernel size was chosen to be of size 3 for all convolution operations. The number of filters, however, varies across the network layers, ranging from 64 to 128. The classifier head layer consists of three fully-connected layers where the hidden layer consists of 86 rectifier units and the last layer contains a different output for each valid action set as shown in Figure 6 (c, d, e). If an agent has only one possible action, it executes it directly without storing the experience in the replay buffer. The training process involves thousands of episodes. Each agent stores its experience tuples in a separate replay buffer associated with its corresponding ResNet. At the beginning of each episode, the random non-uniform occupancy grid generated at the beginning of training process gets reset to its original configuration, and random initial positions and heading directions for each agents are assigned. To encourage exploration in the early stages, we employ a greedy policy π with a high initial exploration rate ϵ . This exploration rate is gradually decreased over time. Each agent evaluates its actions based on the overall reward and updates its approximated value function for the current state and action using (Equation 3).

B. Local, Global and Future Reward

The reward distribution operates in a hybrid setting, where the reward setup is fully cooperative at the global level but competitive at the individual or local level [19]. Each agent receives a local reward (r_l), a global reward (r_g), and a future

reward (r_f) for each action it takes. The sum of these local, global, and future rewards constitutes the overall agent reward (r_i). It should be noted that there are no negative rewards. These three rewards are evaluated based on a few crucial and intuitive questions that helps in shaping the overall reward function and are as follows:

- Is the effort taken by a given agent to obtain a reward worth it, or should it leave that reward for other agents to minimize the reward path length trade-off? This can be described mathematically as:

$$r_l = r_k / (\chi \cdot N_a) \quad (4)$$

Here, r_k is the positive reward obtained by agent i at time step t , divided by the path length χ traveled by agent i to obtain that reward from the last positive reward. This value is further divided by the total number of agents N_a to normalize the local reward (r_l). The reward r_k given to an agent is equal to the level of the cell it visits. For instance, if the cell level is 5, then $r_k = 5$; if the cell level is 0, then $r_k = 0$.

- Is the reward awarded to a given agent distributed at the global level? Does the action of a given agent result in the combined path length being minimized? This can be described mathematically as:

$$r_g = r_t / \mu \quad (5)$$

Here, r_t is the combined total reward obtained by all agents up to time step t , divided by the combined total path length μ of all agents up to time step t .

- Are agents motivated to cover cells that are far away from all their current positions? Do future rewards hold the same relevance to agents as current rewards? This can be described mathematically as:

$$r_f = (M/\zeta) \quad (6)$$

The total number of cells in the entire coverage grid M over total future reward left ζ .

- Therefore overall reward that agent i will receive at time step t will be calculated using:

$$r_i = r_k / (\chi \cdot N_a) + r_t / \mu + (M/\zeta) \quad (7)$$

C. Training

We trained our ResNet-based Q-value function approximators for different image input sizes (ranging from grid sizes 8×8 to 256×256) and varying spray levels (from 4 to 16). Figure 7 shows the training progress for different map sizes and agent numbers. We observed that the total combined path length for grid maps with more agents but the same spray requirements is shorter, indicating that agents learned to allocate tasks evenly by the end of training. Additionally, the training time for maps with more agents is slightly lower, as more agents quickly cover the spray requirements, leading to shorter episode run times. Training 250,000 episodes took only 573 minutes. The total training time and per-episode

Algorithm 1 Multi-agent Q-learning with Multiple ResNets

Initialize Hyperparameters
Initialize Replay memory D_j to capacity D_N and
Instantiate Q_j and target Qt_j ResNets; $j = 1, 2, \dots, 14$
Define loss criterion: $\lambda \leftarrow \text{MSELoss}()$
Initialize: optimizers $\leftarrow$ Adam; Spray matrix S_M
for $\text{eps} = 1, 2, \dots, \text{max_eps}$ **do**
 Set $\text{done} = \text{False}$; **Reset** S_M
 Randomly initialize Agents location and heading
 matrices A_M ; $s_t = \{S_M, A_M\}$
 while $\text{done} \neq \text{True}$ **do**
 Initialize Empty buffer D_t of size N_a
 for $i = 1, 2, \dots, N_a$ **do**
 Preprocess sequence $\phi_{i,t} = \phi(s_t)$ ▷ #1
 Find all possible actions $a_{i,p}$ of agent a_i :
 $a_{i,p} = f_p(\phi_{i,t})$ ▷ #2
 Select Q network; $Q_j = f_{aq}(a_{i,p})$ ▷ #3
 With probability ϵ select action $a_{i,act} \in a_{i,p}$
 else select: $a'_{i,act} = \arg \max_{a_{i,p}} Q_j(\phi_{i,t}, a_{i,p}; \theta)$
 and map action $a_{i,act} = f_{na}(a'_{i,act})$ ▷ #4
 Execute action $a_{i,act}$ and observe reward r_i and
 next state $s_{t+1} = \{S'_M, A'_M\}$
 Preprocess $\phi_{i,t+1} = \phi(s_{t+1})$
 Store tuple $(\phi_{i,t}, a'_{i,act}, r_i, \phi_{i,t+1})$ in D_t
 end for
 for $i = 1, 2, \dots, N_a$ **do**
 $\phi_{i,t+1}, \phi_{i+1,t+1}, \dots, \phi_{N_a,t+1} \leftarrow D_t$ ▷ #5
 Find all possible next actions $a_{i,p+1}$
 $a_{i,p+1} = f_{p+1}(\phi_{i,t+1}, \phi_{i+1,t+1}, \dots, \phi_{N_a,t+1})$
 Select Target network $Qt_i = f_{aq}(a_{i,p+1})$ ▷ #6
 $a'_{i,act+1} = \arg \max_{a_{i,p+1}} Qt_i(\phi_{i,t+1}, a_{i,p+1}; \theta)$
 $T_f = \max_{a_{i,p+1}} Qt_i(\phi_{i,t+1}, a_{i,p+1}; \theta)$
 and map action $a_{i,act+1} = f_{na}(a'_{i,act+1})$
 Store tuple $(\phi_{i,t}, a'_{i,act}, r_i, \phi_{i,t+1}, T_f)$ in D_j
 Update $\phi_{i,t+1} = f_{calc}(\phi_{i,t+1}, a_{i,act+1})$ ▷ #7
 $D_t \leftarrow \phi_{i,t+1}, \phi_{i+1,t+1}, \dots, \phi_{N_a,t+1}$
 end for
 for $j = 1, 2, \dots, 14$ **do**
 if $\text{length}(D_j) \geq \text{Batch size (k)}$ **then**
 Sample random minibatch of transitions
 $(\phi_k, a'_{k,act}, r_k, \phi_{k+1}, T_{f,k})$ from D_j ▷ #8
 $H_f = (Q_j(\phi_k, a'_{k,act}))$
 $H'_f = f_{tar}(H_f, a'_{k,act}, T_{f,k}, r_k)$ ▷ #9
 Backpropagate loss: $\lambda(H_f, H'_f)$
 $++ \text{net_update}_j$
 end if
 if $\text{net_update}_j \geq t_f$ **then**
 $\theta(Qt_j) \leftarrow (1 - \tau) \cdot \theta(Qt_j) + \tau \cdot \theta(Q_j)$ ▷ #10
 $\text{net_update}_j = 0$
 end if
 end for
 end while
end for

TABLE I
DESCRIPTION OF COMMENTS IN ALGORITHM 1

Label	Description
1	Function ϕ transforms the location and heading matrix of the i^{th} active agent to differentiate it from the location and heading matrices of other agents. An example of the output from this transformation function is shown in Figure 5
2	Function f_p takes the global system state $\phi_{i,t}$ of agent a_i and finds all possible actions $a_{i,p}$ that agent a_i can take, given its current position and heading $a_m \in A_M$ and the current position of obstacles around it. For example, the active agent in Figure 5 can only take action to the right or stay at its current position; thus, the set of all possible actions in this case is $a_{i,p} = [\text{West}, \text{Stay}]$.
3	Select network Q_j out of 14 ResNet based on the set of all possible actions $a_{i,p}$ that agent a_i can take. Function f_{aq} maps $a_{i,p}$ to Q_j . Following the previous example where $a_{i,p} = [\text{West}, \text{Stay}]$, f_{aq} outputs 8.
4	Function f_{na} maps the network output to an action. For example, when the selected network is 14 and the network output is 0, this implies the policy network tells the agent to go West.
5	To calculate the Target Q value for the next state-action pair for agent a_i , we must retrieve the next states of all the agents. This is because after executing actions for each agent, their states may change; therefore, relying just on the next global state of the active agent will not result in the true Target Q value.
6	After finding the set of all possible next actions $a_{i,p+1}$ of agent a_i using function f_{p+1} , we select the target Q network Qt_i out of 14 different target Q networks, each corresponding to 14 Q networks.
7	The function f_{calc} calculates the next-to-next state of agent a_i based on the chosen action $a_{i,act+1}$ and plugs it back into the next state of agent a_i . This step might feel non-intuitive at first; however, since we are considering only one agent at a time, its future action will affect the Target Q value of the next agent.
8	After the length of the replay buffer D_j for a particular Q network Q_j exceeds the predefined batch size (k), we sample a mini-batch of state transition tuples from D_j .
9	H_f is the set of Q values for each state-action pair, and H'_f is the set of target Q values. Basically, function f_{tar} replaces the Q value of the selected action $a'_{k,act}$ with the sum of the target Q value T_f and the obtained reward r_k . Here, H'_f represents the first two terms of Equation 1 and H_f represents the last term of Equation 1
10	After t_f updates of network Q_j , we update the parameters of the target network Qt_j with the update factor τ .

duration depend significantly on the map used. All map sizes had evenly spread obstacles and spray levels, with each spray level appearing equally often. To estimate the total combined number of steps required for agents to cover a given non-uniform coverage map when they are spawned at random positions, we calculated the average of the total combined path length for the last few hundred episodes after training stabilized. We compared our training results with the commonly used boustrophedon decomposition path planning method [20], which was modified to allow for our agent turning constraints [21]. This method was further extended by optimizing dif-

TABLE II
DEEP Q-NETWORK HYPERPARAMETERS [8×8 GRID; 4 LEVELS]

Hyperparameters	Value	Description
max_eps	20000	The maximum number of episodes
B	128	Batch size i.e. number of training samples per update
D_N	100000	Replay size of the experience replay dataset
l_r	0.01	Learning rate used by the ADAM optimizer
ϵ	1.0	Initial exploration rate in ϵ greedy exploration policy
ϵ_d and ϵ_{min}	0.9995; 0.001	Exploration decay rate and minimum exploration rate
t_f	500	Frequency of updating the target network

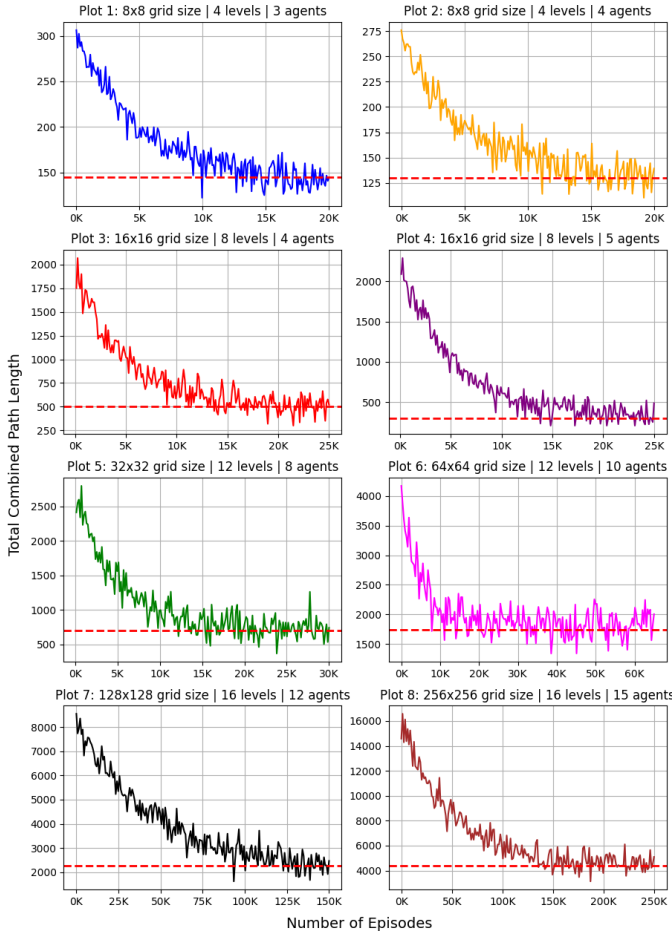


Fig. 7. The figure shows the number of episodes needed to train ResNet-based Deep Q-Network policies for different environments, along with the average path length (indicated by a red dotted line) calculated after training stabilizes. We observe that training in an environment with more agents leads to faster convergence and shorter total combined path lengths, which is an expected outcome.

ferent sweep combinations to find the optimal sweep path for multiple agents and considers obstacles during transitions between cells [22] [23]. We found that for all the training

cases, our method produced shorter path lengths compared to this modified boustrophedon decomposition method with constraints (BDWC). The training results are summarized in Table III, which shows that the BDWC method produced extremely long path lengths for every training example. Our method produced results that were 10 times better than those of the BDWC method for map sizes greater than 16×16 . The training was conducted on a computer with an Intel Core i7 processor, 64 GB of RAM, and an Nvidia RTX 4080.

TABLE III
TRAINING RESULTS FOR DIFFERENT MAPS AND NUMBER OF AGENTS

Map Size (Cell Levels)	Agnts	Training Time	Epi-sodes	MARL Length	BDWC Length
8×8 (4)	3	41 mins	20K	138	304
8×8 (4)	4	38 mins	20K	130	417
16×16 (8)	4	84 mins	25K	312	1987
16×16 (8)	5	80 mins	25K	290	2201
32×32 (12)	8	139 mins	30K	718	11501
64×64 (12)	10	147 mins	65K	1712	16060
128×128 (16)	12	256 mins	150K	2457	28103
256×256 (16)	15	573 mins	250K	4331	45507

IV. RESULTS

After training our 14 ResNets on different grid sizes and varying spray levels, we tested the network's performance on several completely different non-uniform spray maps that the network had never seen before. These maps varied in size, the number of agents, and spray levels. To quantify and measure the accuracy of our method, we created 100 test non-uniform spray maps for each grid size and spray combination and evaluated the trajectories generated by our trained model. Some of these test results are shown in Table IV. The test cases were designed to include varying distributions of different spray levels.

TABLE IV
TESTS RESULTS FOR DIFFERENT MAPS AND NUMBER OF AGENTS

Map Size (Cell Levels)	Agents	MARL Length	BDWC Length	MARL Extra Visits	BDWC Extra Visits
8×8 (6)	1	454	704	12	61
8×8 (6)	4	231	417	6	96
16×16 (5)	6	563	2781	33	2162
16×16 (8)	8	939	3205	20	2335
16×16 (10)	8	1820	1701	25	908
16×16 (12)	10	1930	16060	13	14143
32×32 (5)	10	2451	30103	40	27692
32×32 (12)	12	3831	38507	54	34730
32×32 (16)	15	3022	11501	37	8516
64×64 (12)	18	2724	15059	91	12426
64×64 (16)	20	2919	28919	53	26053
128×128 (10)	15	3104	34019	81	30996
128×128 (12)	25	3510	29013	122	25625
128×128 (16)	30	2817	34109	101	31393
256×256 (10)	10	2963	31045	108	28910
256×256 (16)	35	4901	45020	89	40208

We found that for all test cases, our method produced significantly shorter path lengths compared to the BDWC method. One example test case is shown in Figure 8, where

three agents are located in different areas of the map and are required to spray regions with a spray level requirement of 4. The result generated by the BDWC method is shown in Figure 9, where it is evident that agents travel extra path lengths and visit cells that were either already filled or didn't have any spray requirement. In contrast, the result generated by our method, shown in Figure 10, clearly demonstrates that agents visit fewer cells that were either already filled or didn't have any spray requirement, thus significantly minimizing the total combined path length. Additionally, the task allocation between agents is better, and agents seem to know which cells to fill and which to leave out.

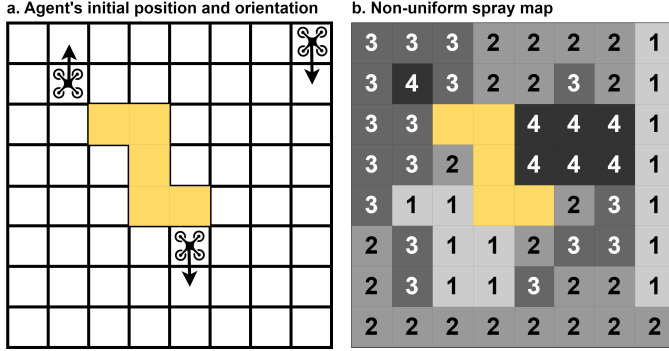


Fig. 8. The Figure (a) shows a test scenario with 3 agents and Figure (b) describes a nonuniform spray map with 4 level spray requirements that these agents need to cover.

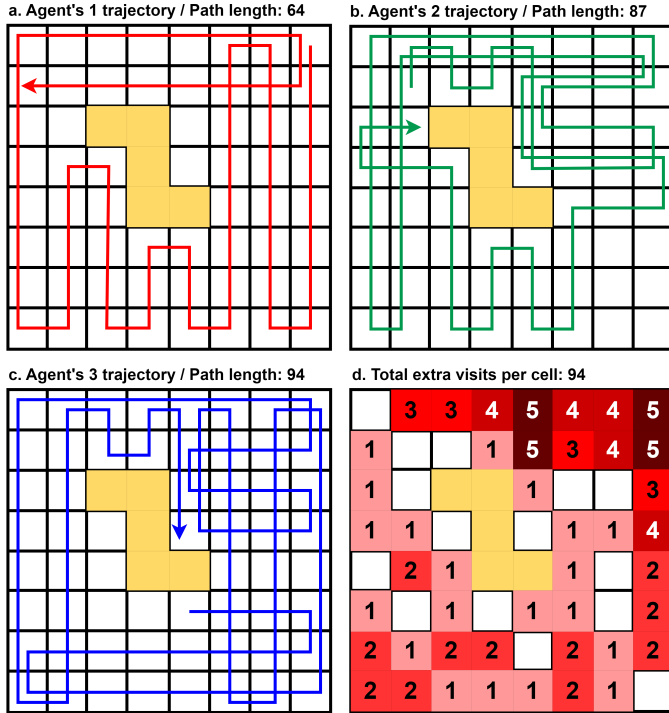


Fig. 9. Figures (a), (b), and (c) show the BDWC algorithm's trajectories for each agent covering the nonuniform spray map in Figure 8. Figure (d) shows excessive steps by all agents, resulting in a total path length of 245.

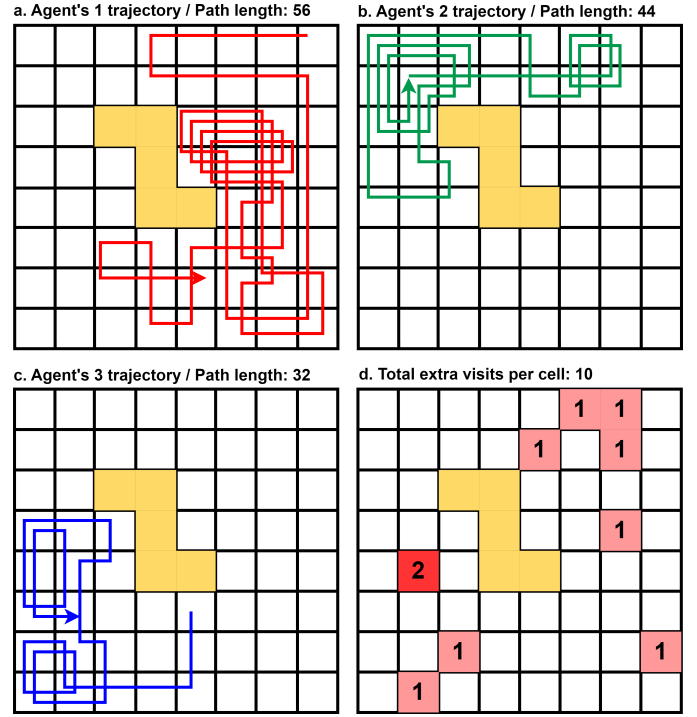


Fig. 10. Figures (a), (b), and (c) show ResNet-generated trajectories on an 8×8 grid covering the nonuniform spray map in Figure 8. Figure (d) shows fewer extra visits than the BDWC algorithm, with tasks allocated efficiently within specific regions.

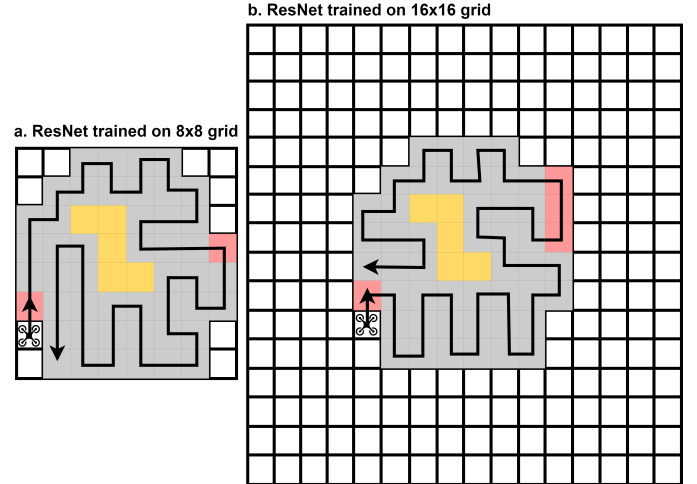


Fig. 11. Figure (a) shows the trajectory generated by a ResNet trained on an 8×8 grid to visit the gray cells once, while Figure (b) shows the path from a ResNet trained on a 16×16 grid. The trajectory from the 16×16 grid has slightly more extra cell visits (show in red), though this is not always the case across different spray maps.

We also found that slight modifications of initial conditions, such as changing an agent's orientation and/or position, generate completely different solutions for the same non-uniform map, but the average total combined path length remains the same. Most importantly, we discovered that a ResNet trained on a larger map size of 16×16 produces equally good results compared to a ResNet trained specifically on a map size of 8×8 that covers the spray region exactly, as shown

in Figure 11. This indicates that once we train our ResNet on large grid sizes with several spray levels and multiple agents, we can use this trained model to generate paths for any smaller grid size with any spray requirement and any number of agents. We also noticed that training the network on both sparse and dense non-uniform spray maps produces better results in many test scenarios compared to a network that was only trained on either one of those non-uniform spray maps.

V. CONCLUSION

Our algorithm performed significantly better than the commonly used boustrophedon method for path planning. We also demonstrated the method's ability to adapt to unforeseen grid environments and the adaptability of our ResNet model on smaller maps, even though it was trained on larger maps with different numbers of agents. Future work will explore using more sophisticated architectures, such as multi-headed attention mechanisms, to investigate better agent collaboration. Another aspect of future work is to conduct a more thorough comparison between our value iteration-based approach and other policy iteration-based approaches. The algorithm's ability should be extended to handle sudden changes in the number of agents in case of the failure of any agent. Additionally, robustness should be ensured in terms of transition probability, meaning the agent should be able to find the optimal path even if it is deflected from the path or if other agents appear in the middle of the operation. Dynamic obstacle avoidance capability is also necessary, as flying multiple UAVs increases the chance of collisions with dynamic obstacles, such as large birds. Therefore, dynamic obstacles could be managed using an already established Model Predictive Controller (MPC) and collision cone approach [24] which will utilize the reference trajectories generated by our ResNet model along with sensory inputs to correct the path avoiding dynamic obstacles.

REFERENCES

- [1] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.
- [2] M. Jana, L. Vachhani, and A. Sinha, "A deep reinforcement learning approach for multi-agent mobile robot patrolling," *International Journal of Intelligent Robotics and Applications*, vol. 6, no. 4, pp. 724–745, 2022.
- [3] D. Mallya, S. Kandala, L. Vachhani, and A. Sinha, "Priority patrolling using multiple agents," in *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 8692–8698.
- [4] C. Ju and H. I. Son, "Multiple UAV systems for agricultural applications: Control, implementation, and evaluation," *Electronics (Switzerland)*, vol. 7, 9 2018.
- [5] F. Nawaz and M. Ornik, "Multiagent, Multitarget Path Planning in Markov Decision Processes," *IEEE Transactions on Automatic Control*, vol. 68, no. 12, p. 7560–7574, Dec. 2023. [Online]. Available: <http://dx.doi.org/10.1109/TAC.2023.3286807>
- [6] P. Sujit, A. Sinha, and D. Ghose, "Team, game, and negotiation based intelligent autonomous uav task allocation for wide area applications," in *Innovations in Intelligent Machines-1*. Springer, 2007, pp. 39–75.
- [7] M. Diaby, "Linear programming formulation of the multi-depot multiple traveling salesman problem with differentiated travel costs," *Traveling Salesman Problem, Theory and Applications. InTech*, New York, NY, pp. 257–282, 2010.
- [8] S. Yadlapalli, W. A. Malik, S. Darbha, and M. Pachter, "A lagrangian-based algorithm for a multiple depot, multiple traveling salesmen problem," *Nonlinear Analysis: Real World Applications*, vol. 10, no. 4, pp. 1990–1999, 2009.
- [9] S. Ghafurian and N. Javadian, "An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesmen problems," *Applied Soft Computing*, vol. 11, no. 1, pp. 1256–1262, 2011. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1568494610000682>
- [10] C. E. Andrade, F. K. Miyazawa, and M. G. Resende, "Evolutionary algorithm for the k-interconnected multi-depot multi-traveling salesmen problem," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 463–470. [Online]. Available: <https://doi.org/10.1145/2463372.2463434>
- [11] J. Ren, X. Huang, and R. N. Huang, "Efficient deep reinforcement learning for optimal path planning," *Electronics*, vol. 11, no. 21, p. 3628, 2022.
- [12] S. Ivić, A. Andrejčuk, and S. Družeta, "Autonomous control for multi-agent non-uniform spraying," *Applied Soft Computing*, vol. 80, pp. 742–760, 2019.
- [13] S. Ivić, B. Crnković, and I. Mezić, "Ergodicity-based cooperative multiagent area coverage via a potential field," *IEEE transactions on cybernetics*, vol. 47, no. 8, pp. 1983–1993, 2016.
- [14] M. Egorov, "Multi-agent deep reinforcement learning," *CS231n: convolutional neural networks for visual recognition*, pp. 1–8, 2016.
- [15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [16] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, no. 1, pp. 1–31, 2012.
- [17] G. J. Laurent, L. Matignon, L. Fort-Piat *et al.*, "The world of independent learners is not markovian," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 15, no. 1, pp. 55–64, 2011.
- [18] J. Liang, "Image classification based on resnet," in *Journal of Physics: Conference Series*, vol. 1634, no. 1. IOP Publishing, 2020, p. 012110.
- [19] Y. Wang, M. Damani, P. Wang, Y. Cao, and G. Sartoretto, "Distributed reinforcement learning for robot teams: A review," *Current Robotics Reports*, vol. 3, no. 4, pp. 239–257, 2022.
- [20] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and service robotics*. Springer, 1998, pp. 203–209.
- [21] J. S. Lewis, W. Edwards, K. Benson, I. Rekleitis, and J. M. O'Kane, "Semi-boustrophedon coverage with a dubins vehicle," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5630–5637.
- [22] R. Bähnamann, N. Lawrance, J. J. Chung, M. Pantic, R. Siegwart, and J. Nieto, "Revisiting boustrophedon coverage path planning as a generalized traveling salesman problem," in *Field and Service Robotics: Results of the 12th International Conference*. Springer, 2021, pp. 277–290.
- [23] N. Karapetyan, K. Benson, C. McKinney, P. Taslakian, and I. Rekleitis, "Efficient multi-robot coverage of a known environment," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1846–1852.
- [24] A. Chakravarthy and D. Ghose, "Obstacle avoidance in a dynamic environment: A collision cone approach," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 28, no. 5, pp. 562–574, 1998.