Cooperative Multi-Robot Hierarchical Reinforcement Learning

Gembong Edhi Setyawan[1], Pitoyo Hartono[2], Hideyuki Sawada[3]
Department of Applied Physics, School of Advanced Science and Engineering[1, 3]
Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan[1, 3]
School of Engineering, Chukyo University[2]
101-2 Yagoto Honmachi, Showa-ku, Nagoya, Aichi, 466-8666 Japan[2]

***Abstract*—Recent advances in multi-robot deep reinforcement learning have made it possible to perform efficient exploration in problem space, but it remains a significant challenge in many complex domains. To alleviate this problem, a hierarchical approach has been designed in which agents can operate at many levels to complete tasks more efficiently. This paper proposes a novel technique called Multi-Agent Hierarchical Deep Deterministic Policy Gradient that combines the benefits of multiple robot systems with the hierarchical system used in Deep Reinforcement Learning. Here, agents acquire the ability to decompose a problem into simpler subproblems with varying time scales. Furthermore, this study develops a framework to formulate tasks into multiple levels. The upper levels function to learn policies for defining lower levels' subgoals, whereas the lowest level depicts robot's learning policies for primitive actions in the real environment. The proposed method is implemented and validated in a modified Multiple Particle Environment (MPE) scenario.**

***Keywords*—Multi-robot system; hierarchical deep reinforcement learning; path-finding; task decomposition**

I. Introduction

Multi-Robot System (MRS) research has attracted significant attention recently due to its advantages over single robots. The benefits include (1) the decrease in time and the improvement in problem-solving efficiency due to the task decomposition, (2) an increase in problem-solving reliability, robustness, and resiliencies as the failures of single robots can be [1]–[3]. Using MRS, numerous prospective applications have been developed in which robots must be able to compete and cooperate, such as formation coordination [4], hide and seek [5], exploration and search [6]–[8], object transportation [9], disaster detection [10], communication networks [11], [12], etc. This study focuses on using MRS technology for exploration and search missions in unknown environments, where robots must collaborate to find the optimal path.

As shown in [13], [14], Reinforcement Learning (RL) is currently extensively employed as a robot learning algorithm that can automatically handle exploration and search problems in unknown environments, both in simulation and physical environments. Multiple robots undertake search and exploration operations in large and complex environments, such as in [15]–[17]. The objective of MRS is to distribute tasks between many robots to increase efficiency. Nowadays, applying RL is an important and challenging subject in MRS, where robots must learn and adapt based on their individual strategies and collective behavior. Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm was proposed as an advancement of RL that may be used for multi-robot learning in which robots can collaborate to solve problems in unknown environments [18].

RL in complex environments is always challenging, hence it serves as the primary motivation for the proposed approach. This paper proposes Multi-Agent Hierarchical Deep Deterministic Policy Gradient (MH-DDPG) as an extension of MADDPG for solving search and exploration problems with many robots in finding the optimal path for each robot in various environmental complexities. Here, hierarchical learning is adopted to learn in complex environments efficiently [13].

The primary contribution of this paper is as follows: (1) MH-DDPG presents a framework that enables multiple robots to learn by sharing environmental information collectively. (2) MH-DDPG proposes a hierarchical learning strategy by assigning different abstraction levels to the problem space, where higher abstraction learnings supply the learning subgoal for the lower ones, which consequently execute automatic task decomposition.

The structure of this article is as follows: Section II examines the previous research and theories relevant to the proposed method. In Section III, the technical context and theory are explained. The proposed MH-DDPG is discussed in detail in Section IV. Sections V and VI present the conducted experiments and the validation of the results. Finally, section VII provides conclusions and potential future works.

II. Related Works

Reinforcement Learning (RL) [19] is increasingly used as learning method to address complicated problems like games [20], [21], and robotics [22], [23]. Q-Learning [24], SARSA [25], and Temporal Differences (TD) [26] are RL algorithms that are predominantly applied to single agents within the Markov Decision Process (MDP) mathematical modeling framework [27].

Traditional RL faces various problems when attempting to solve real-world problems. This issue is known as the "curse of dimensionality", in which data grows exponentially, and computations become costly. Deep Reinforcement Learning (DRL) was introduced to solve this issue, for example, in [20].

This work was supported by JSPS Grants-in-Aid for Scientific Research on Innovative Areas (Research in a proposed research area) 18H05473 and 18H05895.

DRL models complex functions using the advantages of Neural Networks (NN), for example, Deep Q-network (DQN) [20], [28]. However, implementing DQN for robots with many degrees of freedom in action or continuous action space remains challenging. Policy Gradient (PG) [29] has been proposed to solve these issues; nevertheless, PG has its challenges, mainly that it requires extensive training. An actor-critic algorithm [25] combining DQN and PG's benefits was proposed to mitigate these problems. In the actor-critic algorithm, there are two networks: the actor-network, which employs a policy gradient to create optimal policies, and the critic-network, which contains DQN to evaluate the actor's policies. Deep Deterministic Policy Gradient (DDPG) [30] is an actor-critic [31] based algorithm that has been proposed to solve problems in continuous or high-dimensional action spaces.

Multi-Robot System (MRS) is currently attracting much interest because it can solve complex problems that are prohibitively difficult for single-agent systems. Although several attempts have been made to apply the prior algorithm in MRS, it is still not as successful as in single-agent systems. The recent development of MRS operates within the stochastic (Markov) games framework to model mathematical decision-making [32]. The difficulty of developing MRS is that the policies of one robot will impact a non-stationary environment generated by the policies of another agent. Recently, Multi-agent Deep Deterministic Policy Gradient (MADDPG) [18] was presented as a solution to this issue. For each robot, there is a DDPG in the MADDPG. The MADDPG is able to manage the problems of competitive as well as collaborative multi-agent systems.

One attempt to alleviate the learning difficulty is to design a mechanism for hierarchical learning. Hierarchical learning involves decomposing large and complex problems into more manageable subproblems. However, most algorithms need to fix the sub-problems that may hinder problem-solving flexibility. The Hierarchical Reinforcement Learning (HRL) approach of a single robot with discrete action has been proposed in [33], [34]; however, the subgoals need to be assigned manually, while [35] has been designed to be able to find subgoals automatically. Hierarchical Deep Reinforcement Learning (HDRL), [36], [37] proposed systems that can operate with continuous robot action. Algorithms in [33]–[37] work well with a single robot but are unsuitable for multi-robot implementation. Studies in [38], [39] presented a multi-agent hierarchy system with DRL for learning subgoals/skills at higher levels. However, the higher-level environment was manually defined using these approaches.

It is known that MADDPG can handle collaborative multi-robot system problems in a simple environment. However, an algorithm that can improve the learning performance of multi-robot systems is required for more complex environments. In this study, we propose MH-DDPG by developing MADDPG to perform under a hierarchical system. The proposed method is expected to be able to automatically discover subgoals, allowing it to perform better in a high-complexity environment.

III. Technical Preliminaries

This section highlights the theoretical basis for developing a hierarchical MADDPG method for cooperative multi-agent learning in handling complex problems.

A. Markov Decision Process and Reinforcement Learning

The decision-making process of a single robot is often based on Markov Decision Process (MDP). In RL, at each time step *t*, the robot perceives a state S_t from the environment's set of states $\mathcal{S}$ ($S_t \in \mathcal{S}$) and the robot selects an action A_t from the set of actions $\mathcal{A}$ that may be executed in the state S_t ($A_t \in \mathcal{A}$). Here, the next state is decided based on transition probability $\mathcal{P}$ under the learned policy π, as shown in (1). The transition brings the robot to the next state S_{t+1} and gives reward R_{t+1}.

$$\pi(s', r|s, a) = \mathcal{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a) \quad (1)$$

where $s \in \mathcal{S}$ and $s' \in \mathcal{S}$ are particular states that occur at time *t* and *t+1*, $a \in \mathcal{A}$ is the action taken by the robot at time *t* ($a \in \mathcal{A}$), and ($r \in \mathcal{R}$) is the reward received by the robot at time *t+1*.

Typically, the Bellman Equation is used to optimize two functions in RL: the state (V^*) and state-action (Q^*) functions, for which the optimal equation is as follows:

$$V^*(s) = \max_{\pi} R(s, a) + \gamma \sum_{s' \epsilon S} \mathcal{P}^a_{ss'} V^*(s') \quad (2)$$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \epsilon S} \mathcal{P}^a_{ss'} Q^*(s', a') \quad (3)$$

Here, the discount factor ($0 \leq \gamma \leq 1$) is used to express the significance of the future reward value.

B. Stochastic (Markov) Games

In contrast to MDP, which is utilized for a single robot, stochastic games are proposed as a mathematical framework for modeling decision-making in Multi-robot Reinforcement Learning. Stochastic games consist of *N* robots, a set of states containing the state of all robots ($\mathcal{S}$), a set of actions $\mathcal{A}$ from all robots ($A = A_1 x A_2 x... x A_N$), and a set of state transitions (T) which are the transition probability ($\mathcal{P}$) from the current state to the next state for a robot based on the actions taken by all robots ($T: S x A_1 x A_2 x... x A_N \rightarrow \mathcal{P}(S)$). The reward obtained by a robot depends on the actions taken by all robots ($R: S x A_1 x A_2 x... x A_N \rightarrow \mathcal{R}$). The reward function for each robot can be used to classify the type of games. For example, all robots share the same reward function if they play cooperatively. In contrast, when the robots play competitively, one robot aims to maximize the reward while the other attempts to minimize it. In stochastic games, the state value function (V) could be written as follows:

$$V_{i,\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{i,t+k+1} \mid S_{i,t} = s_i\right] \quad (4)$$

C. Q-Learning and Deep Q-Network

Q-Learning is a RL mechanism based on Q-function. Similar to (3), the Q-function is used to compute the expected reward based on the action done by the robot in its current state. The optimum policy is found by maximizing the value of the Q-function. The Q-value is learned iteratively, as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma\left(\max_{a'} Q(s',a') - Q(s,a)\right)\right) \quad (5)$$

where $0 \leq \alpha \leq 1$ denotes the learning rate.

The Q-value for all potential state-action combinations is stored in a Q-table, and thus high computational resources are required in a large number of states and large action space. The necessity for these costly computation resources can be alleviated by replacing the Q-table with a neural network for estimating the Q-value as in Deep Q-Network (DQN).

Q-Network and target Network are the two neural networks that constitute a DQN. The Q-network is used to train robots to predict the optimal Q-value, while the target network is used to forecast the next state based on the sample data and the optimal Q-value from all potential actions in the next state. In addition, DQN has a component called Experience Replay (ER) that stores and generates training data for Q-Network.

The optimal policy for DQN is determined by minimizing the Loss function in (6).

$$L(Q) = \mathbb{E}_{s,a,r,s'}\left[\left(R(s,a) + \gamma \max_{a'} Q^*(s',a'|\bar{\theta}) - Q(s,a|\theta)\right)^2\right] \quad (6)$$

where $\bar{\theta}$ and θ are the parameters for the target network and the Q-network.

D. Policy Gradient

Policy Gradient (PG) is used to enhance DQN's performance in generating optimum policies. In DQN, the robot chooses an action with the maximum Q-value, while in PG, the agent selects an action stochastically according to the probability distribution generated in the output layer.

The PG consists of a neural network known as a policy network, which predicts the probability distribution of actions given the current state. Here, the optimal policy is determined by maximizing the objective function defined as follows:

$$J(\theta) = \sum_{s\in S} d_\pi(s) \sum_{a\in A} \pi_\theta(a,s) Q_\pi(s,a) \quad (7)$$

where $d_\pi(s)$ is the deterministic distribution of the states on π. Here, the objective function $J(\theta)$ can be maximized by adjusting the parameter θ by gradient $\nabla_\theta J(\theta$ as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a,s) Q_\pi(s,a)] \quad (8)$$

E. Deterministic Policy Gradient and Deep Deterministic Policy Gradient

The policy function in PG is always modeled as a stochastic probability distribution of the agent's actions given the current state. The Deterministic Policy Gradient (DPG) has been proposed to model policy as a deterministic decision by the agent in the current state. The objective function in DPG can be written as follows:

$$J(\theta) = \mathbb{E}_{s\sim\rho_\pi}[R(s,\pi_\theta(s))] \quad (9)$$

where ρ_π is discounted state distribution. The gradient of the objective function in DPG can be written as follows:

$$J(\theta) = \mathbb{E}_{s\sim\rho_\pi}\left[\nabla_\theta \pi_\theta(s) \nabla_\theta Q_\pi(s,a)|_{a=\pi_\theta(s)}\right] \quad (10)$$

Deep Deterministic Policy Gradient (DDPG) is an actor-critical algorithm that combines DQN and DPG. DQN is for the actors that operate in discrete action space, while DPG is for the critics that work in continuous action space.

F. Multi-Agent Deep Deterministic Policy Gradient

Multi-Agent Deep Deterministic Policy Gradient (MADDPG) is an expansion of DDPG that adopt an actor-critic algorithm as its fundamental structure. The MADDPG contains multiple robots, each with its neural networks for the actors and the critics, while DDPG only uses a single robot. Similar to DDPG, the actors in MADDPG receive input from the robot's local observations and produce executable action recommendations for the robot. However, in contrast to the critical network in DDPG, the input of critics in MADDPG does come from not only the robot's local observations and actions but also other robots' observations and actions. The critic's output is the Q-value, which is used to evaluate the actor's actions by considering other robots' observations and acts. The network of agents may therefore learn both cooperative and competitive strategies.

IV. Multi-Hierarchies of Multi-Agent Deep Deterministic Policy Gradient

This study proposes Multi-Agent Hierarchical Deep Deterministic Policy Gradient (MH-DDPG) as a new approach that enables learning robots to decompose complex tasks into more manageable subtasks at different time scales. Here, the robots train to learn several levels of policy, each of which has a specific task for the agents to do in parallel.

A. Architecture

MH-DDPG trains robots to hierarchically learn policies based on the architecture shown in Fig. 1. Here, MH-DDPG is comprised of DDPG and experience replay (ER). The number of DDPG and ER depends on the number of agents and hierarchy. For example, suppose *N* and *K* indicate the number of agents and hierarchies, respectively. Consequently, there are *N x K* DDPG in MH-DDPG. Furthermore, the number of ER will equal the number of hierarchies, *K*. The bottom level represents the physical environment in which the robots physically operate. While the higher level, robots are presented by their abstraction.

Formally, the MH-DDPG with *N* agents and *K* levels are defined by the set of state $\mathcal{S}$; the set of joint action $\mathcal{A} = \bigcup_{i=1}^{N} A_{i,t}^{\hbar}$ and the set of joint observations $\mathcal{O} = \bigcup_{i=1}^{N} O_i^{\hbar}$, where $A_{i,t}^{\hbar}$ and $O_{i,t}^{\hbar}$ are actions A_i and observation O_i for each agent i at level $\hbar$ and time t. Each agent will optimize their respective policy at every level to estimate the transition probability $\mathcal{P}$ for selecting an action at time t, such that $\pi = \bigcup_{i=1}^{N} \pi_{i,t}^{\hbar}$, where $\hbar$ is the hierarchy level and $0 \leq \hbar \leq K-1$.

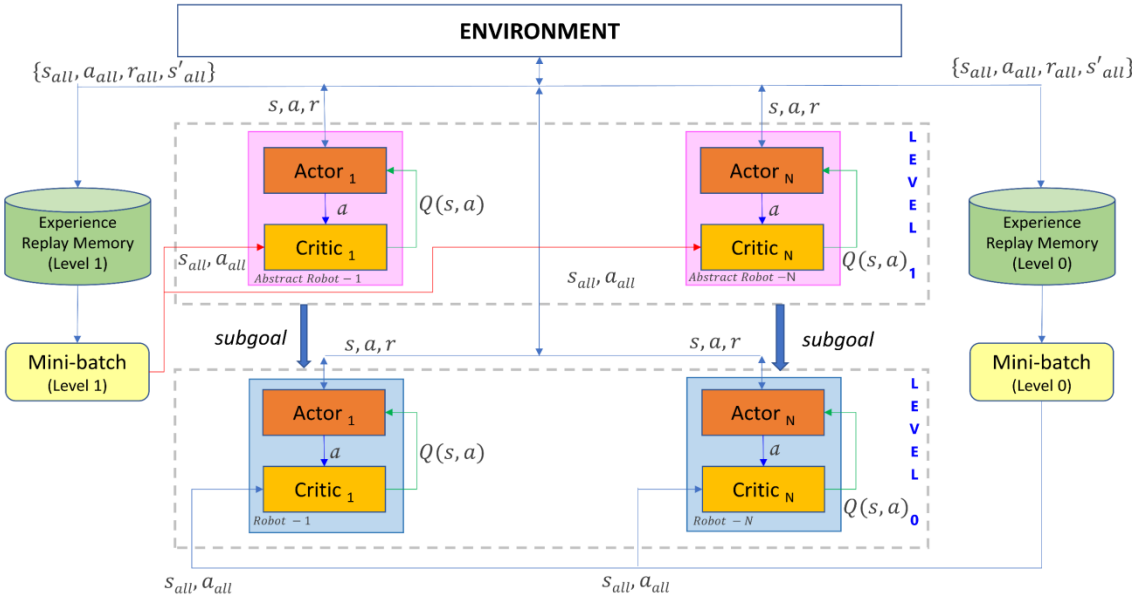


Fig. 1. Architecture of MH-MADDPG.

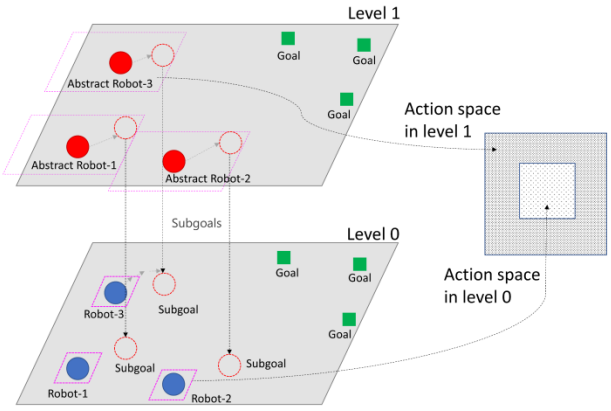


Fig. 2. Illustration Environment with Two Hierarchies.

For illustrating the dynamics of MH-DDPG multi-agent particle environment (MPE) environments will be utilized [40]. One of the MPE scenarios, "simple spread" has been modified here. For example, Fig. 2 depicts the problem that MH-DDPG must address. Simple_spread is an environment with *N* robots and *M* goals (landmarks). Robots are expected to cooperate to accomplish a common objective while avoiding collisions with one another. There are three robots (*N=3*), three goals (*M=3*), and two hierarchies (*K=2*). First, the real problem of the environment is illustrated at level 0, where a blue circle represents the actual robots, and a green rectangle represents the goals. Then, at level 1, an abstraction of level 0, the robots are referred to as abstract robots and symbolized by a red circle. Abstract robots at level 1 possess actions with more capabilities than those of actual robots at level 0. For instance, the actual robots at level 0 have a maximum velocity of 1 pixel per second, while the robot at level 1 is set with a maximum velocity of 10 pixels per second. As seen on the right of Fig. 2, the robot at level 1 has a greater range of distances than the actual robot for each action taken at each step.

The abstract robots are predicted to learn faster than the actual robots in achieving goals since they are less constrained than actual robots (for example, more quickly and with no obstacles). However, remember that abstract robots are only imaginative robots with no capacity to execute physical actions. The task of the abstract robot at level 1 is to learn how to accomplish the main goal best, while at level 0, the task is to learn how to achieve the subgoal optimally. MH-DDPG implicitly assigns different objectives for each level, in which the robots' objective at level 1 is to learn to achieve goals optimally, while the robots' work at level 0 is to learn to achieve subgoals optimally. The subgoal at level 0 is automatically determined from the higher level, which happens when the abstract robot chooses the action $A^1_{i,t}$ based on the policy $\pi^1_{i,t}$ in the current state $S^1_{i,t}$ at time t. The robot will be transitioned to the next state $S^1_{i,t+1}$ and will receive a reward $R^1_{i,t+1}$. Then, the learning shifts to the bottom level, and the next state at level 0 becomes a subgoal for the actual robots. In addition, robots at level 0 engage in learning to achieve these subgoals. When the robots get $S^0_{i,t}$ at level 0, the agent will pick the action $A^0_{i,t}$ based on the policy of $\pi^0_{i,t}$. The robots then transition to $S^0_{i,t+1}$ and is rewarded with $R^0_{i,t+1}$. The learning process at level 0 will continue until the terminal criteria are satisfied. A terminal condition is defined by manually setting the maximum number of steps at level 0. If the terminal requirements are satisfied, learning returns to level 1 to execute the next step at the top level.

B. Learning Dynamic

In MH-DDPG, the multiple robots learn in parallel at all levels. The process of robot learning will start at the top level and flows downward. Robot learning aims to provide optimum policies for each robot at all levels. According to the RL concept that the optimal policy is acquired by maximizing the rewards received by each robot. The reward obtained in the

future by each robot i at level $\hbar$ could be expressed by the following V-function:

$$V_{i,\pi}^{\hbar}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{i,t+1}^{\hbar} \mid S_{i,t}^{\hbar} = s\right] \quad (11)$$

$$V_{i,\pi}^{\hbar}(s) = \sum_{a\epsilon A} \pi_i^{\hbar}(a|s)\left(R_i^{\hbar}(s,a) + \gamma \sum_{s'\epsilon S} \mathcal{P}_{ss'}^{a} V_{i,\pi}^{\hbar}(s')\right) \quad (12)$$

where $r_{i,t+1}^{\hbar}$ is the reward earned by agent i at level $\hbar$ at $t+1$, and $\pi_i^{\hbar}(a|s)$ is the agent policy. Here, the joint action is designed to make the robot's policy dependent on individual policies and joint policies. The definition of the Q-function is as follows:

$$Q_{i,\pi}^{\hbar}(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{i,t+1}^{\hbar} \mid S_{i,t+1}^{\hbar} = s, A_{i,t}^{\hbar} = a\right] \quad (13)$$

$$Q_{i,\pi}^{\hbar}(s,a) = R_i^{\hbar}(s,a) + \gamma \sum_{s'\epsilon S} \mathcal{P}_{ss'}^{a} \sum_{a\epsilon A} \pi_i^{\hbar}(a'|s')\, Q_{i,\pi}^{\hbar}(s',a') \quad (14)$$

The optimal policy is determined by maximizing the value of all actions. According to the Bellman optimality equation, the optimal V-value (V*) and Q-value (Q*) could be written as follows:

$$V_i^{\hbar *}(s) = \max_{\pi_i} R_i^{\hbar}(s,a) + \gamma \sum_{s'\epsilon S} \mathcal{P}_{ss'}^{a} V_i^{\hbar *}(s') \quad (15)$$

$$Q_i^{\hbar *}(s,a) = R_i^{\hbar}(s,a) + \gamma \sum_{s'\epsilon S} \mathcal{P}_{ss'}^{a} Q_i^{\hbar *}(s',a') \quad (16)$$

If the environment consists of N agents and K levels, then the policy set $\pi = \{\pi_1^{\hbar}, \pi_2^{\hbar}, \dots, \pi_N^{\hbar}\}$ that is parameterized by $\theta = \{\theta_1^{\hbar}, \theta_2^{\hbar}, \dots, \theta_N^{\hbar}\}$, where $1 \le i \le N$ and $0 \le \hbar \le K-1$. Then, the gradient of the expected return for each agent i at level $\hbar$ could be expressed as follows:

$$J(\theta_i^{\hbar}) = \mathbb{E}_{x,a\sim D}\left[R\left(s, \pi_i^{\hbar}(s)\right)\right] \quad (17)$$

$$\nabla_{\theta_i^{\hbar}} J(\theta_i^{\hbar}) = \mathbb{E}_{x,a\sim D}\left[\nabla_{\theta_i^{\hbar}} \log \pi_i^{\hbar}(a_i^{\hbar}|O_i^{\hbar}) Q_{i,\pi}^{\hbar}(x^{\hbar}, a_1^{\hbar}, \dots, a_N^{\hbar})\Big|_{a_i^{\hbar}=\pi_i^{\hbar}(o_i^{\hbar})}\right] \quad (18)$$

where $Q_{i,\pi}^{\hbar}(x, a_1^{\hbar}, \dots, a_N^{\hbar})$ is the centralized Q-function at level $\hbar$ that accepts as input all agent actions at level $\hbar$, $a_1^{\hbar}, \dots, a_N^{\hbar}$, and observation x at level $\hbar$ of all agents, $x = (O_1^{\hbar}, \dots, O_N^{\hbar})$, with the output being the Q-value for each agent i at level $\hbar$. Experience Replay buffer $\mathcal{D}$ contains $(x, x', a_1^{\hbar}, \dots, a_N^{\hbar}, r_1^{\hbar}, \dots, r_N^{\hbar})$ where x' is the next state obtained after the agent took action while in state x. The centralize Q-function $Q_{i,\pi}^{\hbar}$ will be updated by minimizing the following loss function:

$$\mathcal{L}(\theta_i^{\hbar}) = \mathbb{E}_{x,a,r,x'}\left[\left(Q_{i,\pi}^{\hbar}(x, a_1^{\hbar}, \dots, a_N^{\hbar}) - y^2\right)\right] \quad (19)$$

$$y = r_i^{\hbar}\, Q_{i,\pi',\hbar}^{\hbar}(x', a_i'^{,\hbar}, \dots, a_N'^{,\hbar})\Big|_{a_j'^{,\hbar}=\pi_j'^{,\hbar}(o_j'^{,\hbar})} \quad (20)$$

where $\pi'^{,\hbar} = \{\pi_{\theta_1'^{,\hbar}}, \dots, \pi_{\theta_N'^{,\hbar}}\}$ is the set of target policy with delayed parameter θ_i' at each level $\hbar$. As the Q-function $Q_{i,\pi}^{\hbar}$ for each agent i is learned independently at all levels, the reward may be determined arbitrarily based on the issue. The algorithm of MH-DDPG is shown in algorithm 1.

Algorithm 1. MH-MADDPG

Initialize: Actor-critic evaluation and target networks for each agent, number of levels K, maximum step H, Replay buffer
For episode = 1 to max-episode, do
  For each agent i, set initial states (S) and goals ($\mathcal{G}$) for each agent
  Train (K-1, S, $\mathcal{G}$)
End for

Function Train($\hbar$ ::level, S ::state, $\mathcal{G}$ ::goal)
  $s \leftarrow S_{i,t}^{\hbar} \leftarrow S$, $g^{\hbar} \leftarrow \mathcal{G}$ (initial, t=0)

  For t = 1 to H **do**
    For each agent n: select action (a_i) where $a_i \leftarrow A_{i,t}^{\hbar}$ based on $\pi_{i,t}^{\hbar}$
    Execute actions $a = (a_1, \dots, a_N)$ and observe reward r and new state s'
    Store (s, a, r, s') in replay buffer $\mathcal{D}$
    If $\hbar > 0$:
      $g^{\hbar-1} \leftarrow s'$
      $Train(\hbar - 1,\ s,\ g^{\hbar-1})$
    End If
    For agent i = 1 to N in level $\hbar$ **do**
      Sample random minibatch of S samples (s, a, r, s') from $\mathcal{D}^{\hbar}$
      Set $y = r_i^{\hbar} + \gamma Q_{i,\pi'}^{\hbar}(s', a_1', \dots, a_N')\big|_{a_i'=\pi_i'^{,\hbar}}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i^{\hbar}) = \frac{1}{S}\sum\left(y - Q_{i,\pi}^{\hbar}(s, a_i, \dots, a_N)\right)^2$
      Update actor using:
        $\nabla_{\theta_i^{\hbar}} J =$
        $\frac{1}{S}\sum \nabla_{\theta_i^{\hbar}} \pi_i^{\hbar}(o_i) \nabla_{a_i} Q_{i,\pi}^{\hbar}(s, a_i, \dots, a_N)\big|_{a_i=\pi_i^{\hbar}(o_i)}$
    End for
    Update target network parameters for each agent i in level $\hbar$:
      $\theta_i'^{,\hbar} \leftarrow \alpha\theta_i^{\hbar} + (1-\alpha)\theta_i'^{,\hbar}$
  End Function

C. State, Observation, and Action Space

Consider an environment with N robots and M goals. Robots and goals have a physical entity represented by X. Based on the original MPE, X is a two-dimensional object characterized by its position and velocity. Furthermore, the state contains polar coordinates that are utilized to identify the robot's relative position to the goals and other robots. An environment with N robots and M goals corresponds to a state space with NxM polar coordinates of robots to the goals ($d_{1,\dots,NxM}^{G}$) and $N-1$ polar coordinates to other robots ($d_{1,\dots,N-1}^{A}$). However, it should be noted that the goals of the bottom level are the subgoals produced at the upper level.

Based on the preceding discussion, the state space $\mathcal{S}$ is a mixture of each level state space: $\mathcal{S} = \cup_{\hbar=0}^{K-1} S^{\hbar}$, where $S^{\hbar} = \{X_{1,\dots,N}, d_{1,\dots,NxM}^{G}, d_{1,\dots,N-1}^{A}\}$.

Then each agent can only observe their own state of the entire state, called observation. The observation space of each agent at each level k is $O_i^{\hbar}(S) = \{X_i\ , d_{i,1,\dots,M}^{G}, d_{i,1,\dots,N-1}^{A}\}$, where i indicates the i^{th} robot.

At $\hbar = 0$, the output layer of the actor networks generates five outputs between 0 and 1 in which each one is associated with a particular action. The five outputs are denoted by u_n, u_l, u_r, u_d, and u_u for no action, move left, right, down, and up, respectively. At $\hbar > 0$, if the action of the abstract robot should have more capabilities than the actual robot, the range u is increased multiplied by the sensitivity rate, therefore $u^{\hbar} = u^0 x\, \mu$, where $1 \leq \hbar \leq K-1$ and μ is the sensitivity with a value more than 1. The sensitivity of the upper level must be larger than the sensitivity of the lower level.

D. Reward Design

The reward is designed to correspond to the learning objectives of the robot. The distance between objects determines the reward design. Suppose that the positions of two object types, A and B, in two dimensions are known. A and B respectively add up to N and M, therefore $A_i = (x_1^i, y_1^i)$ and $B_j = (x_2^j, y_2^2)$, where $1 \leq i \leq N$ and $1 \leq j \leq M$. The following formula may be used to compute the total distance between two types of objects:

$$d(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (21)$$

$$d(A_{1,\dots,N}, B_{1,\dots,M}) = \sum_{i=1}^{N} \sum_{j=1}^{M} \sqrt{\left(x_2^i - x_1^j\right)^2 + \left(y_2^i - y_1^j\right)^2} \quad (22)$$

The design of the reward differs between the bottom and upper levels. The term for the rewards at each level is explained as follows:

1) The goal/subgoal reward: Designed to encourage agents to achieve the Goal/Subgoal. This reward is available at all levels. This reward is utilized at the highest level to promote the abstract robot to accomplish the main goal and at the lowest level to help the robot reach the subgoal. Reward calculations will be based on the distance between all robots and goals using (22) and (23).

$$R(Agent_{i,\dots,N}, G_{j,\dots,M}) = -d(A_{1,\dots,N}, B_{1,\dots,M}) \quad (23)$$

where A = Robot dan B = G (goal/subgoal).

2) Robot relative to other robots Reward: for avoiding collisions between robots. This reward is only used at the lowest level because the abstract robot is unable to detect other robots. This reward term is calculated as follows:

$$R_c(A,B) = \begin{cases} -1; \ if\ d(A,B) \leq A_{size} + B_{size} \\ 0; \ if\ d(A,B) > A_{size} + B_{size} \end{cases} \quad (24)$$

where $d(A,B)$ is the distance between two robots (A and B) that can be calculated by (22) with $i, j=1$.

3) Obstacle reward: Aims to encourage robots to avoid obstacles. Due to the abstract robot's inability to detect obstacles, this reward is only applied at the lowest level. Similar to other robot rewards, the robot must compute the distance between itself and the obstacle to get reward.

$$R_c(A,B) = \begin{cases} -10; \ if\ d(A,O) \leq A_{size} + O_{size} \\ 0; \ if\ d(A,O) > A_{size} + O_{size} \end{cases} \quad (25)$$

where $d(A,O)$ is the distance between robot A and obstacle O that can be calculated by (22) with $i,j=1$.

E. Neural Network Models

Each robot at each level of the MH-DDPG consists of actor and critic networks, structures of which are shown in Fig. 3. Local observations are the inputs for the actor-network, while robot actions represent the output. Therefore, the critic-network uses the observations and actions of all robots as inputs and Q-value as outputs. The Q-value is then used as the training basis for the actor networks. Every network employs the ADAM optimizer with a learning rate (α) and a discount factor (γ).

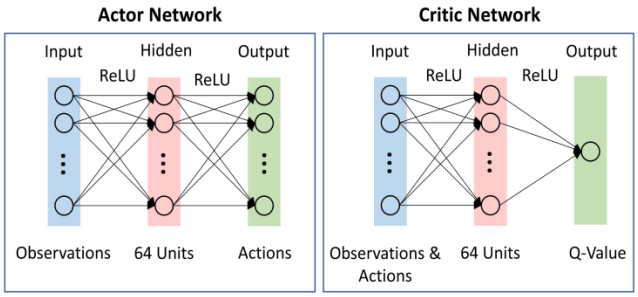
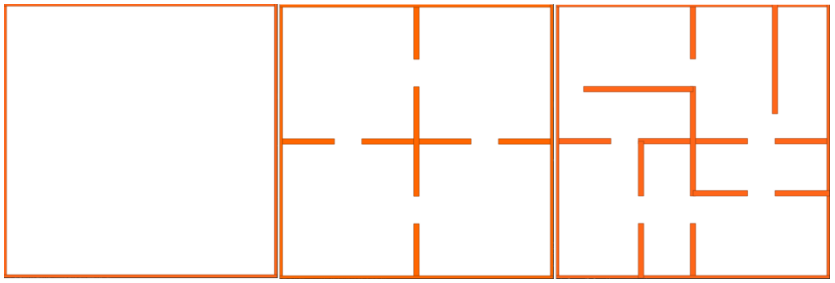


Fig. 3. Neural Network Model.

V. Experimental Environment

We conducted experiments for comparing the DDPG, MADDPG, and MH-DDPG algorithms under the parameters listed in Table I. The experimental environment is set with three robots and three goals, with some obstacles to increase the environment's complexity. Fig. 4 depicts the environment for testing the proposed algorithm. The experiments were conducted on three types of environments with various complexities: low-complexity {Fig. 4(a)}, mid-complexity {Fig. 4(b)}, and high-complexity {Fig. 4(c)}.



(a) Low-Complexity. (b) Mid-Complexity. (c) High-Complexity.

Fig. 4. Illustration of the Experimental Environment.

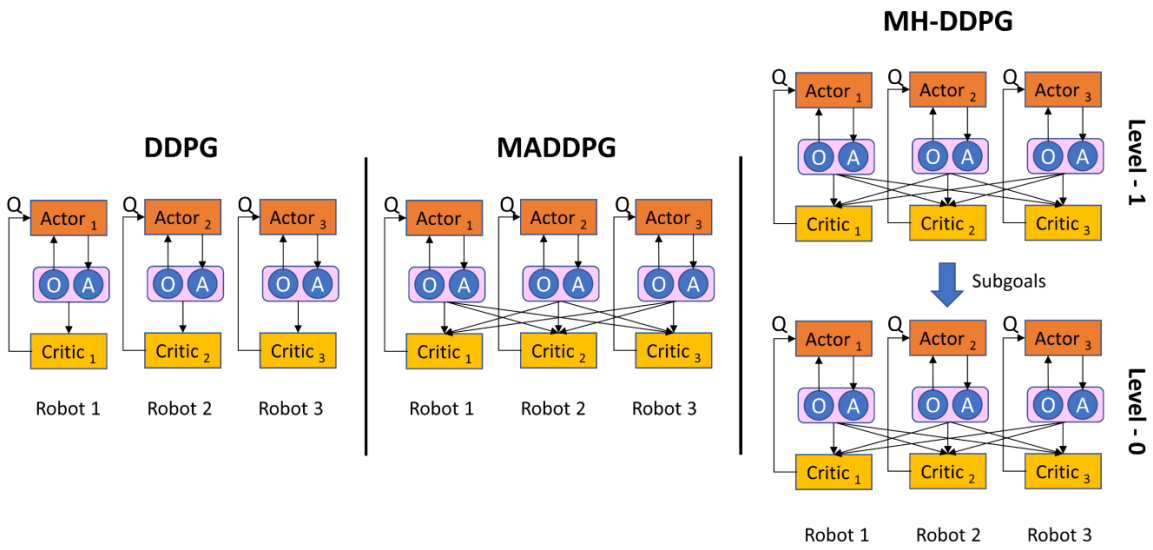


Fig. 5. Block Diagram Comparison for DDPG, MADDPG, and MH-DDPG Employing 3 Robots. Q, O, and a Represent the Q-Value, Observation, and Action, Respectively.

TABLE I. EXPERIMENT-SPECIFIC PARAMETERS FOR DDPG, MADDPG, AND MH-DDPG

Parameters	DDPG	MADDPG	MH-DDPG
Specific Parameters			
The number of robots (N)	3	3	3
The number of levels (K)	-	-	2
The number of actors	3	3	6
The number of critics	3	3	6
The number of ERs	1	1	2
Sensitivity	5	5	5 (level 0), 30 (level 1)
Actor and Critic Networks Parameters			
Number of hidden layers	1	1	1
Number of hidden units	64	64	64
Activation Function	ReLU	ReLU	ReLU
Input Actor Network	Current Observations	Current Observations	Current Observations
Output Actor Network	Action	Action	Action
Input Critic Network	Current Observation and Action	Current Observation and Action	Current Observation and Action
Output Critic Network	Q-value	Q-Value	Q-value
Training parameters			
Optimizer	ADAM	ADAM	ADAM
Learning rate (α)	1e-2	1e-2	1e-2
Discount factor (γ)	0.97	0.97	0.97
Replay buffer size	10^6	10^6	10^6
Minibatch size	1256	1256	1256

Fig. 5 compares the block diagrams of DDPG, MADDPG, and MH-DDPG, illustrating how the algorithm determines the parameter values for the specific parameters given in Table I, except for sensitivity. The values for the sensitivity and the training parameters are empirically determined. From the experiments, the determination of the training parameters in different environments demonstrates that the algorithms are not excessively sensitive to the chosen parameters.

Particularly in MH-DDPG, the designed environment can decompose into K levels. As a preliminary step in the proposed algorithm, this research performs a two-level investigation (K=2). Where the bottom level ($\mathcal{k} = 0$) is the real environment used for actual robots learning, and the top level ($\mathcal{k} = 1$) is the abstract environment used for abstract robot learning. In the environments, the robots must collaborate to accomplish the predetermined goals. The robots' mission will be accomplished if the robots can discover the optimal path for reaching all the goals.

VI. RESULTS

The experiments compare the proposed algorithm against MADDPG and DDPG. The first step is assessing the robots learning performance based on the rewards obtained throughout the learning process.

Fig. 6 depicts the learning curve based on the robot's average reward in each episode. In this experiment, there were 150000 episodes in each environment. The average reward the robot obtains in a low-complexity environment is greater than in mid-complexity and high-complexity environments. A greater average reward indicates that robots in simple environments perform better than in other environments. A low-complexity environment without obstacles makes it easier for robots to reach their goals.

From Fig. 6, it can be observed that MH-DDPG converges faster to the maximum rewards and produces larger reward in each episode than DDPG and MADDPG, indicating that the robots that were trained using MH-DDPG reaches the goals

faster. Fig. 6 also indicates that the superiority of MH-DDPG over DDPG and MADDPG is consistent in complex environments. In a high-complexity environment, the graph also reveals that the average reward value is unstable for DDPG and MADDPG, whereas MH-DDPG remains robust.

Fig. 7 depicts the robot's behavior during the learning process compared to MADDPG and DDPG in a mid-complexity and high-complexity environment at t=0, 10, and 40. The blue circle indicates the actual robot, the green rectangle represents the goals, and the red circle represents the abstract robot found exclusively on MH-DDPG. The abstract robots will generate a subgoal for the lower level. At t=0, the robot begins its first step of learning. Here, the locations of the abstract robot are identical to the positions of the actual robots.

At t=10, robots are learning to achieve all goals. Here, on MH-DDPG, a red circle indicates the presence of an abstract robot. At each instant t, the abstract robot generates a subgoal for the actual robot. In MH-DDPG, Actual robots will first learn to cover subgoals, but in DDPG and MADDPG, robots will learn to cover main goals straight away since there are no subgoals. Abstract robots that cannot detect obstacles might occasionally be located in the same area as the obstacle, as shown in the high-complexity environment at time t=10. This condition is sometimes harmful to the actual robot when it learns to achieve the subgoal since the actual robot cannot reach the subgoal properly, which consequently decreases the associated state-action values.

Finally, at t=40, the final step of learning in a single episode occurs. In DDPG, it is evident that the robots have difficulty cooperating to reach the goal in both mid-complexity and high-complexity environments, which is also consistent with the successful rates shown in Fig. 8. In the mid-complexity environment, the robot tends to go toward one of the goals. Therefore the targeted objectives to cover all goals are often not achieved. This is because the robots work independently and do not share information. In high-complexity, obstacles tend to hinder the robot's ability to achieve the subgoals.

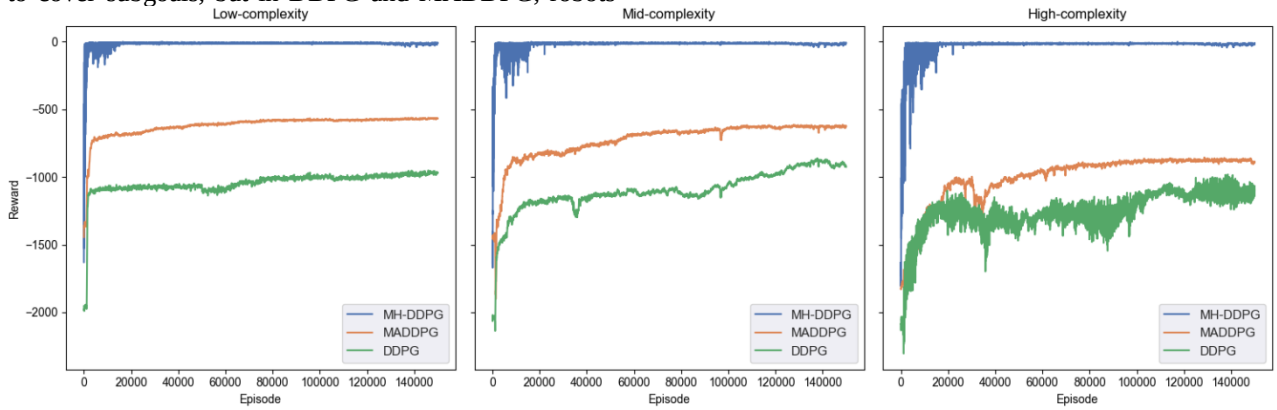


Fig. 6. Reward Graph.

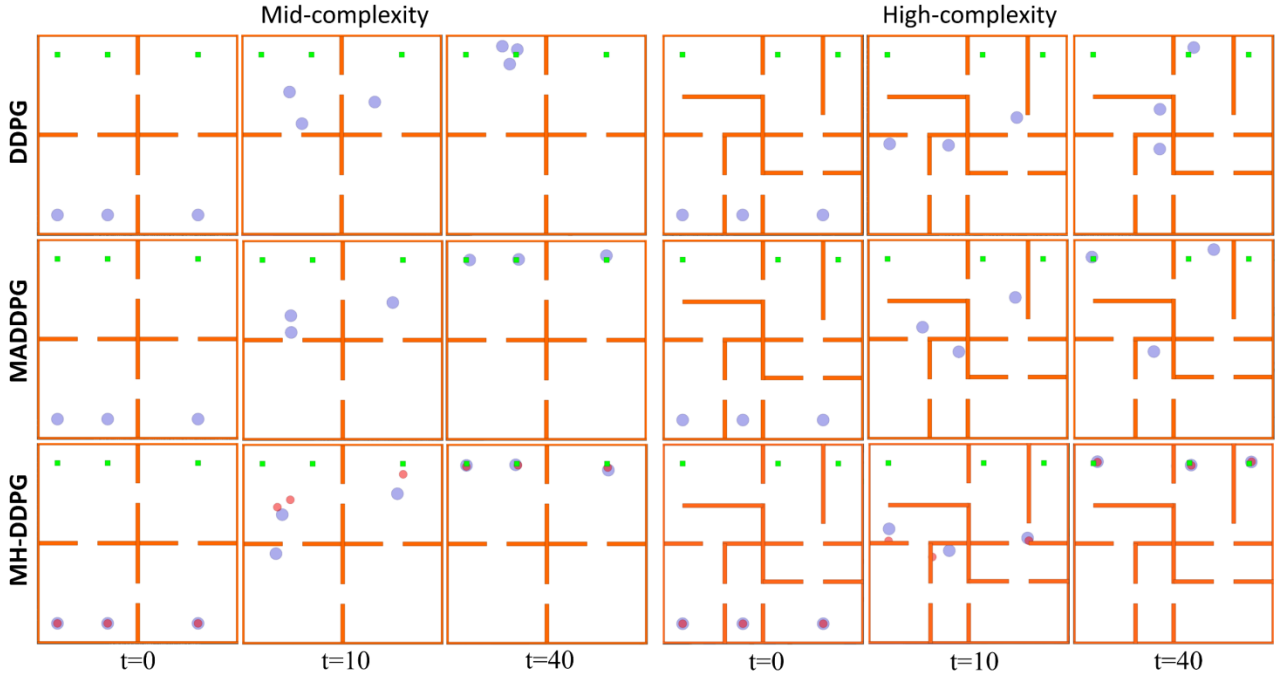


Fig. 7. Comparison between DDPG, MADDPG, and MH-DDPG on the Mid-complexity and High-complexity Environment.

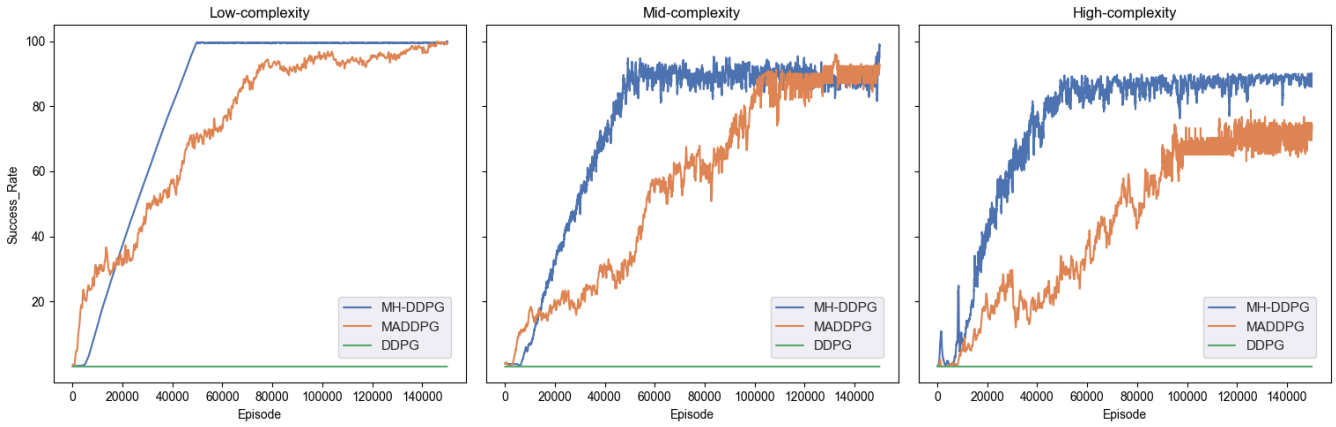


Fig. 8. Success Rate.

From Fig. 7, for MADDPG, it can be observed that the robots can work collectively to achieve goals in a mid-complexity environment, while in high-complexity environments, due to the complex configurations of the obstacles, the robots faced difficulty in the early phase of the learning process as apparent from the fluctuating graph, but gradually stabilize as the learning progresses. In MH-DDPG, subgoals increase the robot's problem-solving ability as they are guided by intermediate objectives that consequently contrained the problem space.

Fig. 8 depicts the average success rate for the respective algorithm in each environment during the learning process. In all environments, the robots with MADDPG and MH-DDPG were able to cooperate and learn policies to achieve their goals, while the robots with DDPG failed to do so. The failure of the robots in DDPG is due to the lack of information sharing between robots; hence, the robots only learn independently and may repeat the failure of other robots. In the low-complexity environment, the average success rates for MADDPG and MH-DDPG are 74.18% and 82.15%, respectively. As the complexity of the environment increases, the average success rate for MADDPG and MH-DDPG decreases, as seen from the graphs for mid and high-complexity environments. MADDPG and MH-DDPG had respective success rates of 56.08% and 73.18% in a mid-complexity environment, while in a high-complexity environment, these success rates were 44.18% and 72.99%, respectively. The results show that MH-DDPG has a greater success rate than MADDPG. This indicates that decomposing the problem environment into many levels is advantageous for maximizing robot learning performance.

VII. Conclusion

MH-DDPG is proposed as a novel framework for multi-robot learning with hierarchical Deep Reinforcement Learning. Here, the robots collectively learn by sharing information about state-action values from their individual runs. In addition, the proposed MH-DDPG provides a mechanism for creating multi-level abstraction, in which higher-level abstraction space allow the robots to execute a kind of “image training” where they may virtually explore the problem space without considering the physical constrains in real-world space. The virtual experiment in abstract space allows the robot to discover the real robots' intermediate goals rapidly. The intermediate goals helps to limit the exploration for the real robots, thus alleviating the curse of dimensionality.

Through some empirical experiments, it can be observed that the proposed MH-DDPG outperforms DDPG and MADDPG in learning efficiency and success rate.

The weakness of the MH-DDPG is that an abstract robot at higher levels is incapable of detecting obstacles. Hence non-realistic subgoals are sometimes produced. This is the cost that needs to be paid for removing the physical constraints in the abstract space. In this preliminary experiment, the abstract robots are given higher speed but are constrained by their inability to detect obstacles, but it does not have to be so. In the following study, experiments will be conducted with various constrained conditions at the higher abstraction levels.

Immediate future research topics include investigating the effect of the number of levels and the number of robots in MH-DDPG. In addition, implementing the proposed learning method into physical robots is also of interest.

References

[1] Z. Yan, N. Jouandeau and A. A. Cherif, “A survey and analysis of multi-robot coordination,” Int. J. Adv. Robot. Syst., vol. 10, pp. 1–18, 2013.

[2] J. Song and S. Gupta, “CARE: Cooperative Autonomy for Resilience and Efficiency of robot teams for complete coverage of unknown environments under robot failures,” Auton. Robots, vol. 44, no. 3–4, pp. 647–671, 2020.

[3] Y. Rizk, M. Awad and E. W. Tunstel, “Cooperative heterogeneous multi-robot systems: A survey,” ACM Comput. Surv., vol. 52, no. 2, 2019.

[4] Z. A. Ali, Z. Han and R. J. Masood, “Collective motion and self-organization of a swarm of uavs: A cluster-based architecture,” Sensors, vol. 21, no. 11, pp. 1–19, 2021.

[5] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powel, B. McGrew and I. Mordatch, “Emergent tool use from multi-agent autocurricula,” 2020.

[6] M. Bakhshipour, M. Jabbari Ghadi and F. Namdari, “Swarm robotics search & rescue: A novel artificial intelligence-inspired optimization approach,” Appl. Soft Comput. J., vol. 57, pp. 708–726, 2017.

[7] J. P. Queralta, J. Taipalmaa, B. C. Pullinen, V. K. Sarker, T. N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju and T. Westerlund, “Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision,” IEEE Access, vol. 8, pp. 191617–191643, 2020.

[8] D. S. Drew, “Multi-Agent Systems for Search and Rescue Applications,” Curr. Robot. Reports, vol. 2, no. 2, pp. 189–200, 2021.

[9] L. Hawley and W. Suleiman, "Control framework for cooperative object transportation by two humanoid robots," Rob. Auton. Syst., vol. 115, pp. 1–16, 2019.

[10] X. Zhou, W. Wang, T. Wang, Y. Lei and F. Zhong, "Bayesian Reinforcement Learning for Multi-Robot Decentralized Patrolling in Uncertain Environments," IEEE Trans. Veh. Technol., vol. 68, no. 12, pp. 11691–11703, 2019.

[11] N. Naderializadeh, J. J. Sydir, M. Simsek and H. Nikopour, "Resource Management in Wireless Networks via Multi-Agent Deep Reinforcement Learning," IEEE Trans. Wirel. Commun., vol. 20, no. 6, pp. 3507–3523, 2021.

[12] J. Sheng, X. Wang, B. Jin, J. Yan, W. Li, T.-H. Chang, J. Wang and H. Zha, "Learning structured communication for multi-agent reinforcement learning," Auton. Agent. Multi. Agent. Syst., vol. 36, no. 2, 2022.

[13] F. Niroui, K. Zhang, Z. Kashino and G. Nejat, "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments," IEEE Robot. Autom. Lett., vol. 4, no. 2, pp. 610–617, 2019.

[14] S. M. Sombolestan, A. Rasooli and S. Khodaygan, "Optimal path-planning for mobile robots to find a hidden target in an unknown environment based on machine learning," J. Ambient Intell. Humaniz. Comput., vol. 10, no. 5, pp. 1841–1850, 2019.

[15] J. Hu, H. Niu, J. Carrasco, B. Lennox and F. Arvin, "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning," IEEE Trans. Veh. Technol., vol. 69, no. 12, pp. 14413–14423, 2020.

[16] R. J. Alitappeh and K. Jeddisaravi, "Multi-robot exploration in task allocation problem," Appl. Intell., vol. 52, no. 2, pp. 2189–2211, 2022.

[17] T. Fan, P. Long, W. Liu and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," Int. J. Rob. Res., vol. 39, no. 7, pp. 856–892, 2020.

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," Adv. Neural Inf. Process. Syst., vol. 2017-Decem, pp. 6380–6391, 2017.

[19] R. S. Sutton and A. G. Barto, Reinforcement Learning: an Introduction, 2nd ed. London, England: The MIT Press Cambridge, Massachusetts, 1998.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level Control through Deep Reinforcement Learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[21] X. Wang, J. Song, P. Qi, P. Peng, Z. Tang, W. Zhang, W. Li, X. Pi, J. He, C. Gao, H. Long and Q. Yuan, "SCC: an efficient deep reinforcement learning agent mastering the game of StarCraft II," 2021.

[22] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn and S. Levine, "Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills," 2021.

[23] M. Dalal, D. Pathak and R. Salakhutdinov, "Accelerating Robotic Reinforcement Learning via Parameterized Action Primitives," in 35th Conference on Neural Information Processing Systems (NeurIPS 2021), 2021, no. NeurIPS.

[24] C. J. C. H. Watkins and P. Dayan, "Q-Learning," Mach. Learn., vol. 8, pp. 279–292, 1992.

[25] G. A. Rummery and M. Niranjan, On-Line Q-Learning Using Connectionist Systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[26] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," Mach. Learn., vol. 3, no. 1, pp. 9–44, 1988.

[27] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming. New Jersey: John Wiley & Sons, Inc., 1994.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," vol. arXiv prep, 2013.

[29] R. S. Sutton, D. McAllester, S. Singh and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in Proceedings of the 12th International Conference on Neural Information Processing Systems, 1999, pp. 1057–1063.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning," 4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc., 2016.

[31] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," Adv. Neural Inf. Process. Syst., pp. 1008–1014, 2000.

[32] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," Mach. Learn. Proc. 1994, pp. 157–163, 1994.

[33] R. S. Sutton, D. Precup and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning Richard," Artif. Intell., vol. 112, no. 1, pp. 181–211, 1998.

[34] A. Bai and S. Russell, "Efficient reinforcement learning with hierarchies of machines by leveraging internal transitions," in IJCAI International Joint Conference on Artificial Intelligence, 2017, vol. 0, pp. 1418–1424. [Online].

[35] G. E. Setyawan, H. Sawada and P. Hartono, "Combinations of Micro-Macro States and Subgoals Discovery in Hierarchical Reinforcement Learning for Path Finding," Int. J. Innov. Comput. Inf. Control, vol. 18, no. 2, pp. 447–462, 2022.

[36] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver and K. Kavukcuoglu, "FeUdal networks for hierarchical reinforcement learning," 34th Int. Conf. Mach. Learn. ICML 2017, vol. 7, pp. 5409–5418, 2017.

[37] Z. Yang, K. Merrick, S. Member, L. Jin, H. A. Abbass and S. Member, "Hierarchical Deep Reinforcement Learning for Continuous Action Control," IEEE Trans. Neural Networks Learn. Syst., vol. 29, no. 11, pp. 5174–5184, 2018.

[38] Z. Liang, J. Cao, W. Lin, J. Chen and H. Xu, "Hierarchical Deep Reinforcement Learning for Multi-robot Cooperation in Partially Observable Environment," Proc. - 2021 IEEE 3rd Int. Conf. Cogn. Mach. Intell. CogMI 2021, pp. 272–281, 2021.

[39] J. Yang, I. Borovikov and H. Zha, "Hierarchical cooperative multi-agent reinforcement learning with skill discovery," Proc. Int. Jt. Conf. Auton. Agents Multiagent Syst. AAMAS, vol. 2020-May, pp. 1566–1574, 2020.

[40] H. Clark and S. Brennan, "Emergence of grounded compositional language in multi-agent populations," 32nd AAAI Conf. Artif. Intell. AAAI 2018, pp. 1495–1502, 2018.