Applied Intelligence (2021) 51:1069–1085
https://doi.org/10.1007/s10489-020-01832-y

Analysis of coordinated behavior structures with multi-agent deep reinforcement learning

Yuki Miyashita[1,2] · Toshiharu Sugawara[1]

Published online: 16 September 2020

Abstract

Cooperation and coordination are major issues in studies on multi-agent systems because the entire performance of such systems is greatly affected by these activities. The issues are challenging however, because appropriate coordinated behaviors depend on not only environmental characteristics but also other agents' strategies. On the other hand, advances in multi-agent deep reinforcement learning (MADRL) have recently attracted attention, because MADRL can considerably improve the entire performance of multi-agent systems in certain domains. The characteristics of learned coordination structures and agent's resulting behaviors, however, have not been clarified sufficiently. Therefore, we focus here on MADRL in which agents have their own deep Q-networks (DQNs), and we analyze their coordinated behaviors and structures for the *pickup and floor laying problem*, which is an abstraction of our target application. In particular, we analyze the behaviors around scarce resources and long narrow passages in which conflicts such as collisions are likely to occur. We then indicated that different types of inputs to the networks exhibit similar performance but generate various coordination structures with associated behaviors, such as division of labor and a shared social norm, with no direct communication.

Keywords Multi-agent deep reinforcement learning · Cooperation · Coordination · Norm · Divisional cooperation

1 Introduction

Cooperation and coordination are important issues in the study of multi-agent systems, because they are essential to achieve the desired autonomous control to improve the overall efficiency in sophisticated cooperative tasks. It is difficult and complicated however, to identify appropriate coordination structures and behavioral rules or strategies, such as social norms and division of work areas. This is because appropriate cooperation and coordination depend highly on the environmental structure, the task characteristics, and the abilities and strategies of individual agents. Furthermore, these factors may vary. Therefore, it is almost impossible to design cooperative behaviors of autonomous agents for real-world applications in advance; rather, the agents themselves are expected to decide appropriate cooperative and coordinated actions based on their experience of joint tasks, thus establishing a cooperation regime.

On the other hand, recent technologies in computer and communication systems have enabled new applications and services that use collaboration among multiple robots, intelligent programs on distributed computers and smartphones, and intelligent sensors and devices. Cooperation and coordination in these applications are often necessary to cover wide environments, execute complex tasks requiring heterogeneous skills, and meet hard deadlines. These types of applications include pickup and delivery services [10, 12], large-area monitoring [6], and patrol and security surveillance [1, 23]. We are also involved in developing applications in which multiple robots operate at construction sites in a cooperative manner to help or work on behalf of human workers and builders. We are attempting to address labor

✉ Yuki Miyashita
y.miyashita@isl.cs.waseda.ac.jp

Toshiharu Sugawara
sugawara@isl.cs.waseda.ac.jp

[1] Computer Science and Communications Engineering, Waseda University, Tokyo 1698555, Japan

[2] Shimizu Corporation, Tokyo 1040031, Japan

shortages,[1] and we expect that robots will be able to help solve such problems. Examples of such tasks are assembling walls, laying floors, and moving heavy building materials at night for the work of human builders the next day.

Meanwhile, deep reinforcement learning (DRL) has produced many successful results for a number of applications, such as robotics [7, 17] and games [8, 14]. Some studies have extended the learning process of multi-agent systems by incorporating single-agent DRL, resulting in *multi-agent deep reinforcement learning* (MADRL) [5, 16]. The learning process in multi-agent systems requires all agents to explore a high-dimensional state-action space that include other agents in observable states. In particular, the behaviors of agents in a multi-agent system are affected by the other agents' behaviors, which may also vary with the learning progress of individual agents. As a result, currently positive training data may be treated as negative training data or noise in the future. Although various studies have indicated that MADRL enables successful learning of some coordinated behaviors, they have not clarified what types of coordination structures and behaviors emerge with MADRL, or how these structures and behaviors are affected by the information observed by agents and the associated inputs fed to their deep Q-networks (DQNs). In general, the detailed behavioral structures of multiple cooperative agents learned with deep neural networks are usually unpredictable; therefore, by analyzing a generated coordination structure, we can understand how robust the coordinated behaviors are and the conditions under which these behaviors may collapse. Such analysis would also help in designing coordination structures in MADRL with desired input information, if we clarify the emergence of coordination structures with various types of input information.

Therefore, we analyzed the coordination structures generated with MADRL for a problem called the *multi-agent pickup and floor laying problem*, which is an abstraction of our target problem in which multiple robots work at a construction site on behalf of human builders. Our challenge is to investigate how agents with distributed concurrent learning using their own DQNs can establish spatial coordination regimes and whether they can generate shared social norms and coordinated behaviors to avoid conflicts. We also attempt to see what types of information in the inputs to the DQNs result in certain coordination and cooperative behaviors. Therefore, we prepare a few types of input structures by combining locally possible views with various local beliefs such as the (estimated) absolute location and the history of an agent's locations (trajectory). We have already reported this issue in the study of Miyashita and Sugawara [29], but that report was based on limited experimental results. Hence, we conducted additional experiments by feeding other types of input structures to the DQNs, enabling deeper analysis of the coordinated behaviors learned by all agents.

[1]A shortage of workers due to a decreased birthrate is a serious problem in some countries, and robots are expected to be used at construction sites to compensate for such labor shortages.

Our experimental results indicate that the agents in distributed MADRL could generate a number of coordinated structures for cooperative work, but the structures were quite different depending on the input information fed to the agents' own DQNs. For example, when the agents had absolute locations in the environment, they formed divisional cooperation by spatial segmentation. When their inputs included the trajectories of their movements, all the agents could find a social norm by which each agent incorporated a one-way rule into its behavior to avoid collisions, without explicit, direct communication. Moreover, agents with only a local view could not generate explicit cooperation structures, but they could still learn opportunistic but flexible synchronized coordinated behaviors to avoid collisions. We also discuss the relationships between the agents' coordination structures and performance in the *pickup and floor laying problem*.

2 Related work

Multi-agent reinforcement learning (MARL) has been extensively studied [2, 3, 21, 28]. Xie et al. [28] acquired agents' cooperative behavior by using extended Q-learning in which agents share the Q-table. Real-world applications of those methods, however, are limited, because real-world problems are dynamic and complicated; thus, the state spaces are too large to learn Q-functions accurately. DRL, which is a kind of reinforcement learning (RL) method in which an agent learns Q-functions by using deep neural networks, has enabled an agent to learn policies for its problem solving behavior in many sophisticated application domains [8, 9, 14, 15, 17]. For example, Lample et al. [8] proposed a learning method using DRL in a 3D first-person shooter (FPS) game by exploiting game features such as knowing where and when enemies are likely to appear. Hasselt et al. [26] proposed an effective method to learn policies by using a dueling network architecture for DRL. It has also been demonstrated that experience replay and prioritized experience replay [19] can enable high-quality learning results by replaying important transactions for training.

Research on MADRL has received much attention in recent years [11, 18, 24] For example, Diallo et al. [4] showed that a large number of agents can behave cooperatively and generate strategic team formations of more than 100 agents by sharing a centralized DQN with teammates in adversarial multi-agent games. Shao et al. [20] proposed a curriculum transfer learning method for MADRL to solve problems with difficult scenarios in StarCraft, a real-time strategy game, and this method accelerates the training process, thereby improving the learning performance. Palmer et al. [16] proposed an extension of MADRL, called *lenient learning*, in which agents possess temperature values for individual state-action pairs and decay the values to avoid the use of outdated pairs. Sartoretti et al. [18] proposed an *asynchronous advantage actor-critic* (A3C) algorithm to enable multiple agents to learn a homogeneous, distributed policy in a multi-robot construction problem. By using that method, agents could learn to work together toward a common goal without explicit communication. To achieve proper cooperation through policy sharing between teammates, Mao et al. [13] presented the *attention multi-agent deep deterministic policy gradient* (ATT-MADDPG) method, which extends the *deep deterministic policy gradient* (DDPG) [22] and actor-critic methods and then showed better scalability and robustness for learning. Unfortunately, those studies have still not clarified how agents' observations with associated local beliefs fed to DQNs affect the coordination behaviors and structures learned in MADRL. Therefore, we address this issue here to clarify what types of strategic cooperative and coordinated behaviors are autonomously established in a distributed manner with the concurrent learning of multiple agents when we vary the types of information input to the agent's DQNs.

3 Problem formulation

We focus on a multiple-robot system to help human workers by, for example, transporting heavy building materials, welding steel columns, attaching ceiling panels, and laying flooring materials at construction sites. Because these work spaces are usually large, a multi-agent system requires cooperation and coordination techniques to operate many robots autonomously and concurrently in complicated environments. In addition, the structures of these environments may change day by day, because some building materials may be placed near a new work space; as a result, building materials may be present in what used to be an empty working space when a task, such as wall installation, is to be performed. In particular, this paper focuses on the control problem of cooperation and coordination among multiple robots for floor-laying work.

To address this control problem, we introduce the *pickup and floor laying problem*, which is an abstraction of a multi-agent problem for multiple robots laying floors material. In this problem, a team of agents is scattered in an environment. Each agent begins by moving to the pickup location for some flooring material. Upon reaching its destination, an agent picks up, carries, and installs the flooring material in a cell where no material has been laid yet. Once the installation is complete, the agent returns to the pickup location. The agents continue this come-and-go process until the flooring materials are spread throughout the installation area. Figure 1a shows an example environment, in which the field of the environment is expressed by $N \times N$ cells. Here, the black circles represent the agents, the black cells represent the obstacles that agents cannot move into, the green cells represent the storage area for picking up flooring material, and the yellow cells represent the installation area, where one piece of flooring material must be installed at each cell. For simplified description, we call a piece of material a *task* and the action of laying it a *task execution*. We also call a cell where no material has been laid yet an *empty cell*.

Next, we introduce discrete time $t \geq 0$, whose unit is called a *tick*. We denote the set of n homogeneous agents as $I = \{1, \cdots, n\}$ and the discrete action space of agents as $A = \{up, right, down, left\}$. An agent $i \in I$ can pick up at most one task (i.e., a piece of material) and carry it to an empty cell by taking one of the actions in $A = \{up, right, down, left\}$. If two or more agents attempt to move into the same cell, one agent selected randomly can successfully move, and the other agent or agents stay at the same location. This is based on the assumption that robots have primary functions to avoid unsafe behaviors and damage, as in a collision. In Fig. 1a, We represent an agent with a task as a circle with a green interior. We assume that there are sufficient tasks in the storage area so that the tasks at any green cell will not run out. Immediately after i picks up a task, i carries it to the installation area and executes it at an empty cell. When i executes a task successfully, i earns a reward $r > 0$. Then, the empty cell at which i executed the task becomes an executed cell (i.e., an installed cell).

The formal expression of our problem can be expressed by a tuple $\langle I, N, m, E, \{S_i\}, \{A_i\}\rangle$. Here, m is the number of empty cells. $E(\ni e)$ is the set of all possible states of the environment, including the states of the cells and all agents. We assume that $\forall i \in I$, an agent can observe the current state $s_{i,t}$ consisting of a limited local area whose center is i itself at time t. Thus, $s_{i,t}$ is a subset of the entire state e_t at t ($s_{i,t} \subset e_t \in E$). Let S_i be the set of possible local states of i. We assume that agents can observe their local areas accurately, so that the local state $s_{i,t}$ at t is correct. We also define $\mathcal{A} = A_1 \times \cdots \times A_n \ni a_t = (a_{1,t}, \ldots, a_{n,t})$ as the agents' joint action space, where A_i is the set of all possible actions of $i \in I$, and we assume that all agents have the same set of actions, i.e., $A_i = A$ for $\forall i \in I$.

The agents may receive a *reward* $r_i(e_t, a_t)$ right after their joint actions a_t in e_t, and then the environment state changes to e_{t+1}. The reward that each agent receives is based only on the current state $e_t \in E$ and the joint action

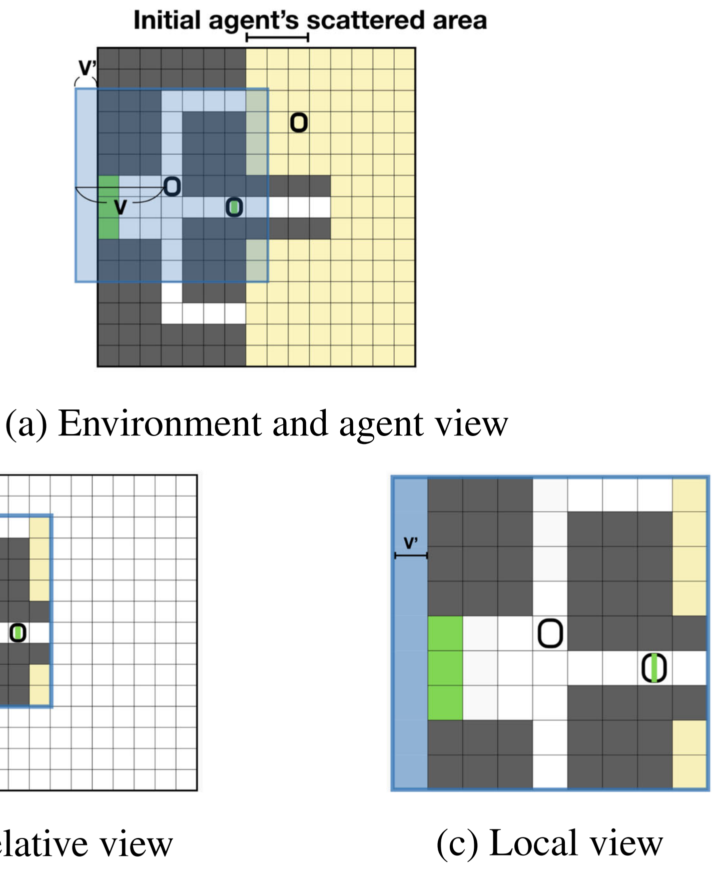
Initial agent's scattered area
V'
V
(a) Environment and agent view
(b) Relative view
V'
(c) Local view

Fig. 1 Example of an environment and relative and local views

$a_t \in \mathcal{A}$. Then, i individually chooses actions according to only the observed local state at t and the learning results so far. Because we focus on MADRL for a *multi-agent pickup and floor laying problem*, the agents individually learn the Q-values and associated *policies* by using their own DQNs to identify cooperative and coordinated behaviors without direct communication. Note that a policy π_i of i is a function from the set of local states S_i to the set of actions, $A = A_i$.

The *pickup and floor laying problem* goes through the following process. The installation area (consisting of 108 cells) and the storage area (consisting of 3 cells) are set as shown in Fig. 1a. Before the process starts (at $t = 0$), all agents in I are placed at random locations in the region of 3×15 cells on the left side of the installation area, not including obstacles (see Fig. 1a). All agents perform the following steps concurrently.

(a) Agent $\forall i \in I$ chooses an action $a_{i,t}$ in e_t according to its policy π_i at t, so $a_{i,t} = \pi_i(s_{i,t}) \in A$.
(b) If i arrives at the storage area (green cells in Fig. 1a), i picks up a task ψ_i and carries it until i arrives at any empty cell in the installation area (yellow cells in Fig. 1a). Then, after i carrying ψ_i arrives at the empty cell, i executes ψ_i (i.e., i lays the piece of flooring material at the cell). Finally, i receives a reward $r_{i,t} = r$, and the empty cell becomes an executed cell.
(c) After all agents take one action at t, the environment e_t changes to its next state, e_{t+1}.
(d) When all empty cells in the installation area have become executed cells at t, or when t exceeds an epoch length H (i.e., $t \geq H$, where H is a positive integer called the *maximal rounds per epoch*), the epoch ends; otherwise, $t \leftarrow t + 1$ and the process returns to step (a) for the next round.
(e) Another epoch begins from step (a) after the environment is initialized, until F_e epochs are complete, where $F_e > 0$ is an integer.

The goal of the agents is to increase the rewards that they receive, and thus, they attempt to lay as many pieces of flooring material as possible in the installation area.

Two particular aspects require certain coordinated activities in the environment (see Fig. 1a). First, the agents must coordinate their behavior to avoid colliding with each other in the long passages connecting the storage and installation areas. Even if agents do not collide, it is costly for an agent to move back into a passage to pass another agent. The second aspect is the behavior in a small, insufficient space. Because the storage area is small, if some agents with tasks try to move right, but other agents try to move left to pick up a task, their movements in opposite directions may also result in collisions. In addition, if many agents gather in this small area, their movements may cause a deadlock. We thus investigate how learning agents with DQNs can generate certain coordinated behaviors to avoid such conflicted and deadlocked states.

4 Learning methods

4.1 Deep reinforcement learning with local observation

A DQN is an RL method using deep neural networks that estimates a Q-function and an associated policy π. We adopt the distributed concurrent learning approach, in which all agents have their own neural networks to learn Q-values.

In conventional Q-learning, an agent learns the action-value function Q to maximize the sum of the cumulative discounted rewards. If the discounted future reward at t is denoted by $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, then the Q-function with the associated policy π is defined as

$$Q^{\pi}(s, a) = \mathbb{E}[R_t | s = s_t, a = \pi(s_t) = a_t],$$

where $0 \leq \gamma < 1$ is the discount factor, and positive integer T is the horizon step. Then, the optimal action value $Q^*(s, a)$ can be defined as

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[R_t | s = s_t, a = \pi^*(s_t) = a_t],$$

where π^* is the optimal policy according to $Q^*(s, a)$. Q-learning [27] is thus a method to approximate the optimal Q-value $Q^*(s, a)$, where

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') | s, a],$$

and $Q^*(s', a')$ is the estimated optimal Q-value for the next state s'. Unfortunately, it is often hard for Q-learning to converge when the problem space is complex and large.

In a DQN, the Q-values are estimated by a deep neural network with associated parameters θ whose values are updated through the agent's experience. At time t, to obtain approximations of the optimal Q-values from the network, the $\theta_{i,t}$ of the network of agent i at t are updated to reduce the mean squared loss function $L_{i,t}(\theta_{i,t})$, which is defined as

$$\begin{aligned} L_{i,t}(\theta_{i,t}) &= \mathbb{E}_{(s_i, a_i, r_i, s_i')}[(r_i + \gamma \max_{a_i'} Q_i(s_i', \arg\max_{a_i'} Q_i(s_i', a_i'; \theta_{i,t}); \theta_{i,t}^-) \\ &\quad - Q_i(s_i, a_i; \theta_{i,t}))^2]. \end{aligned}$$

Note that we use a double DQN [26], meaning that the target network parameters $\theta_{i,t}^-$ are periodically copied from the main Q-network parameters $\theta_{i,t}$ every H time steps (i.e., every epoch). Then, actions are selected using the main Q-network, but the $\theta_{i,t}$ in this Q-network are updated using the Q-value based on the target network's $\theta_{i,t}^-$. Although it may lead to suboptimal policies, we adapt the double DQN to avoid overestimating Q-values. We think that this is particularly important for multiple agents learning coordinated behavior to prevent confusion due to outdated, noisy training data.

Next, we propose feeding the aggregated inputs, which combine the observation $s_{i,t}$ of i at time t and part of the local belief of i as additional information, to the agents' DQNs. This aggregated input is called a *view* and denoted by $v_{i,t}$. The agents decide their next actions by using the policy functions derived from the Q-functions, and the individual Q-values are the outputs from the local DQNs. Therefore, we extend the domain of Q_i and π_i as follows:

$$Q_i : \mathcal{V}_i \times A_i \longrightarrow \mathbb{R}, \text{ and } \pi_i : \mathcal{V}_i \longrightarrow A_i,$$

where $\mathcal{V}_i$ is the set of views of i, i.e., observed states with the local belief of i. This also means that the input to a DQN is $v_{i,t}$ instead of $s_{i,t}$. Note that, if no local information is aggregated into $v_{i,t}$, then $v_{i,t} = s_{i,t}$, and the policy in this paper is identical to the conventional policy. Section 4.3 gives examples of these *views*. Later we explain what kinds of local belief are added to $s_{i,t}$.

4.2 Experience replay

To avoid overfitting and stabilize the parameters, we adopt *experience replay*; thus, the parameters of a DQN are updated through sampling of the agent's past experiences from its local memory at t, $D_{i,t}$. Hence, agent i adds new experience data $c_{i,t} = (v_{i,t}, a_{i,t}, r_{i,t}, v_{i,t+1})$ at time t into its own previous memory $D_{i,t-1} = \{c_{i,t-d_m}, \cdots, c_{i,t-1}\}$, where d_m (> 0) is the memory capacity. After i stores a sufficient number of experiences into its own memory, i updates the parameters $\theta_{i,t}$ every η steps to minimize the value of the loss function $L_{i,t}(\theta_{i,t})$, which is defined by

$$\begin{aligned} L_{i,t}\ (\theta_{i,t}) \\ = \ & \mathbb{E}_{(v_i, a_i, r_i, v_i') \sim U(D_{i,t})}[(r_i + \\ & \gamma \max_{a_i'} Q_i(s_i', \arg\max_{a_i'} Q_i(s_i', a_i'; \theta_{i,t}); \theta_{i,t}^-) \\ & - Q_i(v_i, a_i; \theta_{i,t}))^2], \end{aligned}$$

where $U(D_{i,t})$ is randomly sampled experience data stored within experience memory $D_{i,t}$.

We update the parameters of the DQNs by using gradient descent to decrease the value of the loss function $L_{i,t}(\theta_{i,t})$. In our experiments, We applied RMSprop [25] whose learning rate for parameter updates did not change throughout the experiments. We think that RMSprop is suitable for MADRL because the environment of decentralized learning by a multi-agent system is usually unstable.

4.3 Types of views

Agent i observes the current state $s_{i,t}$ in accordance with a limited local view specified by an *observable range size* V_i, which is a nonnegative integer. We show an example of this observable range, with size $V_i = V$, in Fig. 1a. In the figure, the observable range is represented as a blue square, and i itself is at the center of the observable range. Within this range, an agents can correctly locate all cells, storage and installation areas, other agents, and tasks held by other agents. As shown in Fig. 1a, the observable range may cover areas outside the environment (in this example, the size is $(2V + 1) \times V'$ and V' is 1), but of course, i cannot obtain any information from the outside area. Next, we introduce three kinds of agent views, $v_{i,t}$, that are the input to the DQNs.

4.3.1 Relative view

A *relative view* (RV) $v_{i,t}$ for input to a local DQN is generated by composing its observed state $s_{i,t}$ and the entire map, as shown in Fig. 1b. Because i cannot observe outside its range, the unobservable regions are assumed blank (and thus filled with zeros). The RV of i includes the abstract map of the environment and its current position. The input to the individual DQN consists of five channels of $N \times N$ lattices, as shown in Fig. 2. The first lattice contains the agent's current location (Fig. 2a). The second lattice contains the locations of other agents (Fig. 2b), the third includes obstacles (Fig. 2c), and the fourth represents empty cells in the installation area (Fig. 2d). Finally, the fifth lattice includes tasks carried by agents (including itself) and tasks in the storage area (Fig. 2e). Therefore, the DQN of agent i can know whether i has a task from the fifth lattice. Any characteristic cell is represented as 1, except for obstacle cells, which are represented as -1 in these lattice inputs.

4.3.2 Historic relative view

A *historic relative view* (HRV) is a modified RV generated by combining an RV and part of an agent's own trajectory in the history memorized by individual agents. The HRV of agent i is generated by adding the stored trajectory data to two lattices of the RV input shown in Fig. 2. First, the trajectory data is added into the lattice representing the location of i (Fig. 2a). That is, the current location of i is expressed as 1 and its location k ticks ago is expressed as β^k if β^k is larger than δ, where δ is a parameter cutoff to decide whether to include trajectory data in an HRV, and $\beta \in [0, 1]$ is a decay rate. This means that the trajectory data before a certain tick is not reflected in an HRV. Note that, if an agent was in a certain cell more than twice within the last k ticks, only the larger value (i.e., the more recent one) is represented in the HRV.

Second, when agent i carries a task, its trajectory data is also included in the input of the lattice that expresses the locations of obstacles (Fig. 2c); the values of the cells on a trajectory are also represented by β^k, so they are equal to the values in the lattice expressing the current location and part of the trajectory. Note that agents have to store

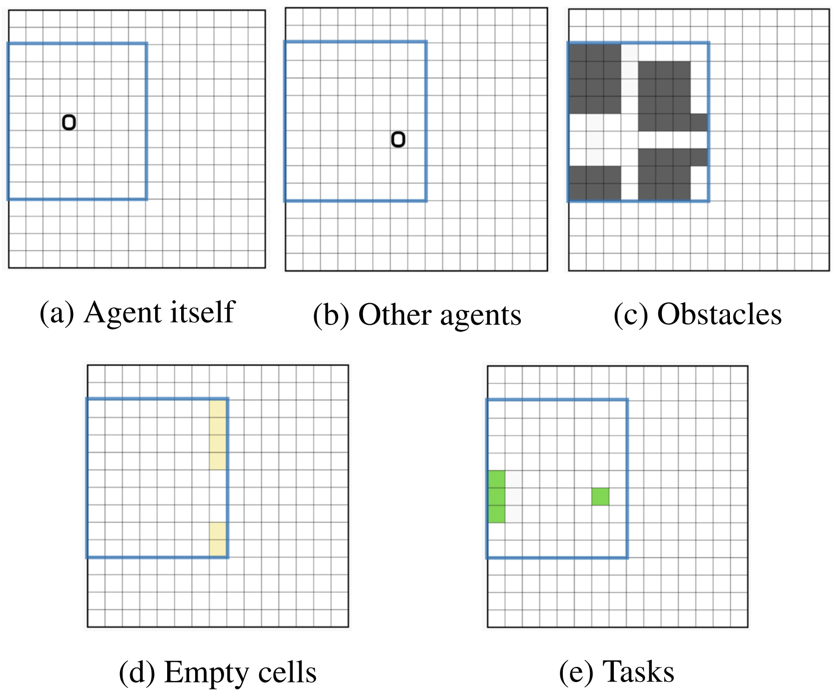

Fig. 2 Input structure for a relative view

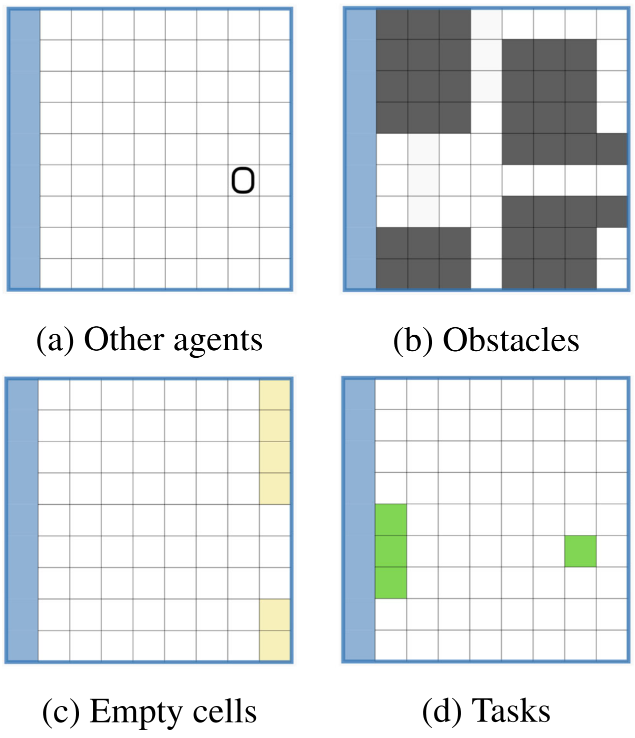

Fig. 3 Input structure for a local view

locations at a time more recent than a predetermined time in the environment to generate an HRV.

4.3.3 Local view

A Local view (LV) is the observed state of agent i ($v_{i,t} = s_{i,t}$) with no additional information, as shown in Fig. 1c, where the area outside the environment within the observable range is shown as blue cells. We assume that i has the ability to see the wall and fill the cells outside the wall with values of -1. Agent i is always located at the center of the its own observation. The input fed to the local DQN consists of four $(2V_i+1)\times(2V_i+1)$ sublattices, as shown in Fig. 3. The first sublattice includes only other agents (Fig. 3a), the second sublattice includes only obstacles (Fig. 3b), and the third sublattice includes empty (i.e., uninstalled) cells (Fig. 3c). The fourth sublattice expresses the locations of tasks that are in the storage area and tasks that some agents (including agent i itself) currently carry (Fig. 3d). Therefore, if i holds a task, it appears at the center of the fourth sublattice.

4.4 Neural network architecture

In MADRL, agents independently decide their next actions by using their own policies derived from local DQNs. Table 1 lists the specifications of the neural network architecture for deep Q-learning. The DQN consists of two convolutional network layers, one max pooling layer, and three fully connected network (FCN) layers. The data structures of the input and output layers are determined in accordance with the channel and type of the agent's *view*; thus, in Table 1, the size of the input lattice/sublattice and the number of lattices/sublattices channels are denoted by parameters M and F_c, respectively. The DQN's outputs are a list of pairs of actions and Q-values, so that each agent can identify the best action with the highest Q-value according to the learning results so far (if there is a tie between several actions, one action is selected at random).

We adopt the ε-greedy strategy with decay, meaning that an agent chooses the best action according to the learning results so far with probability $1-\varepsilon_{i,t}$, or a random action with probability $\varepsilon_{i,t}$. In this paper, agent i decays its greedy parameter $\varepsilon_{i,t}$ by $\varepsilon_{i,t} = \max(\varepsilon_{i,t-1} * \gamma_\varepsilon, \varepsilon_l)$, where γ_ε is the decay rate and ε_l is a lower limit. We set $\varepsilon_{i,0}$ (< 1) to a number near 1 as the initial value, so that the ε-greedy strategy with decay enables agents to take various actions in the earlier learning stages. Gradually, overtime, the agents will make learned choices more often.

5 Experiments and discussion

5.1 Experimental setting

We examined the difference in task execution performance when various types of data were input to agents' DQNs. First, we evaluated the performance on the *pickup and floor laying problem* by agents with the proposed input information (RVs, HRVs, and LVs) and various observable

Table 1 Network architecture

Layer	Input	Filter size	Stride	Activation	Output
Convolutional	$M \times M \times F_c$	2×2	1		$M \times M \times 32$
Convolutional	$M \times M \times 32$	2×2	1		$M \times M \times 32$
Max pooling	$M \times M \times 32$	2×2	2		$M/2 \times M/2 \times 32$
FCN	$M/2 \times M/2 \times 32$			ReLU	512
FCN	512			ReLU	256
FCN	256			Linear	4

Table 2 Learning parameters

Parameter	Value
Discount factor γ_q	0.95
Initial value $\varepsilon_i = \varepsilon_{i,0}$	0.99999
Decay rate γ_ε	0.9999995
Lower limit ε_l	0.02
Momentum for RMSprop, α	0.90
Learning rate for RMSprop, l_r	0.00001
ε for RMSprop, ε_{rms}	1e-07
Parameter update interval in steps η	4

range sizes. To evaluate the performance, we counted the number of tasks executed by all agents and recorded the time required to complete the installation of flooring material. Then, we analyzed what coordination structures emerged depending on the different inputs to individual DQNs. We listed the parameter values used in our experiments in Tables 2 and 3; we set these values to maximize the performance obtained by the learned agents. In these experiments, eight agents moved around to install materials in the environment shown in Fig. 1a; thus, the number of cells to install, m, was 108. The trajectory decay rate β used by agents with HRVs and the lower threshold for the memorized trajectory, δ, are listed in Table 3. From those values, an agent could store its locational data for the past 30 ticks in its memory as the trajectory. Furthermore, in Section 5.5, we discuss whether agents could or could not avoid collisions by constructing coordination structures.

5.2 Performance comparison

First, we investigated whether agents perform effectively by using their own DQNs with RV inputs, what characteristics of cooperative behavior emerge among agents, and how the observable range size V affects their performance and coordinated behavior. Figure 4 shows moving average lines of the total executed tasks by using the values from 100 recent epochs until 25000 epochs, whith the agents having RVs with different size of observation range size of $V = 4, 7, 10,$ and 15. Note that, because $N = 15$, $V = 15$ meant that the agents could correctly observe the entire environment regardless of the view representation.

This figure indicates that the performance could increase quickly for any of the observable range sizes, and that the agents could execute tasks (install flooring materials) in the installation area (which had 108 cells). A closer look at the figure shows that when $V = 4$, the agents often could not complete all the tasks (a few tasks sometimes remained). Figure 4 also indicates that when $V = 7$, the performance improved slightly faster, but when $V = 15$, it improved more slowly. Therefore, there were appropriate values to speed up convergence.

Next, to analyze the performance in detail, the required time to complete the installation was plotted, as shown in Fig. 5, for $V = 4, 7, 10,$ and 15, where each plot is a 100-epoch moving average. When $V = 4$, a few tasks remained incomplete. In this case, for simplicity, we assume that it took 800 ticks to complete the remaining tasks. When $V \geq 7$, the agents could decrease the required time along with the number of epochs in all cases. The performance

Table 3 Parameters and their values

Parameter	Value
Number of agents, n	8
Size of environment, N	15
Number of executed cells, m	108
Reward r per task	1
Observable range size V	4, 7, 10, 15
Mini batch size $\lvert U(D_{i,t})\rvert$	32
Memory capacity d_m	2000
Sum of epochs, F_e	25,000
Epoch length H	800
Lower threshold for trajectory, δ	0.05
Trajectory decay rate β	0.9

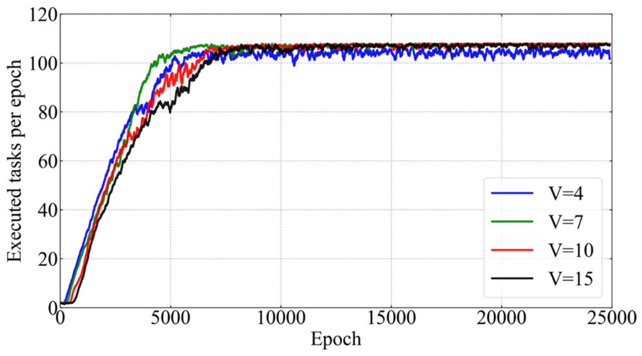

Fig. 4 Executed tasks per epoch (RVs)

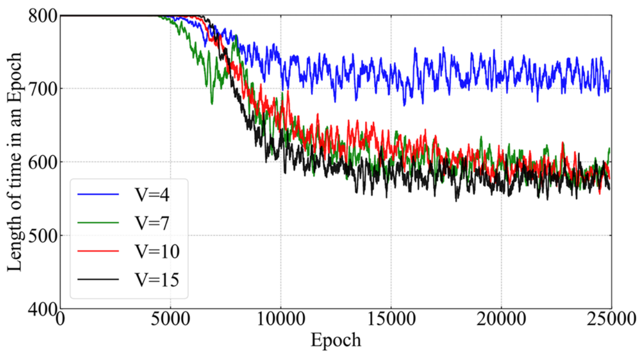

Fig. 5 Required time to completion (RVs)

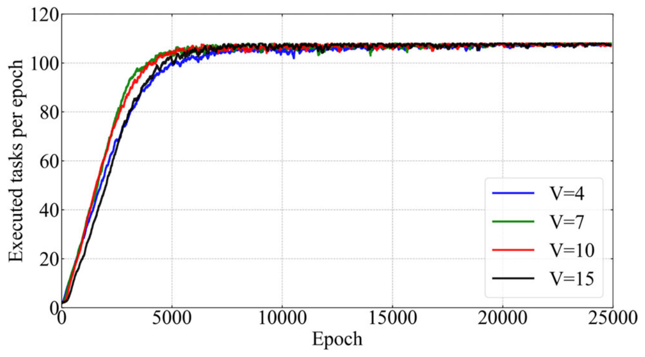

Fig. 6 Executed tasks per epoch (HRVs)

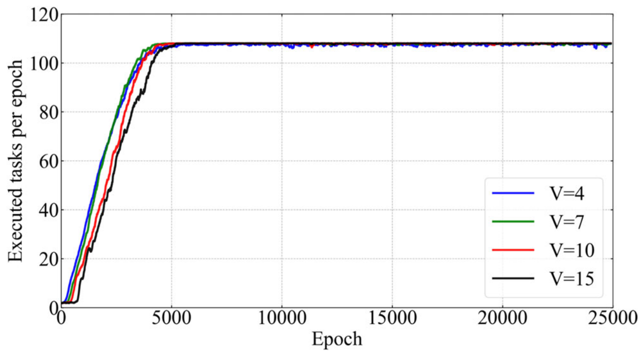

Fig. 8 Executed tasks per epoch (LVs)

looks slightly worse when $V = 15$, but generally, the performances was almost identical except for $V = 4$.

In our second experiment, we verified the performance on collaborative work when all agents used HRVs as inputs to their DQNs. We show the performance results in terms of the number of executed tasks and the time required to completion of all tasks in Figs. 6 and 7, respectively. As seen in the figures, the agents with HRVs improved their performance over epochs like those with RVs did, but their convergence was slightly faster than that of the agents with RVs (Figs. 4 and 5). By comparing Fig. 6 with Fig. 4, however, we can confirm that there was no obvious difference in the time required for the agents to complete the installation, except when $V = 4$, but in that case, the agents could still complete all tasks.

Next, we examined the effect of LVs in a third experiment whose setup was identical to that of the first experiment. Figure 8 plots the total number of executed tasks, indicating that the agents with the LVs could also improve their performance and complete all the installation tasks with any value of V. Comparing these results to those with RVs and HRVs, we find that convergence with LVs was the fastest among all view representations. We also plotted the time required to complete the installation, as shown in Fig. 9. We find that the agents with LVs always exhibited better performance for any observable range size V as compared with the performance by agents with RVs and HRVs. In addition, a close comparison of Figs. 5, 7, and 9 shows that the learning of agents with the LVs was faster than the learning of the others agents, and their convergence time for completing the installation was smaller (i.e., better) than that of the other agents.

5.3 Emergent coordination structures

Our previous three experiments showed that the performance difference among the different views was not so large, but each type of view generated quite different coordination structures. To show these differences, we created the heatmaps, in Figs. 10, 11, and 12, to indicate the numbers of executed tasks by individual agents at various cells when the range size V was 15. Note that these heatmaps were generated from the average numbers of tasks executed for every 50 epochs in the last 5000 epochs (thus, 20,000 to 25,000 epochs). Note also that we only show heatmaps for $V = 15$ here, but in the other cases (except for when $V = 4$), a similar tendency was observed.

Figure 10 shows that agents with RVs as input to their DQNs established divisional cooperation, meaning that each agent found its own locations at which to execute tasks. In the experiment using agents with HRVs, the structure of divisional cooperation appeared but was weaker. On

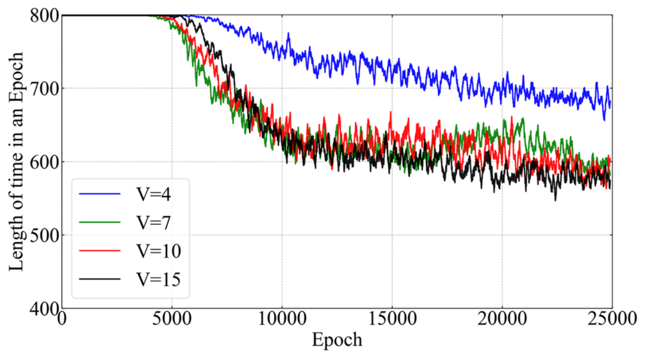

Fig. 7 Required time to completion (HRVs)

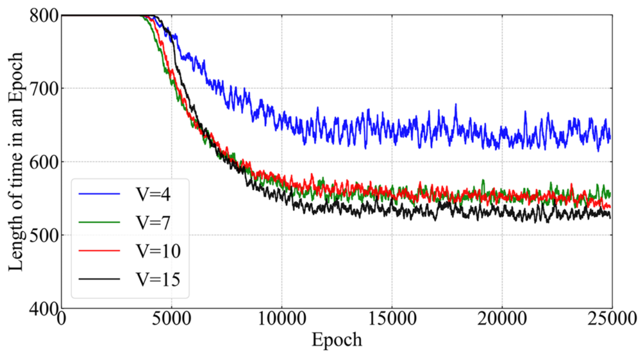

Fig. 9 Required time to completion (LVs)

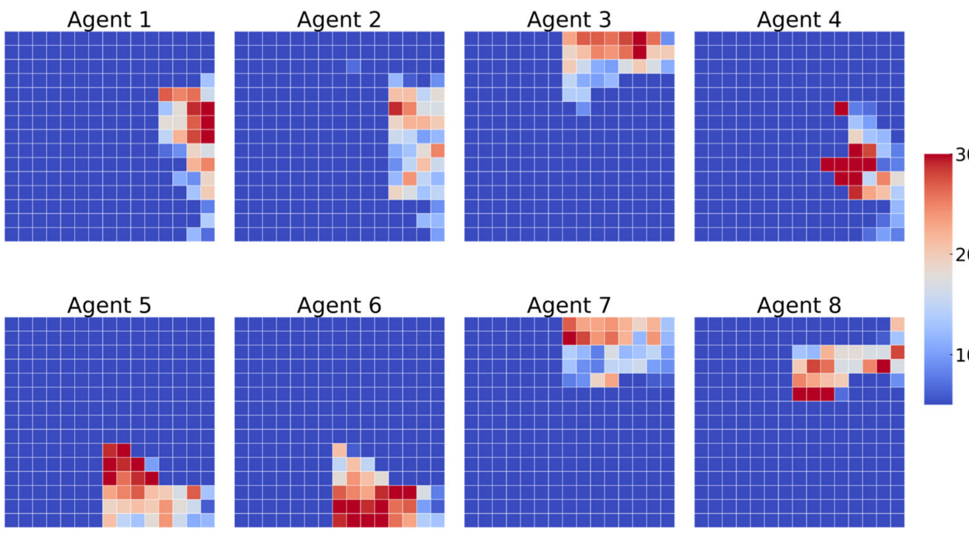

Fig. 10 Executed locations (RVs)

the other hand, the agents with LVs did not have such cooperation structures, and instead, they flexibly executed tasks at any place in the installation area. Division of labor may, in general, be an effective strategy for cooperation, because it prevents duplication of work, unnecessary competition, and collisions. Our experimental results were different, however, and the agents with the LVs achieved the most efficient strategy.

We also found a crucial result that the routes that the agents took turned out to be characteristic for each *view*. For $\forall i \in I$, we counted the number of visits by i to each cell, and we show the results in Figs. 13, 14, and 15 via heat maps in which a darker blue indicates more visits. Because the routes connecting the storage and installation areas were limited, the figures emphasize those routes.

The agents with RVs almost uniquely fixed their routes for the round trip (Fig. 13). These fixed routes were convenient for going back and forth between the work place and the storage area (see Fig. 10). Because there were only three passages, however, between the storage and installation areas, each agent had to share one of the passages with a few other agents; thus, they often collided, returned, or synchronized with other agents to pass. In contrast, Fig. 14 indicates that the agents with HRVs behaved quite differently. All agents shared the center passages, but they were also divided in half in terms of taking the upper or lower passage.

To understand the route selection behavior by the agents with HRVs, we generated the heatmaps shown in Figs. 16 and 17, which indicate the counts of visits when the agents had tasks and did not have tasks, respectively. Here, the darker cells indicate that the corresponding agent visited those cells more frequently. These figures clearly indicate that the agents generated a shared behavior or social *norm* by incorporating a one-way rule to avoid collisions: when they had tasks, they took the center passage, but after

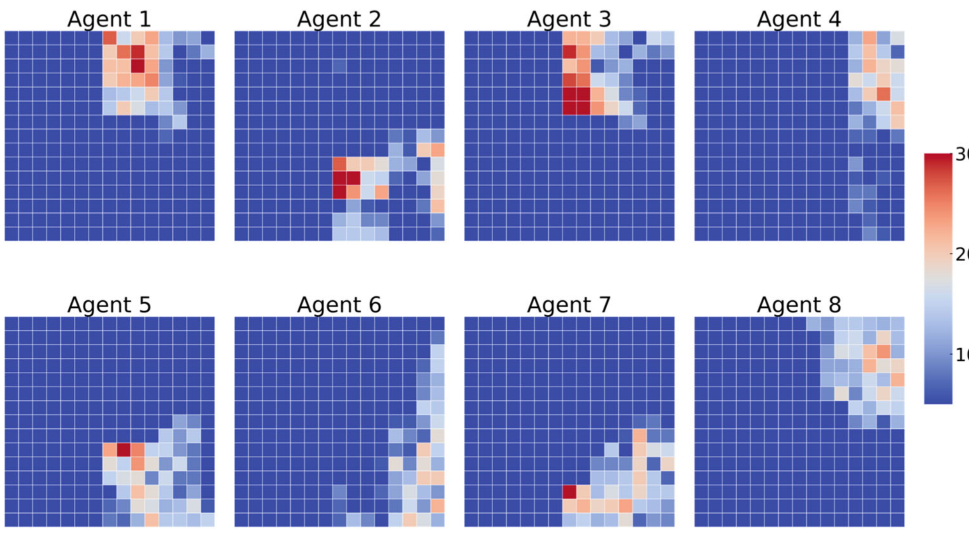

Fig. 11 Executed locations (HRVs)

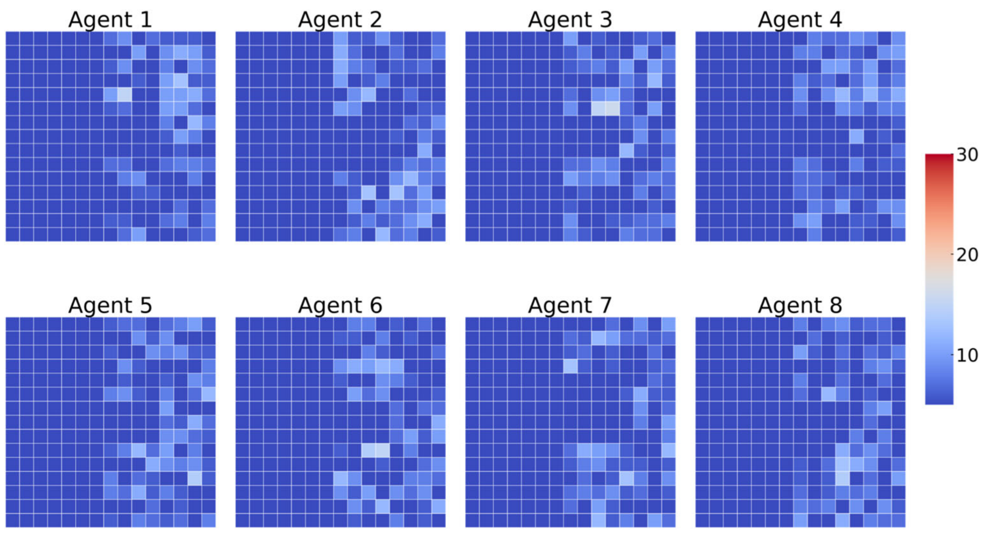

Fig. 12 Executed locations (LVs)

executing tasks, they returned to the storage area by using the other passages. This is more reasonable and easier for coordinated behaviors than the behaviors of the agents with RVs to reduce the inefficiency due to collisions. The routes that they took, however, were often not the shortest to reach the work place or the storage area.

On the other hand, we can confirm that the agents with the LVs took any of the three passages when they moved to the installation area or returned to the storage area as shown in Fig. 15. According to that figure and Fig. 12, they dynamically chose the cells where they should execute tasks without creating their own work places. In addition, when they moved to the work places and returned to the storage areas, they flexibly selected shorter routes. Although not shown here, it could be observed that all the agents moved synchronously along the shortest routes between the storage and installation areas, and such synchronized behavior seemed to reduce collisions. Therefore, the agents whose input to the DQNs were LVs could exhibit better performance than other agents could. Because an LV are simpler and smaller than the other views, we believe that these agents' convergence was faster. A similar phenomenon occurred for all observable range sizes of the LVs. This indicates that the agents constructed flexible coordination to avoid collisions without any specific norm like that identified by the agents with HRVs; furthermore, such flexible coordination resulted in effective work in our experiments, rather than coordination based on a specific norm. Note that similar coordination structures were established by agents with HRVs whose observable range size V was 4. The performance of agents with LVs ($V \geq 7$), however, was much better than that of the agents with the small-sized HRVs.

5.4 Behavior in small spaces

Figures 13, 14, 15, 16, and 17 reflect the agents' behaviors in the small space near the storage area. The agents with

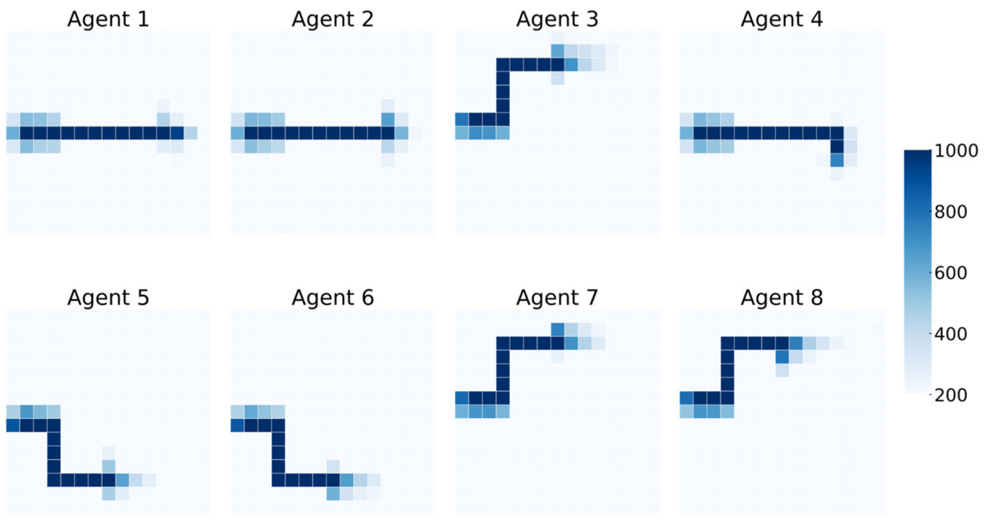

Fig. 13 Locations of each agent's visits (RVs)

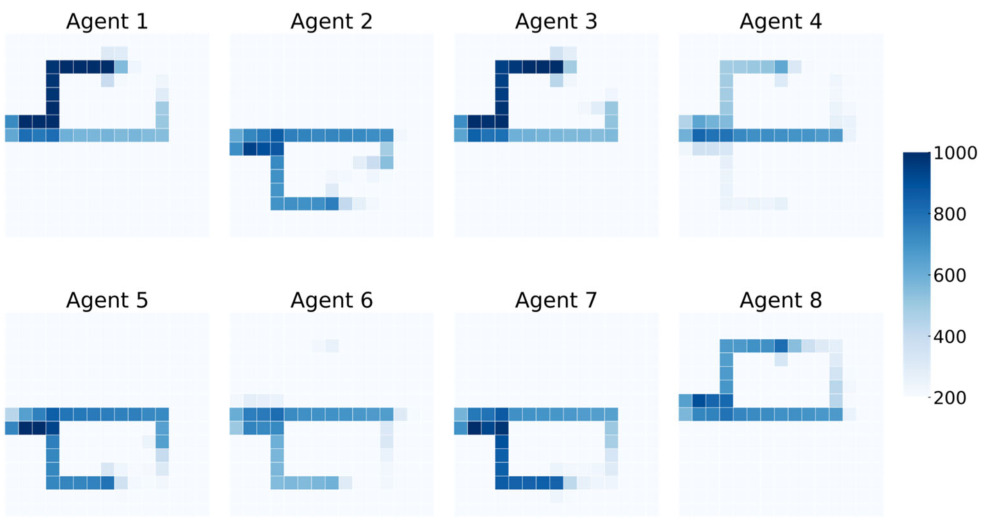

Fig. 14 Locations of each agent's visits (HRVs)

RVs had fixed routes, and thus, they also approached the nearest cell in the storage area. Figure 18 illustrates their movement in detail. Of course, when two agents were likely to collide in this space, one of them detoured; these detours are shown by the light-colored cells next to the dark-colored main routes in Fig. 13. On the other hand, Figs. 14 and 16 show that the agents with HRVs took better routes in this small area: they entered the small space from the upper or lower passages and left along the center passage as shown in Fig. 19. Therefore, they rarely collided, and did not need to detour; indeed, there are few light-colored cells in Fig. 14. Finally, because the agents with LVs choose routes flexibly (see Fig. 15), they might have collided in the small space but could also detour flexibly; thus, they could avoid serious deadlocks.

5.5 Discussion

Our experimental results showed that the agents generated various types of coordination structures by using the different kinds of input information in distributed MADRL. Because of the nature of the *multi-agent pickup and floor laying problem*, actions for collision avoidance in the narrow passages and the insufficiently small space near the storage area reduced the overall performance. Therefore, the agents identified coordinated behaviors that did not cause such actions for collision avoidance. The agents with RVs, which include the absolute locations in the environment, could generate spatial divisional cooperation on the basis of locational segmentation in which agents took specific routes to their work places. The agents with HRVs generated

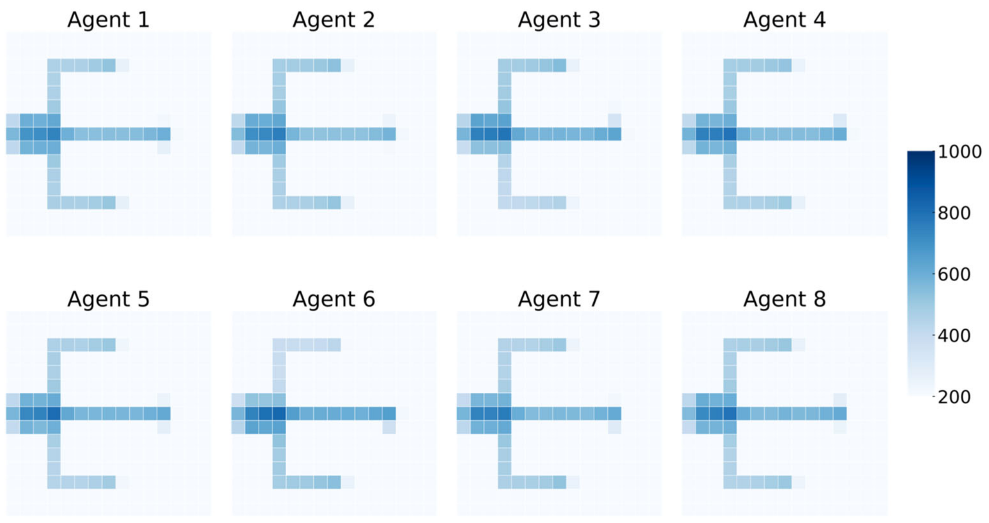

Fig. 15 Locations of each agent's visits (LVs)

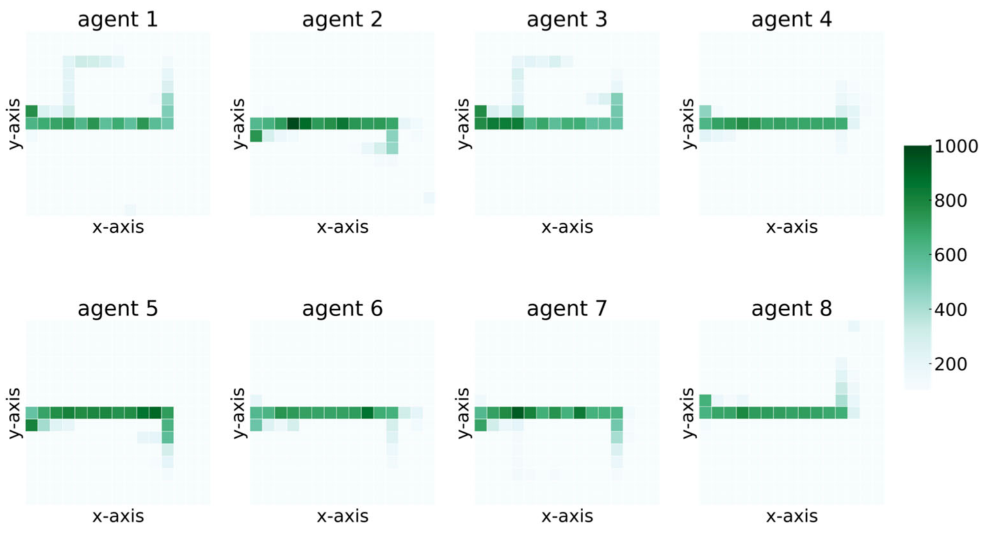

Fig. 16 Locations of each agent's visits with a task (HRVs)

a different coordination structure in which all agents identified shared one-way routes through the environment and used these routes to circulate between the installation and storage areas, but their routes could be slightly longer than the optimal routes. Finally, the agents with LVs did not generate divisional cooperation nor a norm, but they went back and forth between the installation and storage areas in synchronization and always took the (almost) shortest routes. As a result, they achieved better performance in our particular *pickup and floor laying problem*.

We also conducted some experiments with a smaller number of agents to investigate how the number of agents influences the final coordinated behavior. The performances with LVs, RVs and HRVs were all almost identical to those in the experiments described above. The agents with HRVs, however, did not generate a social norm because the incentive to generate it was weakened in the depopulated environment in which agents rarely collided with other agents. Instead, the agents with HRVs generated behaviors similar to those generated by the agents with the RVs. In addition, the agents with RVs and LVs tended to generate similar coordination structures. We also examined the case in which a large number of agents moved around, but some of them did not sufficiently learned their behaviors probably because of redundancy. Although we think that more detailed experiments and discussions are necessary for this issue, we omit it here because it is beyond the scope of this paper.

To see how behaviors for collision avoidance negatively affect performance, we examined the *near collision count* (NCC), which is the number of states immediately before collisions, meaning states in which two agents could not

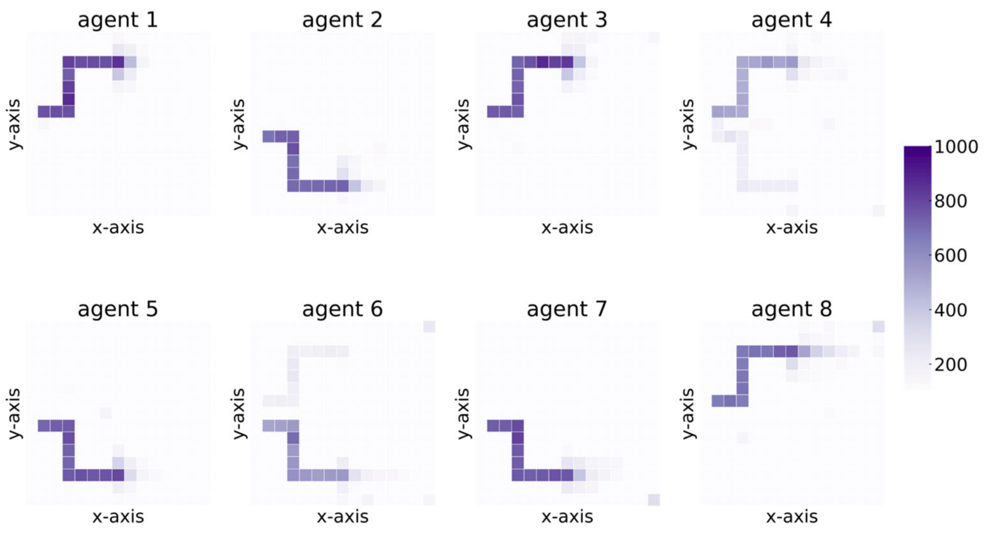

Fig. 17 Locations of each agent's visits without a task (HRVs)

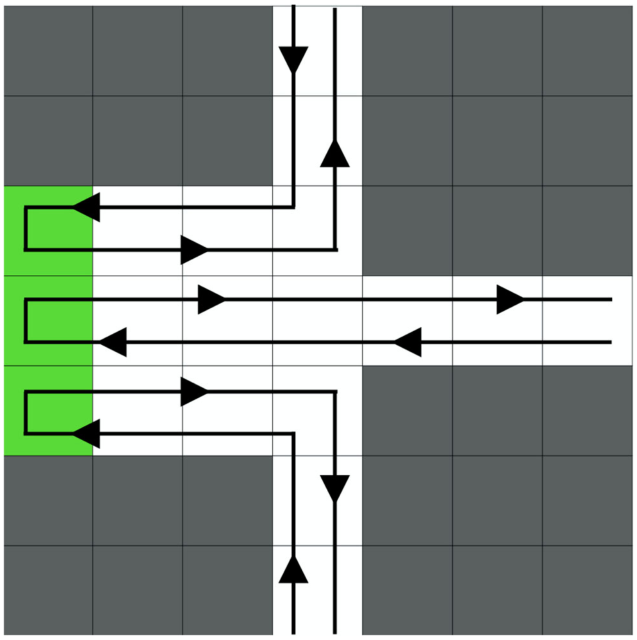
Fig. 18 Agents' movement around storage area (RVs)

move to a neighboring cell at the next tick because they would collide with another agent there. Table 4 lists percentiles of the NCC per epoch for each view representation over 23,000 to 25,000 epochs with $V = 15$. The results indicate that the agents with the HRVs had the lowest NCC, suggesting that movements along circular routes were effective to avoid collisions. Their performance was not as good, however, as that of the LVs because the circular routes forced the agents to take longer paths.

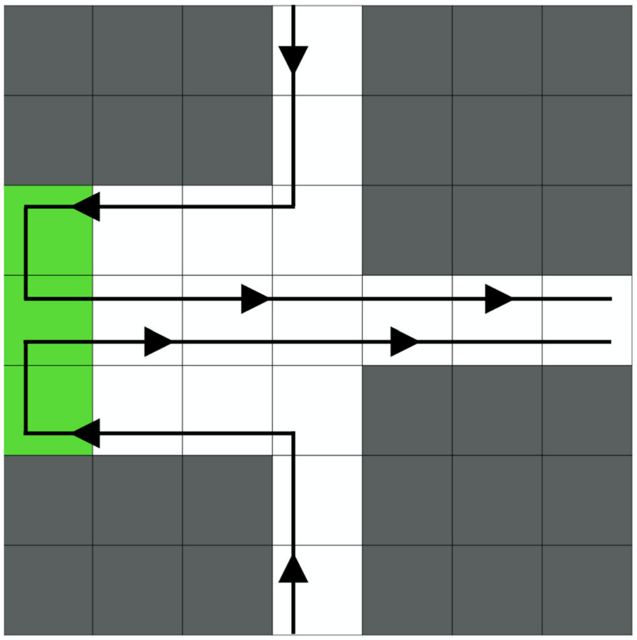
Fig. 19 Agents' movements around storage area (HRVs)

Table 4 Comparison of the near collision count (NCC) per epoch

NCC percentile	RVs	HRVs	LVs
25	303	184	294
50	393	245	342
75	612	408	416

Furthermore, to analyze the NCC in detail, we plotted heatmaps indicating the NCC for each agent at individual cells when $V = 15$, as shown in Figs. 20, 21, and 22. Here, darker orange indicates that the corresponding agent collided with other agents more often. From Fig. 20, we can confirm that the agents with RVs collided with other agents around the storage area and specific narrow passages in which the agents fixed the way for a round trip. Because the agents with RVs always took their own routes, they had fewer collisions, but as eight agents had to share the three passages, the number of collisions often increased there. We thus conclude that the agents with RVs could establish distributed work places to reduce redundant movement in the installation area but could not achieve collision avoidance on the shared passages. Next, as seen from Fig. 21, the agents with HRVs achieved fewer collisions by generating a social norm in which they incorporated a one-way rule. Some agents, however, occasionally broke the social norm (i.e., by going against the one-way rule), thus colliding with other agents around narrow passages. Finally, as shown in Fig. 22, the agents with LVs collided with other agents around the storage area and at the end of the narrow passages. This was because these agents went back and forth between the storage and installation areas in synchronization by using the shortest paths, and they executed the tasks almost concurrently. Thus, agent congestion occurred around the storage area and at the end of the narrow passages in which agents going back to the storage area passed other agents going to the installation area. In our experiments, the coordinated behavior by the agents with HRVs seemed more sophisticated, but the behaviors by the agents with the LVs were the most effective. This was because these agents' routes were usually the shortest, whereas the agents with HRVs were forced to take slightly longer routes because of their regulated behaviors.

We thus clarified the emergence of coordination structure with various type of input information. One contribution to studies of multi-agent systems is to understand possible coordination structures in realistic applications. Designing appropriate coordination structures is a complex task, because we must pursue and analyze the potentials of multiple possible collaborative structures. Our results and methods in this paper can help such a design process. We also think that the results of coordination must

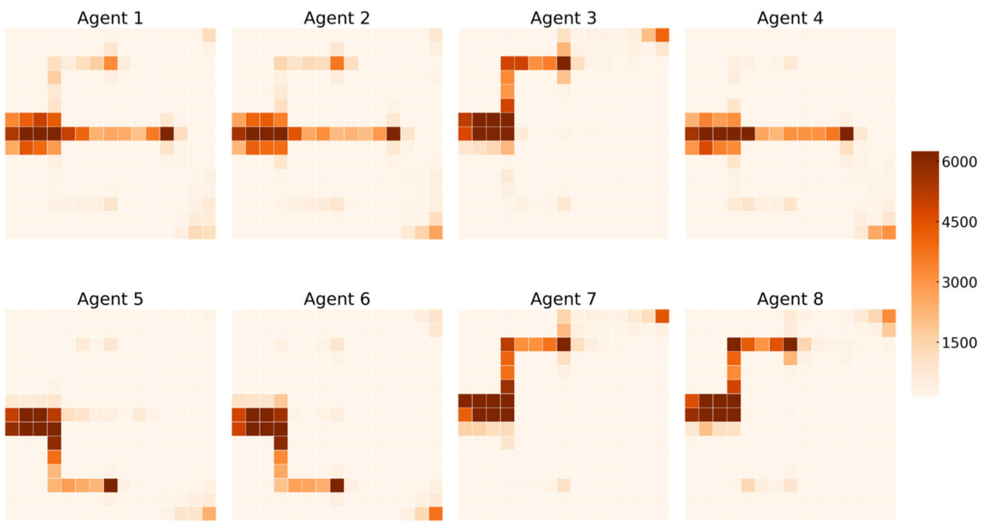

Fig. 20 NCC for each agent's locations (RVs)

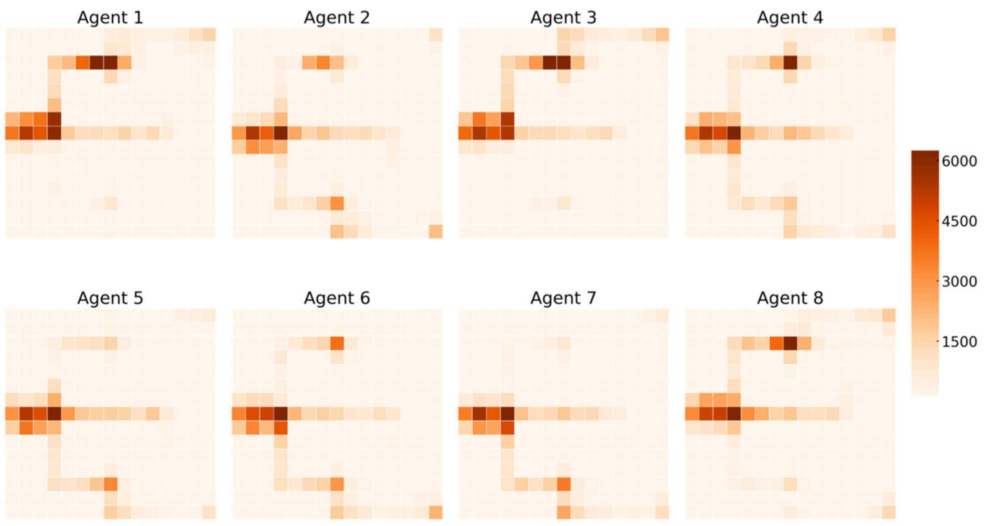

Fig. 21 NCC for each agent's locations (HRVs)

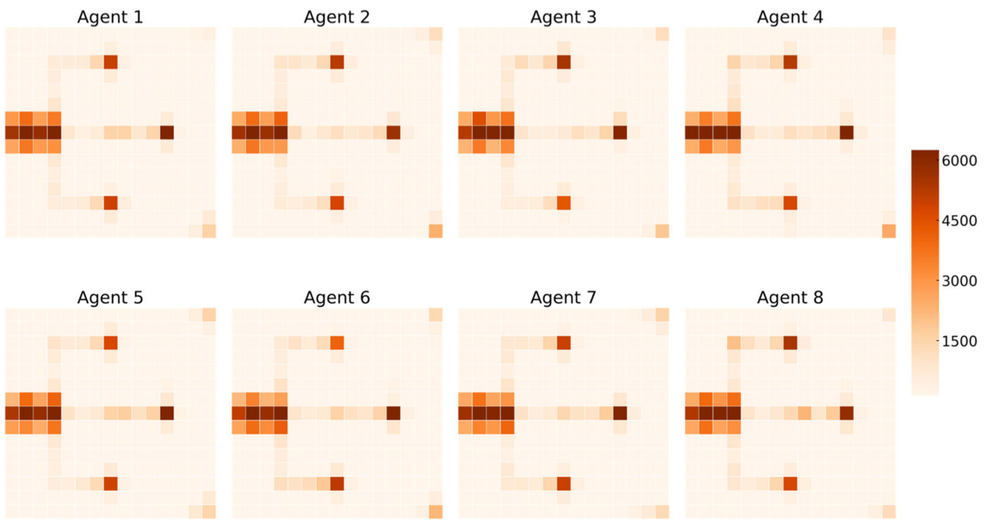

Fig. 22 NCC for each agent's locations (LVs)

be explainable. The behaviors generated by deep neural networks often exhibit good performance, but the reasons are not clear. This kind of obscurity of the results makes it difficult to verify a system's robustness and reliability and to estimate the impact of system changes. Therefore, we have to design and build suitable, explainable coordinated behaviors for real-world applications.

6 Conclusion

pickup and floor laying problem with MADRL, we have presented the emergence of three type of cooperative structures by using different types of inputs (to DQNs) generated from local observations with local beliefs. In our problem formulation, all agents concurrently attempt to learn Q-values, even without explicit communication, to improve their cooperative behavior and earn more rewards by updating their individual DQNs. We showed that various coordination structures and associated behaviors could emerge. When agents had the entire map with their absolute locations, for example, they established segmented work areas for each agent, or they shared a social norm to avoid collisions. In contrast, agents with local views (LVs) as input to the DQNs could not form explicit cooperative structures, but they took appropriate, shorter routes and moved synchronously between the two areas in the problem environment. We found that this behavior also avoided collisions.

In our next study, we plan to verify and improve the scalability, efficiency, and robustness of the agents' learning process when the environment changes over time, because new walls, pillars, floors, etc., are gradually built at a construction site so its structure changes day by day. We will also investigate the coordination structures with other DRL methods, such as recurrent neural networks, the actor-critic method, and extensions of those methods.

Acknowledgements This work was partly supported by JSPS KAKENHI Grant Number 17KT0044.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

References

1. Agmon N, Kraus S, Kaminka GA (2008) Multi-robot perimeter patrol in adversarial settings. In: 2008 IEEE International conference on robotics and automation. IEEE, pp 2339–2345
2. Bloembergen D, Tuyls K, Hennes D, Kaisers M (2015) Evolutionary dynamics of multi-agent learning: a survey. J Artif Intell Res 53:659–697
3. Busoniu L, Babuska R, De Schutter B (2008) A comprehensive survey of multiagent reinforcement learning. IEEE Trans Systems, Man, and Cybernetics Part C 38(2):156–172
4. Diallo EAO, Sugawara T (2018) Learning strategic group formation for coordinated behavior in adversarial multi-agent with double DQN. In: International Conference on Principles and Practice of Multi-Agent Systems. Springer, pp 458–466
5. Foerster J, Nardelli N, Farquhar G, Torr P, Kohli P, Whiteson S et al (2017) Stabilising experience replay for deep multi-agent reinforcement learning. arXiv:1702.08887
6. Giuggioli L, Arye I, Robles AH, Kaminka GA (2018) From ants to birds: A novel bio-inspired approach to online area coverage. In: Distributed autonomous robotic systems. Springer, pp 31–43
7. Gu S, Holly E, Lillicrap T, Levine S (2017) Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp 3389–3396
8. Lample G, Chaplot DS (2017) Playing FPS games with deep reinforcement learning. In: AAAI, pp 2140–2146
9. Levine S, Finn C, Darrell T, Abbeel P (2016) End-to-end training of deep visuomotor policies. J Mach Learning Res 17(1):1334–1373
10. Liu M, Ma H, Li J, Koenig S (2019) Task and path planning for multi-agent pickup and delivery. In: Proceedings of the 18th international conference on autonomous agents and multiagent systems. IFAAMAS, pp 1152–1160
11. Lowe R, Wu Y, Tamar A, Harb J, Abbeel OP, Mordatch I (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in neural information processing systems, pp 6382–6393
12. Luo W, Nam C, Kantor G, Sycara K (2019) Distributed environmental modeling and adaptive sampling for multi-robot sensor coverage. In: Proceedings of the 18th international conference on autonomous agents and multiagent systems. IFAAMAS, pp 1488–1496
13. Mao H, Zhang Z, Xiao Z, Gong Z (2019) Modelling the dynamic joint policy of teammates with attention multi-agent DDPG. In: Proceedings of the 18th international conference on autonomous agents and multiagent systems. IFAAMAS, pp 1108–1116
14. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv:1312.5602
15. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529
16. Palmer G, Tuyls K, Bloembergen D, Savani R (2018) Lenient multi-agent deep reinforcement learning. In: Proceedings of the 17th international conference on autonomous agents and multiagent systems. IFAAMAS, pp 443–451

17. Peng XB, Andrychowicz M, Zaremba W, Abbeel P (2018) Sim-to-real transfer of robotic control with dynamics randomization. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp 1–8
18. Sartoretti G, Wu Y, Paivine W, Kumar TS, Koenig S, Choset H (2019) Distributed reinforcement learning for multi-robot decentralized collective construction. In: Distributed autonomous robotic systems. DARS, Springer, pp 35–49
19. Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. arXiv:1511.05952
20. Shao K, Zhu Y, Zhao D (2018) StarCraft micromanagement with reinforcement learning and curriculum transfer learning. IEEE Transactions on emerging topics in computational intelligence (99):1–12
21. Shoham Y, Powers R, Grenager T (2003) Multi-agent reinforcement learning: a critical survey. Tech. rep. Technical report, Stanford University
22. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st international conference on international conference on machine learning - volume 32: I–387–I–395. ICML'14, JMLR.org
23. Sugiyama A, Sea V, Sugawara T (2018) Emergence of divisional cooperation with negotiation and re-learning and evaluation of flexibility in continuous cooperative patrol problem. Knowledge and Information Systems
24. Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Lanctot M, Sonnerat N, Leibo JZ, Tuyls K, et al. (2017) Value-decomposition networks for cooperative multi-agent learning. arXiv:1706.05296
25. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4(2):26–31
26. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double Q-learning
27. Watkins CJ, Dayan P (1992) Q-learning. Machine learning 8(3-4):279–292
28. Xie M, Tachibana A (2007) Cooperative behavior acquisition for multi-agent systems by Q-learning. In: Foundations of computational intelligence, 2007. FOCI 2007. IEEE Symposium on. pp 424–428. IEEE
29. Miyashita Y, Sugawara T (2019) Coordination in collaborative work by deep reinforcement learning with various state descriptions. In: International conference on principles and practice of multi-agent systems, pp 550–558

Yuki Miyashita received the B.S. degree in Computer Science and Communications Engineering from Waseda University, Tokyo, Japan in 2014, and the M.S degree in Computer Science and Communications Engineering from Waseda University, Tokyo, Japan in 2016 . He is currently pursuing the Ph.D. degree in Science and Communications Engineering at Waseda University, Tokyo, Japan. His current research interests include multi-agent system control, deep learning.

Toshiharu Sugawara is a professor of Department of Computer Science and Engineering, Waseda University, Japan, since April, 2007. He received his B.S. and M.S. degrees in Mathematics, 1980 and 1982, respectively, and a Ph.D, 1992, from Waseda University. In 1982, he joined Basic Research Laboratories, Nippon Telegraph and Telephone Corporation. From 1992 to 1993, he was a visiting researcher in Department of Computer Science, University of Massachusetts at Amherst, USA. His research interests include multi-agent systems, distributed artificial intelligence, machine learning, internetworking, and network management. He is a member of IEEE, ACM, AAAI, Internet Society, The Institute of Electronics, Information and Communication Engineers (IEICE), Information Processing Society of Japan(IPSJ), Japan Society for Software Science and Technology (JSSST), and Japanese Society of Artificial Intelligence (JSAI).