Application of deep reinforcement learning to networked control systems with uncertain network delays

Junya Ikemoto\textsuperscript{1a)} and Toshimitsu Ushio\textsuperscript{1b)}

\textsuperscript{1} Graduate School of Engineering Science, Osaka University 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

\textsuperscript{a)} ikemoto@hopf.sys.es.osaka-u.ac.jp
\textsuperscript{b)} ushio@sys.es.osaka-u.ac.jp

Received January 30, 2020; Revised May 15, 2020; Published October 1, 2020

Abstract: Networked control systems (NCSs) have been attracted much attention thanks to the development of network technology. There are network delays caused by data transmissions in NCSs. These network delays may degrade control performances. In general, the network delays may fluctuate randomly and it is difficult to identify their probability distributions. Moreover, it is difficult to precisely identify models of plants. Thus, we propose a design method of networked controllers using deep reinforcement learning (DRL) taking network delays into consideration. Additionally, we consider the case where sensors cannot observe all state variables of plants. We introduce an extended state and propose a DRL-based controller design method.

Key Words: deep reinforcement learning, policy gradient, network control system, uncertain network delay

1. Introduction

Reinforcement learning (RL) \cite{1} has attracted much attention in various fields. In the control field, many applications of RL have been studied because it is useful to design controllers for plants whose mathematical models are unknown. In \cite{2}, Ichikawa and Ushio applied RL to control of an auto-driven vehicle following after a platoon of human-driven vehicles. In \cite{3}, Kamio et al. proposed multi-agent reinforcement learning system (MARLS) to search ships’ courses as a useful tool. Moreover, a method to modify near-miss courses has proposed as an expansion of MARLS in \cite{4}. In \cite{5,6}, RL-based methods for design of optimal controllers were proposed for an uncertain Markov decision process (MDP) with a control cost and a linear temporal logic (LTL). In \cite{7}, RL has been also applied to control of chaotic systems. However, classical RL algorithms cannot solve the complicated control problems since they are table-based algorithms.

Recently, RL with deep neural networks (DNNs), which is called deep reinforcement learning (DRL), has actively researched. The deep Q-network (DQN) algorithm has achieved impressive results in many Atari 2600 video games using pixels as inputs to the DQN \cite{8}. In this algorithm, a DNN is used in
order to estimate the optimal Q-function with two major techniques: the *experience replay* and the *fitted target Q-function*. It is an extension of *neural fitted Q-learning* [9]. The agent with a DNN can learn successful policies directly from high-dimensional sensory inputs. Thereafter, the DQN has been improved and achieved better performance for playing video games [10, 11]. Furthermore, the DQN is useful for not only playing video games but also controlling complicated systems. In [12], Zhang et al. applied the DQN to control of a three-joint robot manipulator using raw-pixel images observed by an external visual sensor. In [13], Demirel et al. proposed a control-aware scheduling algorithm using the DQN in the case where the size of the communication network is smaller than the size of a system consisting of multiple independent subsystems. Moreover, in [14], Masuda and Ushio applied the hierarchized DQN [15] to a path planning for a parking problem of a 4-wheeled vehicle. However, the DQN cannot deal with a continuous action space since it is assumed that the numbers of the available action is finite. In general, a controller design method using discretization of the continuous action space degrades control performances. Thus, Lillicrap et al. proposed the *deep deterministic policy gradient* (DDPG) algorithm for the continuous action space [16]. It is shown that the DDPG algorithm can solve more than 20 simulated physics tasks. As its applications to control problems, for example, Baumann et al. proposed a method to design an event-triggered controller using the DDPG algorithm in [17] and Duan et al. proposed the method to master the power grid voltage control problem using not only the DQN algorithm but also the DDPG algorithm in [18].

In this paper, we apply DRL to *networked control systems* (NCSs), which consist of a plant, a sensor, a controller, an actuator, and a network. The controller computes control inputs from data sent by the remote sensor, and sends them to the remote actuator. NCSs have been also attracted much attention thanks to development of network technology. In NCSs, there are network delays due to data transmissions. It is necessary to design the controller taking these network delays into consideration because the delays degrades control performances and may make the plant unstable in the worst case. We consider the following NCSs.

- The plant is a nonlinear system whose mathematical model is unknown.
- Network delays fluctuate randomly due to the network routing, where their maximum values are known beforehand.
- The sensor cannot observe all state variables of the plant.

Under these assumptions, we propose a DRL-based design method for a networked controller. The paper is organized as follows. In Section 2, we review standard RL and the DDPG algorithm. In Section 3, we formulate the network control problem considered in this paper. In Section 4, we propose a DRL-based design method of a networked controller. In Section 5, by numerical simulations, we apply the proposed method based on the DDPG algorithm to stabilization of a Lorenz equation at one of equilibrium points. In Section 6, we conclude this paper.

### 1.1 Related work and contribution

In NCSs, it is important to take network delays into consideration. Many design methods of the controller considering network delays have been researched [19–21]. These approaches consist in predicting the state at a time when the control input is received by the actuator. However, in these methods, we need the knowledge of the system. If there is uncertainty about the model of the plant or the network characteristic, we must design the networked controller conservatively. As a way to solve this problem, RL-based control methods are useful. In [22, 23], Fujita and Ushio proposed the RL-based optimal networked control method for a linear system whose parameters are unknown with uncertain fixed network delays. However, we cannot apply these methods to control of nonlinear systems. Thus, in this paper, we use the DRL algorithm in order to design the networked controller for nonlinear systems. Moreover, we assume that network delays fluctuate randomly due to network routing, where their maximum values are known beforehand. Recently, DRL-based networked control methods have been proposed in [13, 17]. However, in these researches, it is not assumed that there are network delays.
In the case where there are network delays, it is necessary for the controller to receive sufficient information about these network delays to learn its control policy. In [24], we proposed the DRL-based control method for the nonlinear system with control input delays. In this method, we use an extended state consisting of the current state and some past determined control inputs for the controller to learn its control policy. Additionally, we considered not only delays due to transmission of control inputs from the controller to the actuator but also delays due to transmission of control outputs from the sensor to the controller. In [25], we showed that the method proposed in [24] is useful in such a case. Furthermore, in NCSs, the sensor may not observe all state variables of the plant, which is called a partial observation and causes degradation of learning performances. For the partial observation, RL algorithms using recurrent neural networks have proposed in [26, 27]. On the other hand, in order to deal with both network delays and the partial observation, we proposed the method using an extended state consisting of not only the current observed output and some past determined control inputs but also some past observed outputs in [28]. In [24, 25, 28], we leveraged the continuous deep Q-learning (CDQL) algorithm [29]. The CDQL algorithm is a simplified method of the DDPG algorithm with a normalized advantage function (NAF) that is a quadratic function approximator of the advantage term of the optimal Q-function. However, the controller may not learn the desirable control policy for a complicated system due to the quadratic approximation of the advantage term. Hence, in this paper, we use the DDPG algorithm to verify that the proposed method is useful not only for the CDQL algorithm but also for the DDPG algorithm that is the standard DRL algorithm with the experience replay and the fixed target Q-network for a continuous action space.

2. Preliminaries

This section reviews standard RL and the DDPG algorithm that is one of DRL algorithms.

2.1 Reinforcement learning (RL)

RL is one of the machine learning methods [1]. The learner is called an agent in RL. The agent interacts with everything outside the agent that is called an environment \( E \) and learns its policy as shown in Fig. 1. At each discrete time \( k \in \{0, 1, \ldots\} \), the agent receives a state of the environment \( s_k \in S \) and determines an action \( a_k \in A \), where \( S \) and \( A \) denote the sets of states and actions, respectively. The agent determines the action \( a_k \) by its policy \( \mu : S \rightarrow A \). Then, at the next time, the state of the environment is transited from \( s_k \) to \( s_{k+1} \). The transition is stochastically caused by a transition dynamics \( p(s_{k+1}|s_k, a_k) \), where \( p(s_{k+1}|s_k, a_k) \) is the conditional probability of the next state \( s_{k+1} \) given by the current state \( s_k \) and the action \( a_k \). The agent receives the next state \( s_{k+1} \) and a reward \( r_k \in \mathbb{R} \) that is given by the following reward function \( R : S \times A \times S \rightarrow \mathbb{R} \).

![Fig. 1. Illustration of interactions between an environment and an agent.](image-url)
Then, a tuple \((s_k, a_k, s_{k+1}, r_k)\) is called an experience. The agent updates its control policy \(\mu\) based on experiences. The goal of RL is for the agent to learn the policy that maximizes the discounted sum of rewards \(G_k = \sum_{i=k}^{\infty} \gamma^{i-k} r_i\), where \(\gamma \in [0,1)\) is a discount factor to prevent the divergence of \(G_k\). On the other hand, it is assumed that transitions of states are stochastic. Thus, we define the following conditional expectation of \(G_k\) as value functions.

\[
V^\mu(s) = \mathbb{E}_{s_{i+1} \sim \rho} [G_k | s_k = s],
\]

\[
Q^\mu(s, a) = \mathbb{E}_{s_{i+1} \sim \rho} [G_k | s_k = s, a_k = a].
\]

Equations (2) and (3) are called a state value function and a state-action value function (Q-function), respectively. Moreover, the following functions are defined as an optimal state value function and an optimal Q-function, respectively.

\[
\forall s \in S, \quad V^*(s) = \max_{\mu} V^\mu(s),
\]

\[
\forall s \in S, \forall a \in A, \quad Q^*(s, a) = \max_{\mu} Q^\mu(s, a),
\]

where the policy \(\mu^*\) is called an optimal policy. In SARSA and Q-learning [1], the agent learns the value function or the optimal value function through interactions with the environment. However, these algorithms are table-based algorithms assuming that both the state set and the action set are finite sets. In the case where states and actions are continuous values, a \(\theta\) gradient method is useful [30]. In this method, we consider a deterministic policy parameterized by a parameter vector \(\theta^\mu\). The parameterized policy is denoted by \(\mu(\cdot; \theta^\mu)\). We define a performance object by \(J(\theta^\mu) = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_i]\). The idea of this method is to adjust the parameter vector \(\theta^\mu\) in the direction of the performance gradient \(\nabla_{\theta^\mu} J(\theta^\mu)\) computed by the following equation [30]:

\[
\nabla_{\theta^\mu} J(\theta^\mu) = \mathbb{E}_{s \sim \rho^\theta} [\nabla_{\theta^\mu} \mu(s; \theta^\mu) \nabla_a Q^\mu(s, a)|_{a = \mu(s; \theta^\mu)}],
\]

where \(\rho^\mu\) is a discounted state distribution defined by

\[
\rho^\mu(s') := \int_s \sum_{i=0}^{\infty} \gamma^i p_0(s)p(s \rightarrow s', i, \mu)ds,
\]

where \(p_0\) is an initial state distribution and \(p(s \rightarrow s', i, \mu)\) is the density at the state \(s'\) after transiting for \(i\) discrete time steps from the state \(s\) by the policy \(\mu\).

Moreover, when we consider off-policy methods where an agent learns a deterministic policy \(\mu\) from trajectories generated by an arbitrary stochastic policy \(\beta\), we use the following approximated performance gradient [30, 31].

\[
\nabla_{\theta^\mu} J(\theta^\mu) \approx \mathbb{E}_{s \sim \rho^\beta} [\nabla_{\theta^\mu} \mu(s; \theta^\mu) \nabla_a Q^\mu(s, a)|_{a = \mu(s; \theta^\mu)}].
\]

Then, we must estimate \(Q^\mu(s, a)\) in Eqs. (6) or (8) to compute \(\nabla_{\theta^\mu} J(\theta^\mu)\). The actor-critic [1] is often used to estimate \(Q^\mu(s, a)\). In this approach, there are two components. One is an actor that determines an action by its parameter vector \(\theta^{\pi}\). The other is a critic that updates an estimation of the Q-function using a set of experiences. Actually, we also use a parametrized function by a parameter vector \(\theta^Q\) in the critic.

### 2.2 Deep deterministic policy gradient (DDPG)

The DQN attracted much attention for learning human level performances on many Atari video games using pixels as inputs to the DQN [8]. However, it is not possible to apply the DQN to problems where the action set is infinite, that is, the action is a continuous variable. Therefore, Lillicrap et al. proposed a DRL algorithm based on a deterministic policy gradient method, called the DDPG algorithm [16], where two deep neural networks are used as function approximators of the actor and the critic. These
deep neural networks are called an *actor network* and a *critic network*, respectively. The parameter vectors of the actor network and the critic network are denoted by $\Theta^{\mu}$ and $\Theta^{Q}$, respectively.

We use the following *TD-error* to adjust the parameter vector $\Theta^{Q}$.

$$L(\Theta^{Q}) = (Q(s_k, a_k; \Theta^{Q}) - t_k)^2,$$

where

$$t_k = r_k + \gamma Q(s_{k+1}, \mu(s_{k+1}; \Theta^{\mu}); \Theta^{Q}).$$

However, the learning method with DNNs tends to be unstable. Hence, we use two major techniques of the DQN algorithm: the experience replay and the fixed target Q-network. In the experience replay, we store the experiences $(s_k, a_k, s_{k+1}, r_k)$ to a *replay buffer* at each discrete time $k + 1$. When the agent updates the parameter vector $\Theta^{Q}$, it selects $N$ experiences randomly and updates $\Theta^{Q}$ by minimizing a TD-error based on these experiences. This technique leads reductions of correlations between experiences. In the fixed target Q-network, we use two deep neural networks that are called a *target actor network* and a *target critic network* for calculating $t_k$. The parameter vectors of these target networks are denoted by $\Theta^{\mu}_{-}$ and $\Theta^{Q}_{-}$, respectively. This technique can make the learning more stable compared with the standard learning by Eq. (10). We use the following *soft update* to adjust parameter vectors $\Theta^{\mu}_{-}$ and $\Theta^{Q}_{-}$.

$$\Theta^{Q}_{-} \leftarrow \eta \Theta^{Q} + (1 - \eta)\Theta^{Q}_{-},$$
$$\Theta^{\mu}_{-} \leftarrow \eta \Theta^{\mu} + (1 - \eta)\Theta^{\mu}_{-},$$

where $\eta$ is a sufficiently small positive constant. Thus, in this paper, we calculate $t_k$ by

$$t_k = r_k + \gamma Q(s_{k+1}, \mu(s_{k+1}; \Theta^{\mu}_{-}); \Theta^{Q}_{-}).$$

The parameter vector $\Theta^{\mu}$ is updated by the following approximated performance gradient.

$$\nabla_{\Theta^{\mu}} J(\Theta^{\mu}) \approx \mathbb{E}_{s \sim \rho^\beta} [\nabla_{\Theta^{\mu}} \mu(s; \Theta^{\mu}) \nabla_{a} Q(s, a; \Theta^{Q}) | a = \mu(s; \Theta^{\mu})].$$

We construct an exploration policy $\mu'$ by adding a noise $\epsilon$ sampled from a noise process $\mathcal{N}$ to our actor policy as follows:

$$\mu'(s) = \mu(s; \Theta^{\mu}) + \epsilon.$$

Finally, the DDPG algorithm is shown in Algorithm 1.

### 3. Network control problem

We consider the networked control of the following nonlinear plant as shown in Fig. 2.

$$\dot{x}(t) = f(x(t), u(t)),$$
$$y_k = h(x(k\Delta)),$$

where

- $x(t) \in \mathcal{X}$ is the state of the plant at time $t \in [0, \infty)$,
- $u(t) \in \mathcal{U}$ is the control input inputted to the plant at time $t \in [0, \infty)$,
- $\Delta > 0$ is the sampling period of the sensor,
- $y_k \in \mathcal{Y}$ is the $k$-th output observed by the sensor at time $k\Delta$ ($k = 0, 1, ...$),
- $f : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{X}$ describes the mathematical model of the plant, and
- $h : \mathcal{X} \rightarrow \mathcal{Y}$ is the output function that is characterized by the sensor.
Algorithm 1 Deep deterministic policy gradient algorithm

1: Randomly initialize the parameter vectors $\Theta^Q, \Theta^\mu$.
2: Initialize parameter vectors of target networks $\Theta^Q \leftarrow \Theta^Q, \Theta^\mu \leftarrow \Theta^\mu$.
3: Initialize a replay buffer $B$.
4: for episode = 1, ..., $M$ do
5: Initialize a random process $N$ for action explorations.
6: Receive initial observation state $s_0$.
7: for $k = 0, ..., K$ do
8: Determine an action $a_k = \mu(s_k; \Theta^\mu) + \epsilon_k$ according to the current policy $\mu$ and the exploration noise $\epsilon_k \sim N$.
9: Execute the action $a_k$.
10: Observe the next state $s_{k+1}$ and receive the reward $r_k$.
11: Store the experience $(s_k, a_k, s_{k+1}, r_k)$ in $B$.
12: for iteration = 1, ..., $I$ do
13: Select $N$ experiences $(s^{(n)}, a^{(n)}, s'^{(n)}, r^{(n)}) (n = 1, ..., N)$ randomly.
14: Set $t^{(n)} = r^{(n)} + \gamma Q(s'^{(n)}, \mu(s'^{(n)}; \Theta^\mu); \Theta^Q)$.
15: Update $\Theta^Q$ by minimizing the loss: $L = \frac{1}{N} \sum_n (t^{(n)} - Q(s^{(n)}, a^{(n)}; \Theta^Q))^2$
16: Update $\Theta^\mu$ using the policy gradient:
17: \[ \nabla_{\Theta^\mu} J(\Theta^\mu) \approx \frac{1}{N} \sum_n \nabla_{\Theta^\mu} \mu(s^{(n)}; \Theta^\mu) \nabla_a Q(s^{(n)}, a^{(n)}; \Theta^Q)|_{a=\mu(s^{(n)}; \Theta^\mu)} \]
18: Update $\Theta^Q$ and $\Theta^\mu$:
19: \[ \Theta^Q \leftarrow \eta \Theta^Q + (1 - \eta) \Theta^Q, \]
20: \[ \Theta^\mu \leftarrow \eta \Theta^\mu + (1 - \eta) \Theta^\mu. \]
21: end for
22: end for
23: end for

\[ X \subseteq \mathbb{R}^n, U \subseteq \mathbb{R}^m, \text{and } Y \subseteq \mathbb{R}^p \] are the plant’s state space, the control input space, and the control output space, respectively. In this paper, we assume that models $f$ and $h$ are unknown, while $n$, $m$, and $p$ are known. Moreover, we assume that there are two types of network delays in the NCS. One is caused by transmissions of observed outputs from the sensor to the controller. The other is caused by...
transmissions of determined control inputs from the controller to the actuator. These $k$-th network delays are denoted by $\tau_{sc,k}$ and $\tau_{cp,k}$, respectively. These network delays randomly fluctuate, where these delays are bounded by maximum delays $\tau_{sc,max} = a\Delta$ and $\tau_{cp,max} = b\Delta$, respectively. We assume that $a$, $b \in \mathbb{N}$ are known and define $\tau := a + b$. We also assume that the packet loss does not occur in the networks and all data are received in the same order as their sending order.

We design a digital networked controller for this system. The $k$-th control input determined by the controller is denoted by $u_k$. The control input $u_k$ is held until the next control input $u_{k+1}$ is received by the actuator, that is,

$$u(t) = u_k \quad (k\Delta + \tau_{sc,k} + \tau_{cp,k} \leq t < (k+1)\Delta + \tau_{sc,k+1} + \tau_{cp,k+1}).$$

(18)

For the NCS, we design the networked controller using the DDPG algorithm.

4. Design of networked controller using DDPG

4.1 State based learning

In this subsection, we consider the case where the sensor can observe all state variables of the plant, that is, $n = p$ and $h(x) = x$ for any $x \in \mathcal{X}$. In an application of RL to control problems, we often regard the unknown plant and the controller as the environment $E$ and the agent, respectively. However, there are uncertain network delays in the communication networks between the plant and the controller. If we ignore network delays and apply RL to design of the controller, the controller cannot learn its control policy. In the RL framework, the state of the environment $E$ must include all information outside the agent. Therefore, we must appropriately determine the environment $E$ and the agent for the NCS.

We consider the worst case, that is, for each $k$, $\tau_{sc,k} = \tau_{sc,max}$, $\tau_{cp,k} = \tau_{cp,max}$. The delays in data transmissions are illustrated in Fig. 3. The sensor observes the state $x_k = x(k\Delta)$ at time $t = k\Delta$. The $k$-th observed state $x_k$ is sent to the controller through the network. The controller receives $x_k$ and determines the $k$-th control input $u_k$ at time $t = (k+a)\Delta$. The control input $u_k$ is sent to the actuator. The actuator receives the $k$-th control input $u_k$ and updates the control input to the plant $u(t) = u_k$ at time $t = (k+\tau)\Delta$. Then, the plant’s state is $x((k+\tau)\Delta)$ that is the future state at

![Diagram of networked control system](image)

**Fig. 3.** Illustration of the delays in data transmissions for the worst case. In this figure, we assume that $\tau_{sc,max} = 2\Delta$ and $\tau_{cp,max} = 3\Delta$. It shows timings of observing states, determining control inputs, and updating control inputs to the plant.
In this subsection, we assume that the sensor cannot observe all state variables of the plant (4.2 Output based learning). This case is called a partial observation. Then, we cannot directly use the plant's state determined by the controller as the action determined by the agent.

On the other hand, if we could identify a mathematical model of the plant \( f \), we would predict the future state \( x((k+\tau)\Delta) \) and determine the \( k \)-th control input \( u_k \) based on available information.

We regard the extended state \( z_k \) as the state of the environment \( E \). We also regard the control input determined by the controller as the action determined by the agent.

In general, network delays do not always take the maximum values \( \tau_{\text{sc}} \) and \( \tau_{\text{cp}} \). However, the controller learns the optimal control policy based on sufficient information \( z_k \) through interactions with the plant.

### 4.2 Output based learning

In this subsection, we assume that the sensor cannot observe all state variables of the plant \( (p < n) \). This case is called a partial observation. Then, we cannot directly use the plant’s state \( x(k\Delta) \). We must estimate the full plant’s state based on available information. If we could identify the models \( f \) and \( h \), we would predict the full state \( x_k \) based on the past control inputs and outputs. Particularly, in the case where the system is a discrete-time linear system, we can estimate the state \( x_k \) based on the current output \( y_k \), past outputs \( y_{k-1}, y_{k-2}, \ldots, y_{k-\varsigma} \), and past control inputs \( u_{k-1}, u_{k-2}, \ldots, u_{k-\varsigma} \), where the length \( \varsigma \) is larger than the observability index \( q \) of the linear system, that is, \( n \leq pq \). Based on this fact, Aangenent et al. proposed a data-based optimal control method [32], Lewis et al. proposed a RL-based method for partially observable dynamic process [33], and Fujita and Ushio applied the Lewis’ method to the design method of an optimal networked controller [23]. The previous studies dealt with linear systems and the estimation of the current state of the linear systems can be done by solving a linear equation consisting of the past control inputs and outputs whose number is larger than the observability index. In nonlinear systems, however, it is difficult to derive a nonlinear equation whose solution is the current state. Thus, we select \( \varsigma \) as a meta-parameter beforehand and give the controller with DNNs past control inputs \( u_{k-1}, u_{k-2}, \ldots, u_{k-\varsigma} \) and control outputs \( y_{k-1}, y_{k-2}, \ldots, y_{k-\varsigma} \) for its learning. In general, if we set the meta-parameter \( \varsigma \) to a small value, the performance of the learned control policy may be limited. On the other hand, if we set the meta-parameter \( \varsigma \) to a large value, the controller may fail to learn the optimal control policy due to the curse of dimensionality. Additionally, we must consider existence of network delays. In the same way as Section 4.1, we consider the worst case, that is, all network delays are \( \tau_{\text{sc}} \) and \( \tau_{\text{cp}} \) as in Fig. 4. When we use past control outputs \( y_{k-1}, y_{k-2}, \ldots, y_{k-\varsigma} \), we must consider past control inputs inputted to the plant for \( (k-\varsigma)\Delta \leq t < (k+\tau)\Delta \), that is, \( u_{k-1}, u_{k-2}, \ldots, u_{k-(\tau+\varsigma)} \). Thus, we define the following extended state \( w_k \in \mathbb{R}^{(\varsigma+1)p+\tau(q+\varsigma)m} \).
∀k \ \tau_{sc,k} = 2\Delta, \ \tau_{cp,k} = 3\Delta

We regard the extended state \(w_k\) as the state of the environment \(E\). We also regard the control input determined by the controller as the action determined by the agent. Moreover, the network delays do not always take the maximum values \(\tau_{sc}^{\text{max}}\) and \(\tau_{cp}^{\text{max}}\). However, the controller learns the optimal control policy based on sufficient information \(w_k\) through interaction with the plant.

We explain our proposed learning algorithm to design the networked controller. The algorithm is shown in Algorithm 2. Let \(S = Y^{k+1} \times U^{\tau+\varsigma}\) and \(A = U\). As shown in Fig. 5, we regard the controller as the agent, and the combination of the past control input list and the past control output list as the environment, respectively. At time \(t = k\Delta + \tau_{sc,k}\), the controller receives the output \(y_k\), generates the extended state \(w_k\), and sends the tuple \((w_{k-1}, u_{k-1}, w_k)\) to the experience sampler as shown between Lines 11 and 16 of Algorithm 2. The experience sampler computes the reward \(r_{k-1} = R(w_{k-1}, u_{k-1}, w_k)\), makes the experience \((w_{k-1}, u_{k-1}, w_k, r_{k-1})\), and stores the experience to the replay buffer \(B\). The reward function \(R\) is defined depending on our control objective. In this paper, it is assumed that the reward function \(R\) is given beforehand. Moreover, the controller determines the control input by

\[
\epsilon_k \sim N
\]

where \(\epsilon_k \sim N\) is the exploration noise as shown in Lines 17 and 18 of Algorithm 2. The controller holds \(w_k\) and \(u_k\) to make the next experience \((w_k, u_k, w_{k+1}, r_k)\) until it receives the next state \(w_{k+1}\).
The controller updates parameter vectors $\Theta^\mu$, $\Theta^{\mu-1}$, $\Theta^Q$, $\Theta^{Q-1}$ asynchronously with sampling experiences. At each time $t = k_p \Delta$ ($k_p \in \mathbb{N}$), the controller updates the parameter vectors as shown between Line 20 and 26 of Algorithm 2.

5. Example

We consider the following Lorenz equation as a plant.

$$\frac{d}{dt} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} 10(x_2(t) - x_1(t)) \\ -x_1(t)x_3(t) + 28x_1(t) - x_2(t) \\ x_1(t)x_2(t) - 8/3x_3(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 2 \\ 0 \\ 3 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}. \quad (23)$$

The state of the plant is denoted by $x(t) = [x_1(t), x_2(t), x_3(t)]^T$ and the control input is denoted by $u(t) = [u_1(t), u_2(t)]^T$. The uncontrolled trajectory is shown in Fig. 6. The goal is the stabilization of the Lorenz equation at one of equilibrium points which we cannot identify. In the following simulations, we assume that the reward functions are given beforehand to meet the goal. In this section, we assume that both $\tau_{sc,k}$ and $\tau_{cp,k}$ are generated by the uniform distribution $U(3\Delta, 5\Delta)$, where we cannot identify this distribution beforehand while these maximum network delays $\tau_{sc,\max} = 5\Delta$ and $\tau_{cp,\max} = 5\Delta$ are known. Thus, in all simulations, we set $\tau = 10$.

We use a critic network with two hidden layers, where all hidden layers have 128 units and all layers are fully connected layers, and an actor network with two hidden layers, where all hidden layers have 128 units and all layers are fully connected layers. The activation functions are ReLU except for the output layers. In regards to the activation functions of the output layers, we use a 7 times weighted hyperbolic tangent function for the actor network and a linear function for the critic network. The two DNNs are shown in Fig. 7. The size of the replay buffer $B$ is $1.0 \times 10^6$ and the minibatch size is $N = 128$. The parameter vectors of the actor network and the critic network are updated 10 times per $4\Delta$ ($I = 10$, $k_p = 4$) by ADAM [34], where learning step sizes are set to $1.0 \times 10^{-5}$ for the actor network and $1.0 \times 10^{-4}$ for the critic network. The update rate of target networks is $\eta = 0.001$ and the discount factor for the Q-function is $\gamma = 0.99$. The final layer weights and biases of both the
Algorithm 2 DDPG-based learning of the networked controller under a partial observation with the extended state consisting past control inputs and outputs.

1: Select the length of the past output sequence $\varsigma$.
2: Randomly initialize the parameter vectors $\Theta^Q, \Theta^\mu$.
3: Initialize parameter vectors of target networks $\Theta^Q_{\text{−}}, \Theta^\mu_{\text{−}} \leftarrow \Theta^Q, \Theta^\mu \leftarrow \Theta^\mu$.
4: Initialize the replay memory $B$.
5: for episode = 1, ..., $M$ do
6: Receive the initial observed output $y_0$.
7: Add the observed output $y_0$ to the past control output list.
8: Generate the initial extended state $w_0$, where $u_i = 0$ ($i < 0$), $y_i = 0$ ($i < 0$).
9: Initialize a random process $N$ for action exploration.
10: for $k = 0, ..., K$ do
11: if $k > 0$ then
12: Receive the $k$-th output $y_k$.
13: Add the observed output $y_k$ to the past control output list.
14: Generate the extended state $w_k$ with past control inputs and outputs, where $u_i = 0(i < 0)$.
15: Send the tuple $(w_{k-1}, u_{k-1}, w_k)$ to the experience sampler.
16: end if
17: Determine the control input $u_k = \mu(w_k; \Theta^\mu) + \epsilon_k$ and send the control input to the actuator.
18: Add the control input $u_k$ to the past control input list.
19: if $k \% k_p = 0$ then
20: for iteration = 1, ..., $I$ do
21: Select $N$ experiences $(w^{(n)}, u^{(n)}, w^{(n)}, r^{(n)})$ ($n = 1, ..., N$) randomly.
22: Set $t^{(n)} = r^{(n)} + \gamma Q(w^{(n)}, \mu(w^{(n)}; \Theta^\mu); \Theta^Q_{\text{−}})$.
23: Update $\Theta^Q$ by minimizing the loss: $L = \frac{1}{N} \sum_n (t^{(n)} - Q(w^{(n)}, u^{(n)}; \Theta^Q))^2$
24: Update $\Theta^\mu$ using the policy gradient:
25: \[
\nabla_{\Theta^\mu} J(\Theta^\mu) \approx \frac{1}{N} \sum_n \nabla_{\Theta^\mu} \mu(w^{(n)}; \Theta^\mu) \nabla_u Q(w^{(n)}, u^{(n)}; \Theta^Q)|_{u=\mu(w; \Theta^\mu)}.
\]
26: Update $\Theta^Q$ and $\Theta^\mu$:
27: $\Theta^Q_{\text{−}} \leftarrow \eta \Theta^Q + (1 - \eta) \Theta^Q_{\text{−}}$.
28: $\Theta^\mu_{\text{−}} \leftarrow \eta \Theta^\mu + (1 - \eta) \Theta^\mu_{\text{−}}$.
29: end for
30: end if
31: end for

Fig. 6. Illustration of the trajectory of the uncontrolled Lorenz equation. Three equilibrium points are shown by red cross marks $[0, 0, 0]^T$, $[6\sqrt{2}, 6\sqrt{2}, 27]^T$, $[-6\sqrt{2}, -6\sqrt{2}, 27]^T$. 

490
actor and critic are initialized from a uniform distribution $[-3.0 \times 10^{-3}, 3.0 \times 10^{-3}]$. We explain the exploration noise process $N$ in Appendix A. The initial state is randomly selected for each episode, where $-15 \leq x_1(0) \leq 15$, $-15 \leq x_2(0) \leq 15$, and $10 \leq x_3(0) \leq 50$.

5.1 State-based learning

In this subsection, we assume that the sensor can observe all state variables of the plant, where the $k$-th observed state is $x_k = [x_1(k\Delta), x_2(k\Delta), 0.08x_3(k\Delta)]^T$. We set the sampling time $\Delta = 2^{-4}$. We assume that the terminal of a learning episode is at time $t = 25$.

First, we consider the design of a networked controller using the current observed state $x_k$ only. It is assumed that the following reward function $R_0: \mathcal{X} \times \mathcal{U} \times \mathcal{X} \rightarrow \mathbb{R}$ is given in order to meet the goal.

$$R_0(x_k, u_k, x_{k+1}) = - \begin{bmatrix} 0.8 & 0.8 & 8.0 \end{bmatrix} \mathcal{K}_3(x_{k+1} - x_k) - \begin{bmatrix} 1.3 & 2.5 \end{bmatrix} \mathcal{K}_2(u_k),$$

(24)

where the function $\mathcal{K}_i: \mathbb{R}^i \rightarrow \mathbb{R}^i$ is given by

$$\forall \mathbf{v} = [v_1, v_2, ..., v_i]^T \in \mathbb{R}^i, \quad (\mathcal{K}_i(\mathbf{v}))[j] = \begin{cases} v_j^2 & \text{if } |v_j| \geq 1 \\ |v_j| & \text{otherwise} \end{cases} \quad (j = 1, ..., i).$$

(25)

The $j$-th element of the vector $\mathbf{b}$ is denoted by $(b)_j$. The absolute value of $v_j$ is denoted by $|v_j|$. The reward given by the reward function $R_0$ is maximum when the plant is stabilized at one of equilibrium points, that is, $x_{k+1} = x_k$ and $u_k = 0$. In the case where for each $k$, $\tau_{sc,k}, \tau_{cp,k} = 0$, the controller can learn its control policy. On the other hand, in the case where for each $k$, $\tau_{sc,k}, \tau_{cp,k} \sim U(3\Delta, 5\Delta)$, the controller cannot proceed with its control policy as shown in Fig. 8. The horizontal axis represents the number of episodes and the vertical axis represents the mean value of the rewards obtained by the control policy after each episode learning, where the initial state is $x(0) = [9.0, -9.0, 18.0]^T$ and the controller controls the plant for $0 \leq t \leq 25$.

Next, we use the extended state $z$ as the state of the environment $E$. It is assumed that the following reward function $R_z: (\mathcal{X} \times \mathcal{U}^*) \times \mathcal{U} \times (\mathcal{X} \times \mathcal{U}^*) \rightarrow \mathbb{R}$ is given in order to meet the goal.
In the case where for each $k$, $\tau_{sc,k}$, $\tau_{cp,k} = 0$, the controller can learn its control policy based on the observed state $x_k$ (blue curve). On the other hand, in the case where for each $k$, $\tau_{sc,k}$, $\tau_{cp,k} \sim U(3\Delta, 5\Delta)$, the controller cannot learn its control policy (orange curve).

Fig. 9. Average learning curve using the extended state $z$. The solid curve and the shade represent the averaged result and the standard deviation over 10 trials, respectively.

$$R_z(z_k, u_k, z_{k+1}) = -r_z^T K_{23}(z_{k+1} - z_k) - \begin{bmatrix} 1.3 \\ 2.5 \end{bmatrix} K_2(u_k),$$

(26)

where $K_{23}$ and $K_2$ are defined by Eq. (25) and the vector $r_z$ is given by

$$r_z = \begin{bmatrix} 0.8 & 0.8 & 8.0 & 0.15 & \cdots & 0.15 \end{bmatrix}^T \in \mathbb{R}^{23}.$$  

(27)

The average learning curve over 10 trials is shown in Fig. 9. The horizontal axis represents the number of episodes and the vertical axis represents the mean value of the rewards obtained by the control policy after each learning episode, where the initial state is $x(0) = [9.0, -9.0, 18.0]^T$ and the controller controls the plant for $0 \leq t \leq 25$. It is shown that the controller can learn a control policy that achieves high rewards using the extended state $z$. Moreover, the time response of the Lorenz equation controlled by the control policy after sufficient learning is shown in Fig. 10. It is shown that
the controller that sufficiently learned its control policy by the proposed method can stabilize the Lorenz equation.

5.2 Output-based learning

In this subsection, we assume that the output function is given by
We assume that $\Delta = 2^{-4}$. First, we use the current output $y_k$ and past control inputs $u_{k-1}, \ldots, u_{k-10}$ as the state of the environment $E$. This state is denoted by $w_k = [y_k^T, u_{k-1}^T, \ldots, u_{k-10}^T]^T$. It is assumed that the following reward function $R_{w^0}: (Y \times U^*) \times U \times (Y \times U^*) \to \mathbb{R}$ is given in order to meet the goal.

$$ R_{w^0}(w_k^0, u_k, w_{k+1}^0) = -r_0^T K_{22}(w_{k+1}^0 - w_k^0) - \begin{bmatrix} 1.3 & 2.5 \end{bmatrix} K_2(u_k), \quad (29) $$

where $K_{22}$ and $K_2$ are defined by Eq. (25) and the vector $r_0$ is given by

$$ r_0 = \begin{bmatrix} 0.8 & 8.0 & 0.15 & \cdots & 0.15 \end{bmatrix}^{\top} \in \mathbb{R}^{22}. \quad (30) $$

The learning curve is shown in Fig. 11. The horizontal axis represents the number of episodes and the vertical axis represents the mean value of the rewards obtained by the control policy after each episode learning, where the initial state is $x(0) = [9.0, -9.0, 18.0]^T$ and the controller controls the plant for $0 \leq t \leq 25$. It is shown that the controller cannot proceed with its control policy using the state $w^0$ under the partial observation.

Next, we use the extended state $w$ as the state of the environment $E$. We set the length of the past output sequence $\zeta = 1, 4, 7$, where these extended states are denoted by $w^1, w^4$, and $w^7$, respectively. It is assumed that the following reward functions $R_{w^i}: (Y^{\zeta+1} \times U^{\zeta+\zeta}) \times U \times (Y^{\zeta+1} \times U^{\zeta+\zeta}) \to \mathbb{R}$ are given in order to meet the goal.

$$ R_{w^1}(w_k^1, u_k, w_{k+1}^1) = -r_1^T K_{26}(w_{k+1}^1 - w_k^1) - \begin{bmatrix} 1.3 & 2.5 \end{bmatrix} K_2(u_k), \quad (31) $$

$$ R_{w^4}(w_k^4, u_k, w_{k+1}^4) = -r_4^T K_{38}(w_{k+1}^4 - w_k^4) - \begin{bmatrix} 1.3 & 2.5 \end{bmatrix} K_2(u_k), \quad (32) $$

$$ R_{w^7}(w_k^7, u_k, w_{k+1}^7) = -r_7^T K_{50}(w_{k+1}^7 - w_k^7) - \begin{bmatrix} 1.3 & 2.5 \end{bmatrix} K_2(u_k), \quad (33) $$

where $K_2, K_{26}, K_{38}$, and $K_{50}$ are defined by Eq. (25) and the vectors $r_1, r_4$, and $r_7$ are given by

$$ r_1 = \begin{bmatrix} 0.8 & 8.0 & 0.8 & 8.0 & 0.15 & \cdots & 0.15 \end{bmatrix}^{\top} \in \mathbb{R}^{26}, \quad (34) $$
Fig. 12. Average learning curve using the extended state $w^1$. The solid curve and the shade represent the averaged result and the standard deviation over 10 trials, respectively.

Fig. 13. Average learning curve using the extended state $w^4$. The solid curve and the shade represent the averaged result and the standard deviation over 10 trials, respectively.

$$r_4 = \begin{bmatrix} 0.8 & 8.0 & \cdots & 0.8 & 8.0 & 0.15 & \cdots & 0.15 \end{bmatrix}^T \in \mathbb{R}^{38},$$ (35)

$$r_7 = \begin{bmatrix} 0.8 & 8.0 & \cdots & 0.8 & 8.0 & 0.15 & \cdots & 0.15 \end{bmatrix}^T \in \mathbb{R}^{50}. $$ (36)

Average learning curves over 10 trials are shown in Figs. 12, 13, and 14, respectively. The horizontal axis represents the number of episodes and the vertical axis represents the mean value of the rewards obtained by the control policy after each episode learning, where the initial state is $x(0) = [9.0, -9.0, 18.0]^T$ and the controller controls the plant for $0 \leq t \leq 25$. It is shown that the controller can learn the control policy that achieves high rewards in all cases $\varsigma = 1, 4, 7$. On the other hand, if we set the meta-parameter $\varsigma$ to a larger value, the controller may fail to learn an
optimal control policy. For example, shown in Fig. 15 is a learning curve in the case where we set $\varsigma = 14$. The extended state is denoted by $w^{14}$. It is assumed that the following reward function

$$ R_{w^{14}}(w_k^{14}, u_k, w_{k+1}^{14}) = -r_{14}^T K_{28}(w_{k+1}^{14} - w_k^{14}) - \begin{bmatrix} 1.3 & 2.5 \end{bmatrix} K_2(u_k), $$

where $K_2$ and $K_{28}$ are defined by Eq. (25) and the vector $r_{14}$ is given by

$$ r_{14} = \begin{bmatrix} 0.8 & 8.0 & \cdots & 0.8 & 8.0 & 0.15 & \cdots & 0.15 \end{bmatrix}^T \in \mathbb{R}^{78}. $$

Although the mean of rewards obtained by the learned control policy is increasing until 200 episodes, it drops after about 350 episodes. Moreover, it is oscillating after 400 episodes. It is shown that a large meta-parameter setting causes unstable learning.

Furthermore, the time response of the Lorenz equation controlled by the control policy after sufficient learning episodes using $w^4$ is shown in Fig. 16. It is shown that the controller that sufficiently
learned its control policy by the proposed method can stabilize the Lorenz equation to the target equilibrium point.

6. Conclusion
In this paper, we proposed a DRL-based learning method to design the networked controller that takes into the consideration two types of network delays caused by data transmissions between the plant and the controller. First, we consider the case where all state are observed and we use the

Fig. 16. Time response of the Lorenz equation by the controller that sufficiently learned its control policy using the extended state $w^4$, where the initial state $x(0) = [9.0, -9.0, 18.0]^T$. The red dotted lines represent the equilibrium point $[x_1, x_2, x_3]^T = [6\sqrt{2}, 6\sqrt{2}, 27]^T$. 
current observed state and the past determined control inputs as the state of the environment. Next, we consider the case where the sensor cannot observe all state variables of the plant and we use not only the current observed output and the past determined control inputs but also the past observed outputs as the state of the environment. As examples, we consider the stabilization of the Lorenz equation and show that the controller learns its control policy by the proposed method. On the other hand, we cannot select the length of the past output sequence theoretically. It is future work to examine how to select the meta-parameter properly. Moreover, it is future work to examine how to select the DRL algorithm according to control problems, namely the DDPG algorithm or the CDQL algorithm. It is also future work to apply our proposed learning method to design of a networked controller for real systems.

Acknowledgments
This work was partially supported by JST-ERATO HASUO Project Grant Number JPMJER1603, Japan and JST-Mirai Program Grant Number JPMJMI18B4, Japan.

Appendix
A. Exploration noise
We use an Ornstein Uhlenbeck process \( \{ \omega_t \}_{t \in [0,T]} \) \cite{35} as the stochastic process that generates exploration noises. The stochastic process is given by the following stochastic differential equation.

\[
d\omega_t = -p_1(\omega_t - p_2) + p_3 dW_t,
\]

where \( p_1, p_2, \) and \( p_3 \) are given parameters and \( \{ W_t \}_{t \in [0,T]} \) is a Wiener process. In this paper, we use noises \( \{ \omega_k \}_{k \in \{0,1,2,...\}} \) generated by the following difference equation based on Eq. (A-1).

\[
\begin{cases}
\omega_k = \omega_{k-1} - p_1(\omega_{k-1} - p_2) + p_3 w,
\omega_0 = 0,
\end{cases}
\]

where \( w \) is a noise generated by a standard normal distribution. We set the parameters \( (p_1, p_2, p_3) = (0.15, 0, 0.5) \) between the first 25 episodes. After that, we set the parameters \( (p_1, p_2, p_3) = (0.15, 0, 0.3) \). Moreover, we double the size of generated noise \( \omega_k \) before 400th episode:

\[
\epsilon_k \leftarrow 2\omega_k.
\]

After that, we weight the generated noise as follows:

\[
\epsilon_k \leftarrow \frac{800}{m} \omega_k,
\]

where \( m (> 400) \) is the current episode.

References
[1] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction Second Edition, MIT Press, 2018.
[2] I. Ichikawa and T. Ushio, “Application of reinforcement learning to adaptive control of connected vehicles,” NOLTA, vol. 10, no. 4, pp. 443–454, 2019.
[3] T. Kamio, K. Mitsubori, T. Tanaka, H. Fujisaka, and K. Haeiwa, “Effects of prior knowledge on multi-agent reinforcement leaning system to find courses of ships,” Australian Journal of Intelligent Information Processing Systems, vol. 12, no. 2, pp. 18–23, 2010.
[4] T. Tomihara, T. Kamio, T. Tanaka, K. Mitsubori, and H. Fujisaka, “Modification of near-miss courses by reinforcement learning to search ships’ courses,” Proc. NOLTA ’19, pp. 633–636, Kuala Lumpur, Malaysia, December 2019.
[5] M. Hiromoto and T. Ushio, “Learning an optimal control policy for a Markov decision process under linear temporal logic specifications,” Proc. IEEE Symposium Series on Computational Intelligence, pp. 548–555, Cape Town, South Africa, December 2015.
[6] Y. Yamakura and T. Ushio, “Application of reinforcement learning to hierarchical control for linear temporal logic constraints,” *IEICE Trans. Fundamentals (Japanese Edition)*, vol. J102-A, no. 10, pp. 254–263, 2019.

[7] S. Gadaleta and G. Dangelmayr, “Optimal chaos control through reinforcement learning,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 9, no. 3, pp. 775–788, 1999.

[8] 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, pp. 529–533, 2015.

[9] M. Riedmiller, “Neural fitted Q iteration - First experiences with a data efficient neural reinforcement learning method,” *Proc. European Conference on Machine Learning*, pp. 317–328, Berlin, Germany, September 2006.

[10] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” *Proc. AAAI 2018*, pp. 3215–3222, New Orleans, USA, February 2018.

[11] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, “Distributed prioritized experience replay,” *arXiv preprint* arXiv:1803.00933, 2018.

[12] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, “Towards vision-based deep reinforcement learning for robotic motion control,” *arXiv preprint* arXiv:1511.03791, 2015.

[13] B. Demirel, A. Ramaswamy, D.E. Quevedo, and H. Karl, “DeepCAS: A deep reinforcement learning algorithm for control-aware scheduling,” *IEEE Control Systems Letters*, vol. 2, no. 4, pp. 737–742, 2018.

[14] N. Masuda and T. Ushio, “Control of nonholonomic vehicle system using hierarchical deep reinforcement learning,” *Proc. NOLTA’17*, pp. 26–29, Cancun, Mexico, December 2017.

[15] T.D. Kulkarni, K.R. Narasimhan, A. Saeedi, and J.B. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” *Proc. NIPS’16*, pp. 3675–3683, Barcelona, Spain, December 2016.

[16] 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,” *arXiv preprint* arXiv:1509.02971, 2015.

[17] D. Baumann, J.-J. Zhu, G. Martius, and S. Trimpe, “Deep reinforcement learning for event-triggered control,” *Proc. IEEE CDC 2018*, pp. 943–950, Florida, USA, December 2018.

[18] J. Duan, D. Shi, R. Diao, H. Li, Z. Wang, B. Zhang, D. Bian, and Z. Yi, “Deep-reinforcement-learning-based autonomous voltage control for power grid operations,” *IEEE Trans. Power Systems*, vol. 35, no. 1, pp. 814–817, 2019.

[19] J.P. Hespanha, P. Naghshtabrizi, and Y. Xu, “A survey of recent results in networked control systems,” *Proceedings of IEEE*, vol. 95, no. 1, pp. 138–162, 2007.

[20] R. Yang, G.-P. Liu, P. Shi, C. Thomas, and M.V. Basin, “Predictive output feedback control for networked control systems,” *IEEE Trans. Industrial Electronics*, vol. 61, no. 1, pp. 512–520, 2014.

[21] H. Gao, T. Chen, and J. Lam, “A new delay system approach to network-based control,” *Automatica*, vol. 44, no. 1, pp. 39–52, 2008.

[22] T. Fujita and T. Ushio, “RL-based optimal networked control considering networked delay of discrete time linear systems,” *Proc. IEEE ECC 2015*, pp. 2481–2486, Linz, Austria, July 2015.

[23] T. Fujita and T. Ushio, “Optimal digital control with uncertain network delay of linear systems using reinforcement learning,” *IEICE Trans. Fundamentals*, vol. E99-A, no. 2, pp. 454–461, 2016.

[24] J. Ikemoto and T. Ushio, “Stabilization of nonlinear systems with uncertain input delays using deep reinforcement learning,” *IEICE Trans. Fundamentals (Japanese Edition)*, vol. J102-A, no. 10, pp. 268–271, 2019.

[25] J. Ikemoto and T. Ushio, “Application of continuous deep Q-learning to networked state-feedback control of nonlinear systems with uncertain network delays,” *Proc. NOLTA’19*, pp. 192–195, Kuala Lumpur, Malaysia, December 2019.
[26] M. Hausknecht and P. Stone, “Deep recurrent Q-learning for partially observable MDPs,” 2015 AAAI Fall Symposium Series Virginia, USA, November 2015.

[27] N. Heess, J.J. Hunt, T. Lillicrap, and D. Silver, “Memory-based control with recurrent neural networks,” arXiv preprint arXiv:1512.04455, 2015.

[28] J. Ikemoto and T. Ushio, “Networked control of nonlinear systems under partial observation using continuous deep Q-learning,” Proc. IEEE CDC 2019, pp. 6793–6798, Nice, France, December 2019.

[29] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep Q-learning with model-based acceleration,” Proc. ICML 2016, pp. 2829–2838, New York, USA, June 2016.

[30] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” Proc. ICML 2014, pp. 387–395, Beijing, China, June 2014.

[31] T. Degris, M. White, and R.S. Sutton, “Off-policy actor-critic,” Proc. ICML 2012, pp. 179–186, Edinburgh, Scotland, June 2012.

[32] W. Aangenent, D. Kostic, B. de Jager, R. van de Molengraft, and M. Steinbuch, “Data-based optimal control,” Proc. IEEE ACC 2005, pp. 1460–1465, Portland, Oregon, USA, June 2005.

[33] F.L. Lewis and K.G. Vamvoudakis, “Reinforcement learning for partially observable dynamic processes: adaptive dynamic programming using measured output data,” IEEE Trans. Systems, Man, and Cybernetics, Part B, vol. 41, no. 1, pp. 14–25, 2011.

[34] D.P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.

[35] G.E. Uhlenbeck and L.S. Ornstein, “On the theory of the brownian motion,” Physical Review, vol. 36, pp. 823–841, 1930.