Deep Reinforcement Learning Based Networked Control with Network Delays for Signal Temporal Logic Specifications

Junya Ikemoto
Engineering Science
Osaka University
Toyonaka, Japan
email address: ikemoto@hopf.sys.es.osaka-u.ac.jp

Toshimitsu Ushio
Engineering Science
Osaka University
Toyonaka, Japan
email address: ushio@sys.es.osaka-u.ac.jp

Abstract—We apply deep reinforcement learning (DRL) to design of a networked controller with network delays to complete a temporal control task that is described by a signal temporal logic (STL) formula. STL is useful to deal with a specification with a bounded time interval for a dynamical system. In general, an agent needs not only the current system state but also the past behavior of the system to determine a desired control action for satisfying the given STL formula. Additionally, we need to consider the effect of network delays for data transmissions. Thus, we propose an extended Markov decision process (MDP) using past system states and control actions, which is called a τd-MDP, so that the agent can evaluate the satisfaction of the STL formula considering the network delays. Thereafter, we apply a DRL algorithm to design a networked controller using the τd-MDP. Through simulations, we also demonstrate the learning performance of the proposed algorithm.

Index Terms—deep reinforcement learning, signal temporal logic, network control systems, network delays

I. INTRODUCTION

Networked control systems (NCSs) have attracted considerable attention owing to the development of network technologies [1]. NCSs are systems with loops closed through networks, as shown in Fig. 1, and have many advantages in various control problems. However, in NCSs, network delays are caused by data transmission between a sensor/actuator and a controller. In conventional model-based controller designs, we identify the mathematical model of a system and network delays beforehand. However, it may be difficult to identify them precisely in real-world problems. Subsequently, *reinforcement learning* (RL) [2] is useful because we can adaptively design a controller through interactions with the system.

RL is a machine learning method used in various fields to solve sequential decision-making problems, and has been studied in the control field because it is strongly associated with optimal control methods from a theoretical point of view. Moreover, RL with *deep neural networks* (DNNs), called *Deep RL* (DRL), has been developed for complicated decision-making problems [3], such as playing Atari 2600 video games [4] and locomotion or manipulation of complicated systems
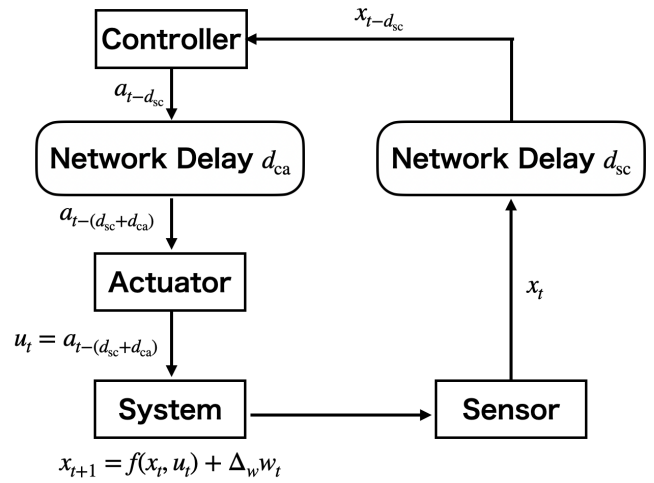

Fig. 1. Illustration of a network control system (NCS). Although the system has many advantages, there exist two types of network delays that may degrade control performance.

[5]–[8]. DRL-based networked controller designs have been proposed [9]–[11]. In [9], Baumann *et al.* proposed a DRL-based event-triggered control method. In [10], Demirel *et al.* proposed a control-aware scheduling algorithm to synthesize an optimal controller for some subsystems. In [11], we proposed DRL-based networked controller designs to stabilize an uncertain nonlinear system with network delays.

On the other hand, in RL-based controller designs, we must design a reward function for desired system behavior beforehand, which is difficult for a temporal high-level control task. To handle the temporal control task, *temporal logic* (TL) [12] is useful. TL is a branch of formal methods and has also been applied to several control problems [13]. *Signal temporal logic* (STL) [14] is particularly useful in designing controllers for dynamical systems as it can specify continuous signals within a bounded time interval. STL has also been studied in the machine learning community. In [15], Ma *et al.*

proposed an STL-based learning framework with knowledge of model properties. Moreover, RL-based controller design methods using STL formulae have been proposed [16]–[19]. In [16], Aksaray *et al.* proposed a Q-learning-based method to design a control policy that satisfies a given STL specification. They introduced an extended *Markov decision process* (MDP), which is called a τ-*MDP*, and designed a reward function to learn a control policy satisfying the STL specification. The extended state of the τ-MDP comprises the current state and the past system state sequence, where the dimension of the extended state depends on the given STL formula. In [17], Venkataraman *et al.* proposed a tractable learning method using a flag state instead of the past state sequence to mitigate the *curse of dimensionality*. However, these methods cannot be directly applied to problems with continuous state and action spaces because they are based on tabular Q-learning. In [18], Balakrishnan *et al.* introduced a *partial signal* and proposed a DRL-based method. In [19], Kapoor *et al.* proposed a model-based DRL method. The model of the system was learned using a DNN, and the controller was designed using a nonlinear model predictive control method. Li *et al.* proposed a policy search algorithm using *truncated linear temporal logic* (TLTL) that does not have a time bound [20], [21].

In this study, we formulate a temporal control specification as an STL formula and propose a DRL-based networked controller design in the presence of networked delays.

Contribution: The main contribution of this paper is the development of a DRL-based networked controller design for satisfying STL specifications with fixed network delays. In this study, it is assumed that we cannot identify the mathematical model of the system and the network delays beforehand, where the bounds of the network delays are known. To design the networked controller, we proposed an extended MDP, which is called a τd-*MDP*, and a practical DRL-based networked controller design using the extended MDP.

Structure: The remainder of this paper is organized as follows. In Section II, we review STL as preliminaries. In Section III, we formulate a networked control problem for a stochastic discrete-time system. In Section IV, we propose a DRL-based networked controller design to satisfy a given STL specification in the presence of networked delays. In Section V, using numerical simulations, we demonstrate the usefulness of the proposed method. In Section VI, we conclude the paper and discuss future work.

Notation: $\mathbb{N}_{\geq 0}$ is the set of non-negative integers. $\mathbb{R}$ is the set of the real numbers. $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. $\mathbb{R}^n$ is the n-dimensional Euclidean space. 0_n is an n-dimensional zero vector. For a set $A \subseteq \mathbb{R}$, $\max A$ and $\min A$ are the maximum value and the minimum value in A if they exist, respectively.

II. SIGNAL TEMPORAL LOGIC

In this study, we describe a desired control task as an STL formula with the following *syntax*.

$$
\begin{aligned}
\Phi &::= G_{[0,T_e]}\phi \mid F_{[0,T_e]}\phi, \\
\phi &::= G_{[t_s,t_e]}\varphi \mid F_{[t_s,t_e]}\varphi \mid \phi \wedge \phi \mid \phi \vee \phi, \\
\varphi &::= \psi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi,
\end{aligned}
$$

where T_e, t_s, and $t_e \in \mathbb{N}_{\geq 0}$ are nonnegative constants for the time bounds of temporal operators. Φ, ϕ, φ, and ψ are STL formulae. ψ is a predicate in the form of $h(x) \leq y$, where $h : \mathcal{X} \to \mathbb{R}$ is a function of a system state, and $y \in \mathbb{R}$ is a constant. The Boolean operators $\neg$, $\wedge$, and $\vee$ are *negation*, *conjunction*, and *disjunction*, respectively. The temporal operators $G_{\mathcal{T}}$ and $F_{\mathcal{T}}$ refer to *Globally* (always) and *Finally* (eventually), respectively, where $\mathcal{T}$ denotes the time bound of the temporal operator in the form of $[t_s, t_e]$, $t_s \leq t_e$. $\phi_i = G_{[t_s^i,t_e^i]}\varphi_i$ or $F_{[t_s^i,t_e^i]}\varphi_i$, $i \in \{1, 2, ..., M\}$ are called *STL sub-formulae*. ϕ comprises multiple STL sub-formulae $\{\phi_i\}_{i=1}^M$.

x_t and $x_{t_1:t_2}$ denote the state at t and the partial trajectory for a discrete-time interval $[t_1, t_2]$, where $t_1 \leq t_2$. The *Boolean semantics* of STL is recursively defined as follows:

$$
\begin{aligned}
x_{t:T} &\models \psi \Leftrightarrow h(x_t) \leq y, \\
x_{t:T} &\models \neg\psi \Leftrightarrow \neg(x_{t:T} \models \psi), \\
x_{t:T} &\models \phi_1 \wedge \phi_2 \Leftrightarrow x_{t:T} \models \phi_1 \wedge x_{t:T} \models \phi_2, \\
x_{t:T} &\models \phi_1 \vee \phi_2 \Leftrightarrow x_{t:T} \models \phi_1 \vee x_{t:T} \models \phi_2, \\
x_{t:T} &\models G_{[t_s,t_e]}\phi \Leftrightarrow x_{t':T} \models \phi, \ \forall t' \in [t+t_s, t+t_e] \\
x_{t:T} &\models F_{[t_s,t_e]}\phi \Leftrightarrow \exists t' \in [t+t_s, t+t_e], \ \text{s.t.} \ x_{t':T} \models \phi,
\end{aligned}
$$

where T ($\geq t_e$) denotes the length of the trajectory.

The *quantitative semantics* of STL is recursively defined as follows:

$$
\begin{aligned}
\rho(x_{t:T}, \psi) &= y - h(x_t), \\
\rho(x_{t:T}, \neg\psi) &= -\rho(x_{t:T}, \psi) \\
\rho(x_{t:T}, \phi_1 \wedge \phi_2) &= \min\{\rho(x_{t:T}, \phi_1), \rho(x_{t:T}, \phi_2)\}, \\
\rho(x_{t:T}, \phi_1 \vee \phi_2) &= \max\{\rho(x_{t:T}, \phi_1), \rho(x_{t:T}, \phi_2)\}, \\
\rho(x_{t:T}, G_{[t_s,t_e]}\phi) &= \min_{t' \in [t+t_s, t+t_e]} \rho(x_{t':T}, \phi), \\
\rho(x_{t:T}, F_{[t_s,t_e]}\phi) &= \max_{t' \in [t+t_s, t+t_e]} \rho(x_{t':T}, \phi),
\end{aligned}
$$

which quantifies how well the trajectory satisfies a given STL formulae [22].

The *horizon length* of an STL formula is recursively defined as follows:

$$
\begin{aligned}
\mathrm{hrz}(\psi) &= 0, \\
\mathrm{hrz}(\phi) &= t_e, \ \text{for} \ \phi = G_{[t_s,t_e]}\varphi, \ \text{or} \ F_{[t_s,t_e]}, \varphi \\
\mathrm{hrz}(\neg\phi) &= \mathrm{hrz}(\phi), \\
\mathrm{hrz}(\phi_1 \wedge \phi_2) &= \max\{\mathrm{hrz}(\phi_1), \mathrm{hrz}(\phi_2)\}, \\
\mathrm{hrz}(\phi_1 \vee \phi_2) &= \max\{\mathrm{hrz}(\phi_1), \mathrm{hrz}(\phi_2)\}, \\
\mathrm{hrz}(G_{[t_s,t_e]}\phi) &= t_e + \mathrm{hrz}(\phi), \\
\mathrm{hrz}(F_{[t_s,t_e]}\phi) &= t_e + \mathrm{hrz}(\phi).
\end{aligned}
$$

hrz(ϕ) is the required length of the state sequence to verify the satisfaction of the STL formula ϕ [23].

III. PROBLEM STATEMENT

We design a networked controller for the following stochastic discrete-time dynamical system as shown in Fig. 1.

$$x_{t+1} = f(x_t, u_t) + \Delta_w w_t, \tag{1}$$

where $x_t \in \mathcal{X}$, $u_t \in \mathcal{U}$, and $w_t \in \mathcal{W}$ are the system state, the control input, and the system noise at the discrete-time $t \in \{0, 1, ...\}$. $\mathcal{X} = \mathbb{R}^{n_x}$, $\mathcal{U} \subseteq \mathbb{R}^{n_u}$, and $\mathcal{W} = \mathbb{R}^{n_x}$ are the state space, the control input space, and the system noise space, respectively. The system noise w_t is an independent and identically distributed random variable with a probability density $p_w : \mathcal{W} \to \mathbb{R}_{\geq 0}$. Δ_w is a regular matrix that is a weighting factor of the system noise. $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is a function that describes the system dynamics. Then, we have the transition probability density $p_f(x'|x, u) := |\Delta_w^{-1}| p_w(\Delta_w^{-1}(x' - f(x, u)))$. The initial state $x_0 \in \mathcal{X}$ is sampled from a probability density $p_0 : \mathcal{X} \to \mathbb{R}_{\geq 0}$. The goal is to design a control policy that satisfies $x_{0:T}^\pi \models \Phi$, where $x_{0:T}^\pi$ is a system trajectory controlled by a control policy π and Φ is a given STL specification.

In the NCS, there exist two types of network delays: a *sensor-to-controller delay* $d_{\text{sc}} \in \mathbb{N}$ caused by the transmission of the observed state and a *controller-to-actuator delay* $d_{\text{ca}} \in \mathbb{N}$ caused by the transmission of a control input computed by the controller. In this study, it is assumed that these delays are uncertain constants bounded by the maximum delays $d_{\text{sc}}^{\max} \in \mathbb{N}$ and $d_{\text{ca}}^{\max} \in \mathbb{N}$, respectively. Then, the controller computes the k-th control input a_k based on the k-th observed state x_k at $t = k + d_{\text{sc}}$. Actually, the control input a_k is input to the system as follows:

$$u_t = \begin{cases} a_k & t = k + d_{\text{sc}} + d_{\text{ca}}, \\ 0_{n_u} & t < d_{\text{sc}} + d_{\text{ca}}, \end{cases} \tag{2}$$

where 0_{n_u} is a zero-vector of $\mathbb{R}^{n_u}$, that is the actuator inputs the control input 0_{n_u} until receiving a_0. Note that the controller cannot control the system until $t = d_{\text{sc}} + d_{\text{ca}}$. The controller computes control inputs $a_0, a_1, ..., a_{T - d_{\text{sc}} - d_{\text{ca}}}$ to satisfy the STL specification.

Furthermore, it is assumed that the mathematical models f and p_w are unknown. Thus, we apply RL to design a networked controller for satisfying the given STL specification Φ. In RL, an agent interacts with an environment and learns its control policy using the past interaction data. In this study, we regard the controller and everything outside the controller as the agent and the environment for RL, respectively. A control input determined by the agent is called a *control action*. However, a standard RL algorithm cannot be directly applied due to the following issues.

(i) The desired control action at each step in order to satisfy the STL specification Φ is determined not only by the current state but also by the past system behavior.

(ii) We must design a reward function to evaluate the satisfaction of the STL specification appropriately.

(iii) The classical RL algorithm cannot deal with continuous state and action spaces directly.

(iv) There exist uncertain network delays in the NCS.

In the next section, we propose a DRL-based controller design that resolves the issues.

IV. DRL-BASED NETWORKED CONTROLLER DESIGN FOR STL SPECIFICATIONS

A. τd-*Markov decision process*

For issue (i), Aksaray *et al.* introduced an extended MDP, which is called a τ-MDP, using a finite state sequence in [16]. For issue (ii), they designed a reward function of the τ-MDP to satisfy a given STL specification using the *log-sum-exp* approximation. They apply the classical Q-learning to design a policy for satisfying the given STL formula. However, their method cannot handle continuous control tasks directly. To resolve issue (iii), we extended the method using a DRL algorithm for problems with a continuous state-action space. In this study, we apply the *soft actor critic* (SAC) algorithm, because it has better sample efficiency and asymptotic performance. Additionally, we must consider network delays. For issue (iv), in [11], we proposed an extended state that comprises a current system state and previously determined control actions. As shown in Fig. 2, we consider the worst case scenario. At $t = k$, the sensor observes the k-th system state x_k, which is transmitted to the agent (controller) through the network. The agent receives the observed state x_k and determines the k-th control action a_k at $t = k + d_{\text{sc}}^{\max}$. The action a_k is sent to the actuator through the network. The actuator receives the k-th control action a_k and updates the control input $u_t = a_k$ at $t = k + d$ ($:= k + d_{\text{sc}}^{\max} + d_{\text{ca}}^{\max}$). Then, it is desirable for the agent to predict the future state x_{k+d} with available information and determine the k-th control action a_k based on the prediction. If we are aware of the system dynamics (1), we can predict the future state. However, the prediction requires not only the system dynamics (1) but also the information u_t, $t \in [k, k + d]$, which is the past control action sequence $a_{k-d:k-1} = a_{k-d}, a_{k-d+1}, ..., a_{k-1}$. Thus, we use not only the extended state proposed in [16] but also previously determined control actions. Actually, the true network delays d_{sc} and d_{ca} are uncertain constants. The agent adapts the true network delays through interactions with the system using sufficient information for predictions in the worst case. For issues (i), (ii), (iii), and (iv), we remodel the interactions between the agent and the system as the following extended MDP, which is called a τd-MDP.

Definition 1 (τd-MDP): Given an STL formula $\Phi = G_{[0, T_e]}\phi$ or $F_{[0, T_e]}\phi$, where hrz(Φ) $= T$ and ϕ comprises multiple STL sub-formulae ϕ_i, $i \in \{1, 2, ..., M\}$. Subsequently, we set $\tau = $ hrz(ϕ) $+ 1$, that is, $T = T_e + \tau - 1$. It is assumed that $d_{\text{sc}}^{\max} + d_{\text{ca}}^{\max} = d$. A τd-MDP is defined by a tuple $\mathcal{M}_{\tau, d} = \langle \mathcal{Z}, \mathcal{U}, p_0^z, p^z, R^z \rangle$, where

- $\mathcal{Z} \subseteq \mathcal{X}^\tau \times \mathcal{U}^d$ is an extended state space. Each extended state is denoted by $z = [(x^\tau)^\top \ (a^d)^\top]^\top$, where $x^\tau = [x^\tau[0]^\top \ x^\tau[1]^\top \ ... \ x^\tau[\tau - 1]^\top]^\top$ and $a^d =$
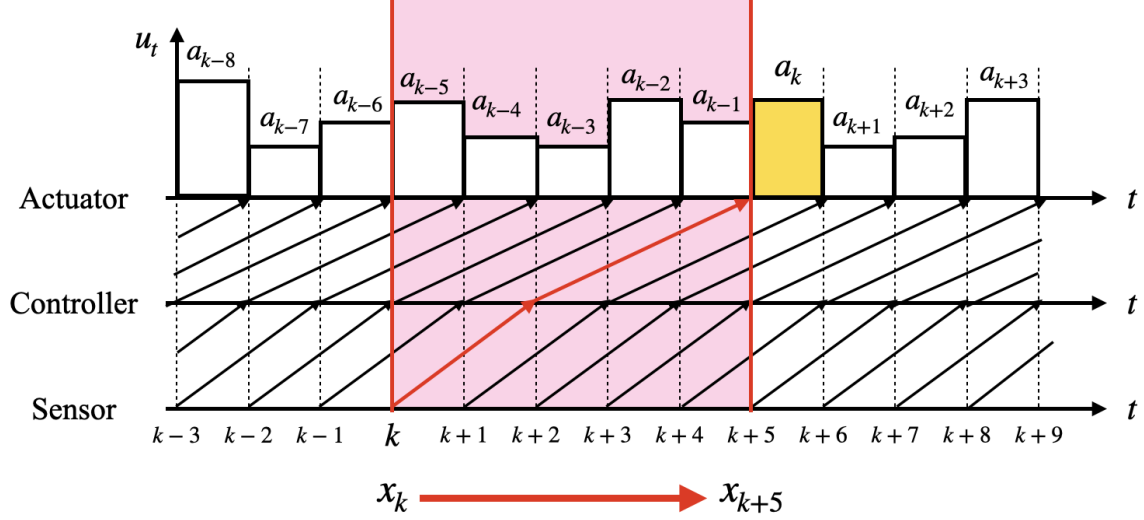
Fig. 2. Illustration of the network delays in data transmissions for the worst case, where $d_{\mathrm{sc}}^{\mathrm{max}} = 2$ and $d_{\mathrm{ca}}^{\mathrm{max}} = 3$ ($d = 5$). At $t = k+2$, the agent predicts x_{k+5} and determines a_k using the k-th state x_k and the past actions $a_{k-5}, ..., a_{k-1}$.

$[a^d[0]^\top \ a^d[1]^\top \ ... \ a^d[d-1]^\top]^\top$ are a past state sequence and a previously determined control action sequence, respectively, that is, $x^\tau[i] \in \mathcal{X}$, $\forall i \in \{0,1,...,\tau-1\}$ and $a^d[j] \in \mathcal{U}$, $\forall j \in \{0,1,...,d-1\}$.

- $\mathcal{U}$ is a control action space.
- p_0^z is a probability density for an initial extended state $z_0 = [(x_0^\tau)^\top \ (a_0^d)^\top]^\top$ with $x_0^\tau[i] = x_0$, $\forall i \in \{0,1,...,\tau-1\}$ and $a_0^d[j] = 0_{n_u}$, $\forall j \in \{0,1,...,d-1\}$, where x_0 is generated from p_0.
- p^z is a transition probability density for the extended state z. In the case where the system state is updated by $x' \sim p_f(\cdot|x,u)$, the extended state is updated by $z' \sim p^z(\cdot|z,a)$ as follows:

$$a^{d'}[j] = a^d[j+1], \ \forall j \in \{0,1,...,d-2\},$$
$$a^{d'}[d-1] = a,$$
$$x^{\tau'}[i] = x^\tau[i+1], \ \forall i \in \{0,1,...,\tau-2\},$$
$$x^{\tau'}[\tau-1] \sim p_f(\cdot|x^\tau[\tau-1], a^{d'}[d-1-d_{\mathrm{sc}}-d_{\mathrm{ca}}]),$$

where $z = [(x^\tau)^\top \ (a^d)^\top]^\top$ and $z' = [(x^{\tau'})^\top \ (a^{d'})^\top]^\top$ are the current extended state and the next extended state, respectively.

- $R_z : \mathcal{Z} \to \mathbb{R}$ is a reward function. Based on [16], it is defined by

$$R_z(z)$$
$$= \begin{cases} -\exp(-\beta \mathbf{1}(\rho(x^\tau, \phi))) & \text{for } \Phi = G_{[0,T_e]}\phi, \\ \exp(\beta \mathbf{1}(\rho(x^\tau, \phi))) & \text{for } \Phi = F_{[0,T_e]}\phi, \end{cases}$$

(3)

where $\beta > 0$ is a reward parameter. The function $\mathbf{1}$: $\mathbb{R} \to \{0,1\}$ is an indicator defined by

$$\mathbf{1}(y) = \begin{cases} 1 & \text{if } y \geq 0, \\ 0 & \text{if } y < 0. \end{cases}$$

The reward function is designed for satisfying the given STL specification using the log-sum-exp approximation [16].

The agent determines a control action according to a stochastic policy $\pi : \mathcal{Z} \to \mathcal{P}(\mathcal{U})$, where $\mathcal{P}(\mathcal{U})$ denotes the set of probability distributions over $\mathcal{U}$. In the SAC algorithm [8], we use the objective with the entropy term as follows:

$$J(\pi) \quad = \quad E_{p_\pi} \left[\sum_{k=0}^{T-d_{\mathrm{sc}}-d_{\mathrm{ca}}} \gamma^k (R_z(z_k) + \alpha \mathcal{H}(\pi(\cdot|z_k))) \right],$$

where $\gamma \in [0,1)$ is a discount factor, p_π is a trajectory distribution by the policy π, $\mathcal{H}$ is the entropy defined by $\mathcal{H}(\pi(\cdot|z)) = E_{a\sim\pi(\cdot|z)}[-\log\pi(a|z)]$, and $\alpha > 0$ is an entropy temperature. The goal is to obtain a control policy π that maximizes the objective. We give the stochastic policy π using a Gaussian with the mean μ_{θ_π} and the standard deviation σ_{θ_π} output by a DNN with *reparameterization trick* [24], which is called an *actor DNN*, whose parameter vector is denoted by θ_π. Additionally, we need to estimate the objective $J(\pi)$. We approximate the object $J(\pi)$ as another DNN, which is called a *critic DNN*, whose parameter vector is denoted by θ_Q. The parameter vector θ_Q is updated using the *experience replay* and the *target network technique* such as the *deep Q-network algorithm* [4]. These techniques can reduce correlation between experience data and make the learning performance stable, respectively.

The parameter vector θ_Q is updated by reducing the following *critic loss function*.

$$J(\theta_Q) = E_{(z,a,z',r)\sim\mathcal{D}}\left[(Q_{\theta_Q}(z,a) - (r + \gamma V_{\theta_Q^-}(z')))^2\right]. \quad (4)$$

The agent selects some experiences from a replay buffer $\mathcal{D}$ randomly for updates of θ_Q. The value $V_{\theta_Q^-}(z')$ is computed by

$$V_{\theta_Q^-}(z') = E_{a'\sim\pi_{\theta_\pi}}\left[Q_{\theta_Q^-}(z',a') - \alpha\log\pi_{\theta_\pi}(a'|z')\right],$$

where $Q_{\theta_Q^-}$ is the *target critic DNN*. The parameter vector θ_Q^- is slowly updated by the following *soft update*.

$$\theta_Q^- \leftarrow \xi\theta_Q + (1-\xi)\theta_Q^-, \quad (5)$$

where $\xi > 0$ is a sufficiently small positive constant. The parameter vector θ_π is updated by decreasing the following *actor loss function*.

$$J(\theta_\pi) = E_{z\sim\mathcal{D}, a\sim\pi_{\theta_\pi}}\left[\alpha\log(\pi_{\theta_\pi}(a|z)) - Q_{\theta_Q}(z,a)\right]. \quad (6)$$

The entropy temperature α is updated by decreasing the following loss function.

$$J(\alpha) = E_{z\sim\mathcal{D}}\left[\alpha(-\log(\pi_{\theta_\pi}(a|z)) - \mathcal{H}_0)\right], \quad (7)$$

where $\mathcal{H}_0$ is a lower bound. For example, in [8], $\mathcal{H}_0$ is set to $-\dim(\mathcal{U})$, where $\dim(\mathcal{U})$ denotes the dimension of the control action space $\mathcal{U}$. Actually, to keep α nonnegative after updates, we exponentiate the parameter.

B. Preprocessing for extended states

As τ becomes larger, the dimension of the extended state z also increases. Thereafter, it is difficult for an agent to learn its policy because of the curse of dimensionality. Thus, we use a preprocess to decrease the dimension of the extended state [17]. Although the preprocess is proposed for grid world problems, it can also be applied to continuous control tasks. We introduce the flag value f^i for each STL sub-formula ϕ_i.
Definition 2: For an extended state $z = [(x^\tau)^\top\ (a^d)^\top]^\top$, the flag value f^i of an STL sub-formula ϕ_i is defined as follows:
(i) For $\phi_i = G_{[t_s^i, t_e^i]}\varphi_i$,

$$f^i = \max\left\{\frac{t_e^i - l + 1}{t_e^i - t_s^i + 1}\ \middle|\ l \in \{t_s^i, ..., t_e^i\}\right.$$
$$\left.\wedge(\forall l' \in \{l, ..., t_e^i\},\ x^\tau[l'] \models \varphi_i)\right\}. \quad (8)$$

(ii) For $\phi_i = F_{[t_s^i, t_e^i]}\varphi_i$,

$$f^i = \max\left\{\frac{l - t_s^i + 1}{t_e^i - t_s^i + 1}\ \middle|\right.$$
$$\left. l \in \{t_s^i, ..., t_e^i\} \wedge x^\tau[l] \models \varphi_i\right\}. \quad (9)$$

Note that $\max\emptyset = -\infty$ and the flag value represents the normalized time lying in $(0, 1] \cup \{-\infty\}$. Intuitively, for $\phi_i = G_{[t_s^i, t_e^i]}\varphi_i$, the flag value indicates the time duration in which φ_i is always satisfied, whereas for $\phi_i = F_{[t_s^i, t_e^i]}\varphi_i$, the flag value indicates the instant when φ_i is satisfied. The flag

---

Algorithm 1 Preprocessing of the extended state z

1: **Input:** The extended state $z = [(x^\tau)^\top\ (a^d)^\top]^\top$ and the STL sub-formulae $\{\phi_i\}_{i=1}^M$.
2: **for** $i = 1, ..., M$ **do**
3:     **if** $\phi_i = G_{[t_s^i, t_e^i]}\varphi_i$ **then**
4:         Compute the flag value f^i by Eq. (8).
5:     **end if**
6:     **if** $\phi_i = F_{[t_s^i, t_e^i]}\varphi_i$ **then**
7:         Compute the flag value f^i by Eq. (9).
8:     **end if**
9: **end for**
10: Set the flag state $\hat{f} = [\hat{f}^1\ \hat{f}^2\ ...\ \hat{f}^M]$ by Eq. (10).
11: **Output:** The preprocessed state
    $\hat{z} = [x^\tau[\tau-1]^\top\ \hat{f}^\top\ (a^d)^\top]^\top$.

Algorithm 2 SAC-based algorithm to design a networked controller with network delays satisfying an STL specification

1: Initialize the parameter vectors of main critic DNNs θ_Q, θ_π.
2: Initialize the parameter vector of a target critic DNN θ_Q^-.
3: Initialize a replay buffer $\mathcal{D}$.
4: **for** Episode $= 1, ..., \text{MAX EPISODE}$ **do**
5:     Initialize the system state $x_0 \sim p_0$.
6:     **for** Discrete-time step $t = 0, ..., T$ **do**
7:         **if** $t \geq d_{\text{sc}}$ **then**
8:             Receive the $(t - d_{\text{sc}})$-th observed state $x_{t-d_{\text{sc}}}$.
9:             Construct the extended state $z_{t-d_{\text{sc}}}$.
10:            Compute the next preprocessed state $\hat{z}_{t-d_{\text{sc}}}$ by **Algorithm 1**.
11:            **if** $t > d_{\text{sc}}$ **then**
12:                Compute the reward $r_{t-d_{\text{sc}}-1} = R_z(z_{t-d_{\text{sc}}-1})$.
13:                Store the experience

$$(\hat{z}_{t-d_{\text{sc}}-1}, a_{t-d_{\text{sc}}-1}, \hat{z}_{t-d_{\text{sc}}}, r_{t-d_{\text{sc}}-1})$$

               in the replay buffer $\mathcal{D}$.
14:            **end if**
15:            Determine the action $a_{t-d_{\text{sc}}}$ based on the state $\hat{z}_{t-d_{\text{sc}}}$.
16:            Send the $(t - d_{\text{sc}})$-th control action $a_{t-d_{\text{sc}}}$ to the actuator.
17:         **end if**
18:         Sample I experiences

$$\{(\hat{z}^{(i)}, a^{(i)}, \hat{z}'^{(i)}, r^{(i)})\}_{i=1,...,I}$$

        from the replay buffer $\mathcal{D}$ randomly.
19:         Update the main DNNs θ_Q, θ_π by Eqs. (4) and (6).
20:         Update the target critic DNN θ_Q^- by Eq. (5).
21:         Update the entropy temperature α by Eq. (7).
22:     **end for**
23: **end for**

TABLE I
DIMENSION OF EXTENDED STATE SPACES

	Without Preprocessing z	With Preprocessing $\hat{z}$
Dimension	$\tau n_x + d n_u$	$n_x + M + d n_u$

values f^i, $i \in \{1, 2, \dots, M\}$ calculated by Eqs. (8) or (9) are transformed into $\hat{f}^i$ as follows:

$$\hat{f}^i = \begin{cases} f^i - \frac{1}{2} & \text{if } f^i \neq -\infty, \\ -\frac{1}{2} & \text{otherwise.} \end{cases} \tag{10}$$

The transformed flag values $\hat{f}^i$ are used as inputs to DNNs to avoid positive biases of the flag values and inputting $-\infty$ to DNNs. We compute the flag value for each STL sub-formula and construct a flag state $\hat{f} = [\hat{f}^1 \ \hat{f}^2 \ \dots \ \hat{f}^M]^\top$, which is called *preprocessing*. We use the preprocessed state $\hat{z} = [x^\tau[\tau-1]^\top \ \hat{f}^\top \ (a^d)^\top]^\top$ instead of the extended state $z = [(x^\tau)^\top \ (a^d)^\top]^\top$. If $M \ll \tau$, we can decrease the dimension of the extended state as shown in Table I. **Algorithm 1** summarizes the above preprocessing.

C. Algorithm

We propose an SAC-based algorithm presented in **Algorithm 2**. From lines 1 to 3, we initialize the parameter vectors of DNNs and a replay buffer $\mathcal{D}$. In line 5, we initialize the state of the system. From lines 6 to 22, the agent interacts with the system and learns its policy for an episode. In line 8, at t ($\geq d_{\text{sc}}$), the agent receives the state $x_{t-d_{\text{sc}}}$. In line 9, the extended state $z_{t-d_{\text{sc}}}$ is constructed using $x_{t-d_{\text{sc}}}$, $z_{t-d_{\text{sc}}-1}$, and $a_{t-d_{\text{sc}}-1}$. In line 10, the preprocessed state $\hat{z}_{t-d_{\text{sc}}}$ is computed by **Algorithm 1**. In line 12, if $t > d_{\text{sc}}$, the reward $r_{t-d_{\text{sc}}-1}$ is computed by Eq. (3). In line 13, the agent stores the experience $(\hat{z}_{t-d_{\text{sc}}-1}, a_{t-d_{\text{sc}}-1}, \hat{z}_{t-d_{\text{sc}}}, r_{t-d_{\text{sc}}-1})$ to the replay buffer $\mathcal{D}$. In line 15, the agent determines an exploration action $a_{t-d_{\text{sc}}}$ based on the preprocessed state $\hat{z}_{t-d_{\text{sc}}}$. In line 16, the agent sends $a_{t-d_{\text{sc}}}$ to the actuator. From lines 18 to 21, the agent updates the DNNs. In line 18, the agent samples I past experiences $\{(\hat{z}^{(i)}, a^{(i)}, \hat{z}'^{(i)}, r^{(i)})\}_{i=1}^{I}$ from the replay buffer $\mathcal{D}$ randomly. From lines 19 to 21, the agent updates the parameter vectors of the DNNs based on the SAC algorithm.

V. EXAMPLE

Consider a two-wheeled mobile robot in the environment shown in Fig. 3. A discrete-time model of the robot is described by

$$\begin{bmatrix} x_{t+1}^{(0)} \\ x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \end{bmatrix} = \begin{bmatrix} x_t^{(0)} + \Delta u_t^{(0)} \cos(x_t^{(2)}) \\ x_t^{(1)} + \Delta u_t^{(0)} \sin(x_t^{(2)}) \\ x_t^{(2)} + \Delta u_t^{(1)} \end{bmatrix} + \Delta_w \begin{bmatrix} w_t^{(0)} \\ w_t^{(1)} \\ w_t^{(2)} \end{bmatrix}, \tag{11}$$

where $x_t = [x_t^{(0)} \ x_t^{(1)} \ x_t^{(2)}]^\top \in \mathbb{R}^3$, $u_t = [u_t^{(0)} \ u_t^{(1)}]^\top \in \mathbb{R}^2$, and $w_t = [w_t^{(0)} \ w_t^{(1)} \ w_t^{(2)}]^\top \in \mathbb{R}^3$. $w_t^{(i)}$, $i \in \{0, 1, 2\}$ is sampled from a standard normal distribution $\mathcal{N}(0, 1)$. In the simulation, we assume that $\Delta = 0.1$ and $\Delta_w = 0.01I$, where I is the identity matrix. The initial state of the system
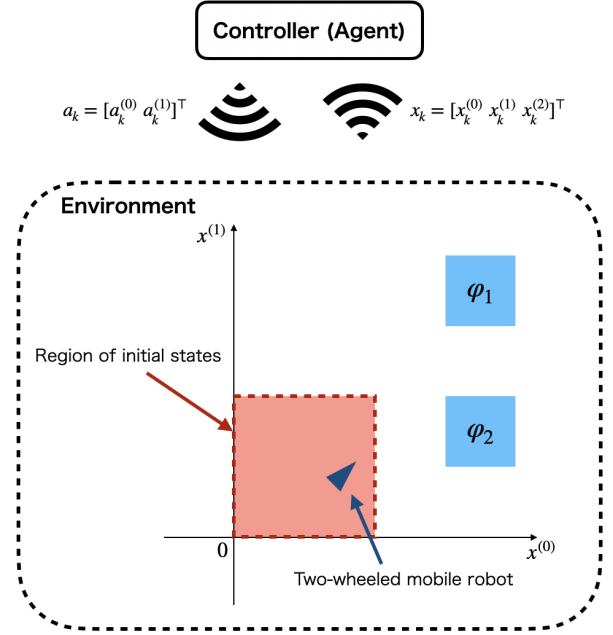

Fig. 3. Environment of the example. The agent learns the optimal policy for the two-wheeled mobile robot under the STL constraint.

is sampled randomly in the region $0 \leq x^{(0)} \leq 2.5$, $0 \leq x^{(1)} \leq 2.5$, $-\pi/2 \leq x^{(2)} \leq \pi/2$.

In this example, we consider the following STL formula.

$$\Phi = G_{[0,900]}(F_{[0,99]}\varphi_1 \wedge F_{[0,99]}\varphi_2), \tag{12}$$

where

$$\varphi_1 = ((3.75 \leq x^{(0)} \leq 5) \wedge (3.75 \leq x^{(1)} \leq 5)),$$
$$\varphi_2 = ((3.75 \leq x^{(0)} \leq 5) \wedge (1.25 \leq x^{(1)} \leq 2.5)),$$

that is, the length of x^τ is $\tau = 100$. It is assumed that $d_{\text{sc}} = 3$ and $d_{\text{ca}} = 4$, where these values are unknown, but we know that $d_{\text{sc}} \leq 5$ and $d_{\text{ca}} \leq 5$ beforehand. Then, the length of the past control action sequence a^d is $d = 10$. In all simulations, the DNNs have two hidden layers, all of which have 256 units, and all layers are fully connected. To mitigate the positive bias in the update of θ_π, the clipped double Q-learning technique [7] is adopted for $Q_{\theta_Q}(\hat{z}, a)$. The activation functions for the hidden layers and the outputs of the actor DNN are the rectified linear unit functions and hyperbolic tangent functions, respectively. We normalize $x^{(0)}$ and $x^{(1)}$ as $x^{(0)} - 2.5$ and $x^{(1)} - 2.5$, respectively. The size of the replay buffer $\mathcal{D}$ is 1.0×10^5, and the size of the mini-batch is $I = 64$. We use *Adam* [25] as the optimizers for all main DNNs and the entropy temperature. The learning rates of all optimizers multiplier are 3.0×10^{-4}. The soft update rate of the target network is $\xi = 0.01$. The discount factor is $\gamma = 0.99$. The target for updating the entropy temperature $\mathcal{H}_0$ is -2. The STL-reward parameter is $\beta = 100$. The agent learns its control policy for 6.0×10^5 steps. The initial entropy temperature is

1.0. For performance evaluation, we introduce the following two indices:

- a **learning curve** shows the mean of returns $\sum_{k=0}^{T-d_{sc}-d_{ca}} \gamma^k R_z(z_k)$ for 100 trajectories, and
- a **success rate** shows the number of trajectories satisfying the given STL constraint for 100 trajectories.

We prepare 100 initial states sampled from p_0 and generate 100 trajectories using the learned policy for each evaluation. All simulations are run on a computer with AMD Ryzen 9 3950X 16-core processor and 32GB of memory and are conducted using the Python software.

A. Effect of network delays

In this section, we demonstrate the effect of using past control actions as a part of an extended state, where we use the preprocessing introduced in Section IV. B. The learning curves and the success rates for the τ-MDP case (without past determined actions) and the τd-MDP case (with past determined actions) are shown in Figs. 4 and 5, respectively. If we do not use past determined actions, the obtained returns become high as the agent updates its policy, as shown in Fig. 4, but the success rate of the learned policy is not increasing as shown in Fig. 5. Conversely, if we use past determined actions, the agent can learn the policy that has a high success rate for the given STL specification. This result concludes that the agent needs not only past states but also past determined actions to learn the policy satisfying the given STL specification with network delays.
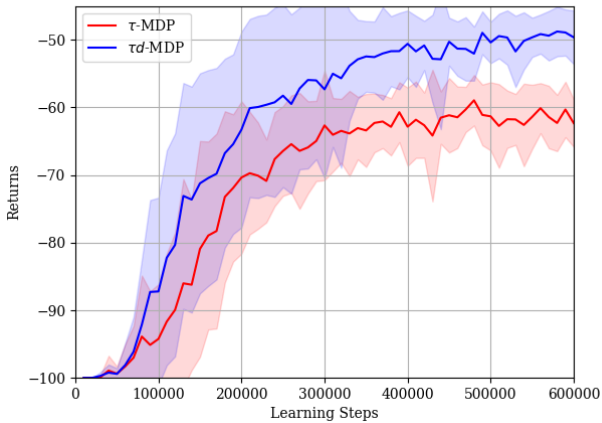
Fig. 5. Success rates for the τ-MDP case and the τd-MDP case. The solid curve and the shade represent the average results and the standard deviations over 15 trials with different random seeds, respectively.

the performance of its policy without preprocessing. Then, the learned policy has a low success rate as shown in Fig. 7. Conversely, in the case with preprocessing, the agent can learn a policy that obtains high returns and a high success rate. The result concludes that preprocessing is necessary for our proposed method under the STL specification with a large τ to mitigate curse of dimensionality.

Fig. 4. Learning curves for the τ-MDP case and τd-MDP case. The solid curve and the shade represent the average results and standard deviations over 15 trials with different random seeds, respectively.

B. Effect of preprocess

In this section, we show the improvement in the learning performance by preprocessing. In the case without preprocessing, the dimension of the extended state is 320 and, in the case with preprocessing, the dimension of the extended state is 25. As shown in Fig. 6, the agent cannot improve
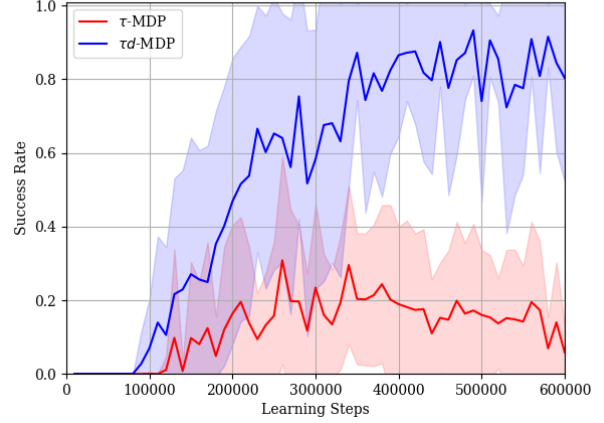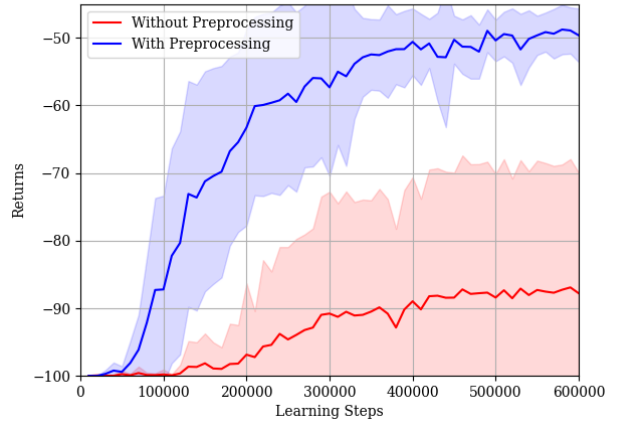
Fig. 6. Learning curves for the cases with and without preprocessing. The solid curve and the shade represent the average results and the standard deviations over 15 trials with different random seeds, respectively.

VI. CONCLUSION

We proposed a DRL-based networked controller design for a given STL specification with network delays. Subsequently, we introduced an extended MDP, which is called a τd-MDP, and proposed a DRL algorithm to design the networked controller. Through numerical simulations, we demonstrated the performance of the proposed method. On the other hand,
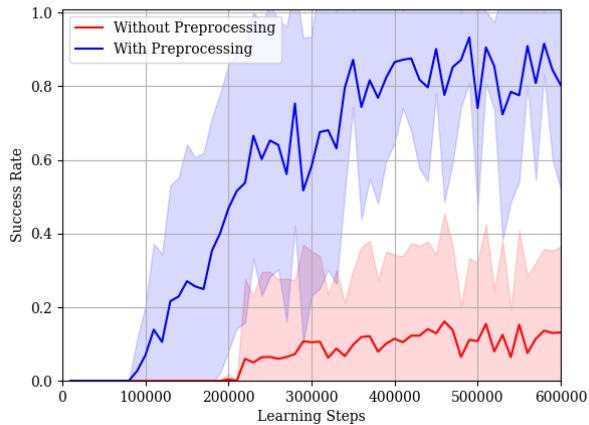
Fig. 7. Success rates for the cases with and without preprocessing. The solid curve and the shade represent the average results and the standard deviations over 15 trials with different random seeds, respectively.

for some STL specifications, the reward may be sparse. Additionally, the syntax in this study is the restrictive compared with the general STL syntax [14]. Solving these issues is an interesting direction for future work. As a practical problem, an extension of the proposed method to a system with network delays that fluctuate randomly is also a future work.

REFERENCES

[1] X.-M. Zhang, Q.-L. Han, X. Ge, D. Ding, L. Ding, D. Yue, and C. Peng, "Networked control systems: A survey of trends and techniques," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 1, pp. 1–17, Jul. 2019.
[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Second Edition)*, MIT Press, 2018.
[3] H. Dong, Z. Ding, and S. Zhang, *Deep Reinforcement Learning Fundamentals, Research and Applications*, Springer, 2020.
[4] 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, Feb. 2015.
[5] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," arXiv Preprint, *arXiv:1502.05477*, 2015.
[6] 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.
[7] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. of ICML 2018*, pp. 1587–1596, July 2018.
[8] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," arXiv Preprint, *arXiv:1812.05905*, 2018.
[9] D. Baumann, J.-J. Zhu, G. Martius, and S. Trimpe, "Deep reinforcement learning for event-triggered control," in *Proc. IEEE 58th Conf. Decis. Control (CDC)*, pp. 943–950, 2018.
[10] B. Demirel, A. Ramaswamy, D. E. Quevedo, and H. Karl, "DEEPCAS: A deep reinforcement learning algorithm for control-aware scheduling," *IEEE Control Syst. Lett.*, vol. 2, no. 4, pp. 737–742, Jun. 2018.
[11] J. Ikemoto and T. Ushio, "Application of deep reinforcement learning to networked control systems with uncertain network delays," *Nonlinear Theory and Its Applications*, IEICE, vol. 11, no. 4, pp. 480–500, Oct. 2020.
[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*, MIT Press, 2008.
[13] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*, Springer, 2017.
[14] O. Maler and D. Nickovic, "Monitoring temporal property of continuous signals," *Formal Techniques, Modeling and Analysis of Timed and Fault-Tolerant Systems*, pp. 71–76, Jan. 2004.
[15] M. Ma, J. Gao, L. Feng, and J. Stankovic, "STLnet: Signal temporal logic enforced multivariate recurrent neural networks," in *Proc. NeurIPS 2020*, 2020.
[16] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, pp. 6565–6570, 2016.
[17] H. Venkataraman, D. Aksaray, and P. Seiler, "Tractable reinforcement learning of signal temporal logic objectives," arXiv preprint, *arXiv:2001.09467*, 2020.
[18] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 3481–3486, 2019.
[19] P. Kapoor, A. Balakrishnan, and J. V. Deshmukh, "Model-based reinforcement learning from signal temporal logic specifications," arXiv preprint, *arXiv:2011.04950*, 2020.
[20] X. Li, C.-I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 3834–3839, 2017.
[21] X. Li, Y. Ma, and C. Belta, "A policy search method for temporal logic specified reinforcement learning tasks," in *Proc. IEEE Amer. Control Conf. (ACC)*, pp. 240–245, 2018.
[22] G. E. Fainekos and G. J. Pappas, "Robust of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.
[23] A. Dokhanchi, B. Hoxha, and G. Fainekos, "On-line monitoring for temporal logic robustness," in *Proc. of International Conference on Runtime Verification*, pp. 231–246, 2014.
[24] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," arXiv preprint, *arXiv:1312.6114*, 2013.
[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint, *arXiv: 1412.6980*, 2014.