A lifting approach to learning-based self-triggered control with Gaussian processes

Wang Zhijun, Kazumune Hashimoto, Wataru Hashimoto and Shigemasa Takai

Graduate School of Engineering, Osaka University, Suita, Japan

ABSTRACT
This paper investigates the design of self-triggered control for networked control systems (NCS), where the dynamics of the plant are initially unknown. Given the nature of the self-triggered control where state measurements are sent to the controller \( a \)-periodically, our proposal involves augmenting the continuous-time dynamics to a novel dynamical model that incorporates inter-event time as a supplemental input. Then, this new model is studied through Gaussian processes (GP) regression. Additionally, we propose a learning-based approach where a self-triggered controller is formulated by minimizing a cost function for the newly learned model, ensuring consideration of inter-sample behavior. The usage of the lifting approach facilitates the application of gradient-based policy updates, allowing for the effective optimization of both control and communication policies. Finally, we provide a numerical simulation to illustrate the effectiveness of the proposed approach.

ARTICLE HISTORY
Received 11 September 2023
Revised 28 November 2023
Accepted 9 December 2023

KEYWORDS
Event-triggered and self-triggered control; Gaussian process regression; optimal control

1. Introduction
In recent years, event-triggered and self-triggered control methods have been the focus of much attention due to their proven ability to conserve communication resources within the realm of networked control systems (NCSs) [1]. Unlike time-triggered control, which consistently and periodically updates control signals, these control types transmit sensor measurements to the controller across the communication network only when it is necessary. Various forms of event/self-triggered controllers have been examined, as outlined in the survey paper [2].

Early attempts applied the principles of input-to-state stability or \( L_2 \) gain performance in designing event/self-triggered control for linear systems [3]. More recently, some strategies have begun incorporating these control types into optimal control scenarios [4–7]. Additionally, recent years have seen considerable research relating to reachability and safety analysis [8].

A considerable number of previous studies involving the event/self-triggered control framework operated on the assumption that the dynamical model of the plant under control was known a priori. This suggests that when these designed controllers are applied to real-world systems, performance results are heavily reliant on the accuracy of the model. However, the true system’s physical dynamics can occasionally be complex and highly nonlinear (and thus unknown a priori). In response to this challenge, numerous studies have proposed an event/self-triggered control framework suitable for unknown transition dynamics; see [9–15]. For instance, Hashimoto et al. [9] offered an iterative method for learning both the system dynamics and the optimal policy based on training data, utilizing a Gaussian processes (GP) regression to learn the dynamics and solving a value iteration algorithm to obtain a self-triggered optimal policy. Furthermore, Umlauft and Hirche [15] explore a technique for designing an event-triggered controller for input-affine systems, in which communication time instants are determined by evaluating a Lyapunov function candidate. In addition, research into a model-free approach to designing event/self-triggered controllers based on a deep reinforcement learning framework has been conducted by Baumann et al. [11], Vamvoudakis et al. [14], and Wang et al. [10].

This paper aims to develop a self-triggered controller within an optimal control-based framework, where the unknown transition dynamics are learned using GP regression. To the best of our knowledge, only one prior study has achieved this objective [9]. However, we argue that this previous work has certain limitations. First, due to the implementation of value iteration via state-space discretization into grid points to approximate the optimal policy, the process generally demands extensive computational resources. Second, Hashimoto et al. [9]
considers a dynamical system represented by \( x_{k+1} = f(x_k, u_k) \) (where \( x_k \) is the state and \( u_k \) is the control input). In this context, the function \( f \) is learned based on training data while the self-triggered controller is operating. However, learning function \( f \) necessitates training data comprised of consecutive states and the control input (i.e. \( x_k, x_{k+1}, u_k \)), meaning that the training data is only available when the inter-event time step is 1. Therefore, all data (state/control inputs) must be discarded if the inter-event time step is larger than 1, rendering this method potentially unsuitable for learning dynamics while operating the self-triggered controller. Thus, the previous work may suffer from both extensive computational resource requirements and inefficient data usage for learning dynamics. To remedy the aforementioned issues, this paper presents a novel optimal control, learning-based approach to designing a self-triggered controller within an optimal control framework using GP regression.

The proposed approach offers several key contributions. First, to efficiently learn dynamics while considering subsequent design of the self-triggered controller, we opt not to learn dynamics as conventionally defined by ordinary differential or difference equations (a common practice in previous literature). Instead, we propose a lifted model in which inter-event time is treated as an additional control input, and it is then learned through GP regression. As will be elaborated upon in subsequent sections, this strategy enhances learning efficiency, since all state information received at the controller can be utilized as training data for learning the dynamics. Second, we propose a finite horizon optimal control problem, enabling us to penalize states between every adjacent triggering instant. For instance, this approach facilitates the optimization of an effective self-triggered controller capable of avoiding obstacle collisions, as will be demonstrated in the numerical simulation. Finally, by employing the lifted dynamics estimated from the training data, we demonstrate that the self-triggered controller can be optimized using a policy gradient algorithm. This algorithm represents a computationally efficient framework for policy derivation, leading to a substantial reduction in computational time compared to the previous work [9].

While our approach draws connections to several prior works [10–12, 14–18], it is important to note that our problem setup is considerably different from them in the following ways. Studies [10, 11, 14] investigate methods for learning event/self-triggered controllers using deep reinforcement learning. Our work diverges from these by providing a model-based solution, where the (lifted) dynamics are learned via GP regression and the optimal self-triggered controller is designed in line with this. Furthermore, our approach stands in contrast to [15], where an event-triggered controller is designed using a Lyapunov function candidate. Instead, we explore a method to design a self-triggered controller based on an optimal control problem using a policy gradient technique. Our approach also has connections with [18], where an optimal control policy is formulated via policy gradient for dynamics learned through GP regression. However, the proposed approach distinguishes itself from [18] in several ways. First, while [18] explores designing a control policy for periodic control, our research focuses on the design of a self-triggered controller that incorporates both control and communication policies. This is made possible through the introduction of the lifted model. Second, we introduce several modifications to the computational graph for policy evaluations and improvements. This arises from altering the cost to incorporate the inter-sample behavior of states while implementing a self-triggered controller. For further details, refer to Section 3.6.

2. Problem formulation

2.1. System description

This paper investigates a Networked Control System (NCS) illustrated in Figure 1. The NCS is composed of a plant and a pair of a controller and a learning agent, which is linked via a communication network. The primary role of the learning agent is to estimate the plant’s dynamics and design the optimal control and communication policies. The controller’s duty is to transmit control inputs to manage the plant, based on the policies established by the learning agent. For the scope of this research, we assume that the communication network operates without packet loss or network delays. This assumption implies that there is no delay between sensing measurements and applying control signals.

The dynamics of the plant are given by the following nonlinear systems:

\[
\dot{x}_t = f(x_t, u_t), \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0),
\]

for all \( t \in \mathbb{R}_{\geq 0} \), where \( x_t \in \mathbb{R}^{n_x} \) and \( u_t \in \mathcal{U} \) are the states and the control inputs at time \( t \) respectively, and
\( \mathcal{U} = [u_{\text{min}}, u_{\text{max}}] \subset \mathbb{R}^{n_u} \) is the set of control inputs. We assume that the initial state \( x_0 \) follows the Gaussian with a given mean \( \mu_0 \) and a covariance matrix \( \Sigma_0 \). Moreover, \( f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_x} \) is a function representing the transition dynamics and unknown apriori.

### 2.2. Overview of learning-based self-triggered control

Let \( t_0, t_1, t_2, \ldots \) with \( t_0 = 0 \) be the communication time instants when the plant transmits the state \( x_{t_n} \) to the controller, and let \( \tau_n = t_{n+1} - t_n \in \mathbb{R}_{>0}, n \in \mathbb{N} \) be the corresponding inter-event times. In this paper, we aim at designing a self-triggered controller so as to reduce the number of communication between the plant and the controller. The basic procedure of the self-triggered control is summarized as follows: for each \( t_n, n \in \mathbb{N} \),

(i) The state \( x_{t_n} \) is measured and transmitted to the controller;

(ii) Based on some policy \( \pi : \mathbb{R}^{n_x} \rightarrow \mathcal{U} \times \mathbb{R}_{>0} \), the controller computes the control input \( u_{t_n} \in \mathcal{U} \) and the inter-event time \( \tau_n \in \mathbb{R}_{>0} \), i.e. \( [u_{t_n}, \tau_n] = \pi(x_{t_n}) \);

(iii) The controller transmits \( \{u_{t_n}, \tau_n\} \) to the plant, and the plant applies \( u_{t_n} \) constantly until the next communication time, i.e. \( u_t = u_{t_n}, \forall t \in [t_n, t_{n+1}) \), where \( t_{n+1} = t_n + \tau_n \).

Due to the fact that the dynamical system (1) is unknown apriori, in this paper we employ a learning-based approach, in which the controller learns the dynamical system based on training data and adaptively updates the policy \( \pi \). A rough sketch of the learning-based self-triggered control is summarized as follows:

**[Step 1]** Using the current policy \( \pi \), implement the self-triggered control (i)–(iii) for a given time period \( T \in \mathbb{R}_{>0} \). While executing the self-triggered controller, the controller stores a set of new training data involving the information of states received from the plant, control inputs and inter-event times.

**[Step 2]** Using the training data, the controller learns the dynamical system and updates the policy \( \pi \). Then, go back to Step 1.

Our goal is to learn the optimal policy \( \pi \) that minimizes a prescribed cost function (defined later in this paper). Moreover, aiming at guaranteeing the positive inter-event times, we will ensure that the designed policy leads to satisfying \( \tau_n \geq \tau_{\text{min}}, \forall n \in \mathbb{N} \) for a given lower bound of the inter-event time \( \tau_{\text{min}} \in \mathbb{R}_{>0} \). A concrete procedure of the proposed approach is elaborated in the next section.

### 3. Proposed approach

#### 3.1. Motivation

A key challenge in learning-based self-triggered control given in the previous section is that the state measurements are transmitted to the controller only at the communication time instants. Thus, the controller is able to utilize the states that are received intermittently from the plant, i.e. \( x_{t_n}, n \in \mathbb{N} \), which indeed makes the learning of both the dynamics and the optimal policy a difficult task. As a naive approach, assuming that the lower bound of the inter-event time \( \tau_{\text{min}} \) is close to 0, one could often set the lowest inter-event time, say \( \tau_n = \tau_{\text{min}} \), so that the controller obtains the consecutive state measurements \( x_{t_n}, x_{t_{n+1}} \) and use them to approximately learn the function \( f(x_t, u_t) \), i.e. \( \dot{x}_{t_n} = f(x_{t_n}, u_{t_n}) \approx (x_{t_n} + \tau_n - x_{t_n})/\tau_n \). Hence, if the controller receives the consecutive states \( x_{t_n}, x_{t_{n+1}} \) with a small enough inter-event time, these states can be utilized as the training data to approximately learn \( f \). However, this approach is clearly inefficient, since the controller is able to learn the dynamical system only for the case when the inter-event time is small enough, i.e. if the inter-event time is selected large, we need to possibly throw away the data since \( (x_{t_n} + \tau_n - x_{t_n})/\tau_n \) is not accurate enough to estimate \( f(x_t, u_t) \). In addition to the above, even if \( f \) could be learned, deriving the optimal policy \( \pi \) that minimizes a given cost function is in general computationally hard based on the knowledge about \( f \). Indeed, the function \( f \) depends on the control input \( u \), but it does not depend on the inter-event time \( \tau \) (i.e. (1) fails to incorporate the information about \( \tau \)), although we need to optimize both \( u \) and \( \tau \) for each state \( x \). This implies that the standard policy gradient algorithm, which is known to be an efficient method to derive the optimal policy, cannot be directly applied based on the knowledge about \( f \).

#### 3.2. Lifting approach for learning-based self-triggered control

In order to efficiently and adaptively learn the dynamical system and the optimal policy while executing the self-triggered control, in this paper we propose to modify a function to be learned as follows. First, note that we have

\[
\begin{align*}
x_{t_{n+1}} = x_{t_n} + \int_{t_n}^{t_{n+1}} f(x_t, u_{t_n}) \, dt, \\
n_1 = n + 1
\end{align*}
\]

where we use the fact that the control input is constant for all \( t \in [t_n, t_{n+1}) \). Then, letting the function \( g \) be given
by \( g(x_{tn}, u_{tn}, \tau_n) = x_{tn} + \int_{t_n}^{t_{n+1}} f(x_t, u_t) \, dt \), we have
\[
x_{tn+1} = g(x_{tn}, v_n), \quad n \in \mathbb{N},
\]
(3)

where \( v_n = [u_{tn}^T, \tau_n]^T \) denotes the extended control input that incorporates both \( u_{tn} \) and \( \tau_n \). Note that the discrete-time system (3) is slightly different from the one that is widely used in the context of nonlinear sampled-data systems (e.g. Equation (5) of [19]) in the sense that the inter-event time \( \tau_n \) is regarded as a supplemental variable (input). Instead of learning \( f \), in this paper we propose to learn the lifted function \( g \) based on the training data. This approach is advantageous over learning \( f \) in the following sense. First, in contrast to the case of learning \( f \), the approach is advantageous over learning \( g \) based on the training data. This approach is advantageous over learning \( f \) in the following sense. First, in contrast to the case of learning \( f \), second, note that the inter-event time \( \tau_n \) is now explicitly given as one of the inputs in \( v_n \) and is incorporated in (3). As such, we can employ a policy gradient algorithm to compute the policy \( \pi \), so that the optimal policy can be derived in a computationally efficient way; for details, see Section 3.6.

### 3.3. Learning the lifted dynamics with Gaussian processes

In this paper, we employ the GP regression in order to learn the lifted dynamics (3). We independently predict each element of (3), which is denoted by \( x_{tn+1,j} = g_j(x_{tn}, v_n), \quad j = 1, 2, \ldots, n_x \), where \( x_{tn+1,j} \) and \( g_j \) are the \( j \)-th element of \( x_{tn+1} \) and \( g \), respectively. We denote by \( D_j = \{ \tilde{X}, Y_j \} \) the dataset to estimate \( g_j \), where
\[
\tilde{X} = \left[ \begin{array}{c} x_{tn+1,1}^x \, v_1^x \, \cdots \, x_{tn+1,n_x}^x \\ \vdots \quad \vdots \quad \cdots \quad \vdots \\ x_{tn+1,1}^\tau \, v_1^\tau \, \cdots \, x_{tn+1,n_x}^\tau \end{array} \right],
\]
(4)
\[
Y_j = [\Delta^*_n, \Delta^*_1, \ldots, \Delta^*_{D-1}]^T,
\]
(5)

with \( \Delta^*_n = x_{tn+1,n}^x - x_{tn,n}^x \), for \( n = 0, \ldots, D - 1 \) and \( D \) is the number of the training data. Then, given \( D_j \), an arbitrary state \( x_{tn} \in \mathbb{R}^{n_x} \) and an extended control input \( v_n \in \mathbb{R}^{n_u+1} \), we can predict \( x_{tn+1,j} \) by the Gaussian distribution as
\[
p(x_{tn+1,j} \mid x_{tn}, v_n, D_j) = \mathcal{N}(x_{tn+1,j} \mid \mu_{n+1,j}, \sigma_{n+1,j}).
\]
(6)

Here \( \mu_{n+1,j} = x_{tn,j} + \mathbb{E}_{g}[\Delta_{n,j}], \sigma_{n+1,j} = \text{var}_{g}[\Delta_{n,j}] \) with \( \Delta_{n,j} = x_{tn+1,j} - x_{tn,j} \) are the mean and the variance of the GP prediction, respectively, and these are given by
\[
\mathbb{E}_{g}[\Delta_{n,j}] = k_{x,j}^T \left( K_j + \sigma_{w,j}^2 I \right)^{-1} Y_j,
\]
(7)

\[
\text{var}_{g}[\Delta_{n,j}] = k_{x,j}^T \left( K_j + \sigma_{w,j}^2 I \right)^{-1} k_{x,j}.
\]
(8)

In these expressions, \( \tilde{x}_n = [x_{tn+1}^x, v_n^x]^T \), \( k_j(\cdot, \cdot) \) is a given kernel parameterized by the hyperparameter \( \theta_j \), \( k_{x,j} = k_j(\tilde{x}_n, \tilde{x}_n) \in \mathbb{R}^{D \times 1} \), and \( K_j \in \mathbb{R}^{D \times D} \) is the kernel matrix whose \( p \times q \)-element is defined as \( K_{j,pq} = k_j(\tilde{x}_p, \tilde{x}_q) \), where \( \tilde{x}_p = [x_{tn+1}^x, v_n^x]^T \). Moreover, \( \sigma_w = \text{diag} \left( \sigma_{w,1}, \ldots, \sigma_{w,n_x} \right) \) is a covariance of the Gaussian distributed white noise for the observation. After obtaining the predictions for all the elements of \( x_{tn+1} \), the whole predictive distribution of \( x_{tn+1} \) is \( p(x_{tn+1} \mid x_{tn}, v_n, (D_j)_{j=1}^n) = \mathcal{N}(x_{tn+1} \mid \mu_{n+1}, \Sigma_{n+1}) \), where
\[
\mu_{n+1} = \begin{bmatrix} \mu_{n+1,1} & \cdots & \mu_{n+1,n_x} \end{bmatrix}^T,
\]
(9)
\[
\Sigma_{n+1} = \text{diag}(\sigma_{n+1,1}, \ldots, \sigma_{n+1,n_x}).
\]
(10)

In other words, the obtained model provides, for given \( x_{tn} \) and \( v_n = [u_{tn}^T, \tau_n]^T \), a prediction of the state at the next communication time, i.e. \( x_{tn+1} \) with \( t_{n+1} = t_n + \tau_n \). Note that, by learning this model and as shown in (4) and (5), we can utilize all the information of the states that are received from the plant, the control inputs, and the inter-event time as the training data to learn the dynamics.

### 3.4. Cost function to be minimized

Let us now define the cost function to be minimized. Again, note that the model obtained in the previous section provides, for given \( x_{tn} \) and \( v_n = [u_{tn}^T, \tau_n]^T \), a prediction of the state at the next communication time \( x_{tn+1} \),
\[
J(\pi_\psi) = \sum_{n=0}^{N-1} \mathbb{E}_{x_{tn}} \left[ c(x_{tn}, \tau_n) \right], \quad x_{t_0} \sim \mathcal{N}(\mu_0, \Sigma_0),
\]
(11)

where \( \pi_\psi : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_x+1} \) denotes a policy parameterized by \( \psi \), and \( \mathbb{E}_{x_{tn}}[\cdot] \) denotes the expectation with respect to \( x_{tn} \) conditioned on the policy \( \pi_\psi \), and \( c : \mathbb{R}^{n_x} \times \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+ \) denotes a given stage cost. (11) implies that the stage cost is added at the communication time instants \( t_n, n = 0, \ldots, N - 1 \), which is in accordance with the transitions for the prediction model given in Section 3.3. Defining the cost function as above allows us to directly apply the policy gradient algorithm (see, e.g., [18]) to optimize \( \pi_\psi \). However, this cost function has a crucial drawback that only triggered instants are considered, which means what happened between any adjacent triggered instants will not be taken into account. For instance, it could happen that the optimized policy makes the resulting state trajectory collide with an obstacle between the triggering instances; for details,
see the simulation result in Section 4. To overcome this shortcoming, we propose to modify the cost function as follows: given $M, N \in \mathbb{N}_0$,

$$J(\pi_\psi) = \sum_{m=0}^{N-1} \mathbb{E}_{x^{t_n}_{m}} \left[ \lambda c_1(t_n) + \sum_{m=0}^{M-1} c_2(x^{t_n}_{m}) \right],$$

$$x^{t_0}_{0} \sim \mathcal{N}(\mu_0, \Sigma_0) \quad (12)$$

where $t_{n,m} = \alpha_m t_{n+1} + (1 - \alpha_m) t_n$ with $\alpha_m = \frac{m}{M}$ for all $m = 0, \ldots, M - 1$. Moreover, $x^{t_n}_{m}$, $m = 0, \ldots, M - 1$, $n = 0, \ldots, N - 1$ is the state of the system by applying the control input $u^{t_n}_{m}$ constantly over the time length of $\alpha_m t_n$ from $x^{t_n}_{0}$, i.e., $x^{t_n}_{m} = g(x^{t_n}_{0}, v^{t_n}_{m})$ with $v^{t_n}_{m} = [u^{t_n}_{m}, \alpha_m t_n]^{\top}$ and $[u^{t_n}_{m}, t_n]^{\top} = \pi_\psi(x^{t_n}_{0})$. Moreover, $c_1$ and $c_2$ represent the stage cost for the communication and the state, respectively, and $\lambda > 0$ is the weight associated with $c_1$. Examples of these cost functions include polynomials and mixtures of Gaussians, so that their expectations can be computed analytically (see Section 3.5). Intuitively, the cost related with control performance linearly interpolates $M - 1$ points between every adjacent triggering instants (i.e. $x^{t_n}_{0}, \ldots, x^{t_n}_{M-1}$). As such, we can take the behavior of the states between every adjacent triggering instants into account.

### 3.5. Long-term prediction and policy evaluation

To evaluate and minimize $J$ in (12), we need to predict the trajectory distribution

$$p(x^{t_n}_{m}), \quad n = 0, \ldots, N - 1, m = 0, \ldots, M - 1 \quad (13)$$

conditioned on the policy $\pi_\psi$. The calculation of (13) requires mapping a probability distribution through the GP model, which is mathematically intractable. Hence, we utilize a moment matching technique (see, e.g. [18]) to approximate (13) by the Gaussians. While a basic procedure follows the approach given in [18], some modifications are necessary in terms of how to cascade the computations for $p(x^{t_n}_{m})$ (as detailed below). Here, we omit technical details of the moment matching technique and provide only its summary and how the computational procedure is different from [18].

Suppose $p(x^{t_{n,0}}) = p(x^{t_0})$ is approximated by a Gaussian distribution. Then, a Gaussian approximation for the distribution $p(v_n) = p(\pi_\psi(x^{t_0}_{n,0}))$ can be computed, and then we can analytically compute Gaussian approximation for the joint distribution $p(x^{t_{n,0}}_{n,0}, v_n) = p(x^{t_{n,0}}_{n,0}, \pi_\psi(x^{t_{n,0}}_{n,0}))$ (for details, see Section 5.5 in [18]). Since $p(v_n)$ is Gaussian and $v^{t_{n,0}}_{m} = [u^{t_{n,0}}_{m}, \alpha_m t_{n,0}]^{\top}$ is obtained from $v_n$ by the following linear transformation

$$v^{t_{n,0}}_{m} = \begin{bmatrix} I_{n_u} \times n_u & 0_{n_u \times 1} \\ 0_{1 \times n_u} & \alpha_m \end{bmatrix} v_n, \quad (14)$$

it follows that $p(v^{t_{n,0}}_{m})$ is also a Gaussian. Since $p(x^{t_{n,0}}_{n,0})$ is Gaussian, we can analytically compute Gaussian approximation for the joint distribution $p(x^{t_{n,0}}_{n,0}, v^{t_{n,0}}_{m})$, where $x^{t_{n,0}}_{n,0} = [x^{t_{n,0}}_{n,0}, v^{t_{n,0}}_{m}]^{\top}$. Finally, the distribution of $x^{t_{n,0}}_{n,0}$ is computed from $p(x^{t_{n,0}}_{0,0}, v^{t_{n,0}}_{m})$ as follows:

$$p(x^{t_{n,0}}_{n,0}) = \int p(x^{t_{n,0}}_{n,0} | \tilde{x}^{t_{n,0}}_{n,0}) p(\tilde{x}^{t_{n,0}}_{n,0}) \, d\tilde{x}^{t_{n,0}}_{n,0}. \quad (15)$$

From the GP model given in Section 3.3, it follows that $p(x^{t_{n,0}}_{n,0} | \tilde{x}^{t_{n,0}}_{n,0})$ is Gaussian. Therefore, by analytically computing the mean and covariance matrix of the right hand side of (15), we can approximate the distribution of $x^{t_{n,0}}_{n,0}$ as a Gaussian distribution, i.e. $p(x^{t_{n,0}}_{n,0}) \approx \mathcal{N}(\mu_{n,0}, \Sigma_{n,0})$. Similarly, we have

$$p(x^{t_{n+1,0}}_{n+1,0}) = \int p(x^{t_{n+1,0}}_{n+1,0} | \tilde{x}^{t_{n+1,0}}_{n+1,0}) p(\tilde{x}^{t_{n+1,0}}_{n+1,0}) \, d\tilde{x}^{t_{n+1,0}}_{n+1,0}, \quad (16)$$

where $\tilde{x}^{t_{n,0}}_{n,0} = [x^{t_{n,0}}_{n,0}, v^{t_{n,0}}_{m}]^{\top}$. From the GP model given in Section 3.3, it follows that $p(x^{t_{n+1,0}}_{n+1,0} | \tilde{x}^{t_{n,0}}_{n,0})$ is Gaussian. Hence, we can approximate $p(x^{t_{n+1,0}}_{n+1,0})$ by the Gaussian distribution by analytically computing the mean and the covariance in the right hand side of (16). Since we can iteratively cascade the above procedure, all of the required state distribution (13) can be obtained with this scheme. The computational graph that summarizes the above procedure is shown in Figure 2.

For comparisons, in Figure 3 we illustrate the computational graph originally proposed in [18]. Note that [18] considers a periodic control execution (i.e. the time interval between the time steps $t$ and $t + 1$ in the above is always the same) and thus it aims at learning only a control policy. As shown in Figure 3, Deisenroth et al. [18] considers computing the state distribution iteratively based on the latest distributions of the state and the control input (i.e. $p(x_t)$ is computed from $p(x_{t-1})$ and $p(u_{t-1}))$. On the other hand, in our approach, the state distribution at the communication time instant is computed based on the distributions of the state and the extended control input at the latest communication time.

![Figure 2](image.png)
(i.e. $p(x_{t+n+1,0})$ is computed from $p(x_{t,n,0})$ and $p(v_n)$). Moreover, all the state distributions between the triggering instants, i.e. $p(x_{t,n,m})$, $m = 1, \ldots, M - 1$ are computed based on the distributions of the state at the latest communication time $p(x_{t,n,0})$ and the extended control input at $t_{n,m}$, i.e. $p(v_{n,m})$ with $v_{n,m} = [\mu_m \alpha_{t_{n,m}}]^{\top}$.

Finally, to evaluate the expected total cost $J$ in (12), it remains to compute the expected values

$$E_{x_{t,n,0}}[c_1(\tau_n)] = \int c_1(\tau_n)N(\mu_{n,0}, \Sigma_{n,0}) \, dx_{t,n,0}$$

$$E_{x_{t,n,m}}[c_2(x_{t,n,m})] = \int c_2(x_{t,n,m})N(\mu_{n,m}, \Sigma_{n,m}) \, dx_{t,n,m}$$

Since $p(v_n) = p(\pi \psi (x_{t,n,0}))$ is Gaussian, $p(\tau_n)$ follows also a Gaussian. Hence, if the cost $c_1$ and $c_2$ are given by, e.g. polynomials, mixtures of Gaussians, we can analytically compute the above expectations.

### 3.6. Gradient based policy improvement

To find policy parameters $\psi$ minimizing $J(\pi \psi)$ in (12), we use gradient information $\delta J(\pi \psi) / \delta \psi$. As with the policy evaluation given in the previous section, we need to provide some modifications from [18], since the computational graph for (12) is different from that of [18]. As a prerequisite, assume that the moments of the control distribution $\mu^v$ and $\Sigma^v$ can be computed analytically and are differentiable with respect to the policy parameters $\psi$. We obtain the gradient $\delta J / \delta \psi$ by repeatedly applying the chain rule. First, we use the notation:

$$\zeta_n = \lambda \eta_n + \sum_{m=0}^{M-1} \epsilon_{n,m},$$

$$\eta_n = E_{x_{t,n,0}}[c_1(\tau_n)],$$

$$\epsilon_{n,m} = E_{x_{t,n,m}}[c_2(x_{t,n,m})].$$

Then the following equations is obtained from (12).

$$\frac{dJ(\pi \psi)}{d\psi} = \sum_{n=0}^{N-1} \frac{d\zeta_n}{d\psi},$$

We first discuss the term $d\epsilon_{n,m}/d\psi$ by adopting the shorthand notation $d\epsilon_{n,m}/d\psi = \{d\epsilon_{n,m}/d\mu_{n,m}, d\epsilon_{n,m}/d\Sigma_{n,m}\}$ for taking the derivative of $\epsilon_{n,m}$ with respect to both mean and covariance of $p(x_{t,n,m}) = N(\mu_{n,m}, \Sigma_{n,m})$ such that the following equation could be derived.

$$\frac{d\epsilon_{n,m}}{d\psi} = \frac{d\epsilon_{n,m}}{d\mu_{n,m}} \frac{d\mu_{n,m}}{d\psi} + \frac{d\epsilon_{n,m}}{d\Sigma_{n,m}} \frac{d\Sigma_{n,m}}{d\psi}.$$  \hspace{1cm} (24)

Next, the predicted mean $\mu_{n,m}$ and covariance $\Sigma_{n,m}$ depend on the moments of $p(x_{t,n,0})$ (see the computational graph in Figure 2) and the controller parameters $\psi$. By applying the chain rule to $dp(x_{t,n,0})/d\psi$ in (20), we obtain

$$\frac{dp(x_{t,n,0})}{d\psi} = \frac{\partial p(x_{t,n,0})}{\partial \mu_{n,0}} \frac{d\mu_{n,0}}{d\psi} + \frac{\partial p(x_{t,n,0})}{\partial \Sigma_{n,0}} \frac{d\Sigma_{n,0}}{d\psi}.$$  \hspace{1cm} (25)

From here onward, we focus on $d\mu_{n,m}/d\psi$, see (24), because the calculation of $d\Sigma_{n,m}/d\psi$ is similar. For $d\mu_{n,m}/d\psi$, we calculate the derivative

$$\frac{d\mu_{n,m}}{d\psi} = \frac{\partial \mu_{n,m}}{\partial \mu_{n,0}} \frac{d\mu_{n,0}}{d\psi} + \frac{\partial \mu_{n,m}}{\partial \Sigma_{n,0}} \frac{d\Sigma_{n,0}}{d\psi} + \frac{\partial \mu_{n,m}}{\partial \Sigma_{n,m}} \frac{d\Sigma_{n,m}}{d\psi}.$$  \hspace{1cm} (26)

And $dp(x_{t,n,0})/d\psi$ in (25) is known from communication time step $n-1$ since $x_{t,n,0}$ can be viewed as $x_{t,n-1,M}$. To compute $d\mu_{n,m}/d\psi$, it remains to compute

$$\frac{\partial \mu_{n,m}}{\partial \psi} = \frac{\partial \mu_{n,m}}{\partial \mu_{n,0}} \frac{d\mu_{n,0}}{d\psi} + \frac{\partial \mu_{n,m}}{\partial \Sigma_{n,0}} \frac{d\Sigma_{n,0}}{d\psi} + \frac{\partial \mu_{n,m}}{\partial \Sigma_{n,m}} \frac{d\Sigma_{n,m}}{d\psi}.$$  \hspace{1cm} (27)

where $\nu_{n,m} \sim N(\mu_{n,m}^v, \Sigma_{n,m}^v)$. The partial derivatives of $\mu_{n,m}^v$ and $\Sigma_{n,m}^v$, i.e. the mean and covariance of $p(\nu_{n,m})$, used in (27) depend on the policy representation. To evaluate $dJ(\pi \psi)/d\psi$, we also need the information of $dp(\tau_n)/d\psi$ in (20), which could be computed by

$$\frac{dp(\tau_n)}{d\psi} = \frac{dp(\tau_n)}{dp(\nu_n)} \frac{d\nu_n}{d\psi} = [0_{1 \times n_u \ 1}] \frac{dp(\nu_n)}{d\psi}.$$
\[
\frac{dp(v_n)}{d\psi} = \frac{dp(v_n)}{dp(x_{i,n\psi})} \frac{dp(x_{i,n\psi})}{d\psi} + \frac{dp(v_n)}{d\psi},
\]  
(28)

where \(dp(x_{i,n\psi})/d\psi\) and \(dp(v_n)/d\psi\) has been discussed in the previous content, and \(\frac{dp(v_n)}{dp(x_{i,n\psi})}\) depends on the policy representation. The individual partial derivatives in (20) to (28) need to apply chain rule to moment matching (for their detailed computations, see the appendix of [18]).

### 3.7. Guaranteeing positive inter-event times and control constraint satisfaction via parametrization

One of the most desirable properties in self-triggered control is to guarantee a positive inter-event time, i.e. there exists a \(\tau_{\text{min}} > 0\) such that \(\tau_n \geq \tau_{\text{min}}\) for all \(n = 0, 1, \ldots\) (see, e.g. [1]). In addition, we constrain that the control input must belong to \(\mathcal{U} = [u_{\text{min}}, u_{\text{max}}]\), i.e. \(u_{\text{in}} \in \mathcal{U}\) for all \(n = 0, 1, \ldots\). In this section, we remark that these properties can indeed be satisfied by a parametrization technique.

Let the policy \(\pi_\psi\) be decomposed as \(\pi_\psi(x) = \{\pi_\psi_i(x), \pi_\psi_u(x)\}\) where \(\pi_\psi_i: \mathbb{R}^{nx} \rightarrow \mathbb{R}_{\geq 0}\) denotes a policy to compute the inter-event time \(\tau\), and \(\pi_\psi_u(x): \mathbb{R}^{nx} \rightarrow \mathbb{R}^{nu}\) denotes a policy to compute the control input \(u\). Now, let us parametrize the policy \(\pi_\psi\) as follows:

\[
\pi_{\psi_i}(x) = \frac{\tau_{\text{max}} + \tau_{\text{min}}}{2} + \frac{\tau_{\text{max}} - \tau_{\text{min}}}{2} \sigma_i(\tilde{\pi}_{\psi_i}(x)),
\]

(29)

for given \(\tau_{\text{min}}, \tau_{\text{max}} > 0\) with \(\tau_{\text{max}} > \tau_{\text{min}}\) and \(\sigma_i(\cdot)\) is a given squashing function that satisfies \(\sigma_i(z) \in [-1, 1]\) for all \(z \in \mathbb{R}\) (see Section 5.1 in [18]), and \(\tilde{\pi}_{\psi_i}: \mathbb{R}^{nx} \rightarrow \mathbb{R}\) denotes a nonlinear policy, which is implemented using a Deterministic Gaussian Process (functionally equivalent to a regularized RBF network, see Section 5.3 in [18]). Parametrizing \(\pi_{\psi_i}\) as above leads to ensuring that \(\pi_{\psi_i}(x) \in [\tau_{\text{min}}, \tau_{\text{max}}]\) for all \(x \in \mathbb{R}^{nx}\), and thus the optimized the policy ensures a positive inter-event time. Similarly, we can ensure a constraint satisfaction of the control input by parametrizing \(\pi_{\psi_u}(x)\) as \(\pi_{\psi_u}(x) = \frac{u_{\text{max}} + u_{\text{min}}}{2} + \frac{u_{\text{max}} - u_{\text{min}}}{2} \sigma_u(\tilde{\pi}_{\psi_u}(x))\) for all \(i = 1, \ldots, nu\), where \(u_{\text{max}}\) and \(u_{\text{min}}\) denote the \(i\)-th element of \(u_{\text{max}}\) and \(u_{\text{min}}\) respectively, and \(\tilde{\pi}_{\psi_u}\) denotes the \(i\)-th element of \(\pi_{\psi_u}\) (i.e. it denotes a policy to compute the \(i\)-th element of \(u\)), \(\sigma_u(\cdot) \in [-1, 1]\) is a given squashing function, and \(\tilde{\pi}_{\psi_u}: \mathbb{R}^{nx} \rightarrow \mathbb{R}\) denotes a preliminary policy with an unconstrained amplitude. Parametrizing \(\pi_{\psi_u}\) as above leads to ensuring that \(\pi_{\psi_u}(x) \in [u_{\text{min}}, u_{\text{max}}]\) for all \(x \in \mathbb{R}^{nx}\), guaranteeing the control constraint satisfaction.

### Algorithm 1 Learning self-triggered controllers

1. **Input**: Characterization of the stage cost functions \(c_1, c_2\), prediction horizon \(N\), Gaussian distribution of the initial state \(\mathcal{N}(\mu_0, \Sigma_0)\), initial policy parameter \(\psi\), step-size of the gradient update \(\alpha > 0\);
2. **Output**: the optimal self-triggered control policy \(\pi_\psi\);
3. Starting from \(x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)\), apply the self-triggered controller for a given time period \(T\), in which the controller generates a random extended control input \(v_n\) for each communication time \(t_n\). Then, record the training data (Section 3.3);
4. **repeat**
5. Using the recorded training data, estimate the lifted dynamics by the GP regression (Section 3.3);
6. **repeat**
7. Use the current policy \(\pi_\psi\) to predict future trajectory distribution and calculate the expected total cost (12) (Section 3.5);
8. Compute gradient information \(df/d\psi\) using (20) to (28);
9. Update the policy parameter as \(\psi \leftarrow \psi - \alpha df/d\psi\);
10. until \(\psi\) converges
11. Using the improved policy \(\pi_\psi\), apply the self-triggered controller for a given time period \(T\) and then record the training data;
12. until task learned

### 3.8. Overall algorithm

Let us now introduce an overall implementation algorithm that jointly learns the dynamics of the plant and the self-triggered controller based on a model-based reinforcement learning framework. The overall algorithm is shown in Algorithm 1. For the initial iteration, the controller generates random control signals (in a self-triggered manner) and apply to the system to record the training data in the form of \(\{(x_{tn}^i, v_n^i)^T, x_{tn+1}^i\}\) according to Section 3.3 (line 3). Then, using the recorded input and output data \(D\), the controller learns the lifted dynamics \(x_{tn+1} \equiv g(x_{tn}, v_n)\) using GP regression (line 5). Next, the algorithm utilizes the current policy and the learned dynamics to predict future trajectory distribution \(p(x_{tn+1})\) from an initial state distribution \(p(x_0)\), and then calculates the expected total cost \(J\) (line 7). Thereafter, the gradient information \(df/d\psi\) is computed and applied to minimize \(J\), yielding an improved policy \(\pi_\psi\). Then execute the control system based on the improved policy, and gather data \(\{(x_{tn}^i, v_n^i)^T, x_{tn+1}^i\}\) during execution. The gathered data is appended to the total training data, and back to line 5, the dynamical model \(x_{tn+1} \equiv g(x_{tn}, v_n)\) is re-trained using the additional training data. Finally, the loop ends and outputs the optimal self-triggered control policy \(\pi_\psi\) when a satisfactory performance is reached.
4. Simulation

4.1. Inverted pendulum

To make comparisons with the previous work [9], we first conducted a simple experiment of an inverted pendulum, whose dynamics are given of the form:

\[
\ddot{\phi} = \frac{u - b\dot{\phi} - \frac{1}{2}mlg \sin \phi}{\frac{1}{4}ml^2 + I},
\]  

(30)

where \(\phi\) is the pendulum angle measured anti-clockwise from the hanging down position, \(g\) is the acceleration of gravity, \(b\) is a friction coefficient and \(I = \frac{1}{12}ml^2\) is the moment of inertia of a pendulum around the pendulum midpoint. In the experiment, we set \(m = 1\) kg, \(l = 1\) m, \(b = 0.01\) and \(g = 9.82\) m/s\(^2\). We then define the system state as \(x_n = [\phi_{tn}, \dot{\phi}_{tn}]^\top\), and the extended control input as \(v_n = [u_{tn}, \tau_{tn}]^\top\). The cost functions are given by \(c_1(\tau) = \tau_{\text{max}} - \tau\), and \(c_2(x) = \exp(-\frac{1}{2}(\text{Tri}(\phi) - \text{Tri}(\pi))^\top Q(\text{Tri}(\phi) - \text{Tri}(\pi)))\), where \(\text{Tri}(\cdot)\) is defined as \(\text{Tri}(\beta) = [\sin \beta, \cos \beta]^\top\) for \(\beta \in \mathbb{R}\). Moreover, we set \(Q = \text{diag}(4, 4), \tau_{\text{min}} = 0.02, \tau_{\text{max}} = 0.6\) and \(M = 1\). The simulation result is given in Figures 4 and 5. These results show that the proposed algorithm can learn an effective controller to stabilize the system towards the inverted position \(\phi = -\pi\) (red dotted line). We can also see from Figure 5 that the inter-event time tends to be larger as \(\lambda\) is selected larger, which is because we penalize more for the communication cost. In addition, it can be seen from Figure 5 that the inter-event time tends to be constant and small as the time evolves (for all cases of \(\lambda\)). This is because the system is unstable around the origin and the control input should be frequently updated so as to keep the state around the origin.

For comparisons, we also conducted the experiment using [9], in which (30) is approximated by the discrete-time system under the time period 0.02. Table 1 shows the number of episodes and total execution time required to learn a controller that can stabilize the system to \(\phi = -\pi\) within an error range 1%. The table shows that the proposed approach achieves a significant reduction of the execution time to learn the controller, which is due to the fact that the previous approach [9] requires state-space discretization to solve the value iteration algorithm. Moreover, the proposed approach is more efficient, in the sense that it requires smaller episodes to learn the controller than [9]. This is because [9] is able to learn the dynamics only when the smallest inter-event time is selected, while the proposed approach can utilize all the data received at the controller to learn the dynamics during execution of the self-triggered controller.

4.2. Vehicle navigation with collision avoidance

To test the benefits of incorporating the cost between the adjacent triggering instants as explained in Section 3.4, we next consider the following vehicle dynamical model:

\[
\dot{x}_1 = u_1 \cos(\theta), \quad \dot{x}_2 = u_1 \sin(\theta), \quad \dot{\theta} = u_2,
\]  

(31)

where \([x_1, x_2]^\top \in \mathbb{R}^2\) is the two-dimensional coordinate of the vehicle, \(\theta \in [0, 2\pi)\) is the angle from the \(x_1\)-axis to the direction of the vehicle, \(u_1\) is the velocity of the vehicle, and \(u_2 \in [\omega_{\text{min}}, \omega_{\text{max}}]\) is the angular velocity. We then define the system state as \(x_n = [x_{1,tn}, x_{2,tn}, \theta_{tn}]^\top\) and the system input as \(v_n = [u_{1,tn}, u_{2,tn}, \tau_{tn}]^\top\). The control goal is to drive the vehicle to a target position in a certain map where obstacles are placed, and it is assumed that the map and obstacle information is known. In addition, the vehicle is set to start from an initial zone and try to reach the target position. The stage cost \(c_2\) is defined as

\[
c_2(x) = c_{ig}(x) + c_{ob}(x),
\]  

(32)

\[
c_{ig}(x) = -K_{ig} \exp\left(-\frac{1}{2}(x - x_{ig})^\top Q(x - x_{ig})\right),
\]  

\[
c_{ob}(x) = \exp(-\frac{1}{2}(x - x_{ob})^\top Q(x - x_{ob}))
\]
Figure 6. The contour map of $c_2(x)$, where $c_2(x)$ has minimal value at $x_{tg}$.

\[ c_{ob}(x) = K_{ob} \sum_{ob \in OB} \exp\left(-\frac{1}{2}(x - x_{ob})^T Q_{ob}(x - x_{ob})\right), \]

where $K_{ob} > K_{tg} > 0$, $Q > 0$ is the weight matrix of the state $x$, $x_{tg} \in \mathbb{R}^3$ is the target state, $OB$ is the collection of all obstacles, and $x_{ob} \in \mathbb{R}^3$, $Q_{ob} > 0$ define the position and shape of obstacles respectively. In this simulation, we set $K_{tg} = 1$, $K_{ob} = 50$, $x_{tg} = [5, 5, 0]^T$, and $Q = \text{diag}(0.04, 0.04, 0)$. The contour map of $c_2(x)$ is shown in the Figure 6. As shown in the figure, we assume that there exist 5 obstacles in the considered region. Moreover, the stage cost $c_1$ is given by $c_1(\tau) = \tau_{\max} - \tau$ with $\tau_{\max} = 0.8$, and the minimum inter-event time is set to $\tau_{\min} = 0.02$.

Performance of the controller learned from Algorithm 1 is given in Figures 7 and 8. It can be seen that when $M = 1$, the vehicle directly goes through an obstacle (i.e. it fails to avoid an obstacle), which is attributed to the fact that when $M = 1$, the expected total cost $J$ degenerates to (11) and, as discussed in Section 3.4, it neglects state trajectories between any two adjacent triggering instants. On the other hand, it is shown from Figure 7 that the resulting trajectory does not collide with any obstacles, which is due to that the cost function is now able to incorporate the cost for states between any two adjacent triggering instants. Figure 8 describes the learning process, the learned system dynamics initially learns from random data, and at this time though the full dynamics has not been learned, it is enough to make the vehicle move roughly towards the target, then some unknown dynamics is detected, thus making the vehicle move more accurately. As shown in Figure 8, in contrast to Figure 7, the resulting trajectory does not collide with any obstacles. Figure 9 depicts how inter-event times change with $\lambda = 0.01, 0.03, 0$. It can be shown that regardless of $\lambda$, the learned controller tends to choose large $\tau_n$ at the beginning, which is mainly due to the fact that the total cost in (12) is defined by summing the
Figure 9. Inter-event time $\tau_n$ against time with $\lambda = 0.01, 0.1, 0$ when episode = 5.

Table 2. Number of training dataset and execution time (in sec) against the number of episodes.

| Episode | 1   | 3   | 5   | 8   | 12  |
|---------|-----|-----|-----|-----|-----|
| Number of data | 30  | 50  | 70  | 100 | 140 |
| Execution time (in sec) | 87  | 78  | 73  | 81  | 93  |

stage costs only at the triggered instants and their interpolations. Reducing the number of triggering leads to the reduction of the total cost, and therefore, minimizing (12) leads to communication reduction even for the case $\lambda = 0$ at the very beginning. However, when the vehicle is about to arrive at the target point, $\tau_n$ falls quickly for the case $\lambda = 0$, and this is because the learned controller chooses not to tune the control inputs $u_1$ and $u_2$ but the inter-event time $\tau_n$ to minimize (12). Using a slightly larger $\lambda$ helps solve this problem; if $\lambda = 0.01, 0.03$, the inter-event time becomes small around $n = 4$, which may be because the vehicle gets close to the obstacle and tries to avoid it. Thereafter, the inter-event time is selected larger since the vehicle is able to move with sufficient safe margin to the obstacles.

Finally, we perform a complexity analysis of the proposed algorithm. The main complexity of the algorithm focuses on the policy improvement, namely lines 6–10 in Algorithm 1. We illustrate in Table 2 the number of the collected training data for each episode and the corresponding execution time (for running lines 6–10) against the number of episodes. This table shows that the execution time decreases for the first few episodes, which is due to that the policy gets close to the (local) optimum and so it does not require many iterations to improve the policy. On the other hand, the execution time tends to increase after a certain episode, which is due to that, even though the policy is mostly converged, it takes longer time to improve the policy as the number of training data increases.

5. Conclusion and future work

In this paper, we studied the self-triggered control for NCSs with unknown transition dynamics. To this end, we lifted the original continuous dynamics to a novel discrete model by taking time as an input and used the GPR to learn the lifted dynamics of the plant. We formulated an optimal control problem where both the cost for the control performance and the communication cost are taken into account. Then, we illustrated that the minimization of the cost will produce an optimal self-triggered controller. In the simulation, detailed analysis of the simulation is given and shows that the proposed approach is effective and enjoys a low computational complexity with respect to the previous approach. On the other hand, the proposed approach takes still tens-to-hundreds of seconds in performing the policy improvement. Our future work is therefore to incorporate some sophisticated learning algorithms (e.g. sparse/recursive GP), aiming to further improve the computational complexity.

Note

1. Here, $\tau_{\max}$ could be selected arbitrary large so as to lengthen the inter-event time.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work is partially supported by Japan Science and Technology Agency (JST) CREST [grant number JPMJCR2012], Japan Science and Technology Agency (JST) ACT-X [grant number JPMJAX23CK], and Japan Society for the Promotion of Science (JSPS) KAKENHI [grant numbers 21K14184, 23KJ1451, and 22KK0155]; Japan Science and Technology Agency [grant number JPMJMS2281].
Notes on contributors

Wang Zhijun received the M.E degree from Osaka University, Japan in 2023. His research interest include model predictive control, networked control systems and machine learning.

Kazumune Hashimoto received the first M.E. degree from the KTH Royal Institute of Technology, Sweden, in 2014, and the second M.E. degree and the Ph.D. degree from Keio University, Japan, in 2015 and 2018, respectively. From 2018 to 2020, he was a Postdoctoral Researcher with the KTH Royal Institute of Technology, Sweden, and Osaka University, Japan. Currently, he is a lecturer at Osaka University, Japan. His research interests include model predictive control, networked control systems, and formal methods for control theory.

Wataru Hashimoto received the M.E degree from Osaka University, Japan in 2021. Currently, his is a PhD student at Osaka University, Japan. His research interest include model predictive control and machine learning.

Shigemasa Takai received the B.E.and M.E. degrees from Kobe University, Kobe, Japan, in 1989 and 1991, respectively, and the Ph.D. degree from Osaka University, Suita, Japan, in 1995. From 1992 to 1998, he was a Research Associate with Osaka University. In 1998, he joined Wakayama University, Wakayama, Japan, as a Lecturer, and became an Associate Professor in 1999. From 2004 to 2009, he was an Associate Professor with Kyoto Institute of Technology, Kyoto, Japan. Since 2009, he has been a Professor with Osaka University, Suita, Japan. His research interests include supervisory control and fault diagnosis of discrete event systems.

References

[1] Heemels W, Johansson K, Tabuada P. An introduction to event-triggered and self-triggered control. In: Proceedings of the 51st IEEE Conference on Decision and Control (IEEE CDC); 2012. p. 3270–3285.
[2] Peng C, Li F. A survey on recent advances in event-triggered communication and control. Inf Sci. 2018;457:113–125. doi: 10.1016/j.ins.2018.04.055
[3] Mazo Jr M, Anta A, Tabuada P. An ISS self-triggered implementation of linear controllers. Automatica. 2010;46(8):1310–1314. doi: 10.1016/j.automatica.2010.05.009
[4] Gommans T, Antunes D, Donkers T, et al. Self-triggered linear quadratic control. Automatica. 2014;50(4):1279–1287. doi: 10.1016/j.automatica.2014.02.030
[5] Hashimoto K, Adachi S, Dimarogonas DV. Event-triggered intermittent sampling for nonlinear model predictive control. Automatica. 2017;81:148–155. doi: 10.1016/j.automatica.2017.03.028
[6] Luo B, Yang Y, Liu D, et al. Event-triggered optimal control with performance guarantees using adaptive dynamic programming. IEEE Trans Neural Netw Learn Syst. 2019;31(1):76–88. doi: 10.1109/TNNLS.2019.290385
[7] Lapandić D, Persson L, Dimarogonas DV, et al. Aperiodic communication for mpc in autonomous cooperative landing. IFAC-Pap. 2021;54(6):113–118. 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021.
[8] Hashimoto K, Saoud A, Kishida M, et al. A symbolic approach to the self-triggered design for networked control systems. IEEE Control Syst Lett. 2019;3(4):1050–1055. doi: 10.1109/LCSYS.7782633
[9] Hashimoto K, Yoshimura Y, Ushio T. Learning self-triggered controllers with Gaussian processes. IEEE Trans Cybern. 2020;51(12):6294–6304. doi: 10.1109/TCSY.2020.2980048
[10] Wang R, Takeuchi I, Kashima K. Deep reinforcement learning for continuous-time self-triggered control. IFAC-Pap. 2021;54(1):203–208.
[11] Baumann D, Zhu JJ, Martius G, et al. Deep reinforcement learning for event-triggered control. In: Proceedings of 57th IEEE Conference on Decision and Control (IEEE CDC); 2018. p. 943–950.
[12] Wang J, Zhang H, Ma K, et al. Neural adaptive self-triggered control for uncertain nonlinear systems with input hysteresis. IEEE Trans Neural Netw Learn Syst. 2021;33(11):6206–6214. doi: 10.1109/TNNLS.2021.3072784
[13] Sahoo A, Xu H, Jagannathan S. Neural network-based event-triggered state feedback control of nonlinear continuous-time systems. IEEE Trans Neural Netw Learn Syst. 2016;27(3):497–509. doi: 10.1109/TNNLS.2015.2416259
[14] Vamvoudakis KG, Mojoodi A, Ferraz H. Event-triggered optimal tracking control of nonlinear systems. Int J Robust Nonlinear Control. 2017;27(4):598–619. doi: 10.1002/rnc.v27.4
[15] Umlauf J, Hirche S. Feedback linearization based on Gaussian processes with event-triggered online learning. IEEE Trans Automat Contr. 2020;65(10):4154–4169. doi: 10.1109/TAC.9
[16] Xue S, Luo B, Liu D, et al. Event-triggered ADP for tracking control of partially unknown constrained uncertain systems. IEEE Trans Cybern. 2022;52(9):9001–9012. doi: 10.1109/TCSY.2021.3054626
[17] Wang Q, Cao J, Liu H. Event-triggered adaptive fuzzy PI control of uncertain fractional-order nonlinear systems with full-state constraints. IEEE Trans Emerg Topics Comput Intell. 2023;7(3):900–911. doi: 10.1109/TETCI.2022.3213708
[18] Deisenroth MP, Fox D, Rasmussen CE. Gaussian processes for data-efficient learning in robotics and control. IEEE Trans Pattern Anal Mach Intell. 2013;37(2):408–423. doi: 10.1109/TPAMI.2013.218
[19] Nesić D, Teel AR. Sampled-data control of nonlinear systems: an overview of recent results. Perspectives in robust control; 2007. p. 221–239.