# A Lifting Approach to Learning-Based Self-Triggered Control with Gaussian Processes

Wang Zhijun

Kazumune Hashimoto

Wataru Hashimoto

Shigemasa Takai

Abstract—This paper investigates the design of self-triggered control for networked control systems (NCS), where the dynamics of the plant is unknown apriori. To deal with the nature of the self-triggered control, in which state measurements are transmitted to the controller *a*-periodically, we propose to lift the continuous-time dynamics to a novel dynamical model by taking an inter-event time as an additional input, and then, the lifted model is learned by the Gaussian processes (GP) regression. Moreover, we propose a learning-based approach, in which a self-triggered controller is learned by minimizing a cost function, such that it can take inter-sample behavior into account. By employing the lifting approach, we can utilize a gradient-based policy update as an efficient method to optimize both control and communication policies. Finally, we summarize the overall algorithm and provide a numerical simulation to illustrate the effectiveness of the proposed approach.

Index Terms—Event-triggered and self-triggered control, Gaussian process regression, Optimal control

#### I. INTRODUCTION

In recent years, event-triggered and self-triggered control have attracted much attention since they are known to be useful strategies for saving resources in networked control systems (NCSs) [1]. In contrast to a time-triggered control that executes a feedback control law in a periodic manner, the event and self-triggered control transmit sensor measurements to the controller over the communication network only when it is needed. Various event/self-triggered controllers have been studied, see [2] for its survey paper. Early works apply the input-to-state stability or  $L_2$  gain performance to design event/self-triggered control [3] for linear systems. More recently, some approaches integrate the event/selftriggered control in optimal control [4]–[6]. In addition, researches related to reachability and safety analysis have been also provided in recent years [7].

In many previous works of the event/self-triggered control framework, the dynamical model of the plant to be controlled is assumed to be known *apriori*, which implies that when designed controllers are applied to real-world control systems, the achieved performance will heavily depend on the modeling accuracy. However, physical dynamics of the true system can be sometimes complex and highly nonlinear (and thus it is unknown *apriori*). Motivated by this problem, several works have suggested an event/self-triggered control framework that can be applied for unknown transition dynamics; see [8]–[14]. For example, [8] provided an iterative procedure of learning both the system dynamics and the optimal policy based on training data. In particular, they employed a Gaussian processes (GP) regression to learn the dynamics and a self-triggered optimal policy was obtained by solving a value iteration algorithm. Moreover, [14] investigates an approach to design an event-triggered controller for input-affine systems, where communication time instants are determined by evaluating a Lyapunov function candidate. In addition, [9]–[13] investigated a model-free approach to designing event/self-triggered controllers based on a deep reinforcement learning framework.

The objective of this paper is to learn a self-triggered controller based on an optimal control-based framework, where the unknown transition dynamics is learned by the GP regression. To the best of our knowledge, such an objective was achieved by only one previous work [8]. However, we argue that this previous work has the following drawbacks. First, since the value iteration was implemented by discretizing the state-space into grid points so as to approximate the optimal policy, it requires heavy computational resources in general. Second, [8] considers the dynamical system of the form:  $x_{k+1} = f(x_k, u_k)$  ( $x_k$  is the state and  $u_k$  is the control input), and then the function f is learned based on the training data while executing the selftriggered controller. However, when learning the function fwe require the training data as the consecutive states and the control input (i.e.,  $x_k, x_{k+1}, u_k$ ), which implies that the training data is available only when the inter-event time step is 1. This means that we have to throw away all the data (state/control inputs) if the inter-event time step is selected larger than 1, and thus it may not be of suitable for learning the dynamics while executing the self-triggered controller. Hence, the previous work has potential drawbacks in terms of both the requirement of computational resources and the inefficiency of data to learn the dynamics.

To address the issues as mentioned above, this paper proposes a novel optimal control, learning-based approach to designing the self-triggered controller with the GP regression. In particular, the proposed approach has the following contributions. First, in order to efficiently learn the dynamics and take the follow-up designing of the self-triggered controller into account, we choose *not* to learn the dynamics originally defined as the ordinary differential or the difference equation (which is typically done in previous works of literature), but instead, we proposed to *lift* to a model, in which an interevent time is regarded as an additional control input, and then such a lifted model is learned by the GP regression. As will be detailed in later sections, this allows us to increase an efficiency of learning the dynamics, since we can utilize all

The authors are with the Graduate School of Engineering, Osaka University (e-mail: wang@is.eei.eng.osaka-u.ac.jp, hashimoto@eei.eng.osaka-u.ac.jp, takai@eei.eng.osaka-u.ac.jp)

the state information received at the controller as the training data to learn the dynamics. Second, we formulate a finite horizon optimal control problem, in which we can penalize the states between every adjacent triggering instants. For example, this allows us to optimize an effective self-triggered controller that can avoid colliding with obstacles, as will be clarified in the numerical simulation. Finally, by employing the lifted dynamics estimated by the training data, we show that the self-triggered controller can be optimized via a policy gradient algorithm, which is one of the computationally efficient framework to derive the policy. This leads to a significant reduction of the computational time in contrast to the previous work [8].

As briefly mentioned above, our approach is also related to several previous works [9]-[15], while the problem setup considered in this paper is significantly different from them in the following ways. [9]-[13] investigate an approach to learn event/self-triggered controllers based on a deep reinforcement learning. Our approach differs from these works, in the sense that we here provide a model-based solution, in which the (lifted) dynamics is learned by the GP regression and the optimal self-triggered controller is designed accordingly. Moreover, in constrast to [14] in which an event-triggered controller is designed based on a Lyapunov function candidate, our approach investigates a way to design a self-triggered controller based on an optimal control problem with a policy gradient technique. In addition, our approach is related to [15], where an optimal control policy is designed via policy gradient for the dynamics learned by the GP regression. The proposed approach differs from [15] in the following sense. First, while [15] investigates a way to design only a control policy, we here investigate a way to design a self-triggered controller that provides both control and communication policies, which will be achieved by introducing the lifted model. Second, we provide several modifications on a computational graph for policy evaluations and policy improvements. This is because we modify a cost such that the inter-sample behavior of the states can be taken into account while executing a self-triggered controller (for details, see Section III-E,F).

*Notation:* Let  $\mathbb{N}$ ,  $\mathbb{N}_{>0}$  be the set of non-negative integers and positive integers respectively. Let  $\mathbb{R}$ ,  $\mathbb{R}_{\geq 0}$ ,  $\mathbb{R}_{>0}$  be the set of reals, non-negative reals and positive reals, respectively. Given  $x_{\min}, x_{\max} \in \mathbb{R}^n$ , let  $[x_{\min}, x_{\max}]$  denote a hyperrectangle with the extreme (i.e., lower-left/upper-right) points  $x_{\min}, x_{\max}$ . For a square matrix Q, we use  $Q \succ 0$  to denote that Q is positive define. Let diag $(a_1, a_2, \ldots, a_h)$  be the diagonal matrix whose elements are give by  $a_1, \ldots, a_h \in \mathbb{R}$ . Let 0 be the matrix of all zeros and I be the identity matrix.

# **II. PROBLEM FORMULATION**

# A. System description

We consider the dynamical system of the form:

$$\dot{x}_t = f(x_t, u_t), \ u_t \in \mathcal{U}, \ x_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \tag{1}$$

for all  $t \in \mathbb{R}_{\geq 0}$ , where  $x_t \in \mathbb{R}^{n_x}$  and  $u_t \in \mathbb{R}^{n_u}$  are the states and the control inputs at time t respectively, and

 $\mathcal{U} = [u_{\min}, u_{\max}] \subset \mathbb{R}^{n_u}$  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,  $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$  is a function representing the transition dynamics, which is unknown *apriori*.

#### B. 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  $\pi : \mathbb{R}^{n_x} \to \mathcal{U} \times \mathbb{R}_{>0}$ , 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.,  $[u_{t_n}^\top, \tau_n]^\top = \pi(x_{t_n})$ ;
- (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 selftriggered 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, we will ensure that the designed policy leads to satisfying  $\tau_n \geq \tau_{\min}$ ,  $\forall n \in \mathbb{N}$  for a given lower bound of the inter-event time  $\tau_{\min} \in \mathbb{R}_{>0}$ , which aims at guaranteeing the positive inter-event times. A concrete procedure of the proposed approach is elaborated in the next section.

# III. PROPOSED APPROACH

#### A. 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  $au_{\min}$  is close to 0, one could often set the lowest inter-event time, say  $\tau_n = \tau_{\min}$ , 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.,  $\dot{x}_{t_n} = f(x_{t_n}, u_{t_n}) \approx (x_{t_n + \tau_n} - x_{t_n}) / \tau_n^{-1}$ . 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 interevent 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.

#### B. 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 selftriggered control, in this paper we propose to modify a *function to be learned* as follows. First, note that we have

$$x_{t_{n+1}} = x_{t_n} + \int_{t_n}^{t_{n+1}} f(x_t, u_{t_n}) \mathrm{d}t,$$
(2)

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  $g(x_{t_n}, u_{t_n}, \tau_n) = x_{t_n} + \int_{t_n}^{t_n + \tau_n} f(x_t, u_{t_n}) dt$ , we have

$$x_{t_{n+1}} = g(x_{t_n}, v_n), \ n \in \mathbb{N},\tag{3}$$

where  $v_n = [u_{t_n}^{\top}, \tau_n]^{\top}$  denotes the *extended* control input that incorporates both  $u_{t_n}$  and  $\tau_n$ . 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, whose training data is available only when the inter-event time is selected small enough as described above, we can make use of all the state information  $x_{t_n}$ ,  $n \in \mathbb{N}$ received at the controller as the training data to learn the dynamics; for details, see Section III-C. 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 III-F.

#### C. 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  $x_{t_{n+1},j} =$  $g_j(x_{t_n}, v_n), j = 1, 2, ..., n_x$ , where  $x_{t_{n+1},j}$  and  $g_j$  are the *j*-th element of  $x_{t_{n+1}}$  and g, respectively. We denote by  $\mathcal{D}_j = \{\tilde{X}, Y_j\}$  the dataset to estimate  $g_j$ , where

$$\tilde{X} = \begin{bmatrix} x_{t_0}^* \\ v_0^* \end{bmatrix}, \begin{bmatrix} x_{t_1}^* \\ v_1^* \end{bmatrix}, \dots, \begin{bmatrix} x_{t_{D-1}}^* \\ v_{D-1}^* \end{bmatrix} \end{bmatrix}, \quad (4)$$

$$Y_{j} = [\Delta_{0,j}^{*}, \Delta_{1,j}^{*}, \dots, \Delta_{D-1,j}^{*}]^{\top},$$
(5)

with  $\Delta_{n,j}^* = x_{t_{n+1},j}^* - x_{t_n,j}^*$  for  $n = 0, \dots, D-1$  and D is the number of the training data. Then, given  $\mathcal{D}_j$ , an arbitrary state  $x_{t_n} \in \mathbb{R}^{n_x}$  and an extended control input  $v_n \in \mathbb{R}^{n_u+1}$ , we can predict  $x_{t_{n+1},j}$  by the Gaussian distribution as

$$p(x_{t_{n+1},j} \mid x_{t_n}, v_n, \mathcal{D}_j) = \mathcal{N}(x_{t_{n+1},j} \mid \mu_{n+1,j}, \sigma_{n+1,j}),$$

where  $\mu_{n+1,j} = x_{t_n,j} + \mathbb{E}_g[\Delta_{n,j}], \sigma_{n+1,j} = \operatorname{var}_g[\Delta_{n,j}]$  with  $\Delta_{n,j} = x_{t_{n+1},j} - x_{t_n,j}$  are the mean and the variance of the GP prediction, respectively, and these are given by

$$\mathbb{E}_{g}[\Delta_{n,j}] = k_{*,j}^{T} \left( K_{j} + \sigma_{w,j}^{2} \mathbf{I} \right)^{-1} Y_{j},$$
  
$$\operatorname{var}_{g}[\Delta_{n,j}] = k_{j}(\tilde{x}_{t_{n}}, \tilde{x}_{t_{n}}) - k_{*,j}^{T} \left( K_{j} + \sigma_{w,j}^{2} \mathbf{I} \right)^{-1} k_{*,j},$$
(6)

where  $\tilde{x}_{t_n} = [x_{t_n}^{\top}, v_n^{\top}]^{\top}$ ,  $k_j(\cdot, \cdot)$  is a given kernel parameterized by the hyperparameter  $\theta_j$ ,  $k_{*,j} = k_j(\tilde{X}, \tilde{x}_{t_n}) \in \mathbb{R}^{D \times 1}$ , and  $K_j \in \mathbb{R}^{D \times D}$  is the kernel matrix whose *p*-*q* element is defined as  $K_{j,pq} = k_j(\tilde{x}_{t_p}^*, \tilde{x}_{t_q}^*)$ , where  $\tilde{x}_{t_n}^* = [x_{t_n}^{*\top}, v_n^{*\top}]^{\top}$ . Moreover,  $\sigma_w = \text{diag}(\sigma_{w,1}, \dots, \sigma_{w,n_x})$  is a covariance of the Gaussian distributed white noise for the observation. After obtaining the predictions for all the elements of  $x_{t_{n+1}}$ , the whole predictive distribution of  $x_{t_{n+1}}$  is  $p(x_{t_{n+1}} \mid x_{t_n}, v_n, \{\mathcal{D}_j\}_{j=1}^{n_x}) = \mathcal{N}(x_{t_{n+1}} \mid \mu_{n+1}, \Sigma_{n+1})$ , where

$$\mu_{n+1} = [\mu_{n+1,1}, \dots, \mu_{n+1,n_x}]^{\top}, \tag{7}$$

$$\Sigma_{n+1} = \operatorname{diag}(\sigma_{n+1,1}, \dots, \sigma_{n+1,n_x}).$$
(8)

In other words, the obtained model provides, for given  $x_{t_n}$  and  $v_n = [u_n^{\top}, \tau_n]^{\top}$ , a prediction of the state at the next communication time, i.e.,  $x_{t_{n+1}}$  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.

#### D. 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  $x_{t_n}$  and  $v_n = [u_n^{\top}, \tau_n]^{\top}$ , a prediction of the state at the next communication time  $x_{t_{n+1}}$ . Hence, as a simple and natural way for defining the cost function, one could consider the following: given  $N \in \mathbb{N}$ ,

$$J(\pi_{\psi}) = \sum_{n=0}^{N-1} \mathbb{E}_{x_{t_n}}^{\pi_{\psi}} \left[ c(x_{t_n}, \tau_n) \right], \ x_{t_0} \sim \mathcal{N}(\mu_0, \Sigma_0), \quad (9)$$

where  $\pi_{\psi} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u+1}$  denotes a policy parameterized by  $\psi$ , and  $\mathbb{E}_{x_{t_n}}^{\pi_{\psi}}[\cdot]$  denotes the expectation with respect to

<sup>&</sup>lt;sup>1</sup>Note that a direct access of f(x, u), or the derivative of the state  $\dot{x}$ , is in general hard for digital computers. Hence, it is of practical to utilize the consecutive states  $x_{t_n+\tau_n}, x_{t_n}$  in order to estimate f.

 $x_{t_n}$  conditioned on the policy  $\pi_{\psi}$ , and  $c: \mathbb{R}^{n_x} \times \mathbb{R}_{\geq 0} \to$  $\mathbb{R}_{>0}$  denotes a given stage cost. (9) 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 III-C. Defining the cost function as above allows us to directly apply the policy gradient algorithm (see, e.g., [15]) 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 IV. To overcome this shortcoming, we propose to modify the cost function as follows: given  $M, N \in \mathbb{N}_{>0}$ ,

$$J(\pi_{\psi}) = \sum_{n=0}^{N-1} \mathbb{E}_{x_{t_{n,m}}}^{\pi_{\psi}} \left[ \lambda c_1(\tau_n) + \sum_{m=0}^{M-1} c_2(x_{t_{n,m}}) \right], \quad (10)$$
$$x_{t_0} \sim \mathcal{N}(\mu_0, \Sigma_0)$$

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

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

To evaluate and minimize J in (10), we need to predict the trajectory distribution

$$p(x_{t_{n,m}}), n = 0, \dots, N-1, m = 0, \dots, M-1$$
 (11)

conditioned on the policy  $\pi_{\psi}$ . The calculation of (11) requires mapping a probability distribution through the GP model, which is mathematically intractable. Hence, we utilize a moment matching technique (see, e.g., [15]) to approximate (11) by the Gaussians. While a basic procedure follows the approach given in [15], some modifications are necessary in terms of how to cascade the computations for  $p(x_{n,m})$ (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 [15].

Suppose  $p(x_{t_{n,0}}) = p(x_{t_n})$  is approximated by a Gaussian distribution. Then, a Gaussian approximation for the distribution  $p(v_n) = p(\pi_{\psi}(x_{t_{n,0}}))$  can be computed, and then we can analytically compute Gaussian approximation for



Fig. 1. The computational graph of  $p(x_{t_{n,m}})$  considered in this paper.



Fig. 2. The computational graph originally proposed in [15].

the joint distribution  $p(x_{t_{n,0}}, v_n) = p(x_{t_{n,0}}, \pi_{\psi}(x_{t_{n,0}}))$  (for details, see Section 5.5 in [15]). Since  $p(v_n)$  is Gaussian and  $v_{n,m} (= [u_{t_n}^{\top}, \alpha_m \tau_n]^{\top})$  is obtained from  $v_n$  by the following linear transformation

$$v_{n,m} = \begin{bmatrix} \mathbf{I}_{n_u \times n_u} & \mathbf{0}_{n_u \times 1} \\ \mathbf{0}_{1 \times n_u} & \alpha_m \end{bmatrix} v_n, \tag{12}$$

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

$$p(x_{t_{n,m}}) = \int p(x_{t_{n,m}} \mid \tilde{x}_{t_{n,m}}) p(\tilde{x}_{t_{n,m}}) \mathrm{d}\tilde{x}_{t_{n,m}}.$$
 (13)

From the GP model given in Section III-C, it follows that  $p(x_{t_{n,m}} | \tilde{x}_{t_{n,m}})$  is Gaussian. Therefore, by analytically computing the mean and covariance matrix of the right hand side of (13), we can approximate the distribution of  $x_{t_{n,m}}$  as a Gaussian distribution, i.e.,  $p(x_{t_{n,m}}) \approx \mathcal{N}(\mu_{n,m}, \Sigma_{n,m})$ . Similarly, we have

$$p(x_{t_{n+1,0}}) = \int p(x_{t_{n+1,0}} \mid \tilde{x}_{t_n}) p(\tilde{x}_{t_n}) \mathrm{d}\tilde{x}_{t_n}, \qquad (14)$$

where  $\tilde{x}_{t_n} = [x_{t_{n,0}}^{\top}, v_n^{\top}]^{\top}$ . From the GP model given in Section III-C, it follows that  $p(x_{t_{n+1,0}} | \tilde{x}_{t_n})$  is Gaussian. Hence, we can approximate  $p(x_{t_{n+1,0}})$  by the Gaussian distribution by analytically computing the mean and the covariance in the right hand side of (14). Since we can iteratively cascade the above procedure, all of the required state distribution (11) can be obtained with this scheme. The computational graph that summarizes the above procedure is shown in Fig.1.

For comparisons, in Fig.2 we illustrate the computational graph originally proposed in [15]. Note that [15] 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 Fig.2, [15] 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 (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.,  $p(x_{t_{m-m}}), m = 1, \ldots, M-1$  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  $v_{n,m} = [u_{t_n}^\top, \alpha_m \tau_n]^\top$ .

Finally, to evaluate the expected the total cost J in (10), it remains to compute the expected values

$$\mathbb{E}_{x_{t_{n,0}}}[c_1(\tau_n)] = \int c_1(\tau_n) \mathcal{N}(\mu_{n,0}, \Sigma_{n,0}) \mathrm{d}x_{t_{n,0}}$$
$$\mathbb{E}_{x_{t_{n,m}}}[c_2(x_{t_{n,m}})] = \int c_2(x_{t_{n,m}}) \mathcal{N}(\mu_{n,m}, \Sigma_{n,m}) \mathrm{d}x_{t_{n,m}}$$

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.

#### F. Gradient Based Policy Improvement

To find policy parameters  $\psi$  minimizing  $J(\pi_{\psi})$  in (10), we use gradient information  $dJ(\pi_{\psi})/d\psi$ . As with the policy evaluation given in the previous section, we need to provide some modifications from [15], since the computational graph for (10) is different from that of [15]. 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  $dJ/d\psi$  by repeatedly applying the chain rule. First, we use the notation:

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

$$\eta_{n} = \mathbb{E}_{x_{t_{n,0}}}[c_{1}(\tau_{n})], \ \epsilon_{n,m} = \mathbb{E}_{x_{t_{n,m}}}[c_{2}(x_{t_{n,m}})].$$
(15)

Then the following equations is obtained from (10).

$$\frac{\mathrm{d}J(\pi_{\psi})}{\mathrm{d}\psi} = \sum_{n=0}^{N-1} \frac{\mathrm{d}\zeta_n}{\mathrm{d}\psi},$$

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

$$\frac{\mathrm{d}\eta_n}{\mathrm{d}\psi} = \frac{\mathrm{d}\eta_n}{\mathrm{d}p(\tau_n)} \frac{\mathrm{d}p(\tau_n)}{\mathrm{d}\psi},$$

$$\frac{\mathrm{d}\epsilon_{n,m}}{\mathrm{d}\psi} = \frac{\mathrm{d}\epsilon_{n,m}}{\mathrm{d}p(x_{t_n,m})} \frac{\mathrm{d}p(x_{t_n,m})}{\mathrm{d}\psi}.$$
(16)

We first discuss the term  $d\epsilon_{n,m}/d\psi$  by adopting the shorthand notation  $d\epsilon_{n,m}/dp(x_{t_{n,m}}) =$   $\{\mathrm{d}\epsilon_{n,m}/\mathrm{d}\mu_{n,m},\mathrm{d}\epsilon_{n,m}/\mathrm{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}}) = \mathcal{N}(\mu_{n,m},\Sigma_{n,m})\$  such that the following equation could be derived.

$$\frac{\mathrm{d}\epsilon_{n,m}}{\mathrm{d}\psi} = \frac{\mathrm{d}\epsilon_{n,m}}{\mathrm{d}\mu_{n,m}}\frac{\mathrm{d}\mu_{n,m}}{\mathrm{d}\psi} + \frac{\mathrm{d}\epsilon_{n,m}}{\mathrm{d}\Sigma_{n,m}}\frac{\mathrm{d}\Sigma_{n,m}}{\mathrm{d}\psi}.$$
 (17)

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 Fig.1) and the controller parameters  $\psi$ . By applying the chain rule to  $dp(x_{t_{n,m}})/d\psi$  in (16), we obtain

$$\frac{\mathrm{d}p(x_{t_{n,m}})}{\mathrm{d}\psi} = \frac{\partial p(x_{t_{n,m}})}{\partial p(x_{t_{n,0}})} \frac{\mathrm{d}p(x_{t_{n,0}})}{\mathrm{d}\psi} + \frac{\partial p(x_{t_{n,m}})}{\partial \psi},$$

$$\frac{\partial p(x_{t_{n,m}})}{\partial p(x_{t_{n,0}})} = \left\{ \frac{\partial \mu_{n,m}}{\partial p(x_{t_{n,0}})}, \frac{\partial \Sigma_{n,m}}{\partial p(x_{t_{n,0}})} \right\}.$$
(18)

From here onward, we focus on  $d\mu_{n,m}/d\psi$ , see (17), because the calculation of  $d\Sigma_{n,m}/d\psi$  is similar. For  $d\mu_{n,m}/d\psi$ , we calculate the derivative

$$\frac{\mathrm{d}\mu_{n,m}}{\mathrm{d}\psi} = \frac{\partial\mu_{n,m}}{\partial\mu_{n,0}}\frac{\mathrm{d}\mu_{n,0}}{\mathrm{d}\psi} + \frac{\partial\mu_{n,m}}{\partial\Sigma_{n,0}}\frac{\mathrm{d}\Sigma_{n,0}}{\mathrm{d}\psi} + \frac{\partial\mu_{n,m}}{\partial\psi}.$$
 (19)

And  $dp(x_{t_{n,0}})/d\psi$  in (18) 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

$$\frac{\partial \mu_{n,m}}{\partial \psi} = \frac{\partial \mu_{n,m}}{\partial p(v_{n,m})} \frac{\partial p(v_{n,m})}{\partial \psi} 
= \frac{\partial \mu_{n,m}}{\partial \mu_{n,m}^v} \frac{\partial \mu_{n,m}^v}{\partial \psi} + \frac{\partial \mu_{n,m}}{\partial \Sigma_{n,m}^v} \frac{\partial \Sigma_{n,m}^v}{\partial \psi},$$
(20)

where  $v_{n,m} \sim \mathcal{N}(\mu_{n,m}^v, \Sigma_{n,m}^v)$ . The partial derivatives of  $\mu_{n,m}^v$  and  $\Sigma_{n,m}^v$ , i.e. the mean and covariance of  $p(v_{n,m})$ , used in (20) depend on the policy representation. To evaluate  $dJ(\pi_{\psi})/d\psi$ , we also need the information of  $dp(\tau_n)/d\psi$  in (16), which could be computed by

$$\frac{\mathrm{d}p(\tau_n)}{\mathrm{d}\psi} = \frac{\mathrm{d}p(\tau_n)}{\mathrm{d}p(v_n)} \frac{\mathrm{d}p(v_n)}{\mathrm{d}\psi} = \begin{bmatrix} \mathbf{0}_{1 \times n_u} & 1 \end{bmatrix} \frac{\mathrm{d}p(v_n)}{\mathrm{d}\psi}, 
\frac{\mathrm{d}p(v_n)}{\mathrm{d}\psi} = \frac{\partial p(v_n)}{\partial p(x_{t_{n,0}})} \frac{\mathrm{d}p(x_{t_{n,0}})}{\mathrm{d}\psi} + \frac{\partial p(v_n)}{\partial\psi},$$
(21)

where  $dp(x_{t_{n,0}})/d\psi$  and  $\partial p(v_n)/\partial \psi$  has been discussed in the previous content, and  $\partial p(v_n)/\partial p(x_{t_n})$  depends on the policy representation. The individual partial derivatives in (16) to (21) need to apply chain rule to moment matching (for their detailed computations, see the appendix of [15]).

# *G.* 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  $\tau_{\min} > 0$  such that  $\tau_n \geq \tau_{\min}$  for all  $n = 0, 1, \ldots$  (see, e.g., [1]). In addition, we constrain that the control input must belong to  $\mathcal{U} = [u_{\min}, u_{\max}]$ , i.e.,  $u_{t_n} \in \mathcal{U}$  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  $\pi_{\psi}(x) = \{\pi_{\psi_{\tau}}(x), \pi_{\psi_{u}}(x)\}$  where  $\pi_{\psi_{\tau}} : \mathbb{R}^{n_{x}} \to \mathbb{R}_{\geq 0}$  denotes a policy

to compute the inter-event time  $\tau$ , and  $\pi_{\psi_u}(x) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ denotes a policy to compute the control input u. Now, let us parametrize the policy  $\pi_{\psi_{\tau}}$  as follows:

$$\pi_{\psi_{\tau}}(x) = \frac{\tau_{\max} + \tau_{\min}}{2} + \frac{\tau_{\max} - \tau_{\min}}{2} \sigma_{\tau}(\tilde{\pi}_{\psi_{\tau}}(x)), \quad (22)$$

for given  $au_{\min}, au_{\max}$  > 0  $^2$  with  $au_{\max}$  >  $au_{\min}$  and  $\sigma_{\tau}(\cdot)$  is a given squashing function that satisfies  $\sigma_{\tau}(z) \in$ [-1,1] for all  $z \in \mathbb{R}$  (see Section 5.1 in [15]), and  $\tilde{\pi}_{\psi_{\tau}}$  :  $\mathbb{R}^{n_x} \to \mathbb{R}$  denotes a preliminary policy with an unconstrained amplitude, such as the one given by a radial basis function (RBF). Parametrizing  $\pi_{\psi_{\tau}}$  as above leads to ensuring that  $\pi_{\psi_{\tau}}(x) \in [\tau_{\min}, \tau_{\max}]$  for all  $x \in \mathbb{R}^{n_x}$ , and thus the optimized the policy ensures a positive interevent time. Similarly, we can ensure a constraint satisfaction of the control input by parametrizing  $\pi_{\psi_u}(x)$  as  $\pi_{\psi_u^i}(x) = \frac{u_{\max,i} + u_{\min,i}}{2} + \frac{u_{\max,i} - u_{\min,i}}{2} \sigma_u(\tilde{\pi}_{\psi_u^i}(x))$  for all  $i = 1, \ldots, n_u$ , where  $u_{\max,i}$  and  $u_{\min,i}$  denote the *i*-th element of  $u_{\max}$  and  $u_{\min}$  respectively, and  $\pi_{\psi^i_\omega}$  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  $\tilde{\pi}^i_{\psi_n}$  :  $\mathbb{R}^{n_x} \to \mathbb{R}$  denotes a preliminary policy with an unconstrained amplitude. Parametrizing  $\pi_{\psi_u}$ as above leads to ensuring that  $\pi_{\psi_u}(x) \in [u_{\min}, u_{\max}]$  for all  $x \in \mathbb{R}^{n_x}$ , guaranteeing the control constraint satisfaction.

#### H. Overall Algorithm

Let us now introduce an overall implementation algorithm that jointly learns the dynamics of the plant and the selftriggered 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  $\{(x_{t_i}^*, v_i^*), x_{t_{i+1}}^*\}$  according to Section III-C (line 3). Then, using the recorded input and output data  $\mathcal{D}$ , the controller learns the lifted dynamics  $x_{t_{n+1}} = g(x_{t_n}, v_n)$  using GP regression (line 5). Next, the algorithm utilizes the current policy and the learned dynamics to predict future trajectory distribution  $\{p(x_{t_{n,m}})\}$  from an initial state distribution  $p(x_{t_0})$ , and then calculates the expected total cost J (line 7). Thereafter, the gradient information  $dJ/d\psi$  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  $\{(x_{t_i}^*, v_i^*), x_{t_{i+1}}^*\}$  during execution. The gathered data is appended to the total training data, and back to line 5, the dynamical model  $x_{t_{n+1}} = g(x_{t_n}, v_n)$  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.

## IV. SIMULATION

# A. Inverted pendulum

To make comparisons with the previous work [8], we first conducted a simple experiment of an inverted pendulum,

# Algorithm 1 Learning self-triggered controllers

- Input: Characterization of the stage cost functions c<sub>1</sub>, c<sub>2</sub>, prediction horizon N, Gaussian distribution of the initial state N(μ<sub>0</sub>, Σ<sub>0</sub>), initial policy parameter ψ, step-size of the gradient update α > 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 III-C);

#### 4: repeat

- 5: Using the recorded training data, estimate the lifted dynamics by the GP regression (Section III-C);
- 6: repeat
- 7: Use the current policy  $\pi_{\psi}$  to predict future trajectory distribution and calculate the expected total cost (10) (Section III-E);
- 8: Compute gradient information  $dJ/d\psi$  using (16) to (21);
- 9: Update the policy parameter as  $\psi \leftarrow \psi \alpha dJ/d\psi$ ;
- 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



Fig. 3. Learned Controller Performance when  $\lambda = 0.01$ 

whose dynamics is given of the form:

$$\ddot{\phi} = \frac{u - b\dot{\phi} - \frac{1}{2}mlg\sin\phi}{\frac{1}{2}ml^2 + I},$$
(23)

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 =1m, b = 0.01 and  $g = 9.82m/s^2$ . We then define the system state as  $x_{t_n} = [\phi_{t_n}, \phi_{t_n}]^{\top}$ , and the extended control input as  $v_n = [u_{t_n}, \tau_n]^{\top}$ . The cost functions are given by  $c_1(\tau) = \tau_{\max} - \tau$ , and  $c_2(x) = -\exp(-\frac{1}{2}(\operatorname{Tri}(\phi) - \operatorname{Tri}(\pi))^\top Q(\operatorname{Tri}(\phi) - \operatorname{Tri}(\pi)))$ , where  $\operatorname{Tri}(\cdot)$  is defined as  $\mathbf{Tri}(\beta) = [\sin \beta, \cos \beta]^{\top}$  for  $\beta \in \mathbb{R}$ . Moreover, we set  $Q = \text{diag}(4, 4), \ \tau_{\min} = 0.02, \ \tau_{\max} = 0.6 \ \text{and} \ M = 1.$  The simulation result is given in Figs.3, 4. 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 Fig.4 that the interevent time tends to be larger as  $\lambda$  is selected larger, which is because we penalize more for the communication cost.

 $<sup>^2 {\</sup>rm Here}, \, \tau_{\rm max}$  could be selected arbitrary large so as to lengthen the interevent time.



Fig. 4. Inter-event time  $\tau_n$  with  $\lambda = 0.01, 0.1, 0$  when episode = 20

For comparisons, we also conducted the experiment using [8], in which (23) is approximated by the discrete-time system under the time period 0.02. Table I 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 [8] 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 [8]. This is because [8] 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.

TABLE I EXECUTION TIME AND NUMBER OF EPISODES TO ACHIEVE STABILITY

	Previous approach [8]	Proposed approach
Execution time	5678s	96s
Number of episodes	10	5

## B. Vehicle navigation with collision avoidance

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

$$\dot{x}_1 = u_1 \cos(\theta), \ \dot{x}_2 = u_1 \sin(\theta), \ \theta = u_2,$$
 (24)

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  $u_2 \in [\omega_{\min}, \omega_{\max}]$  is the angular velocity. We then define the system state as  $x_{t_n} = [x_{1,t_n}, x_{2,t_n}, \theta_{t_n}]^{\top}$  and the system input as  $v_n = [u_{1,t_n}, u_{2,t_n}, \tau_n]^{\top}$ . 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

$$c_2(x) = c_{\rm tg}(x) + c_{\rm ob}(x),$$
  

$$c_{\rm tg}(x) = -K_{\rm tg} \exp\left(-\frac{1}{2}(x - x_{\rm tg})^\top Q(x - x_{\rm tg})\right),$$
  

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



Fig. 5. The contour map of  $c_2(x)$ 

where  $K_{\rm ob} > K_{\rm tg} > 0$ ,  $Q \succ 0$  is the weight matrix of the state  $x, x_{\rm tg} \in \mathbb{R}^3$  is the target state, OB is the collection of all obstacles, and  $x_{\rm ob} \in \mathbb{R}^3$ ,  $Q_{\rm ob} \succ 0$  define the position and shape of obstacles respectively. In this simulation, we set  $K_{\rm tg} = 1, K_{\rm ob} = 50, x_{\rm tg} = [5, 5, 0]^T$ , and  $Q = {\rm diag}(0.04, 0.04, 0)$ . The contour map of  $c_2(x)$  is shown in the Fig.5. 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  $c_1(\tau) = \tau_{\rm max} - \tau$  with  $\tau_{\rm max} = 0.8$ , and the minimum inter-event time is set to  $\tau_{\rm min} = 0.02$ .

Performance of the controller learned from Algorithm 1 is given in Figs. 6 and 7. 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 (9) and, as discussed in Section III-D, it neglects state trajectories between any two adjacent triggering instants. Fig.7 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. Fig.8 depicts how inter-event times change with  $\lambda = 0.01, 0.1, 0$ . 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 (10) is defined by summing the 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 (10) 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  $\lambda = 0$ , 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 (10). Using a slightly larger  $\lambda$  helps solve this problem, but a too large  $\lambda$ causes the algorithm difficult to learn an effective controller.

Finally, we perform a complexity analysis of the proposed algorithm . The main complexity of the algorithm focuses at



Fig. 6. Learned controller performance when  $M = 1, \lambda = 0.01$ .



Fig. 7. Learned controller performance when  $M = 5, \lambda = 0.01$ .

the policy improvement, namely lines 6-10 in Algorithm 1. We use Fig.9 to show that how long running lines 6-10 takes with different M. It can be seen that the running time is approximately linear with M which is attributed to the fact that increasing M will only complexify the computation of J and  $dJ/d\psi$  linearly.

# V. CONCLUSION

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



Fig. 8. Inter-event time  $\tau_n$  with  $\lambda = 0.01, 0.1, 0$  when episode = 5.



Fig. 9. Runtime to take line 6-10 in Algorithm 1 with different M.

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.

#### REFERENCES

- W. Heemels, K. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), 2012, pp. 3270–3285.
- [2] C. Peng and F. Li, "A survey on recent advances in event-triggered communication and control," *Information Sciences*, vol. 457, pp. 113– 125, 2018.
- [3] M. Mazo Jr, A. Anta, and P. Tabuada, "An ISS self-triggered implementation of linear controllers," *Automatica*, vol. 46, no. 8, pp. 1310–1314, 2010.
- [4] T. Gommans, D. Antunes, T. Donkers, P. Tabuada, and M. Heemels, "Self-triggered linear quadratic control," *Automatica*, vol. 50, no. 4, pp. 1279–1287, 2014.
- [5] K. Hashimoto, S. Adachi, and D. V. Dimarogonas, "Event-triggered intermittent sampling for nonlinear model predictive control," *Automatica*, vol. 81, pp. 148–155, 2017.
- [6] K. Hashimoto, S. Adachi, and D. V. Dimarogonas, "Time-constrained event-triggered model predictive control for nonlinear continuous-time systems," in 2015 54th IEEE Conference on Decision and Control (CDC), 2015, pp. 4326–4331.
- [7] K. Hashimoto, A. Saoud, M. Kishida, T. Ushio, and D. V. Dimarogonas, "A symbolic approach to the self-triggered design for networked control systems," *IEEE Control Systems Letters*, vol. 3, no. 4, pp. 1050–1055, 2019.
- [8] K. Hashimoto, Y. Yoshimura, and T. Ushio, "Learning self-triggered controllers with gaussian processes," *IEEE transactions on cybernetics*, 2020.
- [9] R. Wang, I. Takeuchi, and K. Kashima, "Deep reinforcement learning for continuous-time self-triggered control," *IFAC-PapersOnLine*, vol. 54, no. 14, pp. 203–208, 2021.
- [10] D. Baumann, J.-J. Zhu, G. Martius, and S. Trimpe, "Deep reinforcement learning for event-triggered control," in *Proceedings of 57th IEEE Conference on Decision and Control (IEEE CDC)*, 2018, pp. 943–950.
- [11] N. Funk, D. Baumann, V. Berenz, and S. Trimpe, "Learning eventtriggered control from data through joint optimization," *IFAC Journal* of Systems and Control, vol. 16, 2021.
- [12] K. G. Vamvoudakis, A. Mojoodi, and H. Ferraz, "Event-triggered optimal tracking control of nonlinear systems," *The International Journal of Robust and Nonlinear Control*, vol. 27, no. 4, pp. 598– 619, 2017.
- [13] Y. Yang, K. G. Vamvoudakis, H. Ferraz, and H. Modares, "Dynamic intermittent *Q*-learning for systems with reduced bandwidth," in *Proceedings of 2018 IEEE Conference on Decision and Control (IEEE CDC)*, 2018, pp. 924–931.
- [14] J. Umlauft and S. Hirche, "Feedback linearization based on gaussian processes with event-triggered online learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 10, pp. 4154–4169, 2020.

[15] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE transactions* on pattern analysis and machine intelligence, vol. 37, no. 2, pp. 408– 423, 2013.