2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Self-Triggered Markov Decision Processes

Yunhan Huang[1] and Quanyan Zhu[1]

2021 60th IEEE Conference on Decision and Control (CDC) | 978-1-6654-3659-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CDC45484.2021.9682918

***Abstract*— In this paper, we study Markov Decision Processes (MDPs) with self-triggered strategies, where the idea of self-triggered control is extended to more generic MDP models. This extension broadens the application of self-triggering policies to a broader range of systems. We study the co-design problems of the control policy and the triggering policy to optimize two pre-specified cost criteria. The first cost criterion is introduced by incorporating a pre-specified update penalty into the traditional MDP cost criteria to reduce the use of communication resources. A novel dynamic programming (DP) equation called DP equation with optimized lookahead is proposed to solve for the optimal self-triggering policy under this criteria. The second self-triggering policy is to maximize the triggering time while still guaranteeing a pre-specified level of sub-optimality. Theoretical underpinnings are established for the computation and implementation of both policies. Through a gridworld numerical example, we illustrate the two policies' effectiveness in reducing resources consumption and demonstrate the trade-offs between resource consumption and system performance.**

I. Introduction

Recent development in information and communication technologies have led to the implementation of large-scale resource-constrained networked control systems. In these systems, it is desirable to limit the sensor and control communication and computation to instances when a system needs attention [1]. As a result, the self-triggered control paradigm has been proposed to reduce the utilization of communication resources and actuation movements while still maintaining desirable closed-loop behavior for these systems [2]. The self-triggered control abandons the conventional periodic time-triggered implementations. In self-triggered control, the self-triggering policy consists of two sub-policies: the control policy and a triggering mechanism that pre-determines, at an update time, when the control inputs have to be updated the next time. Due to its efficiency in resource-saving, self-triggered control has been studied extensively in the last decades [1]–[6].

The study of self-triggered has been confined to state-space dynamical models, including either linear models [1], [2], [4], [5] or nonlinear models [3], [6] in both(either) continuous-time and(or) discrete-time settings. However, recent developments in technologies such as wireless communication, machine learning, and real-time analytics have broadened the application of Internet of Things (IoTs) beyond control systems to a wide range of areas, including logistics and supply chain [7]–[9], smart cities [10], and wearables [11]. These systems are usually large-scale, equipped with resource-constrained devices, and difficult to be described by state-space dynamic models. Hence, there is an urgent need to incorporate the idea of self-triggering policies in control into a more general dynamic model: Markov Decision Processes (MDP). This incorporation can lead toward a computationally and communicationally more efficient IoT-enabled system.

This paper studies a discrete-time self-triggered MDP where the control[1] policy and the triggering mechanism are co-designed to achieve certain cost criteria. The differences between this work and most existing papers in self-triggered control are three-fold. The first is that we study self-triggered policies for a more general dynamic model, i.e., an MDP model, which allows the extension of the self-triggering policy to a wider range of applications. Second, we address the co-design problem of jointly designing the control policy and the triggering policy. Existing self-triggering methods design the control policy and the triggering policy in an ordered manner, i.e., one designs the control policy first and then design the triggering mechanism while ensuring certain control performance [1], [4]. For example, in [4], the authors pre-set the control gain to be the H_∞ control gain, based on which a triggering mechanism is designed to assure a specified level of $\mathcal{L}_2$ stability. Since the control policy is given without considering the self-triggering nature of the whole policy, it is hard to guarantee that the given control policy is optimal for achieving the minimum number of updates while maintaining certain cost criteria [2]. Here, we address a co-design problem to alleviate the concern regarding the optimality issue. Third, in existing works [1], the analysis of control performance under the self-triggered control paradigm is mostly qualitative, e.g., the analysis of whether one can achieve a certain type of stability or not. The control performance is, in literature, quantified as the decay rate for the Lyapunov function. Only few self-triggering methods provide quantitative analysis for control performance such as $\mathcal{L}_2$ gains [4], quadratic costs [12]–[14]. More recently, T. Gommans et al. studied self-triggered linear-quadratic-gaussian (LQG) control associated with quadratic costs. Our work considers a generic class of cost criteria and proposes self-triggered policies that guarantee a certain optimality level. The contributions of this paper are summarized as follows.

1) We study self-triggered MDP, which extends the idea of self-triggered control into a more generic dynamical model. The genericness of the MDP model enables the application of self-triggering policies into a broader range of systems.
2) We jointly design the control policy and the triggering

[1] Y. Huang and Q. Zhu are with the Department of Electrical and Computer Engineering, New York University, 370 Jay St., Brooklyn, NY. {yh.huang, qz494}@nyu.edu

[1]In this paper, we use the notion "control" and "action" interchangeably.

mechanism that co-optimizes pre-specified cost criteria.

3) We propose two frameworks that produce two co-designed self-triggering policies. The first is introduced by incorporating an update penalty into the traditional MDP cost criteria to reduce the use of communication resources. The second is a greedy reduction of resources used while still guaranteeing any pre-given level of sub-optimality. We establish theoretical underpinnings for the computation and implementation of both policies.
4) Through a gridworld example in both non-windy and windy settings, we show that the proposed policies are efficient in reducing communication resources used while still maintaining a high level of performance.

A. Nomenclature

In this paper, $\mathbb{R}$ and $\mathbb{N}$ represent the set of real numbers and natural numbers, respectively. The expectation operator is denoted by $\mathbb{E}$. And $\Delta t \in \mathbb{N}$ denotes the time steps between two neighboring updates. The letter l is the index for the lth update and t_l is the time instance when the lth update happens. The notation $\mathbb{N}_{[t_l, t_{l+1}]}$ means the intersection of the two sets $\mathbb{N}$ and $[t_l, t_{l+1}]$. The set of non-negative real numbers is denoted by $\mathbb{R}^+$. The notation $\mathcal{A}\backslash\mathcal{B}$ denotes the set $\{x \mid x \in \mathcal{A}, x \notin \mathcal{B}\}$.

II. SELF-TRIGGERED MARKOV DECISION PROCESSES

In this section, we provide the problem formulation for the self-triggered action strategy. We consider a discrete-time MDP defined by a tuple $\{\mathcal{X}, \mathcal{A}, P, c\}$, where $\mathcal{X}$ is the state space, $\mathcal{A}$ is the actions space, P is the time-homogeneous transition probability, and c is the stage-wise cost function. The state space $\mathcal{X}$ and action space $\mathcal{A}$ are both assumed to be Borel subsets of Polish (Banach and separable) spaces. If an action $a \in \mathcal{A}$ is selected at a state $x \in \mathcal{X}$, then a cost $c(x, a)$ is incurred, where without loss of generality, we suppose $c : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^+$. The function c is assumed to be bounded and Borel measurable. The transition probability $P(B|x, a)$ is a Borel function on $\mathcal{X} \times \mathcal{A}$ for each Borel subset B of $\mathcal{X}$, and $P(\cdot|x, a)$ is a probability measure on the Borel σ-field of $\mathcal{X}$ for each $(x, a) \in \mathcal{X} \times \mathcal{A}$.

In classic MDP, the decision process proceeds as follows: at time $t = 0, 1, \cdots$, the current state of the system, x_t, is observed. A decision-maker decides which action, a_t, to choose, the cost $c_t = c(x_t, a_t)$ is incurred, the system moves to the next state following the rule $x_{t+1} \sim P(\cdot|x_t, a)$, and the process continues. The rule that the decision-maker follows to choose an action is called policy. We consider stationary Markov policy ϕ in which all decisions depend only on the current state. A stationary Markov policy ϕ is defined by a measurable mapping $\phi : \mathcal{X} \times \mathcal{A}$. In classic MDP, the goal is to find an optimal stationary Markov policy that minimizes

$$v^{\phi}(x) = \mathbb{E}^{\phi}\left[\sum_{t=0}^{\infty} \beta^t c(x_t, a_t) \middle| x_0 = x\right], \tag{1}$$

where β is a discount factor strictly less than 1, and the expectation is based on the probability distribution on the set of all trajectory $(\mathcal{X} \times \mathcal{A})^{\infty}$, which is uniquely determined by the policy ϕ and the initial state x ([15], pp. 140-141). Define the optimal cost

$$V(x) \coloneqq \inf_{\phi \in \Phi} v^{\phi}(x),$$

where Φ is the set of all stationary policies. A policy ϕ is called optimal if $v^{\phi}(x) = V(x)$ for all $x \in \mathcal{X}$.

A. Self-Triggered Decision Making

In classic MDP, decision making requires persistent transmission of measured state and updates of actions at each time instance $t \in \mathbb{N}$. In this paper, we are interested in constructing a policy that requires less sensing demand, lower communication rate, and less actuator movements [16], while still maintaining certain forms of optimality.

The self-triggering policy is based on holding the current input value for a controlled duration. The self-triggered policy carries the following structure

$$\begin{cases} t_{l+1} & = t_l + \tau(x_{t_l}), \\ a_t & = \pi(x_{t_l}) \in \mathcal{A}, \ t \in \mathbb{N}_{[t_l, t_{l+1})}, \end{cases} \tag{2}$$

where l is the index for the number of triggers, $t_0 \coloneqq 0$, $\tau : \mathcal{X} \to \mathcal{T}, \mathcal{T} \coloneqq \{1, 2, \cdots, \bar{T}\}$, $\bar{T} \in \mathbb{N}$, and $\pi : \mathcal{X} \to \mathcal{A}$. Here, the integer $\bar{T}$ is an arbitrary upper bound on the waiting time for next update. The self-triggering policy, denoted by μ, involves two sub-policies: the timing policy, $\tau(x)$, that determines the next time for updating, and the control policy, $\pi(x)$, that chooses a fixed action to deploy for the next $\tau(x)$ time instances. For convenience, we write $\mu = (\tau, \pi)$ and $\mu : \mathcal{X} \to \mathcal{T} \times \mathcal{A}$.

B. Performance Criteria

This paper introduces two different yet related problems associated with two cost criteria; one is constructed by incorporating a penalty $O \geq 0$ for updating the action into the classic cost criteria defined in eq. (1). The idea of introducing a penalty is originated from costly measurements that have been investigated in the context of LQG optimal control [14], [17] and games [18], [19]. The penalty $O \geq 0$ is a scalar, which we refer to as the update penalty. For instance, if t_l and t_{l+1} are two neighboring updating time, during the time interval $[t_l, t_{l+1}]$, the total update penalty is $\beta^{t_l} O + \beta^{t_{l+1}} O$. Now, we formulate the first problem.

Problem 1. *Find an optimal self-triggering policy μ that minimizes the following cost criterion over an infinite horizon*

$$f^{\mu}(x) = \mathbb{E}^{\mu}\left[\sum_{t=0}^{\infty} \beta^t c(x_t, a_t) + \sum_{l=1}^{\infty} \beta^{t_l} O \middle| x_0 = x\right], \tag{3}$$

where the first term is the accumulated costs in the classic MDP, and the second term is the accumulated costs of updating one's action.

The second cost criteria is similar to that of [2]. That is for a pre-specified sub-optimal performance, we aim to

reduce the number of times the input/output is updated, while maintaining the pre-specified sub-optimal performance. Now, we formulate the second problem as

Problem 2. *Find a policy μ that maximizes the next transmission time $\tau(x)$ subject to the performance guarantee that*

$$v^{\mu}(x) \leq \alpha V(x), \text{ for all } x \in \mathcal{X}, \tag{4}$$

where $\alpha \geq 1$ is a scalar.

Remark 1. *In Problem 1, we introduce an update penalty O to capture the trade-off between the degree of optimality and the usage of sensing/communication resources. The update penalty can be interpreted as a soft constraint on the number of updates. In Problem 2, α serves as a scaling factor that can be selected arbitrarily to balance the consumption of sensing/communication resources and the degradation of performance. There is a hard constraint that requires matainting a certain degree of sub-optimality. When $\alpha = 1$, no degradation of performance is allowed. Solving both problems involves the co-design of the waiting time for next update (through τ) and the chosen action (through π).*

III. THEORETICAL FRAMEWORKS

In this section, by establishing theoretical underpinnings, we pave the way for finding the self-triggering policies that solve the problems. For Problem 1, we formulate a dynamic programming (DP) equation, which we call a DP equation with optimized lookahead. With this equation, we can resort to several effective methods such as value iterations and policy iterations to characterize an optimal self-triggering policy. For Problem 2, we propose a greedy self-triggering policy that reduces the number of updates and maintains a degree of optimality. We show that the proposed policy is well-defined and there exist such a policy for any pre-specified α.

A. Dynamic Programming Equation with Optimized Lookahead

To solve Problem 1, the DP equation with optimized lookahead is derived and presented in this sub-section. The derivation idea is to form consolidated costs, states, and actions between two update time instances, which generates a new discrete-time MDP in the classic setting.

Let $\bar{c}_l$ represent the consolidated costs that correspond to the time period between l-th update and $(l+1)$-th update, i.e., the time period $[t_l, t_{l+1})$. From eq. (2) and eq. (3), we can obtain

$$\bar{c}_l \coloneqq \bar{c}(x_{t_l}, a_{t_l}, \Delta t_l) = \mathbb{E}\left[\sum_{t=0}^{\Delta t_l - 1} \beta^t c(x_{t_l+t}, a_{t_l}) \middle| x_{t_l}, a_{t_l}, \Delta t_l\right],$$

where given a self-triggering policy $\mu = (\pi, \tau)$, the fixed action a_{t_l} is produced by $\pi(x_{t_l})$ and the waiting time Δt_l is generated by $\tau(x_{t_l})$. An application of the Fubini's theorem (principle) and Markov property [20] yields

$$\bar{c}(x, a, \Delta t) = \sum_{t=0}^{\Delta t - 1} \beta^t \mathbb{E}\left[c(x_t, a_t) | x_0 = x, a_t = a, \forall t < \Delta t\right].$$

Furthermore, we define

$$\bar{P}(B|x, a, \Delta T) \coloneqq Prob\left(x_{\Delta t} \in B | x_0 = x, a_t = a, \forall t < \Delta t\right), \tag{5}$$

as the skip-probability that the MDP is in Borel subset B of $\mathcal{X}$, after time Δt, given that the initial condition is $x_0 = x$ and that the action is fixed until Δt. The skip-probability $\bar{P}(B|x, a, \Delta T)$ is a Borel function on $\mathcal{X} \times \mathcal{A} \times \mathcal{T}$ for each Borel subset B of $\mathcal{X}$, which is determined by the one-step transition probability $P(\cdot|x, a)$ defined in Section II.

With the definition of the consolidated stage-wise function $\bar{c}$ and the tower property of conditional expectation, the infinite-horizon cost functional in eq. (3) can be re-written as

$$f^{\mu}(x) = \mathbb{E}\left[\sum_{l=0}^{\infty} \beta^{t_l}\left(\bar{c}(x_{t_l}, \mu(x_{t_l})) + \beta^{\tau(x_{t_l})} O\right) \middle| x_0 = x\right]. \tag{6}$$

Define the optimal cost for Problem 1 as

$$V_{st}(x) \coloneqq \inf_{\mu \in \Phi_{st}} f^{\mu}(x), \tag{7}$$

where Φ_{st} is the set of all self-triggered policies. In the following theorem, we state the DP equation for $V_{st}(\cdot)$.

Theorem 1. *The value function defined by eq. (7) satisfies the following dynamic programming equation:*

$$V_{st}(x) = \inf_{a \in \mathcal{A}, \Delta t \in \mathcal{T}} \mathbb{E}\left[\sum_{t=0}^{\Delta t - 1} \beta^t c(x_t, a) + \beta^{\Delta t}\left(V_{st}(x_{\Delta t}) + O\right) \middle| x_0 = x, a_t = a, \forall t < \Delta t\right], \tag{8}$$

for all $x \in \mathcal{X}$. If there exists a policy $\mu^* = (\tau^*, \pi^*)$ such that

$$V_{st}(x) = \mathbb{E}\left[\sum_{t=0}^{\tau^*(x) - 1} \beta^t c(x_t, \pi^*(x)) + \beta^{\tau^*(x)}\left(V_{st}(x_{\tau^*(x)}) + O\right) \middle| x_0 = x, a_t = \pi^*(x), \forall t < \Delta t\right],$$

for all $x \in \mathcal{X}$, then μ^* is an optimal policy for Problem 1.

Proof. See Appendix A. □

Remark 2. *The DP equation in eq. (8) includes the consolidated state costs, $\sum_{t=0}^{\Delta t - 1} \beta^t c(x_t, a)$, which is the accumulated costs incurred from the current update time instance to the next update time instance, the cost-to-go after Δt-steps of lookahead, $\beta^{\Delta t} V(x_{\Delta x})$, and the penalty for a new update $\beta^{\Delta t} O$. Based on the current measurement x, the DP equation has Δt-steps of lookahead. The number of steps Δt is optimized in order to balance the trade-off between the system performance and the update penalty. Thus, we refer to eq. (8) as the DP equation with optimized lookahead. The optimized number of lookahead steps is the optimal waiting time for next triggering given the penalty O. When $O = 0$, the DP equations gives $V_{st}(x) = V(x), \forall x \in \mathcal{X}$, i.e., the value function is the same as the one in classic MDPs.*

Remark 3 (Computational Methods)**.** *One can resort to methods such as the usual value iteration or the policy iteration [21] to solve the DP equation. In the value iteration approach, given the k-th estimate of the value function, $V_{st,k}(\cdot)$, the next estimate $V_{st,k+1}$ can be computed using eq.* (8)*. Repeat this process until it converges to the fixed-point of eq.* (8)*. The convergence is guaranteed for any given $V_{st,0}$, when $\beta < 1$, in view of the Banach fixed-point theorem (see Theorem 6.2.3. of [21]). And the convergence rate is guaranteed to be $\|V_{st,k} - V_{st}\| \leq (\beta^k/(1-\beta))\|V_{st,0} - V_{st,1}\|$. The actual convergence speed should be faster than the above rate depending on what the update penalty O is.*

With Theorem 1, we can compute the value function $V_{st}(\cdot)$ and the optimal self-triggering policy μ^*. The computation of $V_{st}(\cdot)$ and μ^* is usually off-line, and then μ^* is deployed for online implementation. In the next sub-section, we propose a greedy policy that solves Problem 2, i.e., a policy that reduces the number of updates while maintaining a certain level of sub-optimality.

B. Performance Guaranteed Self-Triggering Policies

In this sub-section, we propose a greedy self-triggering policy μ that achieves the inequality defined in eq. (4). To present the policy, we begin with the following lemma.

Lemma 1. *If a self-triggering policy $\mu = (\pi, \tau)$ achieves the following inequality*

$$\mathbb{E}\left[\sum_{t=0}^{\tau(x)-1} \beta^t c(x_t, \pi(x)) + \alpha\beta^{\tau(x)} V(x_{\tau(x)}) \middle| x_0 = x\right] \leq \alpha V(x), \tag{9}$$

for all $x \in \mathcal{X}$, then we have $v^{\mu}(x) \leq \alpha V(x)$.

Proof. See Appendix B. □

Lemma 1 offers us a convenient way to find an policy that achieves the performance level specified by $\alpha V(x)$ for all $x \in \mathcal{X}$ and for $\alpha \geq 1$. Since the agent aims to reduce the amount of sensing/communication resources (the rate of updating), he/she needs to find, for each $x \in \mathcal{X}$, the maximum $\Delta t_x \in \mathcal{T}$ such that there exists at least an action $a_x \in \mathcal{A}$ so that eq. (9) is satisfied with $\tau(x)$ and $\pi(x)$ replaced by Δt_x and a_x respectively. Then, Problem 2 becomes solving the following problem for each $x \in \mathcal{X}$

$$\begin{aligned} &\max_{\Delta t_x \in \mathcal{T}, a_x \in \mathcal{A}} \Delta t_x \\ &s.t. \quad \text{eq. (9)}, \end{aligned} \tag{10}$$

where in eq. (9), we replace $\tau(x)$ and $\pi(x)$ with Δt_x and a_x respectively.

Theorem 2. *If there exists an optimal policy ϕ^* for the classic MDP such that $v^{\phi^*} = V(x)$, then for any fixed $\alpha \geq 1$, there always exist a feasible set for (10), i.e., the problem (10) is well-defined.*

Proof. See Appendix C. □

Remark 4 (The Greedy Choice Property)**.** *Note that the self-triggering policy for Problem 2 follows the greedy rule. At time t_l. the next update time $t_{l+1} = t_l + \tau(x_{t_l})$ is maximized while ensuring eq.* (9) *without considering the effect of this choice on the number of future updates after t_{l+1}. Different from the greedy policy, the self-triggering policy μ^* from Theorem 1 for solving Problem 1 follows the dynamic programming rule, i.e., current choices are made taking into account the influence of current choices on the future possibilities.*

So far in this section, we have developed Theorem 1 and Theorem 2 to help find the self-triggering policies that can solve Problem 1 and Problem 2. The theorems were developed without specifying the state space $\mathcal{X}$, the action space $\mathcal{A}$, and the transition probabilities, except that we require $\mathcal{X}$ to be Polish and $c(\cdot,\cdot)$ to be bounded and non-negative on $\mathcal{X} \times \mathcal{A}$. Hence, The results are applicable to a variety of models such as LQG control [2], [5], [14], inventory control [8], and queueing systems [9], [22]. The two theorems pave the way for the computation and implementation of the self-triggering policies for various Markov decision processes. In the next section, we present a gridworld example to illustrate the computation and implementation of self-triggering policies using Theorem 1 and Theorem 2.

IV. Computation and Implementation: A Gridworld Case Study

In this section, we consider a rectangular gridworld representation of a simple MDP for illustration purposes. The gridworld environment made up of 4×6 cells is shown in Fig. 1, where grey areas are walls. An agent lives in this gridworld aiming to navigate from the start cell to the target cell. The states, representing the cell the agent lives in, are $\mathcal{X} = \{1, 2, \cdots, 19, 20\}$. There are four actions possible at each state, $\mathcal{A} = \{north, south, east, west\}$. Walls block the agent's path. The actions that would take the agent off the grid or into the walls in fact leave the state unchanged. State $x = 20$ is an absorbing state such that once the agent reaches the target cell, he/she enters the absorbing state with probability one (w.p.1). The agent aims to reach the target as fast as possible. Hence, we define

$$c(x,a) = \begin{cases} 10, & \text{if } x \in \mathcal{X}\backslash\{19,20\}, \\ 0, & \text{if } x \in \{19,20\}. \end{cases} \tag{11}$$

A. A Non-Windy Gridworld

We first consider a non-windy setting where each action deterministically causes the agent to move one cell in the respective direction. Let P^d denotes the transition probabilities in a non-wind setting. For instance, we have $P^d(6|1, north) = 1$. We consider the discount factor $\beta = 0.95$, and the bound on the waiting time for the next update is $\bar{T} = 6$. The update penalty O is subject to change.

We set the initial value function estimate to be $V_{st,0}(x) = 0, \forall x \in \mathcal{X}$. We conduct value iteration using the DP equation

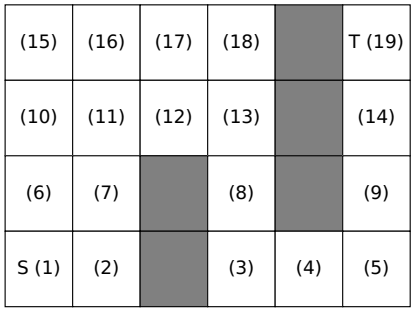


Fig. 1: A gridworld example: Grey areas represent walls, S stands for the start cell, T denotes the target cell, and the integers in the brackets are the indices of states.

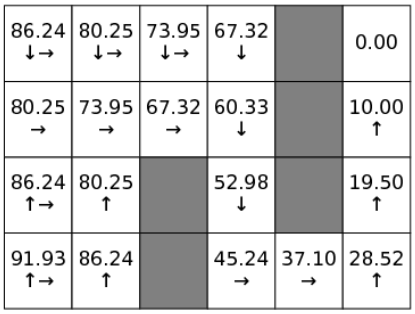


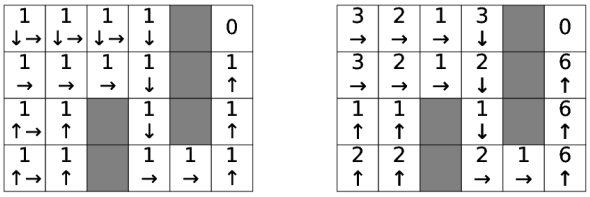
Fig. 2: A non-windy gridworld: The value $V(x)$ (the upper value) and the optimal action $\phi^*(x)$ (the lower pointers) in the classic MDP for each $x \in X$.

with controlled lookahead in eq. (8):

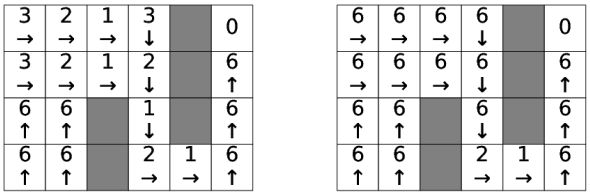
$$V_{st,k+1}(x) = \min_{a\in\mathcal{A},\Delta t\in\mathcal{T}} \mathbb{E}\Bigg[\sum_{t=0}^{\Delta t-1} \beta^t c(x_t,a) + \beta^{\Delta t}\big(V_{st,k}(x_{\Delta t}) + O\big)\Big| x_0 = x, a_t = a, \forall t < \Delta t\Bigg],$$

where every term in the expectation operator can be computed using transition probabilities P^d. The iteration stops when $\|V_{st,k+1} - V_{st,k}\| \le 10^{-5}$, and the results show that the tolerance can be achieved within 25 iterations for every update penalties O we study in this paper.

In Fig. 3, we present the optimal triggering time $\tau(x)^*$ and the optimal control policy $\pi^*(x)$ for each state when the update penalties are $O = 0, 0.1, 40, 80$. As we can see from Fig. 3 (a), when $O = 0$, since there is no update penalty, the optimal triggering time is to update every time, i.e., $\tau^*(x) = 1, \forall x \in \mathcal{X}$, and the optimal control policy is the same as its counterpart in a classic setting, i.e., $\pi^*(x) = \phi^*(x), \forall x \in \mathcal{X}$. The policy offers three paths from the start cell to the target cell: $1 \to 2 \to 7 \to 11 \to \cdots \to 19$, $1 \to 6 \to 7 \to 11 \to \cdots \to 19$, and $1 \to 6 \to 10 \to 11 \to \cdots \to 19$. Each path



(a) The update penalty $O = 0$. (b) The update penalty $O = 0.1$.



(c) The update penalty $O = 40$. (d) The update penalty $O = 80$.

Fig. 3: A non-windy gridworld: The optimal triggering time policy $\tau^*(x)$ (the upper value) and the optimal control policy $\pi^*(x)$ (the lower pointers) for each $x \in \mathcal{X}$ under different update penalties O. (For Problem 1)

takes 12 steps to complete, covers 13 cells, and there are 12 updates.

Suppose a remote controller controls the agent, and the communication between them is expensive. Each communication/update induces an update penalty O. When $O = 0.1$, as is shown in 3 (b), an update is only triggered when there is a need to update the action. For example, when the agent is at state $x = 1$ at time 0, the optimal control policy is heading north, and the optimal waiting time is 2 steps. That means at time $t = 0$, the agent communicates with the controller and is commanded to go north and fix this action for 2 time steps, after which a new update will be sent. Since there is a straight path to the target cell, in a non-windy setting, at states $x = 8, 9, 14$, the controller chooses the maximum allowed waiting time $\bar{T} = 6$. There are few points worth noticing when we compare Fig. 3 (a) and (b): First, when the update penalty $O = 0.1$, the optimal policy, as is shown in Fig. 3, provides one shortest path to the target cell: $1 \to 6 \to 10 \to 11 \to \cdots \to 19$. The path takes 12 time steps to complete, which is the same as when $O = 0$. However, the updates are only triggered when the agent was at states $x = 10, 16, 3, 5$. Hence, the self-triggering policy under $O = 0.1$ requires only 4 updates to achieve the same shortest path as the classic optimal policy. That means the self-triggering policy saves $(12 - 4)/12 = 66.47\%$ of the communication resources required in a classic policy. Second, when the update penalty is $O = 0.1$, at state $x = 1$, going west is no longer an optimal choice since going west requires more updates (5 in this case) to achieve the shortest path. Third, in Fig. 3 (b), the optimal triggering time and the corresponding optimal control at each state always take the agent to the next turning points. For instance, at $x = 10$,

the optimal action is to go east and to fix this direction for 3 steps. This optimal action and optimal waiting time take the agent to state 13, where the agent has to turn south to reach the target cell. There are two reasons to explain this phenomenon: 1. the update penalty is relatively low, compared with the stage cost defined in eq. (11), so that achieving the shortest path within the minimum number of steps is still a priority. 2. In a non-windy setting, the actions deterministically move the agent toward the desired direction, which means the controller can anticipate the agent's trajectory in future steps. Hence, no update is needed between the two turning points.

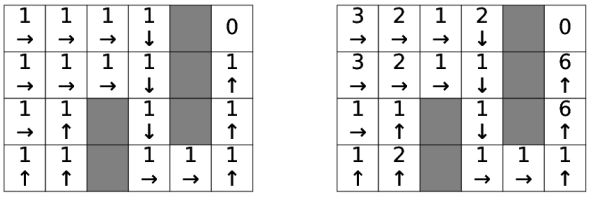
The computed self-triggering policy under $O = 40$ is provided in Fig. 3 (c). The self-triggering policy gives a longer path to reduce the overhead of updating: $1 \rightarrow \cdots \rightarrow 15 \rightarrow \cdots \rightarrow 18 \rightarrow 3 \rightarrow \cdots \rightarrow 19$, which takes 17 time steps (stay at state 15 for 4 time steps due to 6 time steps of going north without update), covers 15 cells, and requires 4 updates to complete. Even though the self-triggering policy requires the same number of updates as the case when $O = 0.1$, the updates are triggered later than their counterparts in the case of $O = 0.1$. Hence, the updates produce less costs due to the discount effect. As the update penalty increases to $O = 80$ (see Fig. 3 (d)), the optimal time policy at most of the states becomes to wait as long as possible for next update, i.e., $\tau^*(x) = \bar{T}$, for $x \in \mathcal{X}\backslash\{3, 4\}$.

B. A Windy Gridworld

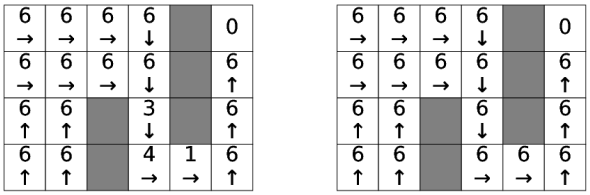
Next, we consider a windy gridworld where the wind takes the agent north 10% of the chance and west 10% of the chance. And 80% of the time, the agent's movement follows its action. In the windy gridworld, the effect of boundaries and walls still applies. The transition probability in a windy setting is defined by P^w. For example, if the agent is at state $x = 11$ and chooses to go east, we have $P^w(12|11, east) = 0.8$, $P^w(10|11, east) = 0.1$, and $P^w(16|11, east) = 0.1$. We run value iterations using the DP equation with controlled lookahead given in eq. (8) under the transition probabilities P^w in the windy environment.

The optimal timing policy and optimal control policy are presented in Fig. 4. One difference in a windy environment is that the control chosen will not deterministically cause the movement of the agent. That means if there is no update, the controller needs to estimate the agent's trajectory, and there exists an estimation error. Hence, we hypothesize that the agent needs to trigger the update more frequently than in a non-windy environment to know his/her location and then adjust his/her control.

Fig. 4 (a) presents the case when there is no update penalty, i.e., $O = 0$. The optimal timing policy is to observe/update every step. The control at state $x = 6$ becomes going east to avoid being taken to the northwest corner by the wind. At states $x = 15, 16, 17$, going south is not an optimal control anymore since if the agent goes south, there is a chance that the wind would take the agent back to the north. When the update penalty is small, i.e., $O = 0.1$, the optimal policy is listed in Fig. 4 (b). There are two points worth mentioning



(a) The update penalty $O = 0$. (b) The update penalty $O = 0.1$.



(c) The update penalty $O = 40$. (d) The update penalty $O = 80$.

Fig. 4: A windy gridworld: The optimal triggering time policy $\tau^*(x)$ (the upper value) and the optimal control policy $\pi^*(x)$ (the lower pointers) for each $x \in \mathcal{X}$ under different update penalties O. (For Problem 1)

when we compare the windy setting and the non-windy setting:

1) When $O = 0.1$, the agent updates more frequently in a windy setting. For example, at $x = 5$, the agent will update the next step in a windy setting, while the agent will update 6 steps later in a non-windy setting. One of the reasons is that in a windy setting, the agent has to update in the next step to make sure he/she goes to state $x = 9$ instead of being blown by the wind to state $x = 4$. This result backs up our hypothesis that the agent in a windy world needs to trigger the update more frequently than in a non-windy environment.
2) When $O = 40$, Fig. 4 (c) shows some interesting and unexpected results. The agent waits longer for the next update in a windy setting than in a non-windy setting shown in Fig. 3 (c). This result contradicts our hypothesis that the agent tends to update more frequently in a noisy environment. For example, if at time t, the agent is at state 11, the next time the agent will update is $t+6$, which is longer than its counterpart in Fig. 3 (c). One explanation is that since the control is to head east, and the wind pushes the agent north or west, there is no need for the agent to update its action. Eventually, the agent will be more likely to be at state 18 or 13 after 6 steps of fixing his/her control of going east.

When $O = 80$, the optimal time policy at every step increases to the maximum allowed waiting time $\bar{T} = 6$ to reduce the update penalties.

C. Performance Guaranteed Policies

In the previous subsections, we solve Problem 1 in the context of a gridworld and obtains the optimal self-triggering policy $\mu^* = (\tau^*, \pi^*)$. Just to remind that we have $\phi : \mathcal{X} \to \mathcal{A}$, which is the policy in the classic setting, and the self-triggering policy $\mu : \mathcal{X} \to \mathcal{T} \times \mathcal{A}$ in self-triggered MDPs. To differentiate the self-triggering policy we obtain for Problem 1 and the policy for Problem 2, we name them $\mu_1^* = (\tau_1^*, \pi_1^*)$ and $\mu_2^* = (\tau_2^*, \pi_2^*)$ respectively.

The self-triggering policy μ_1^* is optimal with respect to a specified update penalty O. However, it does not provide an explicit performance guarantee under the original cost criterion. Instead, the self-triggering policy μ_2^* provides a pre-specified level of performance guarantee.

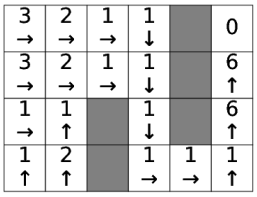
As a result of the discussions in Section III-B, the steps to compute a self-triggering policy μ_2^* for Problem 2 is given as follows:

1) Compute the value function $\{V(x), x \in \mathcal{X}\}$ of the MDP in the classic setting.
2) For each $x \in \mathcal{X}$, select $\Delta t_x = \bar{T}$.
3) Compute

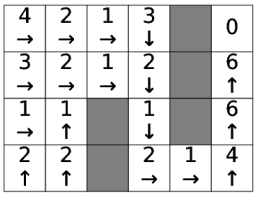
$$\tilde{V}(x) = \min_{a \in \mathcal{A}} \mathbb{E}\left[\sum_{t=0}^{\Delta t_x - 1} \beta^t c(x_t, a) + \alpha \beta^{\Delta t_x} V(x_{\Delta t_x}) \middle| x_0 = x\right],$$
$$a_x^* = \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[\sum_{t=0}^{\Delta t_x - 1} \beta^t c(x_t, a) + \alpha \beta^{\Delta t_x} V(x_{\Delta t_x}) \middle| x_0 = x\right]. \tag{12}$$

4) If $\tilde{V}(x) > \alpha V(x)$, set $\Delta t_x = \Delta t_x - 1$, repeat step 3). Otherwise, $\tau_2^*(x) = \Delta_x$, $\pi_2^*(x) = a_x^*$.

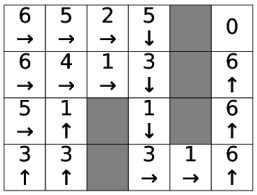
The optimization problem in eq. (12) admits a closed-form solution for models such as LQG control [2] and inventory control [8]. For the windy gridworld model, we compute self-triggering policies following the steps for various levels of sub-optimality. The results are presented in Fig. 5. As we can see from Fig. 5 (a), the self-triggering policy μ_2^* can achieve a full level of optimality, i.e., $v^{\mu_2^*}(x) = V(x), \forall x \in \mathcal{X}$, while requiring less communication/sensing resources. When the level of sub-optimality is $\alpha = 1.1$, as one can see from Fig. 5 (b), at most states, the optimal timing policy is to wait for two or more than two steps for the next update. That means the self-triggering policy μ_2^* can save more than 50% communication/sensing resources while suffering only 10% of performance degradation. If one can tolerate a higher level of degradation, one can set α to a higher value and compute the corresponding self-triggering policy μ_2^*. The cases when $\alpha = 1.4$ and $\alpha = 2$ are presented in Fig. 5 (c) and (d). As one expects, the higher α is (more performance degradation one can tolerate), the fewer updates needed (less communication/resources consumed).



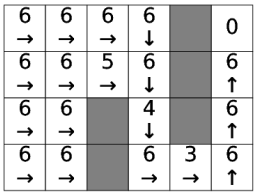
(a) The pre-specified level of sub-optimality penalty $\alpha = 1$.



(b) The pre-specified level of sub-optimality penalty $\alpha = 1.1$.



(c) The pre-specified level of sub-optimality penalty $\alpha = 1.4$.



(d) The pre-specified level of sub-optimality penalty $\alpha = 2$.

Fig. 5: A windy gridworld: The optimal triggering time policy $\tau_2^*(x)$ (the upper value) and the optimal control policy $\pi_2^*(x)$ (the lower pointers) for each $x \in \mathcal{X}$ under various sub-optimality requirements. (For Problem 2)

.

V. Conclusions

In this paper, two self-triggering policies are obtained by proposing two frameworks that convey two different philosophies. Problem 1 introduces a soft constraint, i.e., a update penalty that penalizes frequent use of communication resources and Problem 2 applies a hard constraint on the level of sub-optimality while maximizing the triggering time to resources consumption. Both policies are shown to be effective in reducing the use of communication resources in the gridworld examples. Future endeavors can focus on developing stability guarantees of self-triggered policy for controlled Markov chain, and learning when to trigger, i.e., leveraging reinforcement learning techniques for unknown MDP models [23].

Appendix

A. Proof of Theorem 1

Proof. We prove the theorem by constructing a consolidated Markov decision process problem. A close look at eq. (6) shows that this is a discounted cost discrete-time MDP with discount factor β, Markov states and Markov actions given respectively by

$$X_l = (x_{t_l}, \tilde{t}_l) \in \mathcal{X} \times \{0, 1, 2, \cdots\},$$
$$A_l = (a_{t_l}, \Delta t) \in \mathcal{A} \times \mathcal{T},$$

where $\tilde{t}_l = t_l - l$, the state cost equal to

$$C(X_l, A_l) = \beta^{\tilde{t}_l}\left[\bar{c}(x_{t_l}, a_{t_l}, \Delta t) + \beta^{\Delta t} O\right],$$

and the skip-transition probability defined in eq. (5). Hence, the cost in eq. (6) becomes

$$f^{\mu}(x) = \mathbb{E}\left[\sum_{l=0}^{\infty} \beta^l C(X_l.A_l) \middle| X_l = (x, 0), \mu\right].$$

The consolidated formulation can be treated as a regular Markov decision problem. Note that the Cartesian product of countable countably many polish spaces is still Polish.

Hence, $\mathcal{X} \times \{0, 1, 2, \cdots\}$ is Polish if $\mathcal{X}$ is polish. Thus, the results (mainly the results available to Polish spaces) can be derived from current Markov decision literature [21]. Applying Theorem 6.2.5 and Theorem 6.2.12 of [21], we obtain claims in Theorem 1. □

B. Proof of Lemma 1

Proof. For a given L, let t_{L+1} be the time instance of the $(L+1)$-th update. The accumulated costs before (t_{L+1}) can be written as

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=0}^{t_{L+1}-1} \beta^t c(x_t, a_t)\middle| x_0 = x\right] \\
=&\mathbb{E}\left[\sum_{l=0}^{L} \beta^{t_l} \bar{c}(x_{t_l}, a_{t_l}, t_{l+1} - t_l)\middle| x_0 = x\right] \\
=&\mathbb{E}\left[\sum_{l=0}^{L} \beta^{t_l} \mathbb{E}\left[\sum_{t=t_l}^{t=t_{l+1}-1} \beta^{t-t_l} c(x_t, a_{t_l})\middle| x_{t_l}\right] x_0 = x\right],
\end{aligned} \tag{13}$$

where we use the tower property of conditional expectation to derive the first equality and the second equality follows immediately after some algebraic rearrangements. Suppose at time instance t_l, $l \in \mathbb{N}$, the process is at state x_{t_l}. Let $t_{l+1} = t_l + \tau(x_{t_l})$ and $a_t = \pi(x_{t_l})$ for $t = t_l, t_l + 1, \ldots, t_{l+1} - 1$. From eq. (9), we have

$$\mathbb{E}\left[\sum_{t=t_l}^{t_{l+1}-1} \beta^{t-t_l} c(x_t, a_t)\middle| x_{t_l}\right] \leq \alpha V(x_{t_l}) - \mathbb{E}\left[\alpha \beta^{t_{l+1}-t_l} V(x_{t_{l+1}})\middle| x_{t_l}\right]. \tag{14}$$

Applying eq. (14) into eq. (13) for every $l \leq L$ yields

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=0}^{t_{L+1}-1} \beta^t c(x_t, a_t)\middle| x_0 = x\right] \\
\leq&\mathbb{E}\left[\sum_{l=0}^{L} \beta^{t_l} \alpha \left(V(x_{t_l}) - \beta^{t_{l+1}-t_l} V(x_{t_{l+1}})\right)\middle| x_0 = x\right] \\
=&\alpha \mathbb{E}\left[V(x_{t_0}) - \beta^{t_{L+1}} V(x_{t_{L+1}})\middle| x_0 = x\right] \leq \alpha V(x),
\end{aligned}$$

where we use the fact that $c : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^+$ produces a non-negative $V(\cdot)$, i.e., $V(x) \geq 0, \forall x \in \mathcal{X}$. Since L can be chosen arbitrarily, taking L to infinity, we have $t_l \to \infty$, and by definition of v^μ, $v^\mu(x) \leq \alpha V(x)$ for every $x \in \mathcal{X}$. □

C. Proof of Theorem 2

Proof. To show that there is a feasible set for problem (10), it is sufficient to show that for any $x \in \mathcal{X}$, when $\Delta t_x = 1$, there always exists an action $a_x \in \mathcal{A}$ such that $\mathbb{E}\left[c(x, a_x) + \alpha\beta V(x_1)|x_0 = x, a_0 = a_x\right] \leq \alpha V(x)$. Let ϕ^* be an optimal policy of the classic MDP. Then, by Bellman equation, we have

$$\min_{a \in \mathcal{A}} \mathbb{E}\left[c(x, a) + \beta V(x_1)|x_0 = x, a_0 = a\right] = V(x),$$

where the minimum is attained at $a^* = \phi^*(x)$. That means there exists $a_x = \phi^*(x)$ such that

$$\alpha \mathbb{E}\left[c(x, a_x) + \beta V(x_1)|x_0 = x, a_0 = a_x\right] = \alpha V(x).$$

Since $c(\cdot, \cdot)$ is non-negative, we have

$$\mathbb{E}\left[c(x, a_x) + \alpha\beta V(x_1)|x_0 = x, a_0 = a_x\right] = \alpha V(x).$$

Then for every $x \in \mathcal{X}$, there always exists a $\Delta t_x \in \mathcal{T}$ such that we can find an action $a_x \in \mathcal{X}$ so that (9) is satisfied. □

References

[1] W. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *2012 ieee 51st ieee conference on decision and control (cdc)*. IEEE, 2012, pp. 3270–3285.

[2] 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.

[3] A. Anta and P. Tabuada, "To sample or not to sample: Self-triggered control for nonlinear systems," *IEEE Transactions on automatic control*, vol. 55, no. 9, pp. 2030–2042, 2010.

[4] X. Wang and M. D. Lemmon, "Self-triggered feedback control systems with finite-gain $\mathcal{L}_2$ stability," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 452–467, 2009.

[5] S. Akashi, H. Ishii, and A. Cetinkaya, "Self-triggered control with tradeoffs in communication and computation," *Automatica*, vol. 94, pp. 373–380, 2018.

[6] Y. Gao, P. Yu, D. V. Dimarogonas, K. H. Johansson, and L. Xie, "Robust self-triggered control for time-varying and uncertain constrained systems via reachability analysis," *Automatica*, vol. 107, pp. 574–581, 2019.

[7] S. Yuvaraj and M. Sangeetha, "Smart supply chain management using internet of things(iot) and low power wireless communication systems," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2016, pp. 555–558.

[8] E. A. Feinberg, "Optimality conditions for inventory control," in *Optimization Challenges in Complex, Networked and Risky Systems*. INFORMS, 2016, pp. 14–45.

[9] P. Więcek, E. Altman, and A. Ghosh, "Mean-field game approach to admission control of an $m\backslash m\backslash\infty$ queue with shared service cost," *Dynamic Games and Applications*, vol. 6, no. 4, pp. 538–566, 2016.

[10] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[11] W. Lu, F. Fan, J. Chu, P. Jing, and S. Yuting, "Wearable computing for internet of things: A discriminant approach for human activity recognition," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2749–2759, 2019.

[12] A. Molin and S. Hirche, "On the optimality of certainty equivalence for event-triggered control systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 2, pp. 470–474, 2013.

[13] D. Maity and J. S. Baras, "Optimal event-triggered control of nondeterministic linear systems," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 604–619, 2019.

[14] Y. Huang and Q. Zhu, "Infinite-horizon linear-quadratic-gaussian control with costly measurements," *arXiv preprint arXiv:2012.14925*, 2020.

[15] D. P. Bertsekas and S. Shreve, *Stochastic optimal control: the discrete-time case*. Athena Scientific,Belmont,MA, 1996.

[16] M. Gallieri and J. M. Maciejowski, "l_{asso} mpc: Smart regulation of over-actuated systems," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 1217–1222.

[17] C. Cooper and N. Hahi, "An optimal stochastic control problem with observation cost," *IEEE Transactions on Automatic Control*, vol. 16, no. 2, pp. 185–189, 1971.

[18] Y. Huang and Q. Zhu, "Cross-layer coordinated attacks on cyber-physical systems: A lqg game framework with controlled observations," *European Control Conference*, 2021.

[19] ——, "A pursuit-evasion differential game with strategic information acquisition," *arXiv preprint arXiv:2102.05469*, 2021.

[20] R. Durrett, *Probability: theory and examples*. Cambridge university press, 2019, vol. 49.

[21] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[22] Y. Huang, V. Kavitha, and Q. Zhu, "Continuous-time markov decision processes with controlled observations," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, pp. 32–39.

[23] A. Biedenkapp, R. Rajan, F. Hutter, and M. Lindauer, "Towards temporl: Learning when to act," in *Workshop on Inductive Biases, Invariances and Generalization in Reinforcement Learning (BIG@ ICML'20)*, 2020.