Sleep When Everything Looks Fine: Self-Triggered Monitoring for Signal Temporal Logic Tasks

Chuwei Wang, Xinyi Yu, Jianing Zhao, Lars Lindemann, *Member, IEEE*, and Xiang Yin, *Member, IEEE*

***Abstract*—Online monitoring is a widely used technique in assessing if the performance of the system satisfies some desired requirements during run-time operation. Existing works on online monitoring usually assume that the monitor can acquire system information periodically at each time instant, which may be unnecessarily energy-consuming. In this paper, we proposed a novel *self-triggered* mechanism for model-based online monitoring of discrete-time dynamical system under specifications described by signal temporal logic (STL) formulae. Specifically, instead of sampling the system state at each time instant, a self-triggered monitor can actively determine when the next system state is sampled in addition to its monitoring decision regarding the satisfaction of the task. We propose an effective algorithm for synthesizing such a self-triggered monitor that can correctly evaluate a given STL formula on-the-fly while maximizing the time interval between two observations. We show that, compared with the standard online monitor with periodic information, the proposed self-triggered monitor can significantly reduce observation burden while ensuring that no information of the STL formula is lost. Case studies are provided to illustrate the proposed monitoring mechanism.**

***Index Terms*—Task planning, discrete event dynamic automation systems, planning, scheduling and coordination.**

I. Introduction

AUTONOMOUS systems, such as teams of ground robots or unmanned aerial vehicles, have found widespread applications in diverse areas, ranging from search and rescue missions to smart warehouses. The critical nature of these systems lies in their safety implications: any failure to accomplish their designated tasks could result in catastrophic consequences. However, these systems are inherently susceptible to errors due to their intricate logics and complex dynamics. Therefore, monitoring the safety status of the system stands out as a central task during their operations [1], [2]. In particular, one should halt the system and initiate corrective actions once a failure is detected.

To express the formal requirements of autonomous systems, temporal logic, including linear temporal logic (LTL) or signal temporal logic (STL), stands out as one of the most widely employed tools. This is because it offers a rich and structured framework for describing high-level tasks. Particularly, STL provides an effective tool for the quantitative evaluation of real-value signals in real-time settings [3], [4]. Since the seminal work of Maler [5], STL has undergone extensive developments and applications in the analysis and control of numerous safety-critical systems [6], [7], [8], [9].

Online monitoring is a widely adopted lightweight technique for evaluating the correctness of the system in real-time [10], [11], [12]. In contrast to model checking, which necessitates offline enumeration of all possible behaviors, online monitoring involves a monitor that observes the *partial signals* generated up to the current instant. It can be broadly categorized into *model-free* and *model-based*, depending on the knowledge utilized by the monitor. Specifically, in model-free online monitoring [11], [12], correctness evaluation relies solely on the partial signal, neglecting the model information of dynamic systems. More recently, there has been a surge of interest in model-based online monitoring [13], [14], [15], [16], [17], [18]. This approach, leveraging the dynamic information of the system, enables the monitor to make more precise evaluations regarding the satisfaction of the task.

Concerning the implementation of online monitoring, a fundamental question is how online information is obtained by the monitor. However, this crucial aspect has been neglected by existing works, as they implicitly presuppose that the monitor has complete access to the partial signal (state sequence) generated by the system. This assumption implies that the monitor can acquire system information periodically by, e.g., consistently activating sensors at each time instant. Nevertheless, in numerous applications, such a periodic information acquisition mechanism may prove to be unnecessary. Consistently activating sensors can be either energy-consuming or potentially prone to information leakage. This scenario is also applicable in various daily settings. For instance, a factory manager typically exercises greater caution when a machine approaches the safety boundary, and conversely, adopts a more relaxed stance when there are no hazardous surroundings within their field of view.

Motivated by the preceding discussion, this paper introduces a novel *self-triggered* mechanism for online monitoring of discrete-time dynamical systems under STL specifications. We adopt a model-based setting, assuming that the monitor possesses the dynamic model of the system. However, rather than sampling the system state periodically at each time instant, a self-triggered monitor can actively determine when to sample the

Received 11 June 2024; accepted 12 August 2024. Date of publication 6 September 2024; date of current version 16 September 2024. This article was recommended for publication by Associate Editor J.-H. Lee and Editor C.-B. Yan upon evaluation of the reviewers' comments. This work was supported by the National Natural Science Foundation of China under Grant 62173226, Grant 62061136004, and Grant 61833012. *(Corresponding author: Xiang Yin.)*

Chuwei Wang, Jianing Zhao, and Xiang Yin are with the Department of Automation and Key Laboratory of System Control and Information Processing, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: wangchuwei@sjtu.edu.cn; jnzhao@sjtu.edu.cn; yinxiang@sjtu.edu.cn).

Xinyi Yu and Lars Lindemann are with the Thomas Lord Department of Computer Science, University of Southern California, Los Angeles, CA 90089 USA (e-mail: xinyi.yu12@usc.edu; llindema@usc.edu).

Digital Object Identifier 10.1109/LRA.2024.3455848

next system state, alongside making monitoring decisions about the feasibility of the STL task. As a result, the information acquisition module remains silent when critical information about the task status is not required. Based on the assumption that the environment map is static, we present an algorithm for synthesizing such a self-triggered monitor capable of on-the-fly evaluation of a given STL formula. Specifically, the self-triggered monitor aims to maximize the time interval between two observations while ensuring the successful evaluation of the STL task.

Finally, we note that the concept of self-triggered mechanisms has found widespread applications in the literature, aiming to conserve sensor energy [19], reduce communication bandwidth [20], or enhance information security [21]. In [22], the authors applied event-triggered mechanism to control synthesis of STL tasks. However, to our knowledge, the utilization of self-triggered information acquisition mechanisms for the purpose of online monitoring of complex logic tasks has not been explored in the literature. In [23], model information is used to mitigate the issue of sampling uncertainties in monitoring. However, it considers an offline monitoring setting. Here we use model information for the purpose of predictions rather than interpolations. In the context of predictive online monitoring under partial observation, state estimation processes as well as reachability analysis are also involved [24], [25], [26]. However, no event-triggered mechanism is considered therein.

II. Preliminaries

A. System Model

We consider a discrete-time dynamic system of form

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

where $x_t \in \mathcal{X} \subseteq \mathbb{R}^n$ and $u_t \in \mathcal{U} \subseteq \mathbb{R}^m$ are the system state and the input at time t, respectively, and $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the dynamic function. We assume that the initial state is fixed as $x_0 \in \mathcal{X}$. Given a sequence of inputs $\mathbf{u}_{0:T-1} = u_0 u_1 \cdots u_{T-1} \in \mathcal{U}^T$, the resulting trajectory of the system is the state sequence $\xi(x_0, \mathbf{u}_{0:T-1}) = \mathbf{x}_{1:T} = x_1 \cdots x_T$ such that $x_{i+1} = f(x_i, u_i), i = 0, \ldots, T-1$. Note that our approach treats control inputs from the system user and additional disturbances in the same manner mathematically.

B. Signal Temporal Logic

The specifications of the system are described by STL formulae with bounded-time [5], whose syntax is as follows

$$\psi ::= \top \mid \pi^\mu \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \mathbf{U}_{[a,b]} \psi_2,$$

where $\top$ is the true predicate, π^μ is a predicate whose truth value is determined by the sign of function $\mu : \mathbb{R}^n \to \mathbb{R}$, i.e., π^μ is true iff $\mu(x) > 0$. Notations $\neg$ and $\wedge$ are the standard Boolean operators "negation" and "conjunction", respectively, which can further induce "disjunction" by $\psi_1 \vee \psi_2 := \neg(\neg\psi_1 \wedge \neg\psi_2)$ and "implication" by $\psi_1 \to \psi_2 := \neg\psi_1 \vee \psi_2$. Notation $\mathbf{U}_{[a,b]}$ is the temporal operator "until", where $a, b \in \mathbb{R}_{\geq 0}$ are two time instants. One can also induce temporal operators "eventually" and "always" by $\mathbf{F}_{[a,b]}\psi := \top \mathbf{U}_{[a,b]} \psi$ and $\mathbf{G}_{[a,b]} := \neg\mathbf{F}_{[a,b]}\neg\psi$, respectively.

Given a sequence $\mathbf{x}$, we use notation $(\mathbf{x}, t_0) \models \psi$ to denote the satisfaction for STL formulae ψ at time t_0. The semantics of STL are inductively defined as follows:

$$\begin{array}{lll} (\mathbf{x}, t) \models \pi^\mu & \text{iff} & \mu(\mathbf{x}(t)) > 0 \\ (\mathbf{x}, t) \models \neg\psi & \text{iff} & \neg((\mathbf{x}, t) \models \psi) \\ (\mathbf{x}, t) \models \psi_1 \wedge \psi_2 & \text{iff} & (\mathbf{x}, t) \models \psi_1 \wedge (\mathbf{x}, t) \models \psi_2 \\ (\mathbf{x}, t) \models \psi_1 \mathbf{U}_{[a,b]} \psi_2 & \text{iff} & \exists t' \in [t+a, t+b] : (\mathbf{x}, t') \models \psi_2 \\ & & \text{and } \forall t'' \in [t, t'] : (\mathbf{x}, t'') \models \psi_1 \end{array}$$

We write $\mathbf{x} \models \psi$ whenever $(\mathbf{x}, 0) \models \psi$. The readers are referred to [5] for more details on the semantics of STL.

In the original semantics of $\mathbf{U}_{[a,b]}$, formula ψ_1 needs to be satisfied within interval $[0, a]$. For the sake of convenience, we use new "until" operator $\mathbf{U}'$ by slightly modifying the semantics as: $(\mathbf{x}, t) \models \psi_1 \mathbf{U}'_{[a,b]} \psi_2$ iff

$$[\exists t' \in [t+a, t+b] : (\mathbf{x}, t') \models \psi_2] \wedge [\forall t'' \in [t+a, t'] : (\mathbf{x}, t'') \models \psi_1]$$

Note that, we can express the original until operator $\mathbf{U}$ as $\psi_1 \mathbf{U}_{[a,b]} \psi_2 = \mathbf{G}_{[0,a]} \psi_1 \wedge \psi_1 \mathbf{U}'_{[a,b]} \psi_2$. Hereafter in the paper, we will use this modified version of until operator, and rewrite it directly as $\psi_1 \mathbf{U}_{[a,b]} \psi_2$ by omitting the superscript.

Finally, for predicate π^μ, its satisfaction region is $\mathcal{H}^\mu := \{x \in \mathcal{X} \mid \mu(x) \geq 0\}$ with $\mathcal{H}^{\neg\varphi} = \mathcal{X} \setminus \mathcal{H}^\varphi$ and $\mathcal{H}^{\varphi_1 \wedge \varphi_2} = \mathcal{H}^{\varphi_1} \cap \mathcal{H}^{\varphi_2}$. Therefore, for any Boolean formula φ, we can express its requirement equivalently by $x \in \mathcal{H}^\varphi$.

C. Fragment of STL Formulae

In this work, we consider a fragment of STL formulae such that the overall task is expressed as the conjunction of N sub-formulae without nested temporal operators. Formally, we consider an STL formula of form:

$$\Phi = \bigwedge_{i=1,\ldots,N} \Phi_i, \tag{2}$$

where each Φ_i is either

$$\text{(i) } \mathbf{G}_{[a_i,b_i]} x \in \mathcal{H}_i \quad \text{or} \quad \text{(ii) } x \in \mathcal{H}_i^1 \mathbf{U}_{[a_i,b_i]} x \in \mathcal{H}_i^2.$$

Let $\mathcal{I} = \{1, \ldots, N\}$ be the index set of sub-formulae. We assume that the indices are ordered based on the beginning instant of each sub-formulae, i.e., $a_1 \leq \cdots \leq a_N$. For each time instant t, we denote by $\mathcal{I}_t = \{i \mid a_i \leq t \leq b_i\} \subseteq \mathcal{I}$ the index set of formulae that are effective at time t. Similarly, we define the index sets of sub-formulae effective before and after t by $\mathcal{I}_{<t} = \{i \mid t > b_i\}$ and $\mathcal{I}_{>t} = \{i \mid t < a_i\}$, respectively. For each sub-formulae $i \in \mathcal{I}$, we denote by $\mathbf{O}_i \in \{\mathbf{G}, \mathbf{U}\}$ the unique temporal operator in Φ_i. We define $\mathcal{I}_t^{\mathbf{U}} = \{i \in \mathcal{I}_t \mid \mathbf{O}_i = \mathbf{U}\}$; the same for $\mathcal{I}_t^{\mathbf{G}}$.

D. Remaining Formulae and Feasible Sets

As the system evolves, some sub-formulae may be satisfied or not effective anymore. Throughout the paper, we will use notation $I \subseteq \mathcal{I}$ to denote the index set for those sub-formulae remaining unsatisfied. If $I \cap \mathcal{I}_{<t} = \emptyset$, we define the *$I$-remaining*

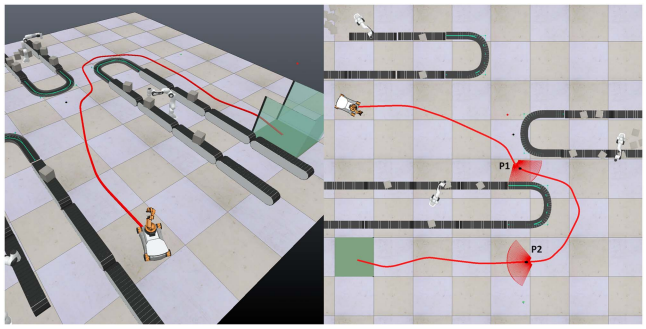


Fig. 1. Motivating example of robot charging mission.

formula at instant t in the form of

$$\hat{\Phi}_t^I = \bigwedge_{i \in I \cap \mathcal{I}_t} \Phi_i^{[t,b_i]} \wedge \bigwedge_{i \in \mathcal{I}_{>t}} \Phi_i, \tag{3}$$

where $\Phi_i^{[t,b_i]}$ is attained from $\Phi_i^{[a_i,b_i]}$ by replacing the starting instant of the temporal operator from a_i to t. Then the *I-remaining feasible set* at instant t, denoted by X_t^I, is defined as the set of states from which the system can possibly satisfy the I-remaining formula, i.e.,

$$X_t^I = \left\{ x_t \in \mathcal{X} \;\middle|\; \begin{array}{c} \exists\, \mathbf{u}_{t:T-1} \in \mathcal{U}^{T-t} \\ \text{s.t. } x_t \xi_f(x_t, \mathbf{u}_{t:T-1}) \models \hat{\Phi}_t^I \end{array} \right\}. \tag{4}$$

Otherwise, if $I \cap \mathcal{I}_{<t} \neq \emptyset$, it means that there are some subformulae that should have been satisfied. In this scenario, we define $X_t^I = \emptyset$ since the overall task is already failed.

Remark 1: Although the STL fragment does not support arbitrary nested temporal operators, it is still more general than the standard reach-avoid task. In particular, it can capture multiple reach-avoid specifications without a predefined reachability order. Many recent works also focus on this fragment; see, e.g., [27], [28].

III. Model-Based Online Monitoring

A. Standard Online Monitor With Periodic Sampling

In the context of online monitoring, the system state is observed by a *monitor* that determines the satisfaction of the STL task dynamically online based on the trajectory generated by the system. Specifically, given an STL formula Φ with time horizon T and a (partial) signal $\mathbf{x}_{0:t} = x_0 x_1 \dots x_t$ (also called *prefix*) up to time instant $t < T$, we say $\mathbf{x}_{0:t}$ is

- *violated* if $\mathbf{x}_{0:t}\xi(x_t, \mathbf{u}_{t:T-1}) \not\models \Phi$ for any $\mathbf{u}_{t:T-1}$;
- *feasible* if $\mathbf{x}_{0:t}\xi(x_t, \mathbf{u}_{t:T-1}) \models \Phi$ for some $\mathbf{u}_{t:T-1}$.

Then an online monitor is a function

$$\mathcal{M} : \mathcal{X}^* \to \{0, 1\} \tag{5}$$

such that, for any prefix $\mathbf{x}_{0:t}$, we have

$$\mathcal{M}(\mathbf{x}_{0:t}) = 1 \quad \Leftrightarrow \quad \mathbf{x}_{0:t} \text{ is violated}, \tag{6}$$

where $\mathcal{X}^*$ denotes the set of all finite sequence over $\mathcal{X}$.

The above defined monitor is referred to as the *periodic monitor* hereafter since it needs to acquire state information x_t periodically at each time instant. In [18], an effective algorithm has been proposed to synthesize a model-based periodic monitor involving both offline computations and online execution. Specifically, for the offline stage, one first uses the model information to pre-compute all possible I-remaining feasible sets that may incur for each time instant. Then for the online execution, one simply tracks the index set I for the remaining formulae, and the monitoring decision can be made by checking whether or not $x_t \in X_t^I$.

B. Motivating Example for Aperiodic Sampling

Before formally formulating the problem, let us consider a motivating example. We consider a nonholonomic unicycle mobile robot working in a factory. It needs to arrive at the charging station, marked by the green region in Fig. 1, in 10 minutes while avoiding the conveyors on its path.

Clearly, a periodic monitor can successful monitor the task by continuously checking whether or not the robot hits the obstacle or reaches the charging station on time. However, it may require unnecessary observations. For example, consider a possible trajectory shown in Fig. 1 and two location points $P1$, $P2$ on it. The sector region around each point is the area the robot can reach within 2 minutes according to its dynamic. If the robot is at $P1$, then it may hit a conveyor in 2 minutes. Meanwhile, when the robot is at $P2$, no collision is possible in 2 minutes. Therefore, to reduce the cost of sensor deployment, the monitor can "sleep" for at least 2 minutes, instead of constantly collecting irrelevant data.

IV. Self-Triggered Monitoring Mechanism

In this section, we formally present the proposed self-triggered monitoring mechanism. Compared with the periodic monitor that passively receives state information at each time instant, a self-triggered monitor will *actively* determine the next time instant for acquiring state information. Specifically, a self-triggered monitor works as follows:

- At each decision instant t (which is determined at the previous decision round), it observes the current state x_t by, e.g., turning on sensors, and makes a monitoring decision 0 or 1 regarding the violation of the STL task;
- Together with the binary monitoring decision, it also determines a number τ specifying after how many time instants the next observation will be made;
- Then the system evolves according to its own dynamic "silently" to the monitor until time instant $t + \tau$, at which the above two steps are repeated.

To formalize the above process, we define an *observation history* as a sequence of states with time stamps of form

$$\hbar := (x_0, \tau_0)(x_1', \tau_1) \dots (x_{n-1}', \tau_{n-1}) x_n' \in (\mathcal{X} \times \mathbb{N})^* \mathcal{X}. \tag{7}$$

Specifically, each x_i' denotes the state at the i-th observation and τ_i denotes the time between observations x_i' and x_{i+1}'. Note that, we use prime in x_i' since it is not the system state at time instant i. Then we denote by $\mathbb{H} := (\mathcal{X} \times \mathbb{N})^* \mathcal{X}$ the set of all observation histories.

With the notion of observation histories, now we are ready to formally define the self-triggered monitors.

Definition 1 (Self-Triggered Monitors): Let $T_{max} < T$ be a number specifying the maximum time the monitor allowed to stay silent. A self-triggered monitor is a function

$$\mathcal{M} : \mathbb{H} \to \{0,1\} \times \{1,\dots,T_{max}\} \tag{8}$$

that makes decisions based on the observation histories. Specifically, for any observation history $\hbar \in \mathbb{H}$, we have $\mathcal{M}(\hbar) := (\mathcal{M}_D(\hbar), \mathcal{M}_\tau(\hbar))$, where

- $\mathcal{M}_D(\hbar) \in \{0,1\}$ is the binary monitoring decision with "0" denoting *feasible* and "1" denoting *violated*; and
- $\mathcal{M}_\tau(\hbar) \in \{1,\dots,T_{max}\}$ is the trigger information indicating the time interval to the next observation.

Once $\mathcal{M}_D(\hbar)$ returns 1, the monitoring process terminates.

Clearly, a self-triggered monitor reduces to the standard periodic monitor when $\mathcal{M}_\tau(\hbar)$ is set to be a single time unit constantly. In this case, the monitor will precisely observe the entire trajectory generated by the system. However, in general, the monitor can only obtain partial information from the underlying trajectory, which is captured by the following definition.

Definition 2 (Induced Observation Histories): Let $\mathcal{M}$ be a self-triggered monitor and $\mathbf{x}_{0:t} = x_0 x_1 \dots x_t$ be a state sequence generated by system (1). The *observation history* of $\mathbf{x}_{0:t}$ *induced* by $\mathcal{M}$, denoted by $\hbar_{\mathcal{M}}(\mathbf{x}_{0:t})$, is defined as the unique sequence

$$\hbar_{\mathcal{M}}(\mathbf{x}_{0:t}) = (x_0,\tau_0)(x'_1,\tau_1)\dots(x'_{n-1},\tau_{n-1})x'_n \in \mathbb{H} \tag{9}$$

such that

1) $\forall i \geq 1 : x'_i = x_{\tau_0+\dots+\tau_{i-1}}$;
2) $\forall i \geq 0 : \mathcal{M}_\tau((x_0,\tau_0)(x'_1,\tau_1)\cdots(x'_{i-1},\tau_{i-1})x'_i) = \tau_i$;
3) $\sum_{i=0}^{n-1}\tau_i \leq t < \sum_{i=0}^{n-1}\tau_i + \mathcal{M}_\tau(\hbar_{\mathcal{M}}(\mathbf{x}_{0:t}))$.

The intuitions of the above definition are as follows. The first condition says that states in the induced observation history are sampled from the original state sequence at those decision time stamps. The second condition says that the time stamp for the i-th observation is determined by $\mathcal{M}$ based on the previous observation history. Finally, the last condition says that there is no more observation after time instant t.

Now, we formulate the self-triggered monitor synthesis problem that we solve in this paper as follows.

Problem 1 (Self-Triggered Monitor Synthesis Problem): Given a dynamic system of form (1) and an STL formula Φ as in (2), design a self-triggered online monitor $\mathcal{M} : \mathbb{H} \to \{0,1\} \times \{1,\dots,T_{max}\}$ such that for any prefix signal $\mathbf{x}_{0:t}$, we have $\mathcal{M}_D(\hbar_{\mathcal{M}}(\mathbf{x}_{0:t})) = 1$ if and only if $\mathbf{x}_{0:t}$ is violated.

We conclude this section by making some remarks regarding the above problem formulation.

Remark 2: Under the self-triggered mechanism, only partial signals are available to the monitor. Therefore, it is possible that the system has two distinct trajectories having the same observation. Therefore, Problem 1 essentially requires that the monitor can always distinguish two trajectories such that one is feasible and the other is violated. Furthermore, since the monitor does not receive any information until the next trigger time instant, any signals within this "silent" interval should also result in the same task status.

Remark 3: In Problem 1, we do not explicitly put any optimization requirement for the trigger interval $\mathcal{M}_\tau(\hbar)$ for each decision instant. In principle, one may want $\mathcal{M}_\tau(\hbar)$ to be as large as possible for the purpose of energy saving. Hereafter, our solution algorithm will follow a greedy strategy, i.e., at each decision instant, the monitor tries to maximize the current trigger interval as long as the ambiguity requirement is satisfied without further considering its future effect.

V. Self-Triggered Online Monitoring Algorithm

In this section, we present our solution for synthesizing self-triggered monitor. Specifically, we first introduce the notion of belief states and their evolution. Then an effective online monitoring algorithm is proposed based on the belief states. Finally, we prove the correctness of the algorithm.

A. Belief States and Predictions

For online monitoring with periodic information, one can always maintain a precise information (x_t, I_t) regarding the current status of the system, where x_t is the current state and $I_t \subseteq \mathcal{I}$ is the index set for the remaining sub-formulae representing the progress of the task. However, for the case of self-triggered monitoring, one of the major challenges is that one does not have this precise information due to unobservable transitions between two sampling instants. Hereafter, we will refer to tuple (x_t, I_t) as an *augmented state* at time instant t. Instead of maintaining a precise augmented state at each instant, the system can only maintain a *belief state* to estimate the current status of the system. Note that, for technical purpose, in each augmented state (x_t, I_t), the progress of the task I_t does not take the effect of current state x_t, which will be carried on for the next instant.

Definition 3 (Belief States): A *belief state* is a set of augmented states. We denote by $\mathbb{B} := 2^{\mathcal{X}\times\mathcal{I}}$ the set of all possible belief states.

Given a belief state, each augmented state (x_t, I_t) in it is an *explanation* of the possible current status of the system. Therefore, a belief state essentially captures all possible explanations based on the partial information. Now suppose that the belief state the monitor holds at instant t is $\boldsymbol{b}$, then for the next instant $t+1$, it can update its belief state as follows:

- If there is no observation at instant $t+1$, then it can make an "open-loop prediction" based on the dynamic of the system as well as the semantics of STL formulae;
- If a new state is observed, i.e., $t+1$ is the triggered instant decided in the previous round, then it can further use the state observation to refine the belief.

Next, we elaborate on more details about how belief states are updated. According to the semantics of STL formulae, the index set can be updated once a new state is reached.

Definition 4 (Index Updates): For each time instant t, we define the *index update function*, denoted as $\mathsf{update}_t : \mathcal{I} \times \mathcal{X} \to \mathcal{I}$, by: for any $I \subseteq \mathcal{I}, x \in \mathcal{X}$, we have

$$\mathsf{update}_t(I,x) = \left\{ i \in I \;\middle|\; \begin{array}{c} [i \in \mathcal{I}_t^{\mathbf{U}} \wedge x \notin \mathcal{H}_i^1 \cap \mathcal{H}_i^2] \\ \vee [i \in \mathcal{I}_t^{\mathbf{G}} \wedge t \neq b_i] \vee [i \notin \mathcal{I}_t] \end{array} \right\}, \tag{10}$$

where, for each $i \in \mathcal{I}_t^{\mathbf{U}}$, we have $\Phi_i = x \in \mathcal{H}_i^1 \mathbf{U}_{[a_i,b_i]} x \in \mathcal{H}_i^2$.

Intuitively, suppose that I is the remaining index set at instant t and x is the state reached at instant t. Then $\mathsf{update}_t(I,x)$ essentially excludes the index set for those sub-formulae with

temporal operator $\mathbf{U}$ that have already been satisfied by reaching state x based on the current index set I. Meanwhile, those sub-formulae with temporal operator $\mathbf{G}$ expired will not be contained anymore.

Now, by combining the dynamic of the system with the update function of the index set, we can define the "dynamic" over the augmented state space.

Definition 5 (Successor Augmented States): Let (x, I) and (x', I') be two augmented states at instants t and $t+1$, respectively. We say (x', I') is a *successor augmented state* of (x, I) from instants t to $t+1$ if

1) there exists $u \in \mathcal{U}$ such that $x' = f(x, u)$; and
2) $I' = \mathsf{update}_t(I, x)$.

We denote by $\mathsf{succ}_t(x, I) \in \mathbb{B}$ the set of all *successor augmented states* of (x, I) from instants t to $t+1$.

Based on the "dynamic" of augmented states, we can define how belief states are evolved. First, without observations between two trigger instants, the belief states can be *predicted* in an open-loop fashion as follow.

Definition 6 (Belief Predictions): We define the (one-step) *belief prediction function* from instants t to $t+1$, denoted as $\mathsf{pred}_t^{t+1} : \mathbb{B} \to \mathbb{B}$, by: for any $\boldsymbol{b} \in \mathbb{B}$, we have

$$\mathsf{pred}_t^{t+1}(\boldsymbol{b}) = \bigcup_{(x,I)\in\boldsymbol{b}} \mathsf{succ}_t(x, I). \tag{11}$$

The multi-step belief prediction function is defined recursively by: for any $\boldsymbol{b} \in \mathbb{B}$ and $k \geq 1$, we have

$$\mathsf{pred}_t^{t+k}(\boldsymbol{b}) = \mathsf{pred}_{t+k-1}^{t+k}(\cdots(\mathsf{pred}_{t+1}^{t+2}(\mathsf{pred}_t^{t+1}(\boldsymbol{b})))). \tag{12}$$

Once observations are made at those triggered instants, the belief states can be *refined* as follows.

Definition 7 (Belief Refinements): We define the *belief refinement function*, denoted as $\mathsf{refine} : \mathbb{B} \times \mathcal{X} \to \mathbb{B}$, by: for any $\boldsymbol{b} \in \mathbb{B}, x \in \mathcal{X}$, we have

$$\mathsf{refine}(\boldsymbol{b}, x) = \{(x', I') \in \boldsymbol{b} \mid x' = x\} = \boldsymbol{b} \cap (\{x\} \times \mathcal{I}). \tag{13}$$

Intuitively, the belief refinement function simply restricts a belief state to a smaller set that is consistent with the current state observed. Note that, compared with the belief prediction function, which is time-dependent, the belief refinement function is time-independent.

B. Main Monitoring Algorithm

First, we introduce two key notions that will be used then present our overall algorithm.

Definition 8 (Safe and Determined Beliefs): Let $\boldsymbol{b} \in \mathbb{B}$ be a belief state at instant t. We say belief state $\boldsymbol{b}$ is

- *safe* (w.r.t instant t) if for any augmented state $(x, I) \in \boldsymbol{b}$, we have $x \in X_t^I$; and
- *determined* (w.r.t instant t) if for any two augmented states $(x, I), (x, I') \in \boldsymbol{b}$ with the same state component, we have $\mathsf{update}_t(I, x) = \mathsf{update}_t(I', x)$.

The intuitions for the above definitions are as follows. For each augmented state (x, I), if $x \notin X_t^I$, then it means that starting from state x, remaining sub-formulae in I is no longer feasible. Therefore, this unsafe circumstance should be avoided for any augmented states in a belief state. On the other hand, a belief state is *determined* means that there will be no ambiguity regarding the task progress once the actual current state is known. Recall that, in each augmented state (x, I), the index set I has not yet taken the effect of x into account. Therefore, the actual index set for sub-formulae remaining is $\mathsf{update}_t(I, x)$ rather that I when x is observed.

We discuss how the online monitor decides when to trigger the next observation. Suppose at time instant $t > 0$, after making an observation, the monitor knows that the current belief state is $\boldsymbol{b}_t$, and it wants to determine an integer $\tau \leq T_{max}$ as the time interval before the next observation. Such integer τ should satisfy the following requirements:

- *Prediction Determinacy:* Since there is no observation until time instant $t+\tau$, the monitor can only predict the belief state for time instant $t+\tau$ as $\hat{\boldsymbol{b}}_{t+\tau} = \mathsf{pred}_t^{t+\tau}(\boldsymbol{b}_t)$. Once the monitor takes observation at $t+\tau$ and observes a new state $x_{t+\tau}$, the belief state will shrink to $\boldsymbol{b}_{t+\tau} = \mathsf{refine}(\hat{\boldsymbol{b}}_{t+\tau}, x_{t+\tau})$. Therefore, in order to capture the exact task progress of the system after making the observation, $\hat{\boldsymbol{b}}_{t+\tau}$ has to be determined.
- *Prediction Safety:* Once the predicted belief state is not safe, the monitor must *immediately* make an observation

Algorithm 1: Self-Triggered Monitor.

Input: Feasible Sets
Output: decision $(\mathcal{M}_\tau, \mathcal{M}_D)$ for each instant
$t \leftarrow 0$; $I \leftarrow \mathcal{I}$; $\hat{\boldsymbol{b}} \leftarrow \{(x_0, I)\}$
while *a new state* x_t *is observed* **do**
  $\boldsymbol{b} \leftarrow \mathsf{refine}(\hat{\boldsymbol{b}}, x_t)$
  if $\boldsymbol{b}$ *is safe* **then**
    $(\mathcal{M}_\tau, \mathcal{M}_D) \leftarrow (\texttt{Trigger-Time}(\boldsymbol{b}, t), 0)$
    $\hat{\boldsymbol{b}} \leftarrow \mathsf{pred}_t^{t+\mathcal{M}_\tau}(\boldsymbol{b})$
    wait for $\mathcal{M}_\tau$ instants and $t \leftarrow t + \mathcal{M}_\tau$
  else
    $\mathcal{M}_D \leftarrow 1$ and **return** "violated"
  end
end

Procedure 1: `Trigger-Time`.

Input: belief state $\boldsymbol{b}$, current instant t
Output: trigger time interval τ
$k \leftarrow 1$; $\tau^\star \leftarrow 1$; $\hat{\boldsymbol{b}} \leftarrow \boldsymbol{b}$
while $k \leq T_{max}$ **do**
  $\hat{\boldsymbol{b}} \leftarrow \mathsf{pred}_{t+k-1}^{t+k}(\hat{\boldsymbol{b}})$
  if $\hat{\boldsymbol{b}}$ *is determined w.r.t.* $t+k$ **then**
    $\tau^\star \leftarrow k$;
  end
  if $\hat{\boldsymbol{b}}$ *is safe w.r.t.* $t+k$ **then**
    $k \leftarrow k+1$
  else
    return $\tau^\star$
  end
end

to resolve ambiguity in the feasibility of task. Otherwise, the system may already become infeasible but the monitor does not issue an alarm on time.

Combining the above two requirements together, based on the current information $\boldsymbol{b}_t$, the largest value for the trigger interval the monitor can select is

$$\tau^\star = \max\{\tau \leq \tau_1 : \mathsf{pred}_t^{t+\tau}(\boldsymbol{b}_t) \text{ is determined}\}, \tag{14}$$

where

$$\tau_1 = \min\{\tau \leq T_{max} : \mathsf{pred}_t^{t+\tau}(\boldsymbol{b}_t) \text{ is not safe}\}. \tag{15}$$

This value $\tau^\star$ can be selected according to procedure `Trigger-Time` by iteratively computing the predicted beliefs and checking their safety and determinacy.

Finally, we present our complete self-triggered online monitoring algorithm for STL tasks in Algorithm 1. We start from the initial instant $t = 0$ and I contains all the indexes in $\mathcal{I}$. At each iteration, we observe the system state and refine the belief state. Then we check if the refined belief state is safe or not. If it is safe, then we continue the monitoring process and employ procedure `Trigger-Time` to compute the next observation instant as well as the predicted belief states. If the refined belief state is not safe, the monitor will issue an alarm and terminate the process.

C. Properties of the Proposed Algorithm

In this subsection, we prove the correctness of Algorithm 1. For any prefix state sequence $\mathbf{x}_{0:t}$, we define its induced augmented state sequence by

$$\zeta(\mathbf{x}_{0:t}) = (x_0, I_0)(x_1, I_1)\dots(x_t, I_t),$$

where for any $k = 1, \dots, t$, I_k is the index set for those sub-formulae who have not yet been satisfied by $x_0 \cdots x_{k-1}$, i.e., $I_k = \{i \in \mathcal{I} \wedge i \notin \mathcal{I}^{\mathbf{G}}_{<k} \mid x_0 \cdots x_{k-1} \not\models \Phi_i\}$. We denote by $\mathsf{last}(\zeta(\mathbf{x}_{0:t})) = (x_t, I_t)$ the last augmented state in the induced sequence. Then for any observation history $\hbar = (x_0, \tau_0)(x'_1, \tau_1)\dots(x'_{n-1}, \tau_{n-1})x'_n$, we define

$$\mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar) = \{\zeta(\mathbf{x}_{0:t}) : \hbar_{\mathcal{M}}(\mathbf{x}_{0:t}) = \hbar\}$$

as the set of augmented state sequence consistent with the observation under monitor $\mathcal{M}$, where $t = \sum_{i=0}^{n-1} \tau_i$. We define $\mathsf{last}(\mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar)) = \{\mathsf{last}(\zeta(\mathbf{x}_{0:t})) \mid \zeta(\mathbf{x}_{0:t}) \in \mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar)\}$.

Note that, the above $\mathsf{last}(\mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar))$ is defined based on the entire observation history. However, Algorithm 1 computes belief in a recursive manner. Formally, still let $\hbar = (x_0, \tau_0)(x'_1, \tau_1)\dots(x'_{n-1}, \tau_{n-1})x'_n$ be an observation history. We define a belief sequence $B(\hbar) = \hat{\boldsymbol{b}}_0\boldsymbol{b}_0\hat{\boldsymbol{b}}_1\boldsymbol{b}_1\cdots\hat{\boldsymbol{b}}_n\boldsymbol{b}_n$ recursively by

- $\hat{\boldsymbol{b}}_0 = \{(x_0, \mathcal{I})\}$;
- $\forall k \geq 0, \boldsymbol{b}_k = \mathsf{refine}(\hat{\boldsymbol{b}}_k, x'_k)$;
- $\forall k \geq 0, \hat{\boldsymbol{b}}_{k+1} = \mathsf{pred}_{T_{k-1}}^{T_k}(\boldsymbol{b}_k)$.

where $T_k = \sum_{i=0}^{k} \tau_i$ with $T_{-1} = 0$. According to Algorithm 1, $\boldsymbol{b}_n$ is the belief state maintained by the monitor following observation history $\hbar$.

The following result states that the belief state maintained by the monitor is indeed the set of all possible augmented states consistent with the observation.

Proposition 1: Let $\hbar = (x_0, \tau_0)(x'_1, \tau_1)\dots(x'_{n-1}, \tau_{n-1})x'_n$ be an observation history under a given monitor $\mathcal{M}$ and $B(\hbar) = \hat{\boldsymbol{b}}_0\boldsymbol{b}_0\hat{\boldsymbol{b}}_1\boldsymbol{b}_1\cdots\hat{\boldsymbol{b}}_n\boldsymbol{b}_n$ be its induced belief state sequence. Then we have $\boldsymbol{b}_n = \mathsf{last}(\mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar))$.

Proof: The result can be proved by induction on the length of the observation history. Due to space constraint, the complete proof is provided in [29]. □

Now we can establish the correctness of Algorithm 1.

Theorem 1: Algorithm 1 correctly solves Problem 1. That is, for any $\mathbf{x}_{0:t}$ generated by system (1), we have

$$\mathcal{M}_D(\hbar_{\mathcal{M}}(\mathbf{x}_{0:t})) = 1 \Leftrightarrow \mathbf{x}_{0:t} \text{ is violated}. \tag{16}$$

Proof: Let $\mathbf{x}_{0:t}$ be the actual prefix signal, $\zeta(\mathbf{x}_{0:t}) = (x_0, I_0)(x_1, I_1)\dots(x_t, I_t)$ be its induced augmented state sequence and $\hbar = \hbar_{\mathcal{M}}(\mathbf{x}_{0:t})$ be the observation history of the monitor. Note that, the actual augmented state sequence $\zeta(\mathbf{x}_{0:t})$ is always contained in $\mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar)$, which also means that $(x_t, I_t) \in \mathsf{last}(\mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar))$. Furthermore, according to Proposition 1, the belief state of the monitor after refinement is $\boldsymbol{b} = \mathsf{last}(\mathcal{M}_{\boldsymbol{b}}^{-1}(\hbar))$.

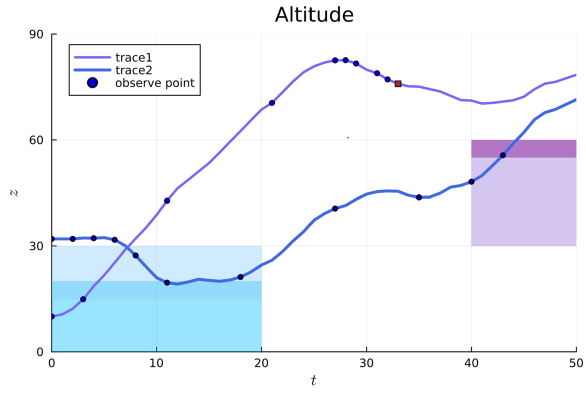
Now, let us argue the correctness of Algorithm 1 as follows. To show the "$\Leftarrow$" direction, suppose that $\mathbf{x}_{0:t}$ is a violated sequence. Then we know that some sub-formulae not yet satisfied must be no longer feasible from x_t, which means that $x_t \notin X_t^{I_t}$. This immediately implies that $\boldsymbol{b}$ is not safe, i.e., the monitor will issue decision $\mathcal{M}_D = 1$. For the "$\Rightarrow$" direction, suppose that $\mathcal{M}_D(\hbar_{\mathcal{M}}(\mathbf{x}_{0:t})) = 1$. This implies that $\boldsymbol{b}$ is not safe, i.e., there exists $(x, I) \in \boldsymbol{b}$ such that $x \notin X_t^I$. If $\boldsymbol{b}$ is a singleton, then $\mathbf{x}_{0:t}$ is a violated sequence. Otherwise, for all $(x_t, I) \in \boldsymbol{b}$, we have $x_t \notin X_t^I$. Based on procedure `Trigger-Time`, $\hat{\boldsymbol{b}}_n$ is determined, x_t must be in the region consistent for index update thus the violation is only concerned with the remaining formulae after update. Thus we prove that $\mathbf{x}_{0:t}$ is a violated sequence. □

VI. Case Studies

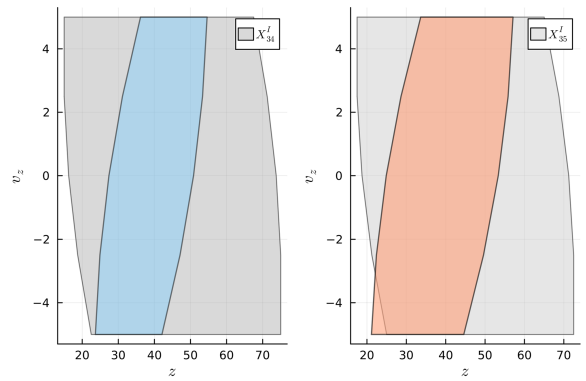
In this section, we demonstrate our approach by case studies. We have implemented the proposed self-triggered monitoring algorithm in **Julia** language [30] and employ package **JuliaReach** [31] for reachability analysis. For nonlinear systems, reachable sets are calculated by over-approximated zonotopes [32]. Note that, for discrete-time systems in (1), a belief state may contain infinite augmented states since the state space is continuous. For the purpose of computation, we represent each belief state $\boldsymbol{b}$ equivalently in the form of $\boldsymbol{b}' = \{(R_{I_1}, I_1), \dots, (R_{I_l}, I_l)\}$, where $I_i \neq I_j, \forall i \neq j$ and $R_I = \{x \in \mathcal{X} \mid (x, I) \in \boldsymbol{b}\}$ is the region of all states whose index set is I. All simulations were performed on a computer with an Intel Core i9-13900H CPU and 32GB of RAM. All codes are available at https://github.com/ChuweiW/st_om.

A. Case Study 1: Drone Altitude Control

We consider a drone for the purpose of gathering air data at various altitudes. Therefore, we consider its altitude state z and velocity v_z, i.e., $x = [z, v_z]^\top \in \mathcal{X} = [0, 100] \times [-5, 5]$. The



(a) Two possible altitude trajectories of the drone.



(b) Left (resp. right) hand side is the safe (resp. unsafe) belief state predicted for t = 34 (resp. t = 35) from t = 27.

Fig. 2. Online monitoring of drone altitude control systems.

discrete-time dynamic of the drone is

$$x_{t+1} = Ax_t + Bu_t \tag{17}$$

where $A = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$ and $\mathcal{U} \subseteq [-2.5, 2.5]$. In order to initialize the process, the drone needs to first reach altitude ranges [0, 20] and [15, 30] in the first 20 time instants to get the permission to take off. Then between $40 \sim 50$ time instants, it should maintain its altitude between [30, 60] until it successfully collects required data at $z \in [55, 60]$. We set the time horizon $T = 50$ and STL task is

$$\psi = \mathbf{F}_{[0,20]} z \in [0, 20] \wedge \mathbf{F}_{[0,20]} z \in [15, 30] \\ \wedge z \in [30, 60] \mathbf{U}_{[40,50]} z \in [55, 60]. \tag{18}$$

In Fig. 2(a), we show two possible trajectories of the drone starting from different initial altitudes. In each trace, blue circles denote those triggered instants at which the monitor takes observations. Specifically, trace 1 fails to achieve the STL task and the red square in the trace denotes the instant from which the task is not longer feasible anymore. One can see that the monitor takes observations more frequently when the system is approaching the violation of the task. On the other hand, trace 2 shows a trajectory along which the STL is achieved. Note that, the monitor takes observations at $t = 27$ and $t = 35$. To illustrate procedure `Trigger-Time`, let us focus on the evolution of the predicted belief state from $t = 34$ to $t = 35$, which is shown in Fig. 2(b). Particularly, we see that the predicted belief state for $t = 34$ is still safe, while the predicted belief state for $t = 35$ is no longer safe as it may contain state that is not feasible. This is why the monitor has to make an observation at $t = 35$.

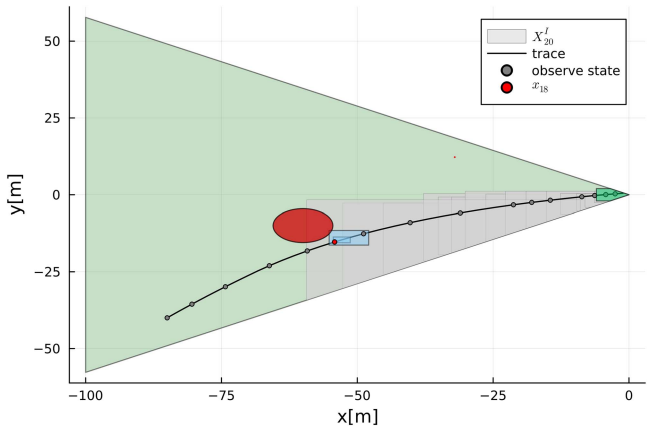


Fig. 3. Online monitoring of spacecraft rendezvous.

B. Case Study 2: Spacecraft Rendezvous

We consider a spacecraft rendezvous problem adopted from [33], where a spacecraft *chaser* aims to approach another spacecraft *target* in the same orbital plane satisfying some desired requirements. The system model of the *chaser* navigating to the *target* can be described by a two-dimensional nonlinear dynamics as follows:

$$\ddot{x} = n^2 x + 2n\dot{y} + \frac{\mu}{r^2} - \frac{\mu}{r_c^3}(r + x) + \frac{u_x}{m_c}$$
$$\ddot{y} = n^2 y - 2n\dot{x} - \frac{\mu}{r_c^3} y + \frac{u_y}{m_c},$$

where system state is $x_k = [x\ y\ \dot{x}\ \dot{y}]^\top$, control inputs provided by the thrusters are $u_k = [u_x\ u_y]^\top$. The parameters are related to geostationary equatorial orbit with $\mu = 3.698 \times 10^{14} \times 60^2 \mathrm{m}^3/\mathrm{min}^2$, $r = 42164\,\mathrm{km}$, $m_c = 500\,\mathrm{kg}$, $n = \sqrt{\frac{\mu}{r^3}}$ and $r_c = \sqrt{(r + x)^2 + y^2}$. We discretize the system with sampling time of 0.5 minutes. The constraints are $x \in \mathcal{X} = [-100, 0] \times [-70, 70] \times [0, 10] \times [0, 10]$ and $u \in \mathcal{U} = [-3, 3] \times [-3, 3]$.

The formal specifications for the rendezvous system are as follows. First, when the distance between *chaser* and *target* is less than 100 m, the *chaser* should keep rendezvousing with relatively stationary velocity and stay within the line-of-sight cone $\mathsf{LOS} = \{(x, y) \mid (y \geq x\tan(30^\circ) \wedge (-y \geq x\tan(30^\circ)\}$. Furthermore, the rendezvous task should be completed within 25 minutes in the sense that the *chaser* is close enough to the *target* with velocity constraints. We define $\mathsf{Goal} = \{(x, y, \dot{x}, \dot{y}) \mid x \in [-6, 0] \wedge y \in [-2, 2] \wedge \dot{x} \in [0, 3] \wedge \dot{y} \in [0, 3]\}$ as the region satisfying the task. Finally, during the rendezvous process, the *chaser* should avoid the **Debris** region represented by the red ball in Fig. 3. Therefore, in terms of STL formula, the mission can be described by:

$$\psi = \mathbf{F}_{[0,25]} x_k \in \mathsf{Goal} \wedge \mathbf{G}_{[0,25]} p \notin \mathsf{Debris} \wedge \mathbf{G}_{[0,25]} p \in \mathsf{LOS},$$

where p denotes $[x, y]^\top$ the position of the *chaser*.

In Fig. 3, we consider a trajectory which completes the task in advance at 40 steps (20min) and the observed states are marked by circles. Under the self-triggered algorithm, the monitor only samples system states for 16 times compared with the periodic sampling strategy of 40 times. For example, consider system state x_{18} marked by red circle. The blue region shows the evolution of belief state starting from this time instant. We see that, at time step 20, the predicted belief state intersects with the debris. This means that, the spacecraft may hit the **Debris** region. Therefore, the monitor cannot "sleep" anymore and has to activate its sensors to solve information ambiguity. Note that, all feasible sets and reachable sets are four dimensional but are projected into two dimensions (position) for plotting.

We analyze the computation time needed for making online monitoring decisions. For the first case study, since its dynamic is very simple, the online reachable set computation can be done very quickly, within 0.01 seconds. For the second case study, the average time for the online computation of reachable sets is 0.1023 seconds over 30 decision-making instants, with a maximum processing time of 0.2396 seconds.

VII. Conclusion

In this paper, we introduced a novel *self-triggered* approach for online monitoring of dynamical systems based on STL specifications. In contrast to conventional online monitoring methods employing periodic information updates, our proposed self-triggered monitor actively determines sampling instants for system states, offering the potential to significantly reduce sensor energy consumption. The key principle behind our trigger time design is to enable the monitor to remain inactive as long as the predicted belief state exhibits no ambiguity regarding the satisfaction of the STL task. The limitation of our approach is that it is primarily suitable for static maps, i.e., the underlying map remains unchanged when the monitor "sleeps". In the future, we would like to extend our approach to the full fragment of STL formulae as well as dynamic maps. Also, we would like to apply the self-trigger mechanism to address control synthesis problems.

References

[1] R. Yan and A. Julius, "Distributed consensus-based online monitoring of robot swarms with temporal logic specifications," *IEEE Robot. Automat. Lett.*, vol. 7, no. 4, pp. 9413–9420, Oct. 2022.

[2] E. Bonnah and K. A. Hoque, "Runtime monitoring of time window temporal logic," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 5888–5895, Jul. 2022.

[3] A. Donzé, T. Ferrere, and O. Maler, "Efficient robust monitoring for STL," in *Proc. Int. Conf. Comput. Aided Verification*, 2013, pp. 264–279.

[4] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, 2010, pp. 92–106.

[5] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *Formal Techn., Modelling Anal. Timed Fault-Tolerant Syst.*, vol. 3253, pp. 152–166, 2004.

[6] M. Ma, E. Bartocci, E. Lifland, J. Stankovic, and L. Feng, "SaSTL: Spatial aggregation signal temporal logic for runtime monitoring in smart cities," in *Proc. ACM/IEEE 11th Int. Conf. Cyber-Physical Syst.*, 2020, pp. 51–62.

[7] M. Srinivasan and S. Coogan, "Control of mobile robots using barrier functions under temporal logic specifications," *IEEE Trans. Robot.*, vol. 37, no. 2, pp. 363–374, Apr. 2021.

[8] V. Kurtz and H. Lin, "Mixed-integer programming for signal temporal logic with fewer binary variables," *IEEE Control Syst. Lett.*, vol. 6, pp. 2635–2640, 2022.

[9] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 3451–3458, Apr. 2022.

[10] H. Ho, J. Ouaknine, and J. Worrell, "Online monitoring of metric temporal logic," in *Proc. Int. Conf. Runtime Verification*, 2014, pp. 178–192.

[11] A. Dokhanchi, B. Hoxha, and G. Fainekos, "On-line monitoring for temporal logic robustness," in *Proc. Int. Conf. Runtime Verification*, 2014, pp. 231–246.

[12] J. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods Syst. Des.*, vol. 51, no. 1, pp. 5–30, 2017.

[13] X. Qin and J. Deshmukh, "Clairvoyant monitoring for signal temporal logic," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, 2020, pp. 178–195.

[14] M. Ma, J. Stankovic, E. Bartocci, and L. Feng, "Predictive monitoring with logic-calibrated uncertainty for cyber-physical systems," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5s, pp. 1–25, 2021.

[15] H. Yoon and S. Sankaranarayanan, "Predictive runtime monitoring for mobile robots using logic-based Bayesian intent inference," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 8565–8571.

[16] L. Lindemann, X. Qin, J. V. Deshmukh, and G. J. Pappas, "Conformal prediction for STL runtime verification," in *Proc. Int. Conf. Cyber-Physical Syst.*, 2023, pp. 142–153.

[17] A. Momtaz, N. Basnet, H. Abbas, and B. Bonakdarpour, "Predicate monitoring in distributed cyber-physical systems," *Int. J. Softw. Tools Technol. Transfer*, vol. 25, pp. 541–556, 2023.

[18] X. Yu, W. Dong, S. Li, and X. Yin, "Model predictive monitoring of dynamical systems for signal temporal logic specifications," *Automatica*, vol. 160, 2024, Art. no. 111445.

[19] C. Nowzari and J. Cortés, "Self-triggered coordination of robotic networks for optimal deployment," *Automatica*, vol. 48, no. 6, pp. 1077–1087, 2012.

[20] F. D. Brunner, W. Heemels, and F. Allgöwer, "Event-triggered and self-triggered control for linear systems based on reachable sets," *Automatica*, vol. 101, pp. 15–26, 2019.

[21] D. Senejohnny, P. Tesi, and C. De Persis, "A jamming-resilient algorithm for self-triggered network coordination," *IEEE Trans. Control Netw. Syst.*, vol. 5, no. 3, pp. 981–990, Sep. 2018.

[22] L. Lindemann, D. Maity, J. Baras, and D. Dimarogonas, "Event-triggered feedback control for signal temporal logic tasks," in *Proc. IEEE Conf. Decis. Control*, 2018, pp. 146–151.

[23] M. Waga, É. André, and I. Hasuo, "Model-bounded monitoring of hybrid systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 6, no. 4, pp. 1–26, 2022.

[24] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Trans. Robot.*, vol. 30, no. 4, pp. 903–918, Aug. 2014.

[25] Y. Chou, H. Yoon, and S. Sankaranarayanan, "Predictive runtime monitoring of vehicle models using Bayesian estimation and reachability analysis," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 2111–2118.

[26] F. Cairoli, L. Bortolussi, and N. Paoletti, "Neural predictive monitoring under partial observability," in *Proc. Int. Conf. Runtime Verification*, 2021, pp. 121–141.

[27] W. Liu, W. Xiao, and C. Belta, "Learning robust and correct controllers from signal temporal logic specifications using barriernet," in *Proc. IEEE Conf. Decis. Control*, 2023, pp. 7049–7054.

[28] F. Chen, M. Sewlia, and D. V. Dimarogonas, "Cooperative control of heterogeneous multi-agent systems under spatiotemporal constraints," *Annu. Rev. Control*, vol. 57, 2024, Art. no. 100946.

[29] C. Wang, X. Yu, J. Zhao, L. Lindemann, and X. Yin, "Sleep when everything looks fine: Self-triggered monitoring for signal temporal logic tasks," 2023, *arXiv:2311.15531*.

[30] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, 2017.

[31] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling, "Juliareach: A toolbox for set-based reachability," in *Proc. Hybrid Syst. Comput. Control*, 2019, pp. 39–44.

[32] M. Althoff, O. Stursberg, and M. Buss, "Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes," *Nonlinear Anal., Hybrid Syst.*, vol. 4, no. 2, pp. 233–249, 2010.

[33] N. Chan and S. Mitra, "Verifying safety of an autonomous spacecraft rendezvous mission," *EPiC Ser. Comp.*, vol. 48, pp. 20–32, 2017.