Control Engineering Practice 143 (2024) 105782

Contents lists available at ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

Signal temporal logic synthesis under Model Predictive Control: A low complexity approach

Tiange Yang, Yuanyuan Zou *, Shaoyuan Li, Xiang Yin, Tianyu Jia

Department of Automation, Shanghai Jiao Tong University, Shanghai, 200240, China

ARTICLE INFO

Keywords:
Temporal logic
Model predictive control
Move blocking scheme

ABSTRACT

In this paper, we focus on the challenging problem of model predictive control (MPC) for dynamics systems with high-level tasks formulated as signal temporal logic (STL). The state-of-art for STL synthesis mainly suffers from limited scalability with respect to the complexity of the task and the planning horizon, hindering the real-time implementation of MPC. This work tackles this issue by STL formula reformulation and input blocking. Specifically, simplifications are applied on disjunctive STL (sub)formulae recursively in the framework of MPC to limit formula size. We show that the simplified STL can be reformulated into mixed integer linear programming (MILP) constraints with a modifiable number of binary variables being required. The move blocking scheme is then employed to further reduce problem complexity by fixing input variables to be constant over several time intervals. In order to trade off the control performance and computational load, a blocking structure design with on-line correction is proposed. The extension of the proposed STL-MPC algorithm to uncertain systems is achieved through STL constraint tightening. Simulations and experiments show the effectiveness of the proposed algorithm.

1. Introduction

In recent years, increasing attention has been paid to control synthesis under complex temporal logic tasks, where strict time and logic constraints are imposed on system behaviors (Liu, Trivedi, Yin, & Zamani, 2022; Tian et al., 2023; Yu, Yin, Li, & Li, 2022). For example, in a patrol mission, an agent must visit several goal regions *in order* and *within given time intervals*, *always* avoid obstacles, and choose *one of* the charging stations to charge the battery *before* entering the terminal region. Temporal logic specifications, such as signal temporal logic (STL) (Maler & Nickovic, 2004), have proven to be adept at expressing these intricate system properties. STL was initially developed for monitoring the expected behavior of physical systems. As a predicate-based logic, STL allows the descriptions of strict time and logic properties on both continuous and hybrid systems. STL also offers diverse quantitative semantics, referred to as robustness, to quantify the extent to which a property is satisfied (or violated) with real values. STL thus has been widely applied to several types of dynamical systems to integrate complex high-level task descriptions with the low-level dynamics, ranging from robotics, industrial systems, and traffic networks (Buyukkocak, Aksaray, & Yazıcıoğlu, 2021; Liu, Wu, Dai, & Lin, 2020; Patil, Hashimoto, & Kishida, 2022).

STL synthesis refers to finding a control policy such that the resulting trajectory of the underlying system satisfies the STL formula while minimizing a given cost. One classical approach is to encode the satisfaction of the specified STL formula into mixed integer linear programming (MILP) constraints (Raman et al., 2014). This method is both sound (any solution found by the method will satisfy the STL formulae) and complete (if there exists a satisfying solution, the method will find it). However, since the encoding introduces at least one binary variable for each predicate at each time step, the resulting MILP problem has exponential computational complexity with respect to the formula horizon. Despite this complexity issue, the MILP-based method has been widely used in temporal logic synthesis with moderate formulae size and achieved good control performance (Farahani, Majumdar, Prabhu, & Soudjani, 2018; Rodionova, Lindemann, Morari, & Pappas, 2022; Sahin, Nilsson, & Ozay, 2020). Efforts have been directed toward reducing the size of MILP problems. A more efficient MILP encoding for STL was designed in Kurtz and Lin (2022) with only a logarithmic number of binary variables being required. In Sun, Chen, Mitra, and Fan (2022), the complexity caused by nonlinear system behaviors was lowered by designing formula-satisfied piece-wise linear reference paths and a corresponding tracking controller. These methods guarantee soundness but still face scalability challenges with lengthy and intricate STL formulae. To avoid MILP encoding entirely, novel techniques including robustness-based optimizations (Haghighi, Mehdipour, Bartocci, & Belta, 2019; Lindemann & Dimarogonas, 2019), reachability

* Corresponding author.
E-mail address: yuanyzou@sjtu.edu.cn (Y. Zou).

https://doi.org/10.1016/j.conengprac.2023.105782
Received 25 June 2023; Received in revised form 24 October 2023; Accepted 7 November 2023

analysis (Yu & Dimarogonas, 2021), control barrier functions (Lindemann & Dimarogonas, 2018) and learning-based approaches (Puranic, Deshmukh, & Nikolaidis, 2021) were employed to derive satisfying system runs with high computational efficiency. However, compared with the MILP encoding, the above-mentioned methods usually lack completeness due to the existence of approximations, so that a feasible solution cannot be guaranteed even if one exists.

Model predictive control (MPC) is a powerful online optimization technique (Bai, Li, & Zou, 2021; Garcia, Prett, & Morari, 1989; Huang, Zheng, & Li, 2022; Xu, Suleman, & Shi, 2023). Due to the ability to cope with strict constraints and complex environments, MPC is an appropriate framework for STL synthesis, and the aforementioned MILP-based and robustness-based approaches can be employed for online problem-solving (Zhou, Yang, Zou, Li, & Fang, 2022; Zhou, Zou, Li, Li, & Fang, 2022). Owing to the existence of temporal and logical properties, the control with STL formulae is historically dependent, that is, previous choices of control actions can impose constraints on the rest of the path. Therefore, STL-MPC is mainly implemented in a shrinking horizon manner (Farahani et al., 2018) to realize historical trajectory tracking: the optimization window is always the whole formula horizon and would not shift with time, and the size of control inputs to be optimized is reduced by one at each time step. However, the size of the required optimization window would grow with the formula horizon and cause significant scalability issues.

Motivated by the discussions above, this paper focuses on STL synthesis via MILP and achieves efficient online computations in the framework of shrinking horizon MPC. First, simplifications on STL (sub)formulae are investigated to limit task size. We formally summarize the rules for computing binary variables required for STL encoding, which are then used to guide the formula simplification process. We illustrate that the simplified STL can be reformulated as MILP constraints, necessitating a modifiable number of binary variables. The input blocking technique (Shekhar & Manzie, 2015; Son, Oh, Kim, & Lee, 2020), which can reduce the problem complexity by constraining groups of adjacent-in-time input variables to have the same value, is then introduced into STL-MPC to further enhance the computational efficiency. Considering both the task completion and computational costs, the blocking structure is initialized offline by solving a pre-designed mixed integer programming (MIP). Slight block corrections are executed during MPC implementation with feasibility guarantees. The control performance of the whole low complexity STL-MPC algorithm is formally analyzed, and the extension to uncertain systems is achieved through STL constraint tightening.

This paper is organized as follows. In Section 2, we introduce STL and formulate the STL-MPC problem. The formula simplification method is proposed in Section 3 and the blocking scheme is further investigated for STL synthesis in Section 4. Section 5 presents the whole low complexity STL-MPC algorithm, analyzes the control performance, and extends the results to uncertain systems. The proposed algorithm is validated in Section 6 through simulations and experiments. We conclude this paper in Section 7.

Notation: Throughout this paper, $\mathbb{R}^n$ denotes the n-dimensional real space. $\mathbb{N}$ denotes the set of non-negative integers. For a positive integer c, $\mathbb{N}_{\geq c}$ indicates the set of integers not less than c and $\mathbb{N}_{\leq c}$ indicates the set of non-negative integers not greater than c. The notation $\mathbf{1}_n$ represents an $n\times 1$ vector of ones. I_n denotes the $n\times n$ identity matrix. The bracket $\lceil a \rceil$ indicates the ceiling function of $a\in\mathbb{R}$. $\otimes$ represents the Kronecker product. For a random variable X, we use $\mathbb{E}[X]$ to denote its expectation. The cardinality of a set Ω is denoted by $|\Omega|$. $(k+i|k)$ represents a prediction of a variable i steps from time k.

2. Preliminaries and problem formulation

2.1. Signal temporal logic

The temporal logic tasks of systems can be captured by STL, whose syntax is defined in a Backus–Naur form (Knuth, 1964) as

$$\phi ::= \mu|\neg\mu|\phi\wedge\varphi|\phi\vee\varphi|G_{[a,b]}\phi|F_{[a,b]}\phi|\phi U_{[a,b]}\varphi, \tag{1}$$

where ϕ and φ are all STL formulae; μ indicates an atomic predicate, which is defined through a predicate function $\alpha:\mathbb{R}^n\to\mathbb{R}$, i.e., μ is true if and only if $\alpha(x)\geq 0$, $x\in\mathbb{R}^n$; $\neg$, $\wedge$, and $\vee$ are Boolean operators negation, conjunction, and disjunction, respectively; $[a,b]$ is a closed time interval with $a,b\in\mathbb{N}$ and $a\leq b$; G, F and U are temporal operators *Globally*, *Finally*, and *Until*, respectively. Specifically, $G_{[a,b]}\phi$ indicates that ϕ must always be true during the time interval $[a,b]$ in the future; $F_{[a,b]}\phi$ indicates that ϕ should become true at some time instants during the time interval $[a,b]$ in the future; and $\phi U_{[a,b]}\varphi$ requires that ϕ must always be true until φ becomes true during time interval $[a,b]$ in the future. Based on the following facts,

$$G_{[a,b]}\phi=\bigwedge_{k'\in[a,b]}F_{[k',k']}\phi, \tag{2}$$

$$F_{[a,b]}\phi=\bigvee_{k'\in[a,b]}F_{[k',k']}\phi, \tag{3}$$

$$\phi U_{[a,b]}\varphi=\bigvee_{k'\in[a,b]}(F_{[k',k']}\varphi\wedge\bigwedge_{k''\in[0,k']}F_{[k'',k'']}\phi), \tag{4}$$

we divide all STL formulae given by (1) into three categories as follows:

(i) Predicates: μ, $\neg\mu$;
(ii) Conjunctive formulae: $\bigwedge_{i=1}^{r}\phi_i$, $G_{[a,b]}\phi$;
(iii) Disjunctive formulae: $\bigvee_{i=1}^{r}\phi_i$, $F_{[a,b]}\phi$, $\phi U_{[a,b]}\varphi$.

The length of an STL formula ϕ, denoted by $len(\phi)$, is the number of the least time steps in the future that is needed to verify the satisfaction of ϕ at the current time, which can be recursively computed as in Belta and Sadraddini (2019). Denote $\xi=x(0)x(1)\dots$ as a discrete-time system run, which is a sequence of system state x. A run ξ satisfies ϕ at time k, denoted by $(\xi,k)\vDash\phi$, if sequence $x(k)x(k+1)\dots$ satisfies ϕ. Accordingly, ξ satisfies ϕ, denoted by $\xi\vDash\phi$, if $(\xi,0)\vDash\phi$. Note that to verify the satisfaction of ϕ with respect to ξ, the length of system run ξ should be equal to or larger than $len(\phi)$.

Assumption 1. We assume that the STL formulae φ are time-bounded, i.e., $len(\varphi)<\infty$, and the related predicate functions α are all affine functions.

2.2. Shrinking horizon MPC problem formulation

In this paper, we consider the following discrete-time linear system:

$$x(k+1)=Ax(k)+Bu(k), \tag{5}$$

where $x(k)\in\mathbb{R}^{d_x}$, $u(k)\in\mathbb{R}^{d_u}$ are the system state and control input at time instant k, respectively, and A and B are matrices of appropriate dimension. Denote $\mathbf{u}_N(k)=[u^{\mathrm{T}}(k|k),\dots,u^{\mathrm{T}}(N-1|k)]^{\mathrm{T}}$ as a control vector consisting of the predicted control inputs from time step k to time step $N-1$. Given an initial state $x(k)$ and a control vector $\mathbf{u}_N(k)$, one can obtain a state sequence $(\xi,k)=x(k)x(k+1|k)\dots x(N|k)$ based on (5). Since (ξ,k) is determined by $x(k)$ and $\mathbf{u}_N(k)$, we rewrite it into a more elaborative way as $\xi(x(k),\mathbf{u}_N(k))$ with a slight abuse of notation.

Considering the historical dependence of STL synthesis, shrinking horizon MPC framework is employed in this paper. For a fixed constant $N\in\mathbb{N}_{\geq 1}$, define $\mathbf{u}_N^{Full}(k)=[u^{\mathrm{T}}(0),u^{\mathrm{T}}(1),\dots,u^{\mathrm{T}}(k-1),u^{\mathrm{T}}(k|k),\dots,u^{\mathrm{T}}(N-1|k)]^{\mathrm{T}}$, where $u(0),\dots,u(k-1)$ are fixed as obtained control inputs up to time $k-1$, and the remaining $u(k|k),\dots,u(N-1|k)$ are to be determined. We then formulate the STL-MPC synthesis problem as follows.

Problem 1. Given system (5), STL task φ, initial state $x(0)$, and a constant $N\geq\mathrm{len}(\varphi)$, find the control input $u(k)$, which is the first element of the vector $\mathbf{u}_N(k)$, for the system at each time step k by solving the following optimization problem, where J is a user-defined cost function:

$$\min_{\mathbf{u}_N(k)} J(x(k),\mathbf{u}_N(k)) \tag{6}$$

$$\text{s.t. } x(k+1)=Ax(k)+Bu(k), \tag{6a}$$

$$\xi(x(0),\mathbf{u}_N^{Full}(k))\vDash\varphi, \tag{6b}$$

Note that some input and state physical constraints can be considered as a special case of STL and incorporated into constraint (6b). For example, an input constraint $\|u(k)\|_\infty \leq h, h \in \mathbb{R}, \forall k \in [0, N]$ can be reformulated into an STL formula $G_{[0,N]}(h - \|u\|_\infty \geq 0)$. As mentioned earlier, many mature solvers exist for Problem 1, but most of them suffer from limited scalability in terms of the complexity and the length of φ. Therefore, our primary goal is to realize more computationally efficient online problem-solving even under long and complex STL formulae.

3. STL simplification and reformulation

In this section, we present an STL simplification method to reduce the complexity of online optimizations. To provide context, we begin by briefly introducing a state-of-the-art MILP-encoding method for STL as follows.

MILP encoding for STL (Kurtz & Lin, 2022): The satisfaction of STL formulae can be encoded into MILP constraints. The main idea is that for a formula ϕ, a continuous variable $q_k^\phi \in [0, 1]$ is introduced such that $q_k^\phi = 1$ if the satisfaction of ϕ is enforced and $q_k^\phi \in [0, 1)$ otherwise. To achieve this, the continuous variables are related to a set of constraints, which can be generated recursively as follows:

(i) MILP encoding for predicates: For a predicate $\mu = \{\alpha(k) \geq 0\}$, continuous variable q_k^μ is introduced. The MILP constraints can be designed as follows such that $q_k^\mu = 1$ enforces the satisfaction of μ at time k, where $L \in \mathbb{R}$ is a sufficiently large positive number:

$$-\alpha(x(k)) \leq L(1 - q_k^\mu).$$

(ii) MILP encoding for conjunctive formulae: For $\phi = \bigwedge_{i=1}^r \phi_i$, continuous indicators q_k^ϕ, $q_k^{\phi_i}, i = 1, \dots, r$ are employed, and the corresponding constraints are designed as

$$q_k^\phi \leq q_k^{\phi_i}, \forall i = 1, \dots, r. \tag{7}$$

In this way, $q_k^\phi = 1$ can enforce $q_k^{\phi_i} = 1$, $i \in [1, r]$ and therefore enforces the satisfaction of the corresponding sub-formula ϕ_i. For *Globally* operation $\varphi = G_{[a,b]}\phi$, the following linear constraints can be constructed similarly based on (2), such that $q_k^\varphi = 1$ enforces the satisfaction of ϕ between future time steps $k + a$ to $k + b$:

$$q_k^\varphi \leq q_{k+k'}^\phi, \ \forall k' = a, \dots, b. \tag{8}$$

(iii) MILP encoding for disjunctive formulae: For $\phi = \bigvee_{i=1}^r \phi_i$, the continuous variables q_k^ϕ, $q_k^{\phi_i}, i = 1, \dots, r$ are employed, which subject to

$$[1 - q_k^\phi, q_k^{\phi_1}, \dots, q_k^{\phi_r}] \in SOS1. \tag{9}$$

Here $SOS1$ denotes vectors that contain exactly one nonzero element, with that element being equal to 1. Consequently, if $q_k^\phi = 1$, it implies that at least one sub-formula ϕ_i within ϕ is enforced to be satisfied. The detailed MILP encoding for (9) can be found in Vielma and Nemhauser (2011), which requires $\lceil \log_2(r + 1) \rceil$ binary variables. Given the statements provided in (3) and (4), the encoding for *Finally* and *Until* operations can be similarly constructed, and is omitted here due to space considerations.

Applying the MILP encoding rules described above, we can encode disjunctive formulae using a logarithmic number of binary variables, while predicates and conjunctive formulae can be encoded without binary variables. The number of binary variables required for the encoding of φ, denoted as N_φ, is summarized in Table 1. An illustrative example of MILP encoding is provided below.

Example 1. Consider STL formula $\varphi = \phi_1 \wedge \phi_2$ with $\phi_1 = F_{[1,15]}\mu_1$ and $\phi_2 = G_{[3,6]}\mu_2$. Given system run ξ, the MILP encoding for $\xi \vDash \varphi$ can be constructed as follows:

(i) First, the continuous variables q_0^φ, $q_0^{\phi_1}$ and $q_0^{\phi_2}$ are employed to indicate the satisfaction of φ, ϕ_1 and ϕ_2, respectively, and are subject to constraints as in (7).

Table 1
Number of required binary variables for the MILP encoding of STL formulae.

STL formula	The number of required binary variables
$\varphi = \mu$	$N_\varphi = 0$
$\varphi = \neg\mu$	$N_\varphi = 0 + N_\mu = 0$
$\varphi = \bigvee_{i=1}^r \phi_i$	$N_\varphi = \lceil \log_2(r + 1) \rceil + \sum_{i=1}^r N_{\phi_i}$
$\varphi = \bigwedge_{i=1}^r \phi_i$	$N_\varphi = \sum_{i=1}^r N_{\phi_i}$
$\varphi = G_{[a,b]}\phi$	$N_\varphi = (b - a + 1)N_\phi$
$\varphi = F_{[a,b]}\phi$	$N_\varphi = \lceil \log_2(b - a + 2) \rceil + (b - a + 1)N_\phi$
$\varphi = \phi_1 U_{[a,b]}\phi_2$	$N_\varphi = \lceil \log_2(b - a + 2) \rceil + \sum_{k'=k+a}^{k+b} [N_{\phi_2} + (k' - k)N_{\phi_1}]$

(ii) For ϕ_1, the continuous variables $q_k^{\mu_1}, k = 1, \dots, 15$ are used to indicate the satisfaction of μ_1 from time 1 to 15, and $\lceil \log_2(15 + 1) \rceil$ binary variables are then employed to realize the SOS1 encoding for *Finally* operation.

(iii) For ϕ_2, the continuous variables $q_k^{\mu_2}, k = 3, \dots, 6$ are introduced to indicate the satisfaction of μ_2 from time 3 to 6, and the *Globally* operation is then encoded based on (8).

Based on the above encoding, if we set $q_0^\varphi = 1$, the satisfaction of φ is enforced with ξ. The number of binary variables involved in the encoding can be calculated as $N_\varphi = N_{\phi_1} + N_{\phi_2} = \lceil \log_2(15 + 1) \rceil + 0 = 4$.

To further limit the required number of binary variables for disjunctive formulae, we propose an STL simplification method outlined below. We note that the simplification method described in this section is non-exhaustive, i.e., there may be other approaches applicable to this problem.

First, for an STL formula defined by (1), we determine the valid time instants for all of its (sub)formulae on the global time scale. Here, the valid time instants of a (sub)formula indicate the moments in time when the (sub)formula should be satisfied, and the global time scale refers to the real-world timeline, as opposed to the relative time requirements $[a, b]$ defined in the STL syntax (1). To facilitate the explanation, time instants on the global timeline are denoted by t in the following discussion. Here is a straightforward example of determining valid time instants for (sub)formulae. Consider the formula $(\xi, t) \vDash G_{[a,b]}\mu$. With the following equivalences:

$$(\xi, t) \vDash G_{[a,b]}\mu \quad \Leftrightarrow \quad \xi \vDash F_{[t,t]}(G_{[a,b]}\mu) \quad \Leftrightarrow \quad \xi \vDash \bigwedge_{t' \in [t+a, t+b]} F_{[t',t']}\mu,$$

we can obtain that formula $G_{[a,b]}\mu$ should be satisfied at global time instant t, and its sub-formulae μ should be satisfied during global time interval $[t + a, t + b]$.

Considering the binary variable calculation rules provided in Table 1, our goal is to decrease the number of valid sub-formulae $\phi_i, i \in [1, r]$ in $\bigvee_{i=1}^r \phi_i$ and reduce the length of valid time interval for $F_{[a,b]}\phi$ and $\phi_1 U_{[a,b]}\phi_2$ during the shrinking horizon MPC implementation. To this end, an ascending time set $\mathcal{T}_k = \{t_1, t_2, \dots\}$, $t_i \in \mathbb{N}, t_i \in [k, N]$ is designed in global time scale at each optimization time step k. The basic idea of the simplification is that, from the perspective of the global time scale, only the fragments of disjunctive (sub)formulae that intersect with time set $\mathcal{T}_k$ are valid. Given an STL formula φ, a signal ξ, and the current time set $\mathcal{T}_k$, the specific simplification procedure under shrinking horizon MPC is summarized as follows.

(i) Obtain the valid time instants of all (sub)formulae of φ on the global time scale.

(ii) For disjunctions $\bigvee_{i=1}^r \phi_i$ in φ: Suppose that $\bigvee_{i=1}^r \phi_i$ should be satisfied by signal ξ at global time instant t, i.e., $\xi \vDash F_{[t,t]}(\bigvee_{i=1}^r \phi_i)$ (here we omit other sub-formulae that exist in φ to focus on the handling of the disjunctions). If $t \in \mathcal{T}_k$, the number of valid sub-formulae remains m. If $t \notin \mathcal{T}_k$, the valid sub-formulae are reduced to those satisfied at time $k - 1$, denoted by $\phi_i, i \in C_k^\vee$, where $C_k^\vee = \{i \mid \xi(x(0), \mathbf{u}_N^{Full*}(k - 1)) \vDash F_{[t,t]}\phi_i\} \subseteq \{1, 2, \dots, r\}$ represents the collection of indicators for sub-formulae satisfied based on the solution obtained at time $k - 1$.

(iii) For *Finally* (sub)formulae $F_{[a,b]}\phi$ and *Until* (sub)formulae $\phi_1 U_{[a,b]}\phi_2$ in φ: Suppose that *Finally* (sub)formula $F_{[a,b]}\phi$ should be satisfied by signal ξ at global time instant t, which can be expressed as $\xi \vDash F_{[t,t]}(F_{[a,b]}\phi)$ or equivalently $\xi \vDash F_{[t+a,t+b]}\phi$ (here we omit other sub-formulae that exist in φ to focus on the handling of the *Finally* operations). If the interval $[t+a, t+b]$ does not intersect with $\mathcal{T}_k$, the valid time interval for the *Finally* formula is reduced to $\mathcal{F}_{k-1} = \{t \mid \xi(x(0), \mathbf{u}_N^{Full*}(k-1)) \vDash F_{[t,t]}\phi\} \subseteq \{t+a, t+a+1, \dots, t+b\}$, which corresponds to the set of global time instants where ϕ is satisfied based on the solution at time $k-1$. If $[t+a, t+b]$ intersects with $\mathcal{T}_k$, the set of valid time instants is updated to $\mathcal{T}_k^F = ([t+a, t+b] \cap \mathcal{T}_k) \cup \mathcal{F}_{k-1}$. Similarly, the simplification for the *Until* formula $\phi_1 U_{[a,b]}\phi_2$ can be performed, with $\mathcal{U}_{k-1} = \{t \mid \xi(x(0), \mathbf{u}_N^{Full*}(k-1)) \vDash \phi_1 U_{[t,t]}\phi_2\}$ and $\mathcal{T}_k^U = ([t+a, t+b] \cap \mathcal{T}_k) \cup \mathcal{U}_{k-1}$.

Based on the above statements, the simplified form of φ under the time set $\mathcal{T}_k$, denoted by $R_k[\varphi]$, is summarized as follows.

Definition 1 (*Simplified STL Formulae*)**.**

$$
\begin{aligned}
R_k[F_{[t,t]}\mu] &= F_{[t,t]}\mu = \alpha(x(t)) \geq 0\\
R_k[F_{[t,t]}(\neg\mu)] &= F_{[t,t]}(\neg\mu) = -\alpha(x(t)) \geq 0\\
R_k[F_{[t,t]}(\bigwedge_{i\in[1,r]}\phi_i)] &= \bigwedge_{i\in[1,r]} R_k[F_{[t,t]}\phi_i]\\
R_k[F_{[t,t]}(\bigvee_{i\in[1,r]}\phi_i)] &= \begin{cases} \bigvee\limits_{i\in C_k^\vee} R_k[F_{[t,t]}\phi_i], & \text{if } t \notin \mathcal{T}_k \\ \bigvee\limits_{i\in[1,r]} R_k[F_{[t,t]}\phi_i], & \text{if } t \in \mathcal{T}_k \end{cases}\\
R_k[F_{[t,t]}(G_{[a,b]}\phi)] &= \bigwedge_{t'=t+a}^{t+b} R_k[F_{[t',t']}\phi]\\
R_k[F_{[t,t]}(F_{[a,b]}\phi)] &= \begin{cases} \bigvee\limits_{t'\in\mathcal{F}_{k-1}} R_k[F_{[t',t']}\phi], & \text{if } [t+a,t+b]\cap\mathcal{T}_k = \varnothing \\ \bigvee\limits_{t'\in\mathcal{T}_k^F} R_k[F_{[t',t']}\phi], & \text{if } [t+a,t+b]\cap\mathcal{T}_k \neq \varnothing \end{cases}\\
R_k[F_{[t,t]}(\phi_1 U_{[a,b]}\phi_2)] &= \begin{cases} \bigwedge\limits_{t'\in\mathcal{U}_{k-1}} (R_k[F_{[t',t']}\phi_2] \wedge \bigwedge\limits_{t''=t}^{t'} R_k[F_{[t'',t'']}\phi_1]), \\ \qquad\qquad \text{if } [t+a,t+b]\cap\mathcal{T}_k = \varnothing \\ \bigvee\limits_{t'\in\mathcal{T}_k^U} (R_k[F_{[t',t']}\phi_2] \wedge \bigwedge\limits_{t''=t}^{t'} R_k[F_{[t'',t'']}\phi_1]), \\ \qquad\qquad \text{if } [t+a,t+b]\cap\mathcal{T}_k \neq \varnothing. \end{cases}
\end{aligned}
$$

The simplifications presented above transform the original formula φ into a new formula $R_k[\varphi]$. It is evident that the language of $R_k[\varphi]$ is a subset of the language of φ. In other words, for a given system run ξ, we have:

$$\xi \vDash R_k[\varphi] \quad \Rightarrow \quad \xi \vDash \varphi.$$

After obtaining a simplified formula $R_k[\varphi]$, we can encode its satisfaction into MILP constraints. The number of required binary variables for the encoding of $R_k[\varphi]$, denoted by $\bar{N}_\varphi^k$, can be calculated based on the rules given in Table 1, and is summarized recursively as in Table 2. It can be deduced that, the MILP encoding for the simplified formula $R_k[\varphi]$ requires $O(\log_2(|\mathcal{T}_k|+1))$ binary variables at each time step k. In contrast, without formula simplification, the encoding results in $O(\log_2(N-k+1))$ binary variables. Since $|\mathcal{T}_k|$ is a customizable constant, our method offers computational speed advantages, particularly when task length N is large.

Remark 1. The design of time set $\mathcal{T}_k$ affects the similarity between simplified tasks and the original tasks, thereby exerting a considerable impact on control optimality. Some heuristic methods for the design of $\mathcal{T}_k$ are provided in the following. One approach involves situating $\mathcal{T}_k$ at the initial horizon of the optimization to maintain fine-grained control actions at the outset, as stated in Section 6.1. Alternatively, one can consider placing and adjusting $\mathcal{T}_k$ during active periods of disjunctive (sub)formulae to enhance control flexibility. Additionally, when there is a high demand for computing speed, removing all binary variables from the optimization problem is preferable. In such cases, $\mathcal{T}_k$ can be chosen as an empty set at each time step k, enabling the reduction of MILP constraints to linear constraints on continuous variables. Theoretical support for the design of time set $\mathcal{T}_k$ is a nontrival research subject in ongoing and future work.

Table 2
Number of required binary variables for the MILP encoding of simplified STL formulae under shrinking horizon MPC.

STL formula	The Number of required binary variables for $R_k[\varphi]$
$\varphi = F_{[t,t]}\mu$	$\bar{N}_\varphi^k = 0$
$\varphi = F_{[t,t]}(\neg\mu)$	$\bar{N}_\varphi^k = 0$
$\varphi = F_{[t,t]}(\bigvee_{i=1}^r \phi_i)$	$\bar{N}_\varphi^k = \begin{cases} \lceil\log_2(r+1)\rceil + \sum\limits_{i\in[1,r]} \bar{N}^k_{F_{[t,t]}\phi_i}, & t \in \mathcal{T}_k \\ \lceil\log_2(\lvert C_k^\vee\rvert+1)\rceil + \sum\limits_{i\in C_k^\vee} \bar{N}^k_{F_{[t,t]}\phi_i}, & t \notin \mathcal{T}_k \end{cases}$
$\varphi = F_{[t,t]}(\bigwedge_{i=1}^r \phi_i)$	$\bar{N}_\varphi^k = \sum_{i=1}^r \bar{N}^k_{F_{[t,t]}\phi_i}$
$\varphi = F_{[t,t]}(G_{[a,b]}\phi)$	$\bar{N}_\varphi^k = \sum_{t'=t+a}^{t+b} \bar{N}^k_{F_{[t',t']}\phi}$
$\varphi = F_{[t,t]}(F_{[a,b]}\phi)$	$\bar{N}_\varphi^k = \begin{cases} \lceil\log_2(\lvert\mathcal{F}_{k-1}\rvert+1)\rceil + \sum\limits_{t'\in\mathcal{F}_{k-1}} \bar{N}^k_{F_{[t',t']}\phi}, & [t+a,t+b]\cap\mathcal{T}_k = \varnothing \\ \lceil\log_2(\lvert\mathcal{T}_k^F\rvert+1)\rceil + \sum\limits_{t'\in\mathcal{T}_k^F} \bar{N}^k_{F_{[t',t']}\phi}, & [t+a,t+b]\cap\mathcal{T}_k \neq \varnothing \end{cases}$
$\varphi = F_{[t,t]}(\phi_1 U_{[a,b]}\phi_2)$	$\bar{N}_\varphi^k = \begin{cases} \lceil\log_2(\lvert\mathcal{U}_{k-1}\rvert+1)\rceil + \sum\limits_{t'\in\mathcal{U}_{k-1}} (\sum\limits_{t''\in[t,t']} \bar{N}^k_{F_{[t'',t'']}\phi_1} + \bar{N}^k_{F_{[t',t']}\phi_2}), & [t+a,t+b]\cap\mathcal{T}_k = \varnothing \\ \lceil\log_2(\lvert\mathcal{T}_k^U\rvert+1)\rceil + \sum\limits_{t'\in\mathcal{T}_k^U} (\sum\limits_{t''\in[t,t']} \bar{N}^k_{F_{[t'',t'']}\phi_1} + \bar{N}^k_{F_{[t',t']}\phi_2}), & [t+a,t+b]\cap\mathcal{T}_k \neq \varnothing \end{cases}$

4. Shrinking horizon move blocked MPC for STL synthesis

To further enhance computational efficiency, we incorporate the move blocking scheme into the STL-MPC synthesis in this section. To make the paper more self-contained, we first give a brief introduction of move blocking mechanism in the following.

4.1. Move blocking scheme

Move blocking is a kind of input parameterization that obtain lower complexity optimization problem formulations by keeping the inputs constant over a specified number of consecutive time steps, referred to as blocks. A schematic illustration of input blocking is shown in Fig. 1. The structure of the blocking is defined by a block matrix S. An admissible blocking matrix is one that allows the representation

$$S = \begin{bmatrix} b_1 & & & \\ & b_2 & & \\ & & \ddots & \\ & & & b_{M_S} \end{bmatrix}_{N\times M_S} = \bigoplus_{q=1}^{M_S} b_q,$$

where $M_S \in \mathbb{N}_{\geq 1}$ represents the number of blocks in S, which is also referred to as the degrees of freedom, indicating the number of free inputs; $b_q = \mathbf{1}_{l(S,q)}$, $q \in [1, M_S]$ denotes a block vector with $l(S,q) \in \mathbb{N}_{\geq 1}$ indicating the length of the qth block in S. If a block has only one element, i.e., $l(S,q) = 1$, it is considered to have no blockage. Denote blocking position set of S as $\{s_1, s_2, \dots, s_{M_S}\}$, where $s_q \in \mathbb{N}_{\leq N}$, $q \in$

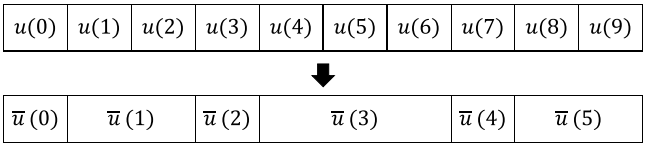


Fig. 1. A schematic illustration of move blocking: $N = 10, M_S = 6$, and the block position set is $\{1, 2, 4, 5, 8, 9\}$.

$[1, M_S]$ indicates the position where the qth block begins. The input variables can then be blocked with the blocking matrix by

$$\begin{aligned}&[u^{\mathrm{T}}(0), u^{\mathrm{T}}(1), \dots, u^{\mathrm{T}}(N-1)]^{\mathrm{T}}\\ =&(S \otimes I_{d_u})[\bar{u}^{\mathrm{T}}(0), \bar{u}^{\mathrm{T}}(1), \dots, \bar{u}^{\mathrm{T}}(M_S-1)]^{\mathrm{T}}.\end{aligned} \tag{10}$$

In this way, an optimization problem subject to (10) can solve for M_S variables $\bar{u}(0), \bar{u}(1), \dots, \bar{u}(M_S-1)$ instead of N original input variables $u(0)$, $u(1)$, …, $u(N-1)$. Let $\mathcal{A}(N, M_S)$ denote a set-valued map, which returns all the admissible blocking matrices that subdivide N elements into M_S blocks. A relaxation of a blocking matrix S is a blocking matrix in which blocks can be amalgamated to S. The set of all the relaxations of S is denoted by

$$\mathcal{X}(S) = \{S' \in \mathbb{R}^{N \times M_{S'}} \mid \exists Q \in \mathcal{A}(M_{S'}, M_S) : S = S'Q, M_{S'} \geq M_S\}.$$

The identity blocking matrix (i.e. no blockage) is a relaxation of any admissible blocking matrix, and each blocking matrix is also a relaxation of itself.

4.2. Input blocking for STL synthesis

When incorporating the blocking scheme into temporal logic control, limiting the degrees of freedom of control inputs can have an impact on system control performance. Consider the following example.

Example 2. In this scenario, we have a single integrator agent with two states representing the agent's position and two control inputs for movement along two directions. The agent is required to visit G_1 within time step 1 to 2, visit G_2 before time step 12, and always stay away from unsafe areas Obs. The STL task can then be formulated as

$$\phi = F_{[1,2]}(x \in \mathrm{G}_1) \wedge F_{[0,12]}(x \in \mathrm{G}_2) \wedge G_{[0,12]}(x \notin \mathrm{Obs}). \tag{11}$$

The trajectories of the agent with full degrees of freedom (no input blocking) and with 3 degrees of freedom (blocking position set $\{1, 2, 7\}$) are shown in Fig. 2. One can see that under the blocking scheme, the agent needs to follow a longer path and exert more control efforts to satisfy the given STL task compared to the full degree of freedom case. Besides, since there are strict time and logic constraints in STL tasks, the selection of the number of blocks and their positions are also crucial.

Motivated by the discussions in Example 2, the following optimization problem, denoted as P_0, is formulated to determine an initial blocking matrix S_0. This matrix aims to balance the computational burden, a specified cost function J, and ensure STL formula satisfaction:

$$\min_{\mathbf{u}_N(0)} J(x(0), \mathbf{u}_N(0)) + \lambda J_b(\mathbf{u}_N(0)), \tag{12}$$

$$\text{s.t. } x(k+1) = Ax(k) + Bu(k), \tag{12a}$$

$$\xi(x(0), \mathbf{u}_N(0)) \vDash \varphi, \tag{12b}$$

where λ denotes a weighting scalar and $J_b(\mathbf{u}_N(0)) = \sum_{t=0}^{N-2} |\mathrm{sign}(\sum_{i=1}^{d_u} |u_i(t+1) - u_i(t)|)|$ indicates blocking cost. Here u_i denotes the ith component of u, and J_b is minimized to decrease the number of changes in input variables. By performing MILP encoding on the STL constraint (12b), P_0 can be transformed into an MIP problem. Solving P_0 yields an initial optimal input vector $\mathbf{u}_N^*(0) = [u^{*\mathrm{T}}(0), u^{*\mathrm{T}}(1), \dots, u^{*\mathrm{T}}(N-1)]^{\mathrm{T}}$, and the initial blocking matrix S_0 can be derived from the structure of $\mathbf{u}_N^*(0)$. Specifically, adjacent decision variables in time that share the same value are constrained to create a block. This allows us to deduce the positions of these blocks and construct S_0 accordingly. Note that as extra binary variables are introduced to model the blocking costs $J_b(\mathbf{u}_N(0))$, solving P_0 may result in a high computational burden. It is acceptable for the initialization since it can be carried out offline. For the subsequent online optimizations, candidate block matrices are provided, as outlined in Definition 2, and the feasibility of these candidate block matrices is further analyzed in Section 5.2.

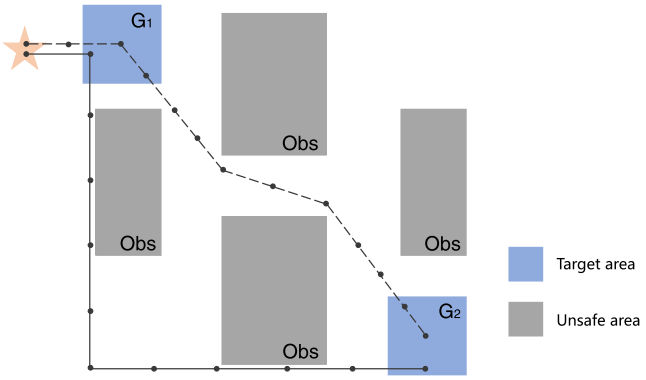


Fig. 2. Workspace under STL task (11). The figure displays agent trajectories using the full degree of freedom input variables (dotted line) and the 3 degrees of freedom input variables (solid line). Both trajectories meet the requirements of task (11), but they differ in terms of control efforts.

Definition 2 (*Candidate Block Matrix*)**.** Define the block shifting function $F(S)$, $S \in \mathcal{A}(N, M_S)$ as

$$F(S) = \begin{cases} [0_{(N-1)\times 1} \;\; I_{N-1}]S, & l(S,1) > 1 \\ [0_{(N-1)\times 1} \;\; I_{N-1}]S \begin{bmatrix} 0_{1\times(M_S-1)} \\ I_{M_S-1} \end{bmatrix}, & l(S,1) = 1 \end{cases}$$

which essentially removes the first row of S and also the first column if $l(S,1) = 1$ to ensure admissibility of the resulting blocking matrix. Suppose that the blocking matrix used at time k is S_k. Then, at time $k+1$, the candidate block matrices are those belonging to the set $\mathcal{X}(F(S_k))$.

5. Low complexity model predictive control algorithm for STL synthesis

In this section, we summarize the proposed STL-MPC algorithm and further analyze its feasibility. We also show that through the use of STL constraint tightening, the algorithm is applicable to uncertain systems, and the STL task completion in the presence of bounded disturbances is guaranteed.

5.1. The proposed algorithm

Building upon the analysis presented in Sections 3 and 4, we formulate the move blocked shrinking horizon MPC problem at time k, denoted by P_k, as follows:

$$\min_{\bar{\mathbf{u}}_{M_{S_k}}(0)} J(x(k), \mathbf{u}_N(k)) \tag{13}$$

$$\text{s.t. } x(k+1) = Ax(k) + Bu(k), \tag{13a}$$

$$\xi(x(0), \mathbf{u}_N^{Full}(k)) \vDash R_k[\varphi], \tag{13b}$$

$$\mathbf{u}_N(k) = (S_k \otimes I_{d_u})\bar{\mathbf{u}}_{M_{S_k}}(0), \tag{13c}$$

where $\bar{\mathbf{u}}_{M_{S_k}}(0) = [\bar{u}^{\mathrm{T}}(0), \bar{u}^{\mathrm{T}}(1), \dots, \bar{u}^{\mathrm{T}}(M_{S_k}-1)]^{\mathrm{T}}$ indicates the reduced input vector from $\mathbf{u}_N(k)$, and $S_k \in \mathcal{X}(F(S_{k-1}))$ denotes the block matrix employed at time k. The overall low complexity STL-MPC algorithm is summarized as in Algorithm 1.

Algorithm 1: Low Complexity Model Predictive Control Algorithm for STL Synthesis

Input: System dynamics (5), initial state $x(0)$, STL task φ, and the global map of the environment.

Initialization: Solve P_0 to obtain S_0 and $\mathbf{u}_N^*(0)$;
while $k \le N-1$ **do**
  Decide $\mathcal{T}_k$ and construct $R_k[\varphi]$ based on Definition 1;
  Update $\mathcal{X}(F(S_{k-1}))$ and choose $S_k \in \mathcal{X}(F(S_{k-1}))$;
  Solve P_k and apply $u^*(k)$;
  $k \leftarrow k+1$;
end

Output: Optimal solutions $u^*(0), \dots, u^*(N-1)$.

5.2. Performance analysis

Given that the satisfaction of $R_k[\varphi]$ implies the satisfaction of the original formula φ, we have that any solution to $P_k, k = 0, \dots, N-1$ can guarantee the satisfaction of φ, which implies the soundness of the proposed algorithm. The feasibility of the algorithm is further analyzed as follows.

Theorem 1. *If there exists a feasible solution for problem P_0 at time $k=0$ and the system is controlled according to Algorithm 1, then all subsequent optimization problems P_k are feasible for $k>0$.*

Proof. Suppose that the initial solution $\mathbf{u}_N^*(0)$ and the initial block matrix S_0 are obtained at time $k=0$. The feasibility of STL constraint (13b) and block constraint (13c) at time $k=1$ is analyzed in the following. For STL constraints, the simplified STL formula $R_1[\varphi]$ is employed in problem P_1, while the non-simplified STL formula φ is employed in problem P_0. According to Definition 1, only disjunctive (sub)formulae, including $\bigvee_{i=1}^r \phi_i$, $F_{[a,b]}\phi$, and $\phi_1 U_{[a,b]}\phi_2$, are simplified. For (sub)formulae $\bigvee_{i=1}^r \phi_i$, the valid formulae are chosen as $\bigvee_{i=1}^r \phi_i$ or $\bigvee_{i\in C_k^\vee} \phi_i$, both of them include the sub-formulae satisfied by $\mathbf{u}_N^*(0)$ at time $k=0$. For $F_{[a,b]}\phi$, the valid time interval is chosen as $\mathcal{F}_{k-1}$ or $\mathcal{T}_k^F$, both of them consist of the time instants that ϕ are satisfied at time $k=0$ (The case for *Until* (sub)formulae is similar). In other words, the (sub)formulae that are satisfied by $\mathbf{u}_N^*(0)$ at time $k=0$ are all involved in the simplified one $R_1[\phi]$ at time $k=1$. Consequently, the control vector

$$\tilde{\mathbf{u}}_N(1) = [u^{*\mathrm{T}}(1|0), \dots, u^{*\mathrm{T}}(N-1|0)]^{\mathrm{T}},$$

which is constructed by removing the first element in $\mathbf{u}_N^*(0)$, directly satisfies (13b) at time $k=1$. We proceed to assess the feasibility of the block constraint (13c). At time $k=1$, according to Definition 2, $F(S_0)$ eliminates the first row of S_0, indicating that the control input $u^*(0)$ has been executed at time $k=1$. Moreover, the block relaxation $\mathcal{X}(F(S_0))$ essentially relaxes the blocking requirements compared with $F(S_0)$. Thus, the form of the control vector $\tilde{\mathbf{u}}_N(1)$ fits all the block structures in $\mathcal{X}(F(S_0))$, implying that constraint (13c) in problem P_1 is satisfied with $\tilde{\mathbf{u}}_N(1)$ for any $S_1 \in \mathcal{X}(F(S_0))$. From the above, $\tilde{\mathbf{u}}_N(1)$ satisfies constraints (13b) and (13c), and thus is a feasible solution for P_1. A similar deduction can be made for $k>1$. We can then conclude that

$$\tilde{\mathbf{u}}_N(k) = [u^{*\mathrm{T}}(k|k-1), \dots, u^{*\mathrm{T}}(N-1|k-1)]^{\mathrm{T}},$$

which is constructed from the solution at time $k-1$, is always a feasible solution for problem P_k at each time step k, i.e., optimization problems P_k are feasible for all $k>0$. ■

5.3. Extension to uncertain case

We then extend the proposed algorithm to include robustness to bounded disturbances. Consider the following discrete-time system with additive uncertainties:

$$x(k+1) = Ax(k) + Bu(k) + w(k), \tag{14}$$

where disturbances $w(0), w(1), \dots$ are assumed to be independent and identically distributed (i.i.d), zero mean stochastic variables with bounded, compact and convex support $\mathcal{W}$.

Let $\mathbf{w}_N(k) = [w^{\mathrm{T}}(k|k), \dots, w^{\mathrm{T}}(N-1|k)]^{\mathrm{T}}$ denote a disturbance vector consisting of the disturbances from time step k to time step $N-1$. Given an initial state $x(k)$, a control vector $\mathbf{u}_N(k)$, and a disturbance vector $\mathbf{w}_N(k)$, one can obtain a stochastic state sequence $\Xi(x(k), \mathbf{u}_N(k), \mathbf{w}_N(k)) = x(k)x(k+1|k)\dots x(N|k)$ based on (14). Define the disturbance vector over the whole planning horizon as $\mathbf{w}_N^{Full}(k) = [w^{\mathrm{T}}(0), w^{\mathrm{T}}(1), \dots, w^{\mathrm{T}}(k-1), w^{\mathrm{T}}(k|k), \dots, w^{\mathrm{T}}(N-1|k)]^{\mathrm{T}}$, where $w(0)$, $w(1)$, $\dots$, $w(k-1)$ are fixed as the observations of disturbances. Similarly to the nominal case, we employ the simplified STL formulae and the move blocking scheme, and the robust online optimization problem at time k can be constructed as follows.

Problem 2. Consider the uncertain system (14) and an STL task φ. Given an initial state $x(0)$ and a constant $N \ge \mathrm{len}(\varphi)$, find control input $u(k)$ for the system at each time step k by solving the following optimization problem, so that the resulting trajectory satisfies φ against uncertainties:

$$\min_{\bar{\mathbf{u}}_{M_{S_k}}(0)} \mathbb{E}[J(x(k), \mathbf{u}_N(k))] \tag{15}$$

$$\text{s.t. } x(k+1) = Ax(k) + Bu(k) + w(k), \tag{15a}$$

$$\Xi(x(0), \mathbf{u}_N^{Full}(k), \mathbf{w}_N^{Full}(k)) \vDash R_k[\varphi], \tag{15b}$$

$$\mathbf{u}_N(k) = (S_k \otimes I_{d_u})\bar{\mathbf{u}}_{M_{S_k}}(0), \tag{15c}$$

where J is a stage cost function with expectation taken with respect to disturbance $\mathbf{w}_N(k)$, and we assume that $\mathbb{E}[J(x(k), \mathbf{u}_N(k))]$ can be evaluated analytically, which is the case for quadratic J.

Due to the existence of disturbances and the complex form of STL formulae, the exact feasible set of the optimizations cannot be characterized directly. In the following, we aim to transform Problem 2 into a sequence of solvable deterministic optimization problems imposed on a nominal system. Specifically, the nominal system corresponding to system (14) is given as

$$z(k+i|k) = A_K z(k+i-1|k) + Bv(k+i-1|k), \tag{16}$$

where z, v denote the nominal state and input, respectively; $A_K = A + BK$ with K being a feedback gain such that A_K is stable. Given an initial nominal state $z(k) = x(k)$ and a nominal input vector $\boldsymbol{v}_N(k)$ at time k, we can obtain a prediction of nominal path $\xi(x(k), \boldsymbol{v}_N(k)) = z(k)z(k+1|k)\dots z(N|k)$ based on (16). Note that as the nominal state $z(k)$ is consistently updated to match the actual state $x(k)$ at each time step k, we consider $\xi(x(0), \boldsymbol{v}_N^{Full}(k))$ to represent the sequence $x(0)\dots x(k)z(k+1|k)\dots z(N|k)$. Choose $u(k) = Kx(k) + v(k)$ and define the error between the actual system (14) and the nominal one (16) as $e(k+i|k) = x(k+i|k) - z(k+i|k)$, we have:

$$e(k+i+1|k) = A_K e(k+i|k) + w(k+i|k),\ e(k|k) = 0. \tag{17}$$

We then transform constraint (15b) into deterministic one imposed on nominal system (16). Before proceeding, we give the following lemma.

Lemma 1. *All the STL formulae given by (1) can be reformulated into the Boolean combination (conjunctions and disjunctions) over multiple atomic predicates.*

Proof. For (sub)formulae μ, $\neg\mu$, $\mu_1 \vee \mu_2$, and $\mu_1 \wedge \mu_2$, the conclusion is trivial. For (sub)formulae with temporal operations $F_{[a,b]}\phi$, $G_{[a,b]}\phi$, and $\phi_1 U_{[a,b]}\phi_2$, we draw the conclusion based on the facts listed in (2)–(4) with $F_{[k',k']}\mu = \{\alpha(x(k')) \geq 0\}$. ■

Let notation $\mathbf{B}^{\varphi}_{m\in[1,M_\varphi]}(\mu_m)$, $M_\varphi \in \mathbb{N}_{\geq 1}, m \in \mathbb{N}_{\geq 1}$ indicate the Boolean combination of M_φ predicates $\mu_1, \dots, \mu_{M_\varphi}$ transformed from φ. Denote $\alpha_m(x) = C_m x(k + i^k_m|k) + d_m, C_m \in \mathbb{R}^{1\times n}, d_m \in \mathbb{R}$ as a predicate function associated with μ_m. Then based on Lemma 1, STL constraint (15b) can be reformulated into the Boolean combination of $M_{R_k[\varphi]}$ predicates as follows:

$$\mathbf{B}^{R_k[\varphi]}_{m\in[1,M_{R_k[\varphi]}]}(C_m x(k + i^k_m|k) + d_m \geq 0), \tag{18}$$

where the number of predicates $M_{R_k[\varphi]}$, the values of C_m and d_m, the time instants $k + i^k_m$, $i^k_m \in \{1, \dots, N-k\}, m = 1, 2, \dots, M_{R_k[\varphi]}$, and the Boolean operators $(\wedge, \vee)$ between predicates are all determined inductively on the structure of $R_k[\varphi]$. Note that each predicate μ_m in the Boolean combination is imposed on system states of a specific time instant $k + i^k_m$, that is, the value of i^k_m changes with the MPC implementation to ensure that $k + i^k_m$ is a fixed value. In addition, once states x are observed at time k, they would appear as fixed true values in (18).

Theorem 2. *For Problem 2, task $R_k[\varphi]$ is satisfied at time k if with $\mathbf{v}_N(k)$, the nominal path $\xi(x(0), \mathbf{v}^{Full}_N(k))$ satisfies the following tightened STL formula:*

$$R^t_k[\varphi] = \mathbf{B}^{R_k[\varphi]}_{m\in[1,M_{R_k[\varphi]}]}(C_m z(k + i^k_m|k) + d_m \geq -\gamma_{i^k_m}),$$

where

$$\gamma_{i^k_m} = \min_{w\in\mathcal{W}} \sum_{s\in[0,i^k_m-1]} C_m A^s_K w. \tag{19}$$

In addition, if $R^t_k[\varphi]$ is satisfied at time k, then $R^t_{k+1}[\varphi], \dots, R^t_{N-1}[\varphi]$ can be satisfied at time $k+1, \dots, N-1$, respectively.

Proof. We begin by the special case where $R_k[\varphi]$ is a predicate and later extend our analysis to the general case. For $R_k[\varphi] = C_m x(k + i^k_m|k) + d_m \geq 0, m = 1$, along with the tightening parameter $\gamma_{i^k_m}$ defined in (19), the following two conclusions can be drawn (Kouvaritakis, Cannon, Raković, & Cheng, 2010): (i) $C_m z(k + i^k_m|k) + d_m \geq -\gamma_{i^k_m} \Rightarrow C_m x(k + i^k_m|k) + d_m \geq 0$; (ii) If inequality $C_m z(k+i^k_m|k)+d_m \geq -\gamma_{i^k_m}$ holds at time k, it can remain satisfied at time $k + j, j = 1, \dots, i^k_m - 1$. The interested reader is referred to Kouvaritakis et al. (2010) for the detailed proof. Extend the conclusions to the Boolean combination of multiple predicates, we have

$$\xi(x(0), \mathbf{v}^{Full}_N(k)) \vDash R^t_k[\varphi] \Rightarrow \Xi(x(0), \mathbf{u}^{Full}_N(k), \mathbf{w}^{Full}_N(k)) \vDash R_k[\varphi].$$

Define F^j as a vector shifting function that removes the first j elements in a vector, i.e., $F^j(\mathbf{v}_N(k)) = [v^{\mathrm{T}}(k + j|k), \dots, v^{\mathrm{T}}(N-1|k)]^{\mathrm{T}}$. It follows that if $\mathbf{v}_N(k)$ is a feasible solution for $\xi(x(0), \mathbf{v}^{Full}_N(k)) \vDash R^t_k[\varphi]$ at time k, the nominal paths generated by $F^1(\mathbf{v}_N(k)), \dots,$ and $F^{N-k-1}(\mathbf{v}_N(k))$ satisfy $R^t_k[\varphi]$ at time $k + j, j = 1, \dots, N-k-1$, respectively, where the tightening parameters $\gamma_{i^k_m}$ for $R_k[\varphi]$ should be updated at each time step according to (19). At time $k+1$, the simplified STL formula is updated as $R_{k+1}[\varphi]$. Based on the simplification rules, we can conclude that the (sub)formulae of $R^t_k[\varphi]$ satisfied at time k are entirely encompassed by $R^t_{k+1}[\varphi]$. Thus, $F^1(\mathbf{v}_N(k))$ is a feasible solution for the satisfaction of $R^t_{k+1}[\varphi]$ at time $k+1$. By recursion, if $R^t_k[\varphi]$ is satisfied at time k, then $R^t_{k+1}[\varphi], \dots,$ and $R^t_{N-1}[\varphi]$ can be satisfied at time $k+1, \dots, N-1$, respectively. ■

Based on Theorem 2, the solvable deterministic optimization problem, denoted by $\hat{P}_k$, can be formulated as:

$$\min_{\bar{\mathbf{v}}_{M_{S_k}}(0)} \mathbb{E}[J(x(k), \mathbf{u}_N(k))] \tag{20}$$

$$\text{s.t. } z(k+1) = A_K z(k) + Bv(k), \tag{20a}$$

$$\xi(x(0), \mathbf{v}^{Full}_N(k)) \vDash R^t_k[\varphi], \tag{20b}$$

$$\mathbf{v}_N(k) = (S_k \otimes I_{d_u})\bar{\mathbf{v}}_{M_{S_k}}(0). \tag{20c}$$

Similar to the nominal case, an initial input solution and a reasonable blocking matrix S_0 is required for $\hat{P}_k$. These can be obtained by solving the following initial problem $\hat{P}_0$:

$$\min_{\mathbf{v}_N(0)} \mathbb{E}[J(x(0), \mathbf{u}_N(0))] + \lambda J_b(\mathbf{v}_N(0))$$
$$\text{s.t. } z(k+1) = A_K z(k) + Bv(k),$$
$$\xi(x(0), \mathbf{v}_N(0)) \vDash \varphi^t,$$

where $J_b(\mathbf{v}_N(0)) = \sum_{t=0}^{N-2} |\text{sign}(\sum_{i=1}^{d_u} |v_i(t+1) - v_i(t)|)|$ with v_i indicating the ith component of v, and φ^t is constructed by tightening φ according to Theorem 2.

So far, through the formula reformulation given in Lemma 1 and constraint tightening in Theorem 2, we transform Problem 2 into solvable deterministic problems $\hat{P}_0$ and $\hat{P}_k, k = 1, \dots, N-1$. By applying Algorithm 1 with problems P_0 and P_k replaced by the problems $\hat{P}_0$ and $\hat{P}_k$, respectively, the STL task satisfaction under uncertainties can be realized with efficient online computation. The feasibility of constraint (20b) with previously obtained solutions is established by Theorem 2, and the design principles of candidate blocking matrices guarantee the satisfaction of constraint (20c). It can then be inferred that problem $\hat{P}_k$ remains feasible at each time step k, underscoring the feasibility of the proposed algorithm in this uncertain scenario.

6. Case study

The algorithm discussed in the previous sections provides flexibility in selecting the time set $\mathcal{T}_k$ and the block matrix $S_k \in \mathcal{X}(F(S_{k-1}))$. A specific selection method is presented in this section. On this basis, numerical simulations are carried out to showcase the effectiveness of the proposed MPC algorithm in nominal cases, while robot experiments are conducted to evaluate the performance in the presence of system uncertainties.

6.1. The partitioned horizon

A specific selection method for time set $\mathcal{T}_k$ and the block matrix $S_k \in \mathcal{X}(F(S_{k-1}))$ is presented in this section. The approach involves dividing the predictive horizon into two segments: the initial sub-horizon and the remaining sub-horizon. To maintain fine-grained control action at the start, there is no blockage applied at the initial sub-horizon. The initial sub-horizon is also chosen as the valid time set $\mathcal{T}_k$ for each time step k. Let H_{is} represent the length of the initial sub-horizon, which is constrained within a range $[l_{is}, u_{is}]$. We then define the time set $\mathcal{T}_k$ at time k as $\mathcal{T}_k = [k, k+H_{is}]$. Let $\mathcal{B}_k \subseteq \mathcal{X}(F(S_{k-1}))$ indicate the collection of feasible block matrices at time k, and let $\tilde{S}^n_k, n \in \mathbb{N}_{\geq 1}$ represent the nth feasible block matrix in $\mathcal{B}_k$. Suppose that the blocking matrix employed at time k is $S_k \in \mathcal{B}_k$. Then at time step $k+1$, the set of feasible block matrices $\mathcal{B}_{k+1} \subseteq \mathcal{X}(F(S_k))$ can be constructed following the steps below:

Step 1: Obtain $F(S_k)$;

Step 2: Split the elements in subsequent block vectors one by one into the initial sub-horizon, such that the length of the initial sub-horizon satisfies $H_{is} \in [l_{is}, u_{is}]$. For each time split:

(i) If the current matrix satisfies $H_{is} < l_{is}$, proceed to the next splitting;

(ii) If the current matrix satisfies $H_{is} \in [l_{is}, u_{is})$, include the current matrix into $\mathcal{B}_{k+1}$ and proceed to the next splitting;

(iii) If the current matrix satisfies $H_{is} \geq u_{is}$, combine the matrix into $\mathcal{B}_{k+1}$, and stop.

We illustrate the construction procedure of $\mathcal{B}_{k+1}$ with the following example.

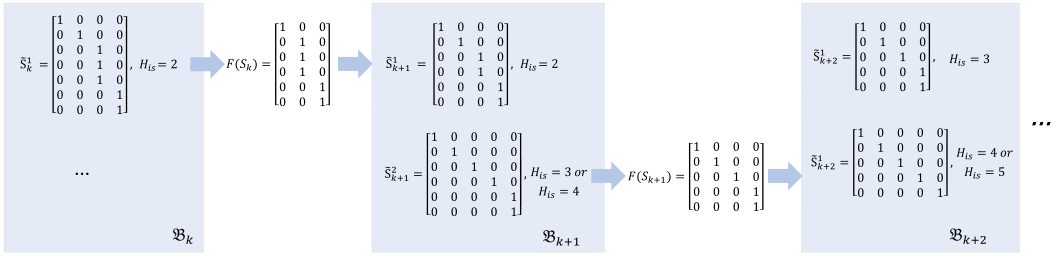


Fig. 3. The construction process of admissible matrix set under partitioned horizon.

Example 3. Suppose that the blocking matrix employed at time k is $S_k = \tilde{S}_k^1$, as shown in Fig. 3. Let $l_{is} = 2, u_{is} = 4$. At time $k+1$, we first remove the first column and also the first row of S_k and obtain $F(S_k)$. Since the length of the current initial sub-horizon is $H_{is} = l(F(S_k), 1) = 1 < l_{is}$, the subsequent block vectors should be split. If we split one element in block vector b_2 of $F(S_k)$ into the initial sub-horizon, we can obtain $\tilde{S}_{k+1}^1$ with $H_{is} = 2$; If we split two elements in block vector b_2 of $F(S_k)$ into the initial sub-horizon, we can obtain $\tilde{S}_{k+1}^2$ with $H_{is} = 3$, (which can also be regarded as $H_{is} = 4$ since $l(\tilde{S}_{k+1}^2, 4) = 1$). Stop splitting. These two matrices compose the feasible block matrix set $\mathcal{B}_{k+1} = \{\tilde{S}_{k+1}^1, \tilde{S}_{k+1}^2\}$. If $\tilde{S}_{k+1}^2$ is chosen as the block matrix at time $k+1$, i.e., $S_{k+1} = \tilde{S}_{k+1}^2$, we can further construct $\mathcal{B}_{k+2} = \{\tilde{S}_{k+2}^1, \tilde{S}_{k+2}^2\}$ by the similar procedure, see Fig. 3.

6.2. Simulation: Nominal case

In this section, we focus on nominal systems and conduct simulations to compare the control performance of the proposed algorithm with state-of-the-art methods.

Scenario 1: Consider an agent governed by double-integrator dynamics:

$$x(k+1) = \begin{bmatrix} I_2 & I_2 \\ 0 & I_2 \end{bmatrix} x(k) + \begin{bmatrix} 0.5I_2 \\ I_2 \end{bmatrix} u(k),$$

where $x = [p^{\mathrm{T}}, o^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^4$, and $p = [p_x, p_y]^{\mathrm{T}} \in \mathbb{R}^2$, $o = [o_x, o_y]^{\mathrm{T}} \in \mathbb{R}^2$, $u = [u_x, u_y]^{\mathrm{T}} \in \mathbb{R}^2$ are the position, the velocity, and the acceleration of the agent, respectively. The input constraint is given as $\|u\|_\infty \le 0.5$. The sampling interval is chosen as 1 s. The workspace for the agent is shown in Fig. 6. The agent must always avoid obstacles Obs, arrive at the terminal region $\mathrm{G}_{\mathrm{terminal}}$ within T_g, choose one of the charging stations, including $\mathrm{G}_{\mathrm{charge1}}$ and $\mathrm{G}_{\mathrm{charge2}}$, to charge the battery before $T_g/2$ and stay there for at least $T_g/10$. The STL task of the agent can be formulated as

$$\begin{aligned}\phi = {} & G_{[0,T_g]}(p \notin \mathrm{Obs}) \wedge F_{[0,T_g]}(p \in \mathrm{G}_{\mathrm{terminal}}) \\ & \wedge F_{[0,T_g/2]} G_{[0,T_g/10]}(p \in \mathrm{G}_{\mathrm{charge1}} \vee p \in \mathrm{G}_{\mathrm{charge2}}),\end{aligned} \tag{21}$$

where the goal regions $\mathrm{G}_{\mathrm{terminal}}$, $\mathrm{G}_{\mathrm{charge1}}$, $\mathrm{G}_{\mathrm{charge2}}$ and obstacle region Obs are defined via a set of conjunctions over linear predicates. The agent is expected to satisfy the specified STL task with minimum control effort $J(x(0), \mathbf{u}_N(0)) = \sum_{i=0}^{N-1} u^{\mathrm{T}}(i)u(i)$. Consider the following four MPC schemes:

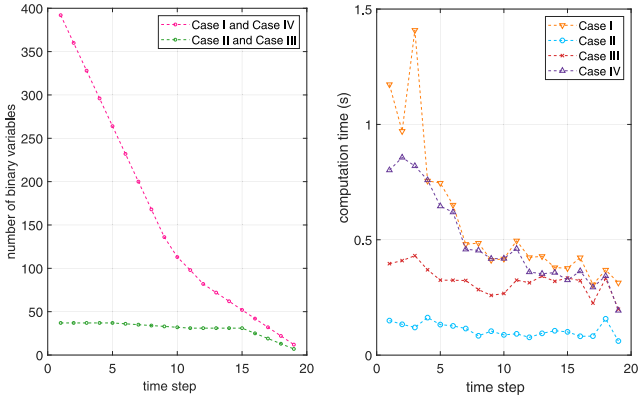
Case I: Control with the classical STL-MPC algorithm in Raman et al. (2014);

Case II: Control with the proposed low complexity STL-MPC algorithm;

Case III: Control with the proposed STL-MPC algorithm, with the blocking matrices set as identity matrices (no input blockage);

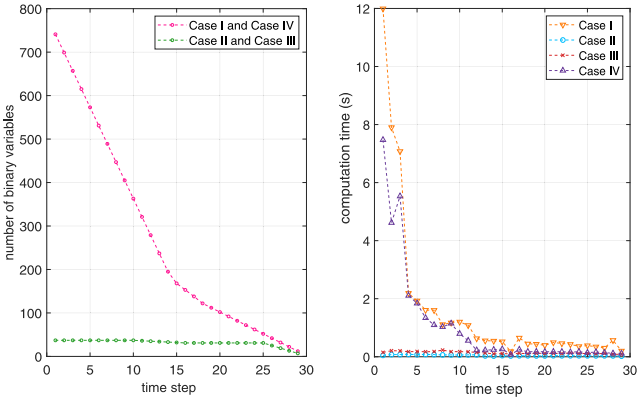
Case IV: Control with the proposed STL-MPC algorithm, with the time set $\mathcal{T}_k$ chosen as the overall task horizon, i.e., $\mathcal{T}_k = [0, T_g]$ (no formula simplification).

The partitioned horizon-based method introduced in Section 6.1 is employed to determine the time set $\mathcal{T}_k$ and block matrix S_k at each time step. Set $H_{is} = 5$ and $\lambda = 0.1$. Under the four MPC schemes,



(a) The number of required binary variables over time. (b) Computation time of online optimizations.

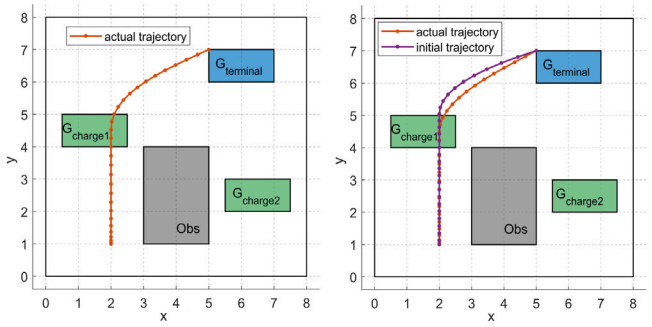
Fig. 4. The number of binary variables and the computation time under four cases when $T_g = 20$.



(a) The number of required binary variables over time. (b) Computation time of online optimizations.

Fig. 5. The number of binary variables and the computation time under four cases when $T_g = 30$.

the involved number of binary variables in the optimization problems and the corresponding online computation time are shown in Fig. 4 and Fig. 5, considering different task length $T_g = 20$ and $T_g = 30$. As observed in Fig. 4(a) and Fig. 5(a), when there is no formula simplification (Case I and Case IV), the number of binary variables decreases over time with shrinking horizon MPC, and accordingly, the computation time shows an exponential decline over time. In contrast, the formula-simplified optimization problems maintain a consistent number of around 35 binary variables (Case II and Case III), resulting in consistently low computation times during MPC implementation. Comparing the simulation results across the four cases, we can conclude



(a) Agent trajectory under Case I. (b) Agent trajectories under Case II.

Fig. 6. Workspace under STL task (21). Agent trajectories are obtained under the method shown in Case I and Case II with $T_g = 30$.

Table 3
The values of cost function under four cases.

	Case I	Case II	Case III	Case IV
$T_g = 20$	0.1566	0.1587	0.1567	0.1586
$T_g = 30$	0.1432	0.1494	0.1467	0.1487

that using input blocking alone may not significantly reduce the computational workload, as evidenced by the computation time comparison between Case I and Case IV. However, by reducing the required number of binary variables through formula simplification, input blocking can further enhance computation speed, as seen in the computation time comparison between Case II and Case III.

The values of the cost function $J(x(0), \mathbf{u}_N^*(0))$ are similar under the four MPC schemes, as shown in Table 3. The moving trajectories of the agent under Case I and Case II with $T_g = 30$ are shown in Fig. 6. Specifically, the initial trajectory in Fig. 6(b) is a predicted trajectory based on the optimization results of the initial problem P_0; and the actual trajectories in Fig. 6(a) and Fig. 6(b) are obtained by executing MPC in Case I and Case II, respectively. Based on the above discussion, we can conclude that the proposed STL-MPC algorithm enhances the efficiency of online optimization with reasonable control performance.

Scenario 2: To further assess the efficacy of the proposed algorithm, we conduct simulations using a more intricate 3D model involving two unmanned aerial vehicles (UAVs). Particularly, consider the state of the jth UAV as $x_j = \left[p_j^{\mathrm{T}}, o_j^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^6$, where p_j and o_j are 3D position and velocity. The inputs $u_j \in \mathbb{R}^3$ are the thrust, roll, and pitch of the UAV. The initial states are specified as $x_1(0) = [0, 10, 4, 0, 0, 0]^{\mathrm{T}}$, $x_2(0) = [9, 1, 0, 0, 0, 0]^{\mathrm{T}}$. The dynamics of UAVs are obtained via linearization around hover and discretization, see Luukkonen (2011) for more details. Similar models have been used for control of real quad-rotors with success (Pant et al., 2015).

Two UAVs are tasked with a surveillance mission of three regions, see Fig. 7. The overall specification is of the form $\varphi = \bigwedge_{i=1}^{3} \varphi_i$ where:

(i) UAV 1 should reach and stay in zone $\mathrm{G_A}$ all the time from 10 to 30 time units while always avoid zone $\mathrm{G_B}$, $\varphi_1 = G_{[10,30]}(p_1 \in \mathrm{G_A}) \wedge G_{[0,60]}(p_1 \notin \mathrm{G_B})$;

(ii) UAV 2 should eventually reach zone $\mathrm{G_B}$ any time between 2 and 60 time units while always avoid zone $\mathrm{G_A}$, $\varphi_2 = F_{[2,60]}(p_2 \in \mathrm{G_B}) \wedge G_{[0,60]}(p_2 \notin \mathrm{G_A})$;

(iii) Zone $\mathrm{G_C}$ should be surveilled, i.e., either one or both UAVs must be within $\mathrm{G_C}$ all the time from 35 to 60 time units, $\varphi_3 = F_{[35,60]}(p_1 \in \mathrm{G_C} \wedge p_2 \in \mathrm{G_C})$.

We implement the proposed algorithm for this STL satisfaction synthesis problem and conduct a performance comparison with the methods presented in Raman et al. (2014) and Kurtz and Lin (2022). The simulations are all executed within the framework of shrinking horizon MPC. Set $N = 60$, $J(x(k), \mathbf{u}_N(k)) = \sum_{i=k}^{N-1} u_1^{\mathrm{T}}(i)u_1(i) + u_2^{\mathrm{T}}(i)u_2(i)$, $H_{is} = 15$, and $\lambda = 0.1$. We observe that the method introduced in Raman et al. (2014) fails to find a solution for this problem, running continuously for over 10,000 s without termination. It thus has been excluded from further comparisons. Both the method in Kurtz and Lin (2022) and our approach successfully achieve task satisfaction with similar control efforts, resulting in cost values of 9.21 and 9.17, respectively. The trajectories of UAVs under the proposed algorithm is shown in Fig. 7 from different angles. We observe that the method from Kurtz and Lin (2022) requires a computation time of approximately 200 s during the initial optimization stage, while our proposed algorithm consistently demonstrates significantly reduced computation times of around 0.7 s at each time step during MPC implementation. These results underscore the efficiency of our algorithm.

6.3. Experiment: Uncertain case

We extend our investigation to evaluate the effectiveness of the proposed STL-MPC algorithm under uncertain conditions and conduct experiments using a TurtleBot3 mobile robot. We use VICON positioning system to locate the robot and the regions in the environment. A workstation (Intel Core i5-10400 2.90 GHz CPU and 8 GB of RAM) is utilized to process the data from VICON and calculate the control inputs for the robot. The information transactions between the VICON, the workstation, and the robots are established through a Robot Operating System. Robot model

$$x(k+1) = \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} u(k) + w(k) \tag{22}$$

is employed to calculate the discrete optimal location points $x = [p_x, p_y]^{\mathrm{T}} \in \mathbb{R}^2$, and the transition between two location points is achieved by the activation of low-level continuous controllers of the robot. The stochastic disturbance w models measurement and tracking error, which is truncated from a Gaussian distribution with mean zero, standard deviation 0.8 and satisfies $\|w\|_\infty \leq 0.2$. The robot's workspace in simulations and experiments is illustrated in Fig. 8 and Fig. 10, respectively, where the gray areas represent obstacles that the robot cannot enter and the green areas denote the target regions. The motion task of the robot is to visit three goal regions $\mathrm{G_1}$, $\mathrm{G_2}$, and $\mathrm{G_3}$ within specified time, while avoiding obstacles. Especially, keys in $\mathrm{G_1}$ should be collected before entering $\mathrm{G_2}$. This task can be formulated into STL formula as follows:

$$\begin{aligned} \varphi = &(x \notin \mathrm{G_2}) U_{[0,27]}[G_{[0,3]}(x \in \mathrm{G_1})] \wedge F_{[0,27]}[G_{[0,3]}(x \in \mathrm{G_2})] \wedge \\ &F_{[0,27]}[G_{[0,3]}(x \in \mathrm{G_3})] \wedge G_{[0,30]}(x \notin \mathrm{Obs}). \end{aligned} \tag{23}$$

We implement the proposed MPC algorithm for this STL satisfaction synthesis problem, and compare the performance with the methods proposed in Raman et al. (2014) and Kurtz and Lin (2022). Note that STL constraint tightening is applied in all cases to ensure solvable deterministic optimization problems. Set initial state $x(0) = [5.5, 1.5]^{\mathrm{T}}$, $H_{is} = 6$, and $\lambda = 0.1$. We observe that the method proposed in Raman et al. (2014) fails to find a solution for this problem and, therefore, is excluded from further comparisons. We conduct 100 simulation instances, and the average computation time at each optimization are depicted in Fig. 9. Notably, the computation time decreases with time under the method in Kurtz and Lin (2022), while our proposed algorithm consistently maintains a lower computation time throughout the process. Specifically, under the method in Kurtz and Lin (2022), the maximum computation time occurs at $k = 3$, which is 87.33 s with a variance of 7.76 (the average computation time at $k = 3$ is 84.29 s). However, with our proposed algorithm, the maximum computation time is observed at $k = 1$, and it is only 0.47 s with a variance of 0.014 (the average computation time at $k = 1$ is 0.32 s). In addition, all generated trajectories under two approaches satisfy the specified STL task against uncertainties, and two representative robot simulation trajectories are shown in Fig. 8. The proposed algorithm is then implemented by TurtleBot3 mobile robot, with the safe distance from obstacles adjusted based on the robot's size. Snapshots of some

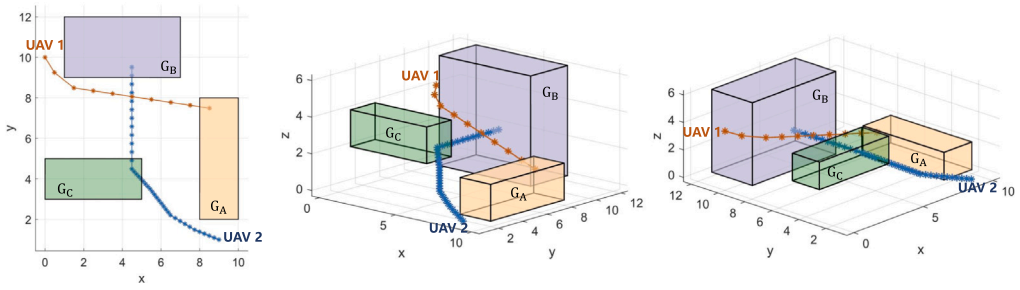


Fig. 7. 3D representation of the map for multi-UAV surveillance task from different angles. The trajectories of UAV 1 (orange lines) and UAV 2 (blue lines) are obtained using the proposed STL-MPC algorithm, which satisfy the specified surveillance mission.

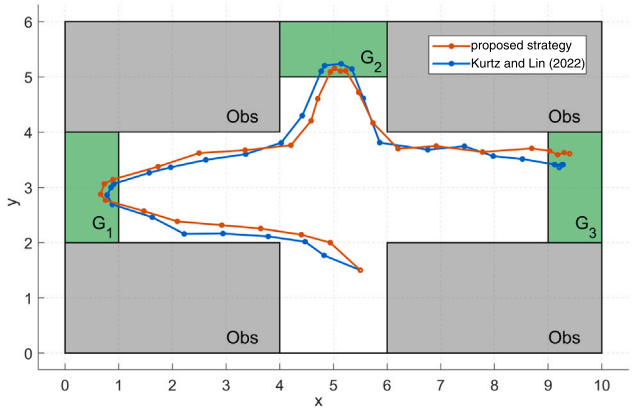


Fig. 8. Workspace under STL task (23). Simulation trajectories of the robot are obtained with the proposed STL-MPC algorithm and the encoding method in Kurtz and Lin (2022). Both trajectories satisfy the task, but with different computational costs.

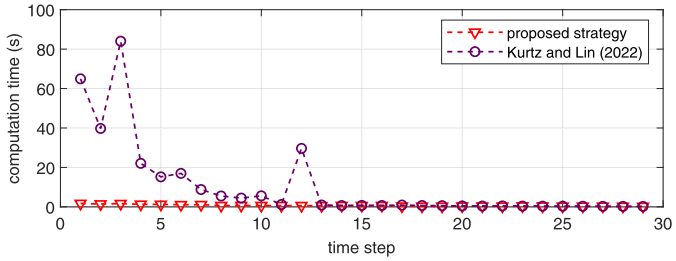


Fig. 9. Computation time of online optimizations under STL task (23) with the proposed strategy and the encoding method in Kurtz and Lin (2022).

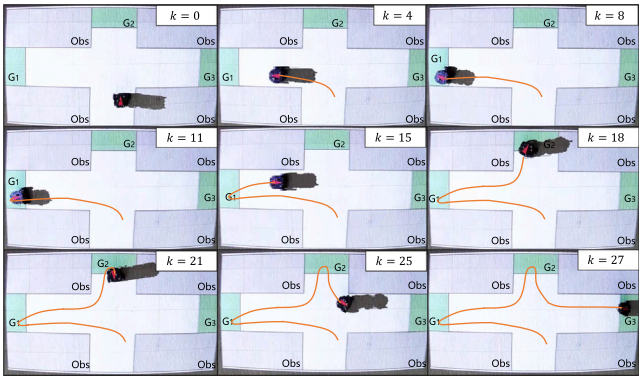


Fig. 10. Snapshots during the experiment. The robot is guided by the proposed STL-MPC algorithm, with the orange lines indicating its historical trajectories before the current sampling instant.

important time instants during the experiment are given in Fig. 10. We can observe that the robot successfully arrives at G_1, G_2, and G_3 in sequence while avoiding obstacles, and the computational speed also meets the sampling requirement. In conclusion, the proposed STL-MPC algorithm realizes efficient online problem-solving with performance guarantees.

7. Conclusion

This paper achieved efficient MILP problem-solving for STL in the framework of shrinking horizon MPC by formula simplification and input blocking. The soundness and feasibility of the proposed algorithm were formally analyzed, and the extension to uncertain systems was achieved by STL constraint tightening. Simulations and experiments verified the effectiveness of the STL-MPC algorithm. We are exploring several extensions and variations in ongoing and future work, including distributed control scheme and more complex tasks. Moreover, the selection of time set $\mathcal{T}_k$ has significant effects on the optimality of STL synthesis, which is currently chosen heuristically in this work. Theoretical support for time set design would also be an important subject of future research.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This paper is supported by the National Natural Science Foundation of China (NSFC) (62173224) and the founding from Beijing Advanced Innovation Center for Intelligent Robots and Systems, China.

References

Bai, T., Li, S., & Zou, Y. (2021). Distributed MPC for reconfigurable architecture systems via alternating direction method of multipliers. *IEEE/CAA Journal of Automatica Sinica*, *8*(7), 1336–1344.

Belta, C., & Sadraddini, S. (2019). Formal methods for control synthesis: An optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, *2*, 115–140.

Buyukkocak, A. T., Aksaray, D., & Yazıcıoğlu, Y. (2021). Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates. *IEEE Robotics and Automation Letters*, *6*(2), 1375–1382.

Farahani, S. S., Majumdar, R., Prabhu, V. S., & Soudjani, S. (2018). Shrinking horizon model predictive control with signal temporal logic constraints under stochastic disturbances. *IEEE Transactions on Automatic Control*, *64*(8), 3324–3331.

Garcia, C. E., Prett, D. M., & Morari, M. (1989). Model predictive control: Theory and practice–a survey. *Automatica*, *25*(3), 335–348.

Haghighi, I., Mehdipour, N., Bartocci, E., & Belta, C. (2019). Control from signal temporal logic specifications with smooth cumulative quantitative semantics. In *2019 IEEE conference on decision and control* (pp. 4361–4366).

Huang, M., Zheng, Y., & Li, S. (2022). Distributed economic model predictive control for an industrial fluid catalytic cracking unit ensuring safe operation. *Control Engineering Practice*, *126*, Article 105263.

Knuth, D. E. (1964). Backus normal form vs. backus naur form. *Communications of the ACM*, *7*(12), 735–736.

Kouvaritakis, B., Cannon, M., Raković, S. V., & Cheng, Q. (2010). Explicit use of probabilistic distributions in linear predictive control. *Automatica*, *46*(10), 1719–1724.

Kurtz, V., & Lin, H. (2022). Mixed-integer programming for signal temporal logic with fewer binary variables. *IEEE Control Systems Letters*, *6*, 2635–2640.

Lindemann, L., & Dimarogonas, D. V. (2018). Control barrier functions for signal temporal logic tasks. *IEEE Control Systems Letters*, *3*(1), 96–101.

Lindemann, L., & Dimarogonas, D. V. (2019). Robust control for signal temporal logic specifications using discrete average space robustness. *Automatica*, *101*, 377–387.

Liu, S., Trivedi, A., Yin, X., & Zamani, M. (2022). Secure-by-construction synthesis of cyber-physical systems. *Annual Reviews in Control*, *53*, 30–50.

Liu, Z., Wu, B., Dai, J., & Lin, H. (2020). Distributed communication-aware motion planning for networked mobile robots under formal specifications. *IEEE Transactions on Control of Network Systems*, *7*(4), 1801–1811.

Luukkonen, T. (2011). Modelling and control of quadcopter. *Independent Research Project in Applied Mathematics, Espoo*, *22*(22).

Maler, O., & Nickovic, D. (2004). Monitoring temporal properties of continuous signals. In *Formal techniques, modelling and analysis of timed and fault-tolerant systems* (pp. 152–166).

Pant, Y. V., Abbas, H., Mohta, K., Nghiem, T. X., Devietti, J., & Mangharam, R. (2015). Co-design of anytime computation and robust control. In *2015 IEEE real-time systems symposium* (pp. 43–52). IEEE.

Patil, S. V., Hashimoto, K., & Kishida, M. (2022). Traffic flow control at signalized intersections using signal spatio-temporal logic. In *2022 IEEE 61st conference on decision and control* (pp. 1051–1058). IEEE.

Puranic, A. G., Deshmukh, J. V., & Nikolaidis, S. (2021). Learning from demonstrations using signal temporal logic in stochastic and continuous domains. *IEEE Robotics and Automation Letters*, *6*(4), 6250–6257.

Raman, V., Donzé, A., Maasoumy, M., Murray, R. M., Sangiovanni-Vincentelli, A., & Seshia, S. A. (2014). Model predictive control with signal temporal logic specifications. In *2014 IEEE conference on decision and control* (pp. 81–87).

Rodionova, A., Lindemann, L., Morari, M., & Pappas, G. J. (2022). Combined left and right temporal robustness for control under stl specifications. *IEEE Control Systems Letters*, *7*, 619–624.

Sahin, Y. E., Nilsson, P., & Ozay, N. (2020). Multirobot coordination with counting temporal logics. *IEEE Transactions on Robotics*, *36*(4), 1189–1206.

Shekhar, R. C., & Manzie, C. (2015). Optimal move blocking strategies for model predictive control. *Automatica*, *61*, 27–34.

Son, S. H., Oh, T. H., Kim, J. W., & Lee, J. M. (2020). Move blocked model predictive control with improved optimality using semi-explicit approach for applying time-varying blocking structure. *Journal of Process Control*, *92*, 50–61.

Sun, D., Chen, J., Mitra, S., & Fan, C. (2022). Multi-agent motion planning from signal temporal logic specifications. *IEEE Robotics and Automation Letters*, *7*(2), 3451–3458.

Tian, D., Fang, H., Yang, Q., Guo, Z., Cui, J., Liang, W., et al. (2023). Two-phase motion planning under signal temporal logic specifications in partially unknown environments. *IEEE Transactions on Industrial Electronics*, *70*(7), 7113–7121.

Vielma, J. P., & Nemhauser, G. L. (2011). Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Mathematical Programming*, *128*(1), 49–72.

Xu, B., Suleman, A., & Shi, Y. (2023). A multi-rate hierarchical fault-tolerant adaptive model predictive control framework: Theory and design for quadrotors. *Automatica*, *153*, Article 111015.

Yu, P., & Dimarogonas, D. V. (2021). Hierarchical control for uncertain discrete-time nonlinear systems under signal temporal logic specifications. In *2021 60th IEEE conference on decision and control* (pp. 1450–1455). IEEE.

Yu, X., Yin, X., Li, S., & Li, Z. (2022). Security-preserving multi-agent coordination for complex temporal logic tasks. *Control Engineering Practice*, *123*, Article 105130.

Zhou, X., Yang, T., Zou, Y., Li, S., & Fang, H. (2022). Multiple sub-formulae cooperative control for multi-agent systems under conflicting signal temporal logic tasks. *IEEE Transactions on Industrial Electronics*.

Zhou, X., Zou, Y., Li, S., Li, X., & Fang, H. (2022). Distributed model predictive control for multi-robot systems with conflicting signal temporal logic tasks. *IET Control Theory & Applications*, *16*(5), 554–572.