# Multi-agent Motion Planning from Signal Temporal Logic Specifications

Dawei Sun[1], Jingkai Chen[2], Sayan Mitra[1], and Chuchu Fan[2]

***Abstract*—We tackle the challenging problem of multi-agent cooperative motion planning for complex tasks described using signal temporal logic (STL), where robots can have nonlinear and nonholonomic dynamics. Existing methods in multi-agent motion planning, especially those based on discrete abstractions and model predictive control (MPC), suffer from limited scalability with respect to the complexity of the task, the size of the workspace, and the planning horizon. We present a method based on *timed waypoints* to address this issue. We show that timed waypoints can help abstract nonlinear behaviors of the system as safety envelopes around the reference path defined by those waypoints. Then the search for waypoints satisfying the STL specifications can be inductively encoded as a mixed-integer linear program. The agents following the synthesized timed waypoints have their tasks automatically allocated, and are guaranteed to satisfy the STL specifications while avoiding collisions. We evaluate the algorithm on a wide variety of benchmarks. Results show that it supports multi-agent planning from complex specification over long planning horizons, and significantly outperforms state-of-the-art abstraction-based and MPC-based motion planning methods. The implementation is available at https://github.com/sundw2014/STLPlanning.**

***Index Terms*—Task and Motion Planning; Path Planning for Multiple Mobile Robots or Agents.**

## I. Introduction

THE capability of performing automatic task and motion planning according to high-level specifications is what people usually expect from an intelligent and autonomous robotic system. These high-level task specifications usually consist of temporal and logical rules and need cooperative solutions of multiple agents. It is not straightforward to directly derive a specific sequence of locations to visit for each agent from these high-level specifications. Therefore, synthesizing correct-by-construction plans and control strategies from these complicated specifications has been an open problem, especially when the planning horizon is long and the robotic systems have complex dynamics [1], [2]. Fortunately, Temporal Logic (TL), especially Signal Temporal Logic (STL) provides a mathematically precise language for specifying tasks and rules over continuous signals with explicit time semantics [3]. Such a formal description of the task enables automatic control action synthesis.

Methods based on discrete abstractions and model predictive control (MPC) are two representative approaches for motion planning from TL specifications. Abstraction-based methods discretize the state space and create an abstract bisimilar graph or automaton, on which the actual planning is performed. MPC-based methods discretize the trajectory with a fixed time step, and the states at each time step are viewed as the decision variables of an optimization problem. Although these methods have achieved success in a wide range of applications, some obvious disadvantages prevents them from being widely adopted for solving realistic robotic planning problems: To use abstraction-based methods, one needs to construct the bisimilar graph, which heavily relies on domain expertise. Moreover, the number of abstracted states would potentially grow exponentially fast as the dimensionality of the state space increases and cause significant scalability issues. As for MPC-based methods, the number of time steps needed might be too large for long-horizon planning. In Sec. IV, we compare the proposed method with those two methods.

In this paper, we propose a novel synthesis method which tackles the aforementioned challenges. Inspired by the method in [4] where the authors use piece-wise linear (PWL) reference paths and tracking controllers to solve simple reach-avoid synthesis problems, we show that using PWL reference paths one can also handle more expressive STL specifications. A PWL path is defined by a sequence of time-stamped waypoints. Our method can automatically reason over the STL formula by recursively encoding constraints over the time-stamped waypoints according to the syntax of the STL formula. Moreover, we define the multi-agent STL, which can be used to specify tasks that need be completed cooperatively by a group of agents. Given such a multi-agent STL formula, our proposed method can automatically assign sub-tasks to each agent such that they cooperate efficiently without collision. Because the tracking error of the tracking controller is taken into account when encoding the constraints, it can be shown that any solution that satisfy our encoded constraints can give the desired PWL paths: Agents following the PWL reference paths are guaranteed to satisfy the given multi-agent STL specification and are collision-free. More importantly, our constraints are all linear because of the PWL structure. Therefore, we can find optimal solutions by solving mixed-integer linear programming (MILP) problems, which can be

Manuscript received: September 9, 2021; Revised December 12, 2021; Accepted January 6, 2022.

This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments. Sun and Mitra were supported by research grants from the National Security Agency's Science of Security (SoS) program and National Science Foundation's Formal Methods in the Field (FMITF) program. Fan was supported by MIT-IBM Watson AI Lab.

[1]Dawei Sun and Sayan Mitra are with University of Illinois at Urbana-Champaign, Champaign, IL, 61820 USA. {daweis2, mitras}@illinois.edu

[2]Jingkai Chen and Chuchu Fan are with Massachusetts Institute of Technology, Cambridge, MA, 02139 USA. {jk_chen, chuchu}@mit.edu

Digital Object Identifier (DOI): see top of this page.

effectively handled by off-the-shelf solvers such as Gurobi[1].

We evaluate the proposed method on 8 benchmark synthesis problems with a variety of different scenarios. We compare with both abstraction-based and MPC-based methods [5], [6]. Empirical results show that our method outperforms other state-of-the-art methods in terms of running time and quality of the planned paths, not to say that our method can handle much more general STL formulas. For example, our method is order of magnitude faster than `stlcg` [5], which is MPC-based and solves the optimization problem with gradient decent. Also, we implement and compare with another MPC-based algorithm proposed in [6]. The MPC-based method failed in some cases due to the large number of decision variables, while our method successfully found a solution.

### A. Related work

Robot motion planning is a large and active research area [7], [8], [9], [10], and planning from TL specifications has received significant attention [1], [11]. Abstraction-based approaches have stood out as a systematic framework of finding control policies [1], [12]. However, the abstraction step of these approaches heavily relies on domain expertise and is hard to be automated. Among all the planning methods that can handle TL specifications, the closest to ours is the one proposed in [13], which discretizes the workspace into regions, constructs a graph with regions as nodes, and finally searches for a valid path on the graph.

Another class of synthesis approaches for STL is based on model predicative control (MPC), for example, [6], [14], [15], [16], [17]. In these approaches, a time-step is fixed and the decision variables of the optimization are just the state at each step. Both the dynamics and the STL specifications are encoded as constraints of the optimization problem. Thus, it is challenging to handle real-world robots with complicated dynamics. Another drawback of these approaches is that the number of time steps needed might be too large for long-horizon planning. The proposed approach tackles this problem by using time-stamped waypoints instead of a fixed time step, which is also similar to event-based control (e.g., [18]) in the sense that each waypoint can be viewed as an event and between two consecutive events the control input does not change. Similar idea has been studied in [19], where the users use zeroth-order hold control, i.e., the control signal is held at a time instant (waypoint) for a variable interval. Different from the proposed approach, it uses control barrier functions to ensure satisfaction between timed waypoints.

Sampling-based methods have also been used to solve planning problems for multi-agent systems and/or STL specifications. STyLuS$^*$ [20] is a scalable algorithm for multi-agent optimal control with temporal logics. [21] utilizes RRT to plan paths for long-term LTL goals with short-term reactive specifications. In [22], the authors proposes the spatio-temporal RRT* algorithm which can handle STL specifications containing only "always" operators. In [23], the authors extend the RRT$^*$ algorithm with biased space-time sampling and guided steering, and the algorithm is able to efficiently grow the RRT tree along the direction of increasing STL satisfaction.

[1]https://www.gurobi.com/

## II. Preliminaries and Problem Statement

Let $\mathbb{R}$ and $\mathbb{Z}^+$ be the real numbers and positive integers respectively. For a vector $x \in \mathbb{R}^n$, $x^{(i)}$ is its $i^{\text{th}}$ entry, $\|x\|$ is its Euclidean norm, $\|x\|_1$ is its one-norm, and $B_\epsilon(x) := \{y \in \mathbb{R}^n \mid \|y - x\| \leq \epsilon\}$ is the $\epsilon$-ball centered at $x$. Given a matrix $H \in \mathbb{R}^{n\times m}$ and a vector $b \in \mathbb{R}^n$, $\texttt{Poly}(H, b)$ denotes the convex polytope $\{x \in \mathbb{R}^m \mid H \cdot x \leq b\}$. $H^{(i)}$ is the $i^{\text{th}}$ row of $H$, and $\texttt{Row}(H)$ denotes the number of rows in $H$, which is also the number of faces of the polytope. For $N \in \mathbb{Z}^+$, denote $\{1, \cdots, N\}$ by $[N]$.

### A. STL for multi-agent specifications

Let $\mathcal{W} := \mathbb{R}^d$ be the workspace. Given a vector-valued function $\mu$ defined on $\mathcal{W}$, an *atomic predicate* can be defined based on $\mu$ and is denoted by $\pi^\mu$. For a point $x \in \mathcal{W}$, we say that $x$ satisfies $\pi^\mu$ (written as $x \vDash \pi^\mu$) iff. $\mu(x) \geq 0$. In this paper, we are only interested in atomic predicates that indicate whether or not a point is in a polytope. That is, $\mu$ is always of the form $\mu(x) = b - H \cdot x$. Then, $x \vDash \pi^\mu$ iff. $x \in \texttt{Poly}(H, b)$. Also, $x \nvDash \pi^\mu$ iff. $x \notin \texttt{Poly}(H, b)$. Atomic predicates only characterize standalone points in the workspace. However, we are more interested in predicates that can characterize trajectories. Let $p : \mathbb{R}_{\geq 0} \mapsto \mathcal{W}$ be the position trajectory of a robot, which is a function of time. Let $(p, t)$ be the suffix of $p$ at $t$, i.e., $(p, t)(s) = p(s + t)$. Next, STL is defined based on the atomic predicates.

**Definition 1** (Signal Temporal Logic (STL))**.** *An STL formula is defined with the following syntax:*

$$\begin{aligned}\varphi ::= \pi^\mu | \neg\pi^\mu | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 \\ | \Diamond_{[a,b]}\varphi | \Box_{[a,b]}\varphi | \varphi_1 \mathcal{U}_{[a,b]}\varphi_2 | \varphi_1 \mathcal{R}_{[a,b]}\varphi_2\end{aligned} \tag{1}$$

*where $\varphi, \varphi_1, \varphi_2$ are STL formulas, and $0 \leq a \leq b < \infty$ denote time intervals. Here, the temporal operators $\Diamond, \Box, \mathcal{U}, \mathcal{R}$ are called "eventually", "always", "until", and "release" respectively. Formally, the validity of an STL formula with respect to a trajectory $p : \mathbb{R}_{\geq 0} \mapsto \mathcal{W}$ is defined as follows.*

$$\begin{aligned}
p \vDash \varphi &\Leftrightarrow (p, 0) \vDash \varphi \\
(p, t) \vDash \pi^\mu &\Leftrightarrow \mu(p(t)) \geq 0 \\
(p, t) \vDash \neg\pi^\mu &\Leftrightarrow (p, t) \nvDash \pi^\mu \\
(p, t) \vDash \varphi_1 \wedge \varphi_2 &\Leftrightarrow (p, t) \vDash \varphi_1 \wedge (p, t) \vDash \varphi_2 \\
(p, t) \vDash \varphi_1 \vee \varphi_2 &\Leftrightarrow (p, t) \vDash \varphi_1 \vee (p, t) \vDash \varphi_2 \\
(p, t) \vDash \Diamond_{[a,b]}\varphi &\Leftrightarrow \exists t' \in [t + a, t + b], (p, t') \vDash \varphi \\
(p, t) \vDash \Box_{[a,b]}\varphi &\Leftrightarrow \forall t' \in [t + a, t + b], (p, t') \vDash \varphi \\
(p, t) \vDash \varphi_1 \mathcal{U}_{[a,b]}\varphi_2 &\Leftrightarrow \exists t' \in [t + a, t + b], (p, t') \vDash \varphi_2 \\
&\quad \wedge \forall t'' \in [t, t'], (p, t'') \vDash \varphi_1 \\
(p, t) \vDash \varphi_1 \mathcal{R}_{[a,b]}\varphi_2 &\Leftrightarrow \forall t' \in [t + a, t + b], (p, t') \vDash \varphi_2 \\
&\quad \vee \exists t'' \in [t, t'], (p, t'') \vDash \varphi_1
\end{aligned}$$

Please note that in the above syntax, negation can only be applied to atomic predicates. This is known as the *Negation Normal Form* and is not restrictive because any STL formula

can be put in this form [7]. The above definition of STL only characterizes a single trajectory $p$. Next, we define the multi-agent STL (MA-STL), which extends the notion of STL to cases where multiple trajectories are considered.

**Definition 2** (Multi-agent STL (MA-STL))**.** *An $N$-agent STL formula is defined recursively with the following syntax:*

$$\Psi ::= \pi_i^{\varphi} | \Psi_1 \wedge \Psi_2 | \Psi_1 \vee \Psi_2,$$

*where $\Psi_1, \Psi_2$ are $N$-agent STL formulas, and $\pi_i^{\varphi}$ assigns a single-agent STL $\varphi$ to agent $i$. Formally, the validity an MA-STL w.r.t. a group of trajectories $(p_1, \cdots, p_N)$ is as follows.*

$$\begin{aligned}
(p_1, \cdots, p_N) \vDash \pi_i^{\varphi} \quad &\Leftrightarrow \quad p_i \vDash \varphi, \\
(p_1, \cdots, p_N) \vDash \Psi_1 \wedge \Psi_2 \quad &\Leftrightarrow \quad (p_1, \cdots, p_N) \vDash \Psi_1 \\
&\qquad \text{and } (p_1, \cdots, p_N) \vDash \Psi_2, \\
(p_1, \cdots, p_N) \vDash \Psi_1 \vee \Psi_2 \quad &\Leftrightarrow \quad (p_1, \cdots, p_N) \vDash \Psi_1 \\
&\qquad \text{or} \quad (p_1, \cdots, p_N) \vDash \Psi_2.
\end{aligned}$$

**Remark.** *Firstly, MA-STL enables implicit task assignment. For example, let $\{\mathcal{G}_i\}_{i=1}^M$ be $M$ goals, which are atomic predicates defining some polytopes in the workspace. The MA-STL formula $\Psi = \bigwedge_{j=1}^{M} \bigvee_{i=1}^{N} \pi_i^{\Diamond_{[0,T]} \mathcal{G}_j}$ assigns tasks to the agents implicitly. That is, it does not assign specific tasks to each agent, but requires each goal to be visited by at least one agent. As will be shown in Sec. IV, with the proposed planning algorithm, agents can figure out the optimal assignment automatically. Secondly, please also note that MA-STL is only a syntactic sugar in the sense that it is a subset of the STL formulas defined over the joint state space of the multi-agent system. In this subset, temporal operations can only be applied to a single agent at a time.*

### *B. Tracking controllers for the agents*

In practice, the robots used to complete a task usually have complicated and nonlinear dynamics, which makes it difficult to directly synthesize the correct control input for them. As a famous aphorism goes: "all problems in computer science can be solved by another level of indirection", we exploit a separation of concerns that exists in the robot control synthesis problem so that complexity of specifications (tasks) and that of the dynamics can be dealt with separately. Specifically, we assume that a tracking controller is given for each agent such that it can track *any* reference path under *any* bounded disturbances with an known tracking error $\epsilon > 0$.[2] That is, the distance between the actual position of the robot and the desired position on the reference path is always upper bounded by $\epsilon$. Many techniques can be used for obtaining such a tracking controller and the corresponding tracking error, for example, control Lyapunov functions [24] or control contraction metrics [25]. Here, the tracking controller is an abstraction (or the so-called "indirection") layer that wraps the underlying dynamics such that the closed-loop system has a uniform behavior (characterized by the tracking error bound), and thus makes the design of motion planners easier.

[2]This is true when the reference paths satisfy the requirements of the controller, for example, the velocity is bounded.

Obviously, controlled by the tracking controller, the actual trajectory of the agent will be in a tube centered at the reference path. If one can show that every trajectory in this tube satisfy the specification, then it can be guaranteed that the actual trajectory of the agent will satisfy the specification in the presence of any bounded disturbances. To this end, we define the robustness of trajectories.

**Definition 3** ($\epsilon$-robust)**.** *A group of trajectories $(p_1, \cdots, p_N)$ is said to be $\epsilon$-robust with respect to a property for some $\epsilon > 0$, if the property holds for all $(\hat{p}_1, \cdots, \hat{p}_N)$ satisfying $\sup_t \|\hat{p}_i(t) - p_i(t)\|_2 \leq \epsilon$, $\forall i \in [N]$.*

### *C. The MA-STL motion planning problem*

Next, we define the multi-agent motion planning problem. Intuitively, the goal is to find a group of reference paths that are $\epsilon$-robust to a given MA-STL specification and free of inter-agent collisions. Assume that $N$ agents are involved and $T$ is the time bound. Denote the size of the $i$-th agent by $s_i > 0$. That is, at position $p \in \mathcal{W}$, the agent is completely contained in a ball around it of radius $s_i$, i.e., $B_{s_i}(p)$. Then, the planning problem is defined as follows.

**Definition 4** (MA-STL Motion Planning)**.** *A MA-STL motion planning problem is defined by a tuple*

$$\langle p_1^{init}, \cdots, p_N^{init}, \Psi \rangle,$$

*where $p_i^{init} \in \mathcal{W}$ is the initial position of agent $i$ and $\Psi$ is an MA-STL formula. The problem is to find a group of reference paths $(p_1, \cdots, p_N)$ satisfying the following conditions:*

1) *(Initial conditions) $p_i(0) = p_i^{init}$, $\forall i \in [N]$.*
2) *(No inter-agent collisions) $\forall t \in [0, T]$, $\forall i, j \in [N]$ and $i \neq j$, $B_{s_i+\epsilon}(p_i(t)) \cap B_{s_j+\epsilon}(p_j(t)) = \varnothing$.*
3) *(STL Satisfaction) $(p_1, \cdots, p_N)$ are $\epsilon$-robust w.r.t. $\Psi$.*

Instead of searching for the reference paths among all possible functions of time, the proposed approach restricts its search space to piece-wise linear (PWL) paths. In the rest of the paper, we refer to the reference PWL path for agent $i$ as $S_i$. Formally, PWL paths are defined as follows.

**Definition 5** (Piece-wise Linear Path)**.** *A piece-wise linear path $S_i$ in the workspace $\mathcal{W}$ is a function $S_i : \mathbb{R}_{\geq 0} \to \mathcal{W}$ that maps a time instant $t$ to a position $S_i(t) \in \mathcal{W}$. It is constructed from a sequence of time-stamped waypoints $\{(t_{i,k}, p_{i,k})\}_{k=0}^{K_i}$ such that $S_i(t) = p_{i,k-1} + \frac{p_{i,k} - p_{i,k-1}}{t_{i,k} - t_{i,k-1}}(t - t_{i,k-1})$ for $t \in [t_{i,k-1}, t_{i,k}]$. Here, $0 = t_{i,0} \leq t_{i,1} \leq \cdots \leq t_{i,K}$ are the time stamps, and $(t_{i,k}, p_{i,k}) \in \mathbb{R}_{\geq 0} \times \mathcal{W}$ is called the $k^{th}$* **waypoint** *of path $S_i$. The restriction of $S_i$ on the $k^{th}$ time interval $[t_{i,k-1}, t_{i,k}]$ is called the $k^{th}$* **segment** *of $S_i$ and is denoted by $S_i^{(k)}$.*

## III. Solving the planning problems using MILP

**Overview of the approach.** We formulate the problem of finding PWL paths satisfying the specifications (inter-agent collision avoidance and STL satisfaction) as a constrained

optimization problem and solve its mixed-integer linear programming (MILP) encoding using off-the-shelf optimizers such as Gurobi. The optimization problem is as follows:

$$\begin{aligned} \min_{\mathcal{C}} \quad & \mathcal{L}(\mathcal{C}) \\ \text{s.t. } & (S_1, \cdots, S_N) \text{ satisfy the conditions in Def. 4.} \end{aligned} \tag{2}$$

where $\mathcal{C} := \bigcup_{i=1}^{N} \bigcup_{k=0}^{K_i} \{t_{i,k}, p_{i,k}\}$ is the set of variables representing the time stamps and waypoints on the PWL reference paths $(S_1, \cdots, S_N)$, and $\{K_i\}_{i=1}^{N}$ are constants. Here, $\mathcal{L}$ is a linear cost function. For example, one can minimize the total travel time, $\mathcal{L}(\mathcal{C}) = \sum_{i=1}^{N} t_{i,K_i}$. One can also minimize the makespan using $\mathcal{L}(\mathcal{C}) = T_{makespan}$ with extra linear constraints $T_{makespan} \geq t_{i,K_i}$, $i = 1, \cdots, N$.

In order to solve the above optimization problem as a MILP problem, the constraint in Eq. (2) must be transformed into *a conjunction of linear constraints*, where each constraint is of the form $\texttt{LE} \geq 0$, and $\texttt{LE}$ is a linear expression of the decision variables. In addition to the aforementioned continuous variables $\mathcal{C}$, the decision variables of the MILP problem will contain another set of variables $\mathcal{B}$ that are the binary variables introduced when encoding logic relations. Also, this transformation must be sound, i.e., the feasible set defined by the linear constraints should be a subset of the original feasible set in Eq. (2).

In our approach, we first convert the original constraint in Eq. (2) into a *linear constraint formula (LCF)*, which is a logic sentence of atomic formulas connected by conjunction or disjunction operators. The atomic formulas are of the form $\texttt{LE}_{\mathcal{C}} \geq 0$, where $\texttt{LE}_{\mathcal{C}}$ is a linear expression of the continuous variables $\mathcal{C}$. Then, the disjunctions in the LCF are eliminated using the big-M method, and binary variables $\mathcal{B}$ are introduced in this step. After eliminating all the disjunctions, the LCF becomes a conjunction of linear constraints.

This section is structured as follows. We first show how to transform the STL satisfaction and inter-agent collision avoidance into LCFs in Section III-A and Section III-B respectively. In Section III-C, we show the overall algorithm.

### A. Encoding MA-STL satisfactions with LCFs

In this section, we consider the problem of encoding an MA-STL specification $\Psi$ with LCFs. Recall the syntax of MA-STL in Definition 2. In an MA-STL formula $\Psi$, there are only conjunction and disjunction operations in addition to single-agent STL formulas. Thus, if we can find an LCF for each single-agent STL formula $\pi_i^{\varphi}$ in $\Psi$, then these LCFs can be directly combined with conjunctions and disjunctions to get the LCF for $\Psi$. Hence, in this section, we only consider the encoding of single-agent STL formulas, and the subscript $i$ is omitted for simplicity. In conclusion, given an STL formula $\varphi$ and the tracking error $\epsilon > 0$, we aim at obtaining an LCF over the time-stamped waypoints $\{t_k, p_k\}_{k=0}^{K}$ such that if this LCF is true then the PWL path is $\epsilon$-robust to $\varphi$.

Such an LCF can be constructed inductively. We will construct an LCF for each of the $K$ segments of $S$ and denote them by $z_i^{\varphi}$, $i = 0, 1, \cdots, K-1$. We want $z_i^{\varphi}$ to have a strong soundness property: $z_i^{\varphi}$ is true $\implies \forall t \in [t_i, t_{i+1}], (p, t) \vDash \varphi$ for any trajectory $p$ deviating from $S$ up to the tracking error $\epsilon$, i.e., starting from any time point on the segment, $\varphi$ is satisfied robustly. Once we obtain such LCFs for $\varphi$, $z_0^{\varphi}$ is just the LCF we ultimately want. Fortunately, LCFs with such a property can be encoded inductively starting from the atomic predicates and their negations.

For an atomic predicate $\varphi = \pi^{\mu}$ or its negation $\neg\pi^{\mu}$, where $\mu(x) := b - H \cdot x$, it is easy to construct $z_i^{\varphi}$ by shrinking or bloating $\texttt{Poly}(H, b)$ as follows.

$$\begin{aligned} z_i^{\pi^{\mu}} = \bigwedge_{j=1}^{\texttt{Row}(H)} \Big( & \big(b^{(j)} - H^{(j)} \cdot p_i - \epsilon \|H^{(j)}\|_2 \geq 0\big) \\ & \wedge \big(b^{(j)} - H^{(j)} \cdot p_{i+1} - \epsilon \|H^{(j)}\|_2 \geq 0\big) \Big); \end{aligned} \tag{3}$$

$$\begin{aligned} z_i^{\neg\pi^{\mu}} = \bigvee_{j=1}^{\texttt{Row}(H)} \Big( & \big(H^{(j)} \cdot p_i - b^{(j)} - \epsilon \|H^{(j)}\|_2 \geq 0\big) \\ & \wedge \big(H^{(j)} \cdot p_{i+1} - b^{(j)} - \epsilon \|H^{(j)}\|_2 \geq 0\big) \Big). \end{aligned} \tag{4}$$

It is easy to verify that the constructed $z$ formulas have the aforementioned soundness property. Intuitively, Eq. (3) requires both endpoints of the $i$-th segment of $S$ to be in the shrunk polytope, which is sufficient for the whole segment to be in the polytope. Eq. (4) requires both endpoints to be on the outside of at least one face of the bloated polytope, which is sufficient for the whole segment to be outside the polytope.

For non-atomic predicates, its $z$ formula will depend on the $z$ formulas of its sub-predicates, e.g., $z^{\Box_{[a,b]}\varphi}$ depends on $z^{\varphi}$. The principle behind the design of the encoding rules is induction: we should guarantee that the aforementioned soundness property holds for the resulting $z$ formula if it holds for all the $z$ formulas of the sub-predicates (i.e., the $z$ formulas on the right-hand side of the below encoding rules).

For conjunctions and disjunctions, it is simply $z_i^{\varphi_1 \wedge \varphi_2} = z_i^{\varphi_1} \wedge z_i^{\varphi_2}$; $z_i^{\varphi_1 \vee \varphi_2} = z_i^{\varphi_1} \vee z_i^{\varphi_2}$.

Temporal operators are handled as follows.

$$z_i^{\Box_{[a,b]}\varphi} = \bigwedge_{j=0}^{K-1} \big([t_j, t_{j+1}] \cap [t_i + a, t_{i+1} + b] \neq \varnothing \Rightarrow z_j^{\varphi}\big); \tag{5}$$

$$\begin{aligned} z_i^{\Diamond_{[a,b]}\varphi} = & (t_{i+1} - t_i \leq b - a) \\ & \wedge \bigvee_{j=0}^{K-1} \big([t_j, t_{j+1}] \cap [t_{i+1} + a, t_i + b] \neq \varnothing \wedge z_j^{\varphi}\big); \end{aligned} \tag{6}$$

$$\begin{aligned} z_i^{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2} = & (t_{i+1} - t_i \leq b - a) \wedge \\ & \bigvee_{j=0}^{K-1} \Big( [t_j, t_{j+1}] \cap [t_{i+1} + a, t_i + b] \neq \varnothing \wedge z_j^{\varphi_2} \\ & \wedge \bigwedge_{l=0}^{j} \big([t_l, t_{l+1}] \cap [t_i, t_{i+1} + b] \neq \varnothing \implies z_l^{\varphi_1}\big) \Big); \end{aligned} \tag{7}$$

$$\begin{aligned} z_i^{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2} = \bigwedge_{j=0}^{K-1} \Big( & \big([t_j, t_{j+1}] \cap [t_i + a, t_{i+1} + b] \neq \varnothing \\ & \Rightarrow z_j^{\varphi_2}\big) \vee \bigvee_{l=0}^{j-1} \big([t_l, t_{l+1}] \cap [t_{i+1}, t_{i+1} + b] \neq \varnothing \wedge z_l^{\varphi_1}\big) \Big). \end{aligned} \tag{8}$$

With the above rules of encoding, we can encode any STL formula as an LCF as follows. As mentioned earlier, $z_0^{\varphi}$ is what we ultimately want. In order to obtain $z_0^{\varphi}$, all of its dependencies on other $z$ formulas have to be resolved. Therefore, the algorithm runs recursively. The recursion stops at atomic predicates since they do not depend on any other $z$ formulas as shown in Eq. (3) and Eq. (4).

In order to prove the aforementioned soundness property, we proceed by induction. The base cases are the atomic predicates (Eq. (3) and (4)), for which we have provided some intuitions earlier. Then, the induction step has to be verified for each non-atomic predicate. The verification is straightforward but tedious. Here, we only verify the one for the "$\Box$" operation in Eq. (5). The induction hypothesis is that the soundness property holds for all $z$ formulas on the RHS of Eq. (5). Considering any trajectory $p$ deviating from $S$ up to $\epsilon$, by induction hypothesis, if $z_j^{\varphi}$ is true, then $\forall t \in [t_j, t_{j+1}]$, $(p,t) \vDash \varphi$. For any $t \in [t_i, t_{i+1}]$ and any $t' \in [t+a, t+b]$, we must have that $t' \in [t_i+a, t_{i+1}+b]$. Now, assume that $z_i^{\Box_{[a,b]}\varphi}$ is true. Let $j$ be such that $t' \in [t_j, t_{j+1}]$. Then, $[t_i + a, t_{i+1} + b] \cap [t_j, t_{j+1}] \neq \varnothing$. According to the encoding, this implies that $z_j^{\varphi}$ is true. By induction hypothesis, we have that $(p,t') \vDash \varphi$. To summarize, if $z_i^{\Box_{[a,b]}\varphi}$ is true, then $\forall t \in [t_i, t_{i+1}]$, $\forall t' \in [t+a, t+b]$, $(p,t') \vDash \varphi$, which is equivalent to say that $\forall t \in [t_i, t_{i+1}]$, $(p,t) \vDash \Box_{[a,b]}\varphi$. Thus, we have proved the soundness property for $z_i^{\Box_{[a,b]}\varphi}$. A complete proof can be found in Appendix of [26].

**Remark.** *With the above proof, it should be clear that although the encoding rules are stronger than we would need, i.e., $z_i^{\phi}$ encodes satisfaction over the entire segment, and thus make the problem harder to solve, it is indeed necessary. Otherwise, the induction does not hold.*

### *B. Encoding inter-agent collision avoidance with LCFs*

We consider the problem of encoding the inter-agent collision avoidance with LCFs. Specifically, we aim at obtaining an LCF such that if this LCF is true, then at any time, the distance between any two agents is safe. First, let us consider how to encode the specification that two time-stamped line segments are at least $\epsilon$ away from each other, which will be the building block for encoding the inter-agent collision avoidance specification. Consider two time-stamped line segments, $\texttt{SEG}_1$ and $\texttt{SEG}_2$. Let the endpoints of $\texttt{SEG}_1$ be $(t_{11}, p_{11})$ and $(t_{12}, p_{12})$. Similarly, $(t_{21}, p_{21})$ and $(t_{22}, p_{22})$ are the endpoints of $\texttt{SEG}_2$. Define a function $\texttt{safe}()$ mapping them to an LCF as follows.

$$\begin{aligned}\texttt{safe}(\texttt{SEG}_1, \texttt{SEG}_2, \epsilon) := ([t_{11}, t_{12}] \cap [t_{21}, t_{22}] = \varnothing) \\ \vee \left( \left\| \frac{p_{11}+p_{12}}{2} - \frac{p_{21}+p_{22}}{2} \right\|_1 \geq \right. \\ \left. \left\| \frac{p_{11}-p_{12}}{2} \right\|_1 + \left\| \frac{p_{21}-p_{22}}{2} \right\|_1 + \epsilon\sqrt{d} \right),\end{aligned}$$

where $d$ is the dimensionality of the workspace. Intuitively, if the above LCF is true, either of the following two conditions is true. 1) the two segments are disjoint in the time dimension; or 2) in the spatial dimension, the distance between the two centers is greater than the summation of the half-lengths of the two segments with a margin $\epsilon$, and thus they are disjoint. Then, the specification that all the agents will not collide with each other is encoded as follows.

$$z_{\texttt{inter}} = \bigwedge_{\substack{i,j=1 \\ i \neq j}}^{N} \bigwedge_{\substack{k=1,\ldots,K_i \\ l=1,\ldots,K_j}} \texttt{safe}\left( S_i^{(k)}, S_j^{(l)}, 2\epsilon + s_i + s_j \right),$$

which is, again, an LCF of the decision variables $\bigcup_{i=1}^{N} \bigcup_{k=0}^{K_i} \{t_{i,k}, p_{i,k}\}$. Recall that $s_i$ is the size of agent $i$. Please also note that we use 1-norm instead of 2-norm in the encoding to make the resulting expression linear (or at least piece-wise linear). Also, a formal proof of soundness can be found in Appendix of [26].

### *C. Overall algorithm*

In this section, we show the overall algorithm. Each step of the algorithm is explained in the following.

**Construct an AND-OR tree.** In Section III-A and Section III-B, we have shown how to transform the STL satisfaction and inter-agent collision avoidance to LCFs. These LCF formulas can be further merged with conjunctions and disjunctions into a single LCF. Such an LCF can be represented as an AND-OR tree. There are three types of nodes in the tree, AND nodes (i.e., conjunctions), OR nodes (i.e., disjunctions), and leaf nodes. Each AND or OR node has a finite number of children. Each leaf node refers to a linear expression $\texttt{LE}$.

**Additional constraints.** Firstly, we need additional constraints for the time instants. For each PWL path $S_i$, we need $0 = t_{i,0} \leq t_{i,1} \leq \cdots \leq t_{i,K_i} \leq T$, where $T$ is a constant specified by the user. Secondly, the maximum velocity of the PWL paths should also be constrained. For each PWL path $S_i$,

$$\|p_{i,k+1} - p_{i,k}\|_1 \leq \texttt{vmax} * (t_{i,k+1} - t_{i,k}),\ k = 0, 1, \cdots, K_i - 1,$$

where $\texttt{vmax}$ is a constant specified by the user. Finally, the PWL path must start from the initial position of the agent, i.e., $p_{i,0} = p_i^{\texttt{init}}$. Please note that all these constraints are linear and can be easily merged into the AND-OR tree.

**Create MILP constraints from the AND-OR tree.** In order to create MILP constraints, we have to eliminate all the disjunctions in the tree so that the whole tree is converted into a conjunction of linear constraints, i.e., $\bigwedge_i \texttt{LE}_i \geq 0$. Then, we can add each $\texttt{LE}_i \geq 0$ as a linear constraint to the MILP optimizer. To eliminate the disjunctions, we use the big-M method. For example, given an OR node, $\bigvee_{i=1}^{n} \texttt{LE}_i \geq 0$, we introduce $n$ binary variables $z_i,\ i = 1, \cdots, n$. Then, it can be shown that the conjunctive form $(\bigwedge_{i=1}^{n} \texttt{LE}_i + (1 - z_i) \cdot M \geq 0) \wedge (\sum_{i=1}^{n} z_i \geq 1)$ is equivalent to the original disjunctive form, where $M$ is a large enough positive constant. Intuitively, if $z_i = 1$, then $\texttt{LE}_i + (1 - z_i) \cdot M \geq 0$ becomes the original constraint $\texttt{LE}_i \geq 0$. On the other hand, if $z_i = 0$, then $\texttt{LE}_i \geq 0$ is disabled since $\texttt{LE}_i + M \geq 0$ is trivially true regardless of the value of $\texttt{LE}_i$. Finally, $\sum_{i=1}^{n} z_i \geq 1$ enforces that at least one of the constraints is enabled.

**Putting it all together.** The algorithm first creates continuous variables in the optimizer, which represents the waypoints. Then, it constructs the AND-OR tree of the linear constraints. Next, disjunctions in the tree are eliminated, and the tree is converted into a list of linear constraints, which are then added to the optimizer. After the optimizer finds a feasible solution, the values of the continuous variables are returned, which determine the PWL paths.

**Complexity.** The computational cost of solving a MILP problem is mostly determined by the number of binary variables. Therefore, we analyze the number of binary variables introduced for encoding an STL formula $\varphi$ with respect to a PWL path of length $K$. Due to the use of the big-M method, each child of an OR node in the AND-OR tree introduces a binary variable. As in Eq. (6-8), each $z$ formula consists of $\mathcal{O}(K)$ disjunctions.[3] Since each segment has a $z$ formula, we will have $\mathcal{O}(K^2)$ disjunctions in order to encode a single operation. Let $|\varphi|$ be the number of operators in $\varphi$. The complexity of the proposed approach is $\mathcal{O}(K^2 \cdot |\varphi|)$. For MPC-based methods (e.g., [27]), the complexity of encoding is $\mathcal{O}(N \cdot |\varphi|)$, where $N$ is the number of time steps. Although the proposed approach has a quadratic complexity while the MPC-based approach has a linear complexity, in many practical cases, the time horizon is long (and hence $N$ is large) but the task can be completed with very few line segments. In these cases, $K^2 \ll N$ and the proposed method drastically outperforms MPC-based methods, which is empirically verified in the experiments in Sec. IV. On the other hand, in cases where the time horizon is small but the required number of segments is somehow large, using a MPC-based approach could be a better choice.

## IV. Experimental evaluation

We evaluate the proposed approach on several benchmark scenarios and compare it with several other methods. The algorithm is implemented in Python, and the Gurobi optimizer is used for solving the MILP problems. The implementation is available at https://github.com/sundw2014/STLPlanning. As for the robot dynamics, we use the Dubins vehicle model. All experiments were conducted on a Linux workstation with two Intel Xeon Silver 4110 CPUs and 32 GB RAM.

### A. Benchmarks

Some of the benchmarks were borrowed from the motion planning literature [28], [5]. We also designed several other benchmarks in order to show the ability of the proposed method to handle complicated MA-STL specifications. Specifically, the following benchmarks are used.

`stlcg-1` is from [5]. As shown in Fig. 1a, a robot starting from the bottom-left corner is asked to visit the upright corner. It is also asked to visit and avoid some regions in the middle. Denote the four regions by $Y$ (yellow), $B$ (blue), $G$ (green), and $R$ (red) respectively. The task is specified using an STL formula $\left(\Diamond_{[0,T]}\Box_{[0,5]}R\right) \wedge \left(\Diamond_{[0,T]}\Box_{[0,5]}G\right) \wedge \left(\Box_{[0,T]}\neg B\right)$. Please note that in the original benchmark, region $B$ is a circle, and here we replace it with its circumscribed square. `stlcg-2` uses the same environment as in `stlcg-1` but with a different STL specification $\left(\Diamond_{[0,T]}\Box_{[0,5]}Y\right) \wedge \left(\Box_{[0,T]}\neg G\right) \wedge \left(\Box_{[0,T]}\neg B\right)$.

`doorpuzzle-1` is from [28]. As shown in Fig. 1c, a robot is asked to visit the goal region (blue). However, there are walls (black) and doors (red) in the environment. Before being able to open a door, the robot has to visit the correspondingly numbered green region to pick the key. Denote the goal by $G$, the wall by $W$, the doors by $D_1, \cdots, D_5$, and the keys by $K_1, \cdots, K_5$. Then, the task can be specified using an STL formula $\left(\Diamond_{[0,T]}G\right) \wedge \left(\Box_{[0,T]}\neg W\right) \wedge \left(\bigwedge_{i=1}^{5} \neg D_i \mathcal{U}_{[0,T]} K_i\right)$. `doorpuzzle-2` is a similar scenario from [28] with 6 doors.

`rover-1` and `rover-2` are designed to evaluate the ability to handle complicated multi-agent STL specifications. As shown in Fig. 1e, rovers are asked to visit the goal regions (green) to make scientific observations while conforming to the following rules: 1) Every rover should visit the charging station (blue) within $t_c$ time units every time they leave the charging station; 2) After visiting a goal region, the rover should visit a transmitter (yellow) within $t_d$ time units, to transmit the collected data to the remote control; 3) The rovers should avoid the walls (black) and each other. Denote the charging station by $C$, the walls by $W$, the transmitters by $S_1$ and $S_2$, and the goals by $G_1, \cdots, G_4$. Then, the rule of charging can be encoded as $\varphi_1 := \Box_{[0,T]}(\neg C \implies \Diamond_{[0,t_c]} C)$. The rule of transmitting can be encoded as $\varphi_2 := \Box_{[0,T]}(\bigvee_{i=1}^{4} G_i \implies \Diamond_{[0,t_d]} \bigvee_{i=1}^{2} S_i)$. The rule of avoiding walls can be encoded as $\varphi_3 := \Box_{[0,T]}\neg W$. Assuming that $N$ rovers are involved, the MA-STL specification encoding the task is $\Psi = \bigwedge_{i=1}^{N} \pi_i^{(\varphi_1 \wedge \varphi_2 \wedge \varphi_3)} \wedge \bigwedge_{j=1}^{4} \bigvee_{i=1}^{N} \pi_i^{\Diamond_{[0,T]} G_j}$. We set $N = 1$ and 2 for `rover-1` and `rover-2` respectively.

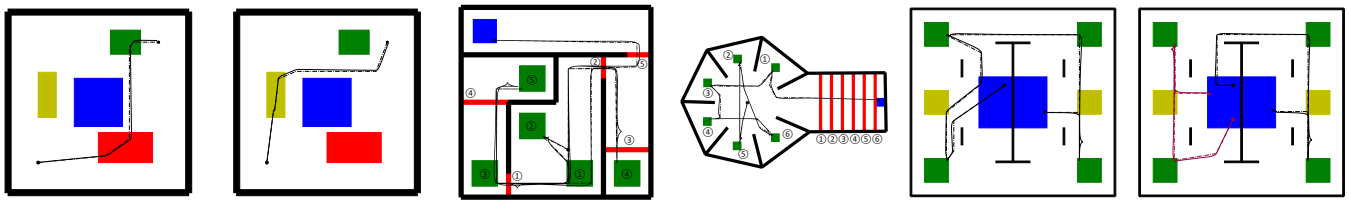
`wall-1` and `wall-2` are designed to evaluate the ability to arrange multiple agents to avoid collisions. As shown in Fig. 2a and Fig. 2b, a group of agents are asked to visit some goal regions, but there is a narrow door in the middle of the map. In order to avoid collisions, the agents have to figure out an order for them to go through the door. Let the wall (black) be $W$, and the goals be $G_1, \cdots, G_4$. The task is specified as $\Psi = \bigwedge_{i=1}^{4} \pi_i^{\Box_{[0,T]}\neg W \wedge \Diamond_{[0,T]} G_i}$.

### B. Comparison with other methods

We compared our method with others, including an MPC-based method [6] and an abstraction-based method. Please note that although [13] is the closest work to ours, the authors did not provide a publicly available implementation of their approach. Also, the benchmarks `doorpuzzle-1` and `doorpuzzle-2` are borrowed from [28], but the authors did not either provide an implementation of their algorithm or report the run time of their algorithm on these two benchmarks. Therefore, we were not able to compare with these approaches. The details of the setup are as follows.

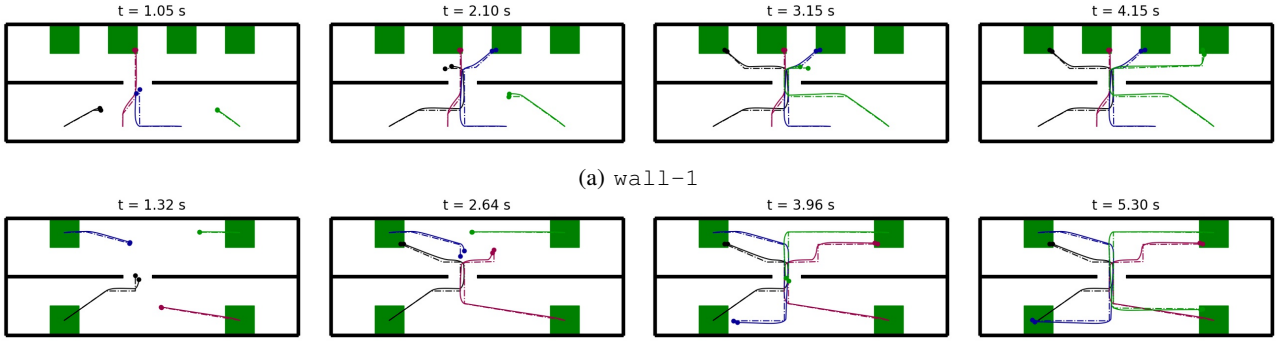
As stated earlier, the lengths of the PWL paths, i.e., $K_1, K_2, \cdots, K_N$, are constants. In the experiments, we set $K = K_1 = K_2 = \cdots = K_N$. Obviously, $K$ should be large enough, otherwise, the problem is not feasible. Thus, we start from $K = \underline{K}$, where $\underline{K}$ is an initial guess by the

[3] Please note that in the encoding of $\mathcal{R}$, although there are $\mathcal{O}(K^2)$ disjunctions for a single segment, after merging the repeated ones, there are only $\mathcal{O}(K)$ necessary disjunctions.



(a) `stlcg-1` (b) `stlcg-2` (c) `doorpuzzle-1` (d) `doorpuzzle-2` (e) `rover-1` (f) `rover-2`

Figure 1: Benchmarks and results. Dashed-lines are the PWL paths found by the proposed method; Solid lines are the actual trajectories tracking the PWL paths; The circle on the trajectory is the starting point, and star is the end.




(a) `wall-1`

(b) `wall-2`

Figure 2: Benchmarks and results. Dashed-lines are the PWL paths found by the proposed method; Solid lines are the actual trajectories tracking the PWL paths. For each benchmark, we show four snapshots of the simulation with a clock in the title. The dots indicate the current locations of the agents, and agents are marked in different colors.

user according to the task. If the problem is infeasible, we increment $K$ by 1 until the problem becomes feasible.

In [6], the authors proposed an MPC-based method of planning paths from STL specifications. It also models the planning as an MILP problem. The decision variables are just the states of the system at $\Delta t, 2\Delta t, \cdots, \lceil \frac{T}{\Delta t} \rceil \Delta t$, where the time step $\Delta t > 0$ is a small constant specified by the user. The dynamics of the robots are encoded as constraint of the MILP problem. Furthermore, the authors of [6] proposed a group of rules with which an STL formula can be converted into linear constraints. In the experiments, we set $\Delta t = 0.1$. To make the comparison fair, we use a very simple dynamics $\dot{x} = u$ for MPC, i.e., an integrator. As for the inter-agent collision avoidance requirement, we represent it as constraints that at each time step, the distance between any pair of agents must be greater than a threshold. Obviously, the performance of MPC highly relies on the time horizon $T$. However, we do not have an idea of how large $T$ should be for completing each benchmark. In order to determine a good time bound that is not too large but large enough for completing the task, we first run our algorithm with $T = 1000$ which is large enough for all the benchmarks in this section. Our algorithm returns a PWL path with the optimized travel time[4]. Then the makespan of the planned PWL paths is used as the $T$ when running the MPC-based algorithm. Therefore, both the proposed method and the MPC-based method need a pre-process to determine $K$ or $N$. To make the comparison clear and fair, we did not include the time spent for this pre-process in TABLE I.

[4]The solution is not exactly optimal. The precision of the solution depends on one of Gurobi's arguments, "`MIPGap`". We always uses the same "`MIPGap`" for MPC and our method.

| Benchmark | Ours (s) | MPC (s) | ABS (s) |
|---|---|---|---|
| `stlcg-1` | **0.855** | 6.5 | N/A |
| `stlcg-2` | **0.175** | 4.0 | N/A |
| `doorpuzzle-1` | **49.5** | TO | N/A |
| `doorpuzzle-2` | **175.6** | 2102.5 | N/A |
| `rover-1` | **180.5** | TO | N/A |
| `rover-2` | **101.6** | 2733.7 | N/A |
| `wall-1` | **20.8** | 113.7 | 50.7 |
| `wall-2` | 172.9 | 163.7 | **79.1** |

Table I: Run time on benchmarks. MPC failed in some cases due to time out (TO). ABS does not support general STL scenarios and can only handle the last two benchmarks.

We also implement an abstraction-based method based on [29], [30], which uses a MILP-based approach for optimal task assignment and ordering, and leverages the priority-based search to plan collision-free trajectories to achieve all the assigned tasks. It does not support general STL specifications but supports the tasks in `wall-1` and `wall-2`.

### C. Observations

The results are summarized in TABLE I. Planned paths can be found in Fig. 1 and Fig. 2. Some observations are in order. Firstly, the proposed method can correctly solve planning problems with complex STL specifications for multiple agents (up to 4) while other methods in comparison failed in some

cases. Secondly, the proposed method outperforms other methods in almost all cases in terms of run time. Thirdly, as shown in Fig. 1 and Fig. 2, because the tracking error is taken into account when planning the PWL paths, the actual trajectories of the robots satisfy the STL specification although they deviate from the reference PWL paths. Also, results show that our algorithm is able to correctly figure out the logical ordering of events with temporal constraints, then automatically assign tasks to each agent and do essential arrangement to avoid inter-agent collision. It is also worth mentioning that the tool (`stlcg`) proposed in [5] also uses a fixed time step and uses gradient decent to minimize the violation of the STL specification. It takes minutes to find paths for its two benchmark scenarios, `stlcg-1` and `stlcg-2`, while our method takes less than one second.

Furthermore, we evaluated the proposed approach on selected benchmarks, including `doorpuzzle-1`, `doorpuzzle-2`, `wall-1`, and `wall-2`, with real-world robots on the Robotarium [31] platform. For the real robots, we use the official tracking controller provided by the Robotarium team, and the tracking error is estimated from simulations using the official simulator. Experiments show that with the proposed approach and the tracking controller, the robots can safely complete the tasks. Videos can be found in the supplementary material.

## V. Conclusion

We introduced a novel method to synthesize long-horizon motions of multi-agent robotic systems for STL specifications. Our method can effectively encode complex specifications and support long-time horizon synthesis due to the combinatorial use of PWL reference paths and guaranteed tracking controller. We plan to further reduce the complexity of the encoding rules and support planning for larger-scale problems.

## References


[1] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[2] R. Majumdar, N. Yoshida, and D. Zufferey, "Multiparty motion coordination: from choreographies to robotics programs," *arXiv preprint arXiv:2010.05484*, 2020.

[3] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.

[4] C. Fan, K. Miller, and S. Mitra, "Fast and guaranteed safe controller synthesis for nonlinear vehicle models," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 629–652.

[5] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2020.

[6] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 81–87.

[7] S. M. LaValle, *Planning algorithms*. Cambridge univ. press, 2006.

[8] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[9] R. Ghosh, C. Hsieh, S. Misailovic, and S. Mitra, "Koord: a language for programming and verifying distributed robotics application," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–30, 2020.

[10] R. Ghosh, J. P. Jansch-Porto, C. Hsieh, A. Gosse, M. Jiang, H. Taylor, P. Du, S. Mitra, and G. Dullerud, "Cyphyhouse: A programming, simulation, and deployment toolchain for heterogeneous distributed coordination," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6654–6660.

[11] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, "Robots that use language," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 25–55, 2020.

[12] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[13] A. T. Buyukkocak, D. Aksaray, and Y. Yazıcıoğlu, "Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1375–1382, 2021.

[14] S. S. Farahani, V. Raman, and R. M. Murray, "Robust model predictive control for signal temporal logic synthesis," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 323–328, 2015.

[15] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2015, pp. 772–779.

[16] Y. E. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1189–1206, 2019.

[17] A. M. Jones, K. Leahy, C. I. Vasile, S. Sadradinni, Z. Serlin, R. Tron, and C. Belta, "Scalable and Robust Deployment of Heterogenenous Teams from Temporal Logic Specifications," in *International Symposium on Robotics Research (ISRR)*, Hanoi, Vietnam, October 2019.

[18] D. Gundana and H. Kress-Gazit, "Event-based signal temporal logic synthesis for single and multi-robot tasks," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3687–3694, 2021.

[19] G. Yang, C. Belta, and R. Tron, "Continuous-time signal temporal logic planning with control barrier functions," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 4612–4618.

[20] Y. Kantaros and M. M. Zavlanos, "STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, vol. 39, no. 7, 2020.

[21] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 1002–1028, 2020.

[22] J. Karlsson, F. S. Barbosa, and J. Tumova, "Sampling-based motion planning with temporal logic missions and spatial preferences," *IFAC-PapersOnLine*, vol. 53, no. 2, 2020, 21st IFAC World Congress.

[23] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3840–3847.

[24] E. J. Rodríguez-Seda, C. Tang, M. W. Spong, and D. M. Stipanović, "Trajectory tracking with collision avoidance for nonholonomic vehicles with acceleration constraints and limited sensing," *The International Journal of Robotics Research*, vol. 33, no. 12, 2014.

[25] D. Sun, S. Jha, and C. Fan, "Learning Certified Control using Contraction Metric," in *Conference on Robot Learning*, 2020.

[26] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications — extended version," 2021, available at https://www.daweisun.me/static/MA-STL.pdf.

[27] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th international conference on hybrid systems: Computation and control*, 2015, pp. 239–248.

[28] W. Vega-Brown and N. Roy, "Admissible abstractions for near-optimal task and motion planning," in *IJCAI*, 2018.

[29] K. Brown, O. Peltzer, M. A. Sehr, M. Schwager, and M. J. Kochenderfer, "Optimal sequential task assignment and path finding for multi-agent robotic assembly planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[30] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019.

[31] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1699–1706.