arXiv:2309.05632v1 [cs.RO] 11 Sep 2023

MAPS^2: Multi-Robot Anytime Motion Planning under Signal Temporal Logic Specifications

Mayank Sewlia, Christos K. Verginis, and Dimos V. Dimarogonas

***Abstract*—This article presents MAPS^2: a distributed algorithm that allows multi-robot systems to deliver coupled tasks expressed as Signal Temporal Logic (STL) constraints. Classical control theoretical tools addressing STL constraints either adopt a limited fragment of the STL formula or require approximations of min/max operators, whereas works maximising robustness through optimisation-based methods often suffer from local minima, relaxing any completeness arguments due to the NP-hard nature of the problem. Endowed with probabilistic guarantees, MAPS^2 provides an anytime algorithm that iteratively improves the robots' trajectories. The algorithm selectively imposes spatial constraints by taking advantage of the temporal properties of the STL. The algorithm is distributed, in the sense that each robot calculates its trajectory by communicating only with its immediate neighbours as defined via a communication graph. We illustrate the efficiency of MAPS^2 by conducting extensive simulation and experimental studies, verifying the generation of STL satisfying trajectories.**

I. Introduction

AUTONOMOUS robots possess the capability to solve significant problems when provided with a set of guidelines. These guidelines can be derived from either the physical constraints of the robot, such as joint limits, or imposed as human-specified requirements, such as pick-and-place objects. An efficient method of imposing such guidelines is the utilisation of logic-based tools, which enable reasoning about the desired behaviour of robots. These tools help us describe the behaviour of a robot at various levels of abstraction, such as interactions between its internal components to the overall high-level behaviour of the robot [1]. This strong expressivity helps us efficiently encode complex mission specifications into a logical formula. Recent research has focused on utilising these logic-based tools to express maximal requirements on the behaviour of robots. Once these requirements are established, algorithms are developed to generate trajectories that satisfy them.

Examples of logic-based tools include formal languages, such as Linear Temporal Logic (LTL), Metric Interval Temporal Logic (MITL), and Signal Temporal Logic (STL). The main distinguishing feature between these logics is their ability to encode time. While LTL operates in discrete-time and discrete-space domain, MITL operates in continuous-time domain but only enforces qualitative space constraints. On the other hand, STL allows for the expression of both qualitative and quantitative semantics of the system in both continuous-time and continuous-space domains [2]. STL thus provides a natural and compact way to reason about a robot's motion as it operates in a continuously evolving space-time environment. Although STL presents certain challenges, such as not being able to cast into a discrete-time discrete-space paradigm for which tractable planning algorithms already exist, we consider STL-based tasks due to the additional logical structures provided by STL over continuous-time signals and the availability of a robustness metric to determine the degree of satisfaction of a formula [3].

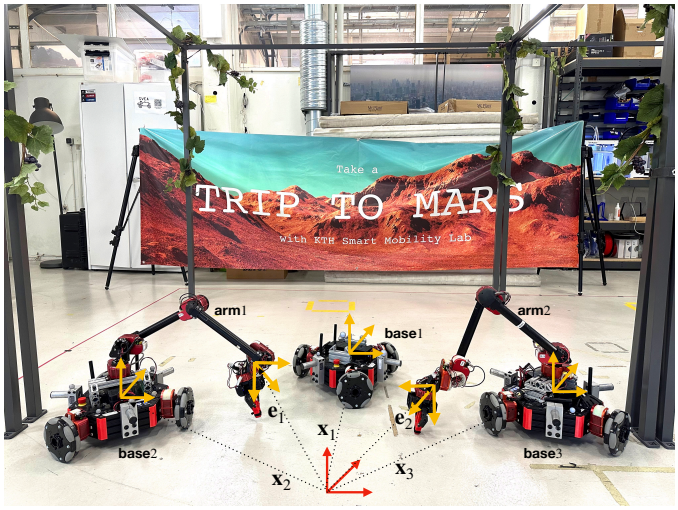


Figure 1: Experimental setup with three mobile bases and two 6-dof manipulators

Another important property of autonomous robots is their ability to coordinate and work in teams. The use of multiple robots is often necessary in situations where a single robot is either insufficient, the task is high-energy demanding, or unable to physically perform certain tasks. However, multi-robot systems present their own set of challenges, such as communication overload, the need for a central authority for commands, and high computational demands. The challenge, therefore, is to derive solutions for multi-robot problems utilising logic-based tools, ensuring the achievement of specified high-level behaviour. The problem becomes more complex when interdependent constraints are imposed between the robots. This complexity is amplified when using a purely distributed approach, i.e., when each robot computes its own actions by communicating only with its neighbours, without

*This work was supported by the ERC CoG LEAFHOUND, the Swedish Research Council (VR), the Knut och Alice Wallenberg Foundation (KAW) and the H2020 European Project CANOPIES.

M. Sewlia and D. V. Dimarogonas are with Division of Decision and Control, School of EECS, KTH Royal Institute of Technology, 100 44 Stockholm, Sweden. {sewlia, dimos}@kth.se

Christos K. Verginis is with Division of Signals and Systems, Department of Electrical Engineering, Uppsala University, Uppsala, Sweden. christos.verginis@angstrom.uu.se

access to the actions of other robots. Such an approach is necessary to prevent communication network congestion and minimise computational overheads.

In this article, we propose an algorithm that addresses the multi-robot motion planning problem subject to coupled STL tasks. The algorithm encodes these constraints into an optimisation function and selectively activates them based on the temporal requirements of the STL formula. While doing so, each robot only communicates with its neighbours and iteratively searches for STL satisfying trajectories. The algorithm proposed is called **MAPS**^{2} - **M**ulti-Robot **A**nytime Motion **P**lanning under **S**ignal Temporal Logic **S**pecifications. The article's contributions are summarised as follows:

- The algorithm ensures distributed trajectory generation to satisfy STL formulas that consist of coupled constraints for multiple robots.
- It reduces conservatism by eliminating the need for approximations, samples in continuous time to avoid abstractions, and achieves faster convergence by warm starting with initial trajectories.
- It incorporates a wide range of coupled constraints (both linear and nonlinear) into the distributed optimisation framework, enabling the handling of diverse tasks such as pick-and-place operations and time-varying activities like trajectory tracking.
- We present extensive simulation and hardware experiments that demonstrate the execution of complex tasks using MAPS^2.

Additionally, the algorithm presented is sound, meaning that it produces a trajectory that meets the STL formula and is complete, meaning that it will find such a trajectory if one exists.

In our prior study [4], we addressed the STL motion planning problem for two coupled agents. There, we extended the conventional Rapidly-exploring Random Trees (RRT) algorithm to sample in both the time and space domains. Our approach incrementally built spatio-temporal trees through which we enforced space and time constraints as specified by the STL formula. The algorithm employed a sequential planning method, wherein each agent communicated and waited for the other agent to build its tree. In contrast, the present work addresses the STL motion planning problem for multiple robots. Here, our algorithm adopts a distributed optimisation-based approach, where spatial and temporal aspects are decoupled to satisfy the STL formula. Instead of constructing an incremental tree, as done in the previous work, we introduce a novel metric called the *validity domain* and initialise the process with an initial trajectory. In the current research, we only incorporate the STL parse tree and the Satisfaction variable tree from our previous work (Section III-B here). Additionally, we present experimental validation results and introduce a novel STL verification architecture.

The rest of the paper is organised as follows. Section II presents the related work, Section III presents the notations and necessary preliminaries, Section IV formulates the problem of this work, Section V presents the STL inclusion along with the underlying assumptions and important definitions, Section VI presents the main algorithm MAPS^2 along with analyses of the algorithm, Section VIII presents the experimental validation on a real multi-robot setup, while Section VII presents simulations. Finally, Section IX concludes the paper.

II. Related Work

In the domain of single-agent motion planning, different algorithms have been proposed to generate safe paths for robots. Sampling-based algorithms, such as CBF-RRT [5], have achieved success in providing a solution to the motion planning problem in dynamic environments. However, they do not consider high-level complex mission specifications. Works that impose high-level specifications in the form of LTL, such as [6]–[9], resort to a hybrid hierarchical control regime resulting in abstraction and explosion of state-space. While a mixed integer program can encode this problem for linear systems and linear predicates [10], the resulting algorithm has exponential complexity, making it impractical for high-dimensional systems, complex specifications, and long duration tasks. To address this issue, [11] proposes a more efficient encoding for STL to reduce the exponential complexity in binary variables. Additionally, [12] introduces a new metric, discrete average space robustness, and composes a Model Predictive Control (MPC) cost function for a subset of STL formulas.

In multi-agent temporal logic control, works such as [13], [14] employ workspace discretisation and abstraction techniques, which we avoid in this article due to it being computationally demanding. Some approaches to STL synthesis involve using mixed-integer linear programming (MILP) to encode constraints, as previously explored in [15]–[17]. However, MILPs are computationally intractable when dealing with complex specifications or long-term plans because of the large number of binary variables required in the encoding process. The work in [18] encodes a new specification called multi-agent STL (MA-STL) using mixed integer linear programs (MILP). However, the predicates here depend only on the states of a single agent, can only represent polytope regions, and finally, temporal operations can only be applied to a single agent at a time. In contrast, this work explores coupled constraints between robots and predicates are allowed to be of nonlinear nature.

As a result, researchers have turned to transient control-based approaches such as gradient-based, neural network-based, and control barrier-based methods to provide algorithms to tackle the multi-robot STL satisfaction problem [11]. Such approaches, at the cost of imposing dynamical constraints on the optimisation problem, often resort to using smooth approximations of temporal operators at the expense of completeness arguments or end-up considering only a smaller fragment of the syntax [19]–[22]. STL's robust semantics are used to construct cost functions to convert a synthesis problem to an optimisation problem that benefits from gradient-based solutions. However, such approaches result in non-smooth and non-convex problems and solutions are prone to local minima [23]. In this work, we avoid approximations and consider the full expression of the STL syntax. The proposed solution adopts a purely geometrical approach to the multi-robot STL planning problem.

III. Notations and Preliminaries

The set of natural numbers is denoted by $\mathbb{N}$ and the set of real numbers by $\mathbb{R}$. With $n \in \mathbb{N}$, $\mathbb{R}^n$ is the set of n-coordinate real-valued vectors and $\mathbb{R}^n_+$ is the set of real n-vector with non-negative elements. The cardinality of a set A is denoted by $|A|$. If $a \in \mathbb{R}$ and $[b, c] \in \mathbb{R}^2$, the Kronecker sum is defined as $a \oplus [b, c] = [a+b, a+c] \in \mathbb{R}^2$. We further define the Boolean set as $\mathbb{B} = \{\top, \bot\}$ (True, False). The acronym *DOF* stands for degrees of freedom.

A. Signal Temporal Logic (STL)

Let $\mathbf{x} : \mathbb{R}_+ \to \mathbb{R}^n$ be a continuous-time signal. Signal temporal logic [2] is a predicate-based logic with the following syntax:

$$\varphi = \top \mid \mu^h \mid \neg\varphi \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \mid \varphi_1 \wedge \varphi_2 \tag{1}$$

where φ_1, φ_2 are STL formulas and $\mathcal{U}_{[a,b]}$ encodes the operator *until*, with $0 \leq a < b < \infty$; μ^h is a predicate of the form $\mu^h : \mathbb{R}^N \to \mathbb{B}$ defined by means of a vector-valued predicate function $h : \mathbb{R}^N \to \mathbb{R}$ as

$$\mu^h = \begin{cases} \top & h(\mathbf{x}(t)) \leq 0 \\ \bot & h(\mathbf{x}(t)) > 0 \end{cases}. \tag{2}$$

The satisfaction relation $(\mathbf{x}, t) \models \varphi$ indicates that signal $\mathbf{x}$ satisfies φ at time t and is defined recursively as follows:

$$\begin{aligned}
&(\mathbf{x}, t) \models \mu^h && \Leftrightarrow h(\mathbf{x}(t)) \leq 0 \\
&(\mathbf{x}, t) \models \neg\varphi && \Leftrightarrow \neg((\mathbf{x}, t) \models \varphi) \\
&(\mathbf{x}, t) \models \varphi_1 \wedge \varphi_2 && \Leftrightarrow (\mathbf{x}, t) \models \varphi_2 \wedge (\mathbf{x}, t) \models \varphi_2 \\
&(\mathbf{x}, t) \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 && \Leftrightarrow \exists t_1 \in [t+a, t+b] \text{ s.t. } (\mathbf{x}, t_1) \models \varphi_2 \\
& && \quad \wedge \forall t_2 \in [t, t_1], (\mathbf{x}, t_2) \models \varphi_1.
\end{aligned}$$

We also define the operators *disjunction*, *eventually*, and *always* as $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathcal{F}_{[a,b]}\varphi \equiv \top \mathcal{U}_{[a,b]}\varphi$, and $\mathcal{G}_{[a,b]}\varphi \equiv \neg\mathcal{F}_{[a,b]}\neg\varphi$, respectively. Each STL formula is valid over a time horizon defined as follows.

Definition 1 ([24]). *The time horizon* $\mathrm{th}(\varphi)$ *of an STL formula* φ *is recursively defined as,*

$$\mathrm{th}(\varphi) = \begin{cases} 0, & \text{if } \varphi = \mu \\ \mathrm{th}(\varphi_1), & \text{if } \varphi = \neg\varphi_1 \\ \max\{\mathrm{th}(\varphi_1), \mathrm{th}(\varphi_2)\}, & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ b + \max\{\mathrm{th}(\varphi_1), \mathrm{th}(\varphi_2)\}, & \text{if } \varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2. \end{cases} \tag{3}$$

In this work, we consider only time bounded temporal operators, i.e., when $\mathrm{th}(\varphi) < \infty$. In the case of unbounded STL formulas, it is only possible to either falsify an *always* operator or satisfy an *eventually* operator in finite time, thus we consider only bounded time operators.

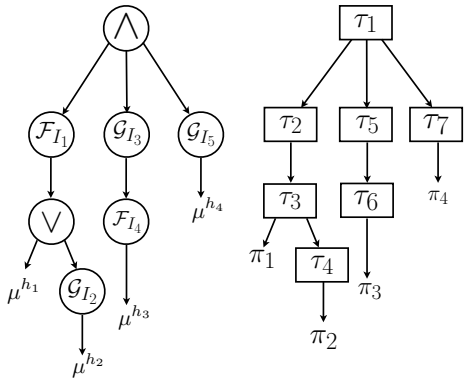


Figure 2: STL parse tree and satisfaction variable tree for the formula in (4).

B. STL Parse tree

An STL parse tree is a tree representation of an STL formula [4]. It can be constructed as follows:

- Each node is either a temporal operator node $\{\mathcal{G}_I, \mathcal{F}_I\}$, a logical operator node $\{\vee, \wedge, \neg\}$, or a predicate node $\{\mu^h\}$, where $I \subset \mathbb{R}$ is a closed interval;
- temporal and logical operator nodes are called *set* nodes;
- a root node has no parent node and a leaf node has no child node. The leaf nodes constitute the predicate nodes of the tree.

A path in a tree is a sequence of nodes that starts at a root node and ends at a leaf node. The set of all such paths constitutes the entire tree. A subpath is a path that starts at a set node and ends at a leaf node; a subpath could also be a path. The resulting formula from a subpath is called a subformula of the original formula. In the following, we denote any subformula of an STL formula φ by $\bar{\varphi}$. Each set node is accompanied by a satisfaction variable $\tau \in \{+1, -1\}$ and each leaf node is accompanied by a predicate variable $\pi = \mu^h$ where h is the corresponding predicate function. A signal $\mathbf{x}$ satisfies a subformula $\bar{\varphi}$ if $\tau = +1$ corresponding to the set node where the subpath of $\bar{\varphi}$ begins. Subsequently, $\tau(\textit{root}) = +1 \Leftrightarrow (\mathbf{x}, t) \models \varphi$ where *root* is the root node of φ. An analogous tree of satisfaction and predicate variables can be drawn, called *satisfaction variable tree*. The satisfaction variable tree borrows the same tree structure as the STL parse tree. Each set node from the STL parse tree maps uniquely to a satisfaction variable τ_i and each leaf node maps uniquely to a predicate variable π_i, where i is an enumeration of the nodes in the satisfaction variable tree. An example of construction of such trees is shown below.

Example 1. *The STL parse tree and the satisfaction variable tree for the STL formula*

$$\varphi = \mathcal{F}_{I_1}\Big(\mu^{h_1} \vee \mathcal{G}_{I_2}(\mu^{h_2})\Big) \wedge \mathcal{G}_{I_3}\mathcal{F}_{I_4}(\mu^{h_3}) \wedge \mathcal{G}_{I_5}(\mu^{h_4}). \tag{4}$$

are shown in Figure 2. From the trees, one obtains the implications $\tau_2 = +1 \implies (\mathbf{x}, t) \models \mathcal{F}_{I_1}\Big(\mu^{h_1} \vee \mathcal{G}_{I_2}(\mu^{h_2})\Big)$, *and* $\tau_7 = +1 \implies (\mathbf{x}, t) \models \mathcal{G}_{I_5}(\mu^{h_4})$.

IV. Problem Formulation

We start by presenting the multi-robot system and defining the types of constraints, followed by presenting the topology and finally stating the problem being addressed.

A. Multi-robot system

Consider a multi-robot system with states $\mathbf{x} = [\mathbf{x}_1^\top, \mathbf{x}_2^\top, \dots]^\top$ where each robot $i \in \mathbf{V} = \{1, \dots, \mathbf{N}\}$ consists of states $\mathbf{x}_i \in \mathbb{R}^{n_i}$ subject to state constraints $\mathbf{x}_i \in \mathcal{X}_i \subseteq \mathbb{R}^{n_i}$. Some robots further need to satisfy coupled state constraints $\mathbf{x}_i \in \mathcal{X}_{ij}(\mathbf{x}_j)$. Then, we call the robot j a neighbour of the robot i and write $j \in \mathcal{N}_i$ where $\mathcal{N}_i$ is the set of all such neighbours of robot i. These state constraints are the task specifications derived from the STL formula i.e., they represent tasks to be executed by the robots. Additionally, these constraints can entail obstacle avoidance and are therefore outlined within the STL formula. The state constraints are defined by the inequalities

$$\begin{aligned} {}_\alpha h_i(\mathbf{x}_i) &\leq 0, \quad \alpha \in \{1, 2, \dots, r_i\} \\ {}_\beta h_{ij}(\mathbf{x}_i, \mathbf{x}_j) &\leq 0, \quad j \in \mathcal{N}_i, \quad \beta \in \{1, 2, \dots, s_i\} \end{aligned} \tag{5}$$

where ${}_\alpha h_i$ and ${}_\beta h_{ij}$ are continuous functions; ${}_\alpha h_i : \mathbb{R}^{n_i} \to \mathbb{R}$ specifies constraints on robot i and r_i is the number of such constraints, and, ${}_\beta h_{ij} : \mathbb{R}^{n_i} \times \mathbb{R}^{n_j} \to \mathbb{R}$ specifies coupled constraints between robots i and j with s_i being the number of such constraints. The state constraint sets are then defined as,

$$\begin{aligned} \mathcal{X}_i &:= \{x_i \in \mathbb{R}^{n_i} |_\alpha h_i(\mathbf{x}_i) \leq 0\} \\ \mathcal{X}_{ij}(\mathbf{x}_j) &:= \{x_i \in \mathbb{R}^{n_i} |_\beta h_{ij}(\mathbf{x}_i, \mathbf{x}_j) \leq 0\}. \end{aligned}$$

We further consider that ${}_\beta h_{ij}(\mathbf{x}_i, \mathbf{x}_j) = {}_\beta h_{ji}(\mathbf{x}_j, \mathbf{x}_i)$. The inequalities ${}_\alpha h_i$ and ${}_\beta h_{ij}$ are predicate function inequalities of the form (2). A predicate function inequality $h(\mathbf{x}_i(t)) \leq 0$ corresponds to state constraint ${}_\alpha h_i(\mathbf{x}_i) \leq 0$ and $h(\mathbf{x}_i(t), \mathbf{x}_j(t)) \leq 0$ corresponds to the state constraint ${}_\beta h_{ij}(\mathbf{x}_i, \mathbf{x}_j) \leq 0$. Next, we state a common assumption regarding the STL formula.

Assumption 1. *The STL formula is in* positive normal form *i.e., it does not contain the negation operator.*

The above assumption does not cause any loss of expression of the STL syntax (1). As shown in [17], any STL formula can be written in positive normal form by moving the negation operator to the predicate.

B. Graph topology

The coupled state constraints ${}_\beta h_{ij}$ define an undirected graph over the multi-robot system. The graph is given by $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$ where $\mathbf{E} = \{(i, j) | j \in \mathcal{N}_i\}$ is the set of edges; $\mathbf{E}$ defines the communication links between the robots in $\mathbf{V}$.

C. Problem statement

Let $\mathcal{W} \subset \mathbb{R}^{\sum_i n_i}$ be defined as the workspace in which the robots operate, and let $\mathcal{S} \subseteq \mathcal{W}$ be a compact set where a trajectory $\mathbf{y} : [0, \mathrm{th}(\varphi)] \to \mathcal{S}$ satisfies the STL formula (as in (1)). The set $\mathcal{S}$ is referred to as the satisfiable set. It is assumed that obstacles are defined in the STL formula, making $\mathcal{S}$ the free space and ensuring that any continuous trajectories within $\mathcal{S}$ satisfies the STL formula. Moreover, we have the following feasibility assumption:

Assumption 2. *The set $\mathcal{S}$ is nonempty, i.e., $\mathcal{S} \neq \emptyset$.*

We consider the following problem formulation.

Problem 1. *Given an STL formula φ that specifies tasks in a multi-robot system with $\mathbf{N}$ robots, design a distributed algorithm to find the trajectory $\mathbf{y} = [\mathbf{y}_1^\top, \mathbf{y}_2^\top, \dots, \mathbf{y}_\mathbf{N}^\top]^\top : [0, \mathrm{th}(\varphi)] \to \mathcal{S}$ for each robot $i \in \{1, \dots, N\}$, by only communicating with neighbours $j \in \mathcal{N}_i$.*

V. STL Inclusion

This section presents the STL inclusion within our problem framework. First, we delve into including spatial constraints in Section V-A, followed by temporal inclusion in Section V-B.

A. Distributed Optimisation

The planning problem is solved in a distributed way where each robot maximises a local optimality criterion. All robots solve their local optimisation problem by communicating with their neighbours. For robot i, the constraints (5) are cast into the cost function F^i as

$$F^i = \sum_{\alpha=1}^{r_i} \frac{1}{2} \max\left(0, {}_\alpha h_i\right)^2 + \sum_{\beta=1}^{s_i} \frac{1}{2} \max\left(0, {}_\beta h_{ij}\right)^2, \tag{6}$$

and the resulting optimisation problem takes the form

$$\min F^i \tag{7}$$

whose solution $\mathbf{x}_i^\star$ satisfies $F^i(\mathbf{x}_i^\star) = 0$. The robots solve their respective optimisation problem cooperatively in a distributed manner via inter-neighbour communication. This makes the problem distributed, as every interaction between robots is part of the communication graph. The optimisation problem will be used in the main algorithm presented in Section VI ALgorithm 1, where we detail the approach to solve (7). Given the nature of the optimisation problem, there is a trade off between robustness and optimisation performance: since $\mathbf{x}^\star$ converges to the boundaries imposed by the STL formula constraints making it vulnerable to potential perturbations. However, introducing a slack variable into the equation can enhance robustness, albeit at the cost of sacrificing completeness arguments.

Example 2. *Consider a system with 3 agents and the corresponding states $\{x_1, x_2, x_3\}$, and let the STL formula be: $\varphi = (\|x_1 - x_2\| > 5) \wedge (\|x_2 - x_3\| < 2)$; then, the functions F^i, for $i \in \{1, 2, 3\}$, are,*

$$\begin{aligned} F^1 &= \frac{1}{2} \max(0, 5 - \|x_1 - x_2\|)^2 \\ F^2 &= \frac{1}{2} \max(0, 5 - \|x_1 - x_2\|)^2 + \frac{1}{2} \max(0, \|x_2 - x_3\| - 2)^2 \\ F^3 &= \frac{1}{2} \max(0, \|x_2 - x_3\| - 2)^2. \end{aligned}$$

Note that the optimisation problem here only considers the spatial aspect and temporal inclusion is discussed below.

B. Validity Domain

We now introduce the concept of *validity domain*, a time interval associated with every predicate and defined for every path in the STL formula. This interval represents the time domain over which each predicate applies and is defined as follows:

Definition 2. *The* validity domain $\mathrm{vd}(\bar{\varphi})$ *of each path* $\bar{\varphi}$ *of an STL formula* φ*, is recursively defined as*

$$\mathrm{vd}(\bar{\varphi}) = \begin{cases} 0, & \text{if } \bar{\varphi} = \mu^h \\ \mathrm{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \neg\bar{\varphi}_1 \\ [a,b], & \text{if } \bar{\varphi} = \mathcal{G}_{[a,b]}\mu^h \\ a \oplus \mathrm{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \mathcal{G}_{[a,b]}\bar{\varphi}_1, \bar{\varphi}_1 \neq \mu^h \\ t^\star + T^\star \oplus \mathrm{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \mathcal{F}_{[a,b]}\bar{\varphi}_1. \end{cases} \tag{8}$$

where $T^\star \in [a,b]$ and $t^\star = \{t \mid (\bar{\varphi}, t) \models \mathcal{F}_{[a,b]}\bar{\varphi}_1\}$ is a variable with initial value 0 that changes over time and captures the last instance of satisfaction for the eventually operator. This is necessary due to the redundancy of the eventually operator. We must ascertain the specific instances where the eventually condition is met to ensure finding a feasible trajectory. Additionally, we need to maintain the history of $t^\star$ for nested temporal operators which require recursive satisfaction. The validity domain is determined for each path of an STL formula in a hierarchical manner, beginning at the root of the tree, and each path has a distinct validity domain. The number of leaf nodes in an STL formula is equivalent to the total number of validity domains. In Definition 2, we do not include the operators $\wedge$ and $\vee$ because they do not impose temporal constraints on the predicates and thus inherit the validity domains of their parent node. If there is no parent node, operators $\wedge$ and $\vee$ inherit the validity domains of their child node.

The validity domain is specially defined in the following cases. If a path contains only predicates, the validity domain of μ^h is equal to the time horizon of φ (i.e., $\mathrm{vd}(\mu^h) = \mathrm{th}(\varphi)$). Furthermore, if a path contains nested formulas with the same operators, such as $\bar{\varphi} = \mathcal{G}_{[1,10]}\mathcal{G}_{[0,2]}\mu^h$, then the validity domain of $\bar{\varphi}$ is equal to the time horizon of the path (i.e., $\mathrm{vd}(\bar{\varphi}) = \mathrm{th}(\bar{\varphi})$). For example, $\mathrm{vd}(\mathcal{G}_{[1,10]}\mathcal{G}_{[0,2]}\mu^h) = \mathrm{th}(\bar{\varphi}) = [1, 12]$.

Example 3. *Consider the following examples of the validity domain:*

- $\varphi_1 = \mathcal{G}_{[5,10]}\mu^h$*, then* $\mathrm{vd}(\varphi_1) = [5, 10]$*, which is the interval over which* μ^h *must hold.*
- $\varphi_2 = \mathcal{F}_{[5,10]}\mu^h$*, then* $t^\star = 0$ *,* $T^\star \in [5, 10]$ *and* $\mathrm{vd}(\mu^h) = 0$*. Therefore,* $\mathrm{vd}(\varphi_2) = T^\star \in [5, 10]$ *is the instance when* μ^h *is required to hold.*
- $\varphi_3 = \mathcal{F}_{[5,10]}\mathcal{G}_{[0,2]}\mu^h$*, then* $t^\star = 0$*,* $T^\star \in [5, 10]$*,* $\mathrm{vd}(\mathcal{G}_{[0,2]}\mu^h) = [0, 2]$*. Therefore,* $\mathrm{vd}(\varphi_3) = 0 + T^\star \oplus [0, 2] = [T^\star, T^\star + 2]$ *is the interval over which* μ^h *needs to hold such that* φ_3 *is satisfied.*
- $\varphi_4 = \mathcal{G}_{[2,10]}\mathcal{F}_{[0,5]}\mu^h$*, then* $a = 2$ *and* $\mathrm{vd}(\varphi_4) = 2 \oplus \mathrm{vd}(\mathcal{F}_{[0,5]}\mu^h) = 2 + 0 + T^\star$ *where* $T^\star \in [0, 5]$*. For example, if* $T^\star = 1$*, then* $\mathrm{vd}(\varphi_4) = 3$ *is the time instance when* μ^h *needs to hold. Once* $\mu^h = \top$*, then* $t^\star = T^\star$ *and the new* $\mathrm{vd}(\varphi_4) = 2 + 1 + T^\star$ *where* $T^\star \in [0, 5]$*.*
- $\varphi_5 = \mathcal{F}_{[0,100]}\mathcal{G}_{[5,10]}\mathcal{F}_{[0,1]}\mu^h$*, then* $t^\star = 0$*,* $T^\star \in [0, 100]$ *and* $\mathrm{vd}(\varphi_5) = T^\star + a \oplus \mathrm{vd}(\mathcal{F}_{[0,1]}\mu^h)$*. Suppose* $T^\star = 50$*, then* $\mathrm{vd}(\varphi_5) = 55 \oplus \mathrm{vd}(\mathcal{F}_{[0,1]}\mu^h)$ *and so on.*

Regarding the STL formula in equation (4)*, the validity domains are defined for the following paths:* $\mathcal{F}_{I_1}\mu^{h^1}$*,* $\mathcal{F}_{I_1}\mathcal{G}_{I_2}\mu^{h^2}$*,* $\mathcal{G}_{I_3}\mathcal{F}_{I_4}\mu^{h^3}$*, and* $\mathcal{G}_{I_5}\mu^{h^4}$*.*

We use the following notational convenience in this work: if a parent node of a leaf node of a path φ is an *eventually* operator we denote the corresponding validity domain by $\mathrm{vd}^F()$, and, if the parent node of a leaf node of a path φ is an *always* operator we denote the corresponding validity domain by $\mathrm{vd}^G()$. The notation $\mathrm{vd}^F()$ indicates that the predicate needs to hold at some instance in the said interval, and $\mathrm{vd}^G()$ indicates that the predicate needs to hold throughout the interval.

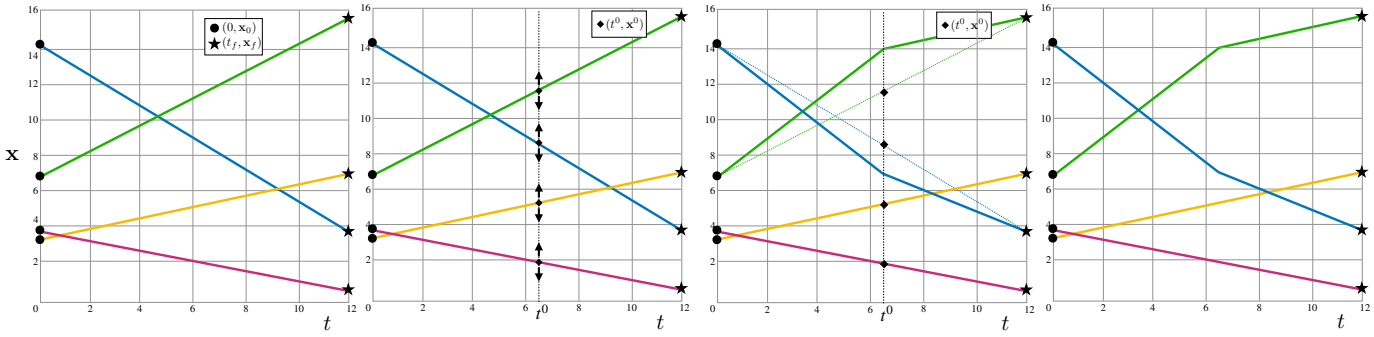
In the next Section, we present how to integrate the validity domain with the optimisation problem in (7), completing thus the spatial and temporal integration.

VI. Main Results

In this section, we present the algorithm for generating continuous trajectories that meet the requirements of a given Signal Temporal Logic (STL) formula φ. The algorithm is executed by the robots offline in a distributed manner, in the sense that they only communicate with their neighbouring robots. The algorithm builds a tree $\mathcal{T}_i = \{\mathcal{V}_i, \mathcal{E}_i\}$ for robot i where $\mathcal{V}_i$ is the vertex set and $\mathcal{E}_i$ is the edge set. Each vertex $z \in \mathbb{R}_+ \times \mathbb{R}^{n_i}$ is sampled from a space-time plane.

In what follows, we give a high-level description of the algorithm. The general idea is to start with an initial trajectory that spans the time horizon of the formulas $\mathrm{th}(\varphi)$, then repeatedly sample random points along the trajectory and use gradient-based techniques to find solutions that satisfy the specification at these points. More specifically, the algorithm begins by connecting the initial and final points $z_0^i = \{0, \mathbf{x}_0^i\}$ and $z_f^i = \{t_f^i, \mathbf{x}_f^i\}$ with a single edge $\mathcal{E}_i^0 = (z_0^i, z_f^i)$. The algorithm then randomly selects a time instant $t^0 \in [0, \mathrm{tf}(\varphi)]$ and uses linear interpolation to determine the states of each robot at that time, denoted by $\mathbf{x}^0$. The robots then solve the distributed optimisation problem (7) to find new positions $\mathbf{x}^\star$ that meet the specification at time t^0. The algorithm then repeats this process at a user-specified time density, updating the trajectories as necessary. The result is a trajectory that asymptotically improves the task satisfaction of the STL formula.

Example 4. *Before we get into the technical details, let us consider an example of 4 agents, represented by the colours* blue, green, yellow *and* magenta*, to illustrate the procedure. Suppose, at a specific instance in time, say* t^0*, the STL formula requires agent 1 (blue) and agent 2 (green) to be more than 6 units apart and agent 3 (yellow) and agent 4 (magenta) to*



(a) Initial trajectory (b) A random time instance t^0 (c) Optimal solution $\mathbf{x}^\star$ (d) Shaped trajectory

Figure 3: Illustration of the proposed algorithm

be closer than 6 units i.e., for $\epsilon > 0$,

$$G_{[t^0-\epsilon,t^0+\epsilon]}\Big(\big((\textit{blue and green are farther than 6 units apart})\wedge (\textit{yellow and magenta are closer than 6 units})\big)$$

We begin the process by connecting the initial and final points z_0^i *and* z_f^i *with an initial trajectory for all agents, as shown in Figure 3a. Each agent's vertex set is* $\mathcal{V}_i$ *and consists of the start and end points denoted by* z_0^i *and* z_f^i *respectively, while its edge set is* $\mathcal{E}_i$ *which contains only one edge connecting the start and end points. The initial trajectory of agent* i *is the tree* $\mathcal{T}_i = \{\mathcal{V}_i, \mathcal{E}_i\}$. *From the initial line trajectory, the algorithm randomly selects a point at time instance* t^0 *from the entire time domain and use linear interpolation to determine the state of each agent at that time. The agents solve* (7) *using the initial position* $\mathbf{x}^0$ *to find new position* $\mathbf{x}^\star$, *as seen in Figure 3b. As shown in Figure 3c, the distributed optimisation problem* (7) *is solved, resulting in a solution* $\mathbf{x}^\star$, *in which agent 1 and agent 2 are positioned so that they are more than 6 units apart and agent 3 and agent 4 remain undisturbed. The latter is the result of using functions of the form* $1/2\max(0, h_{ij})^2$, *and since agent 3 and agent 4 already satisfy the requirements, i.e.,* $h_{ij} < 0$, *the function is valued 0. The newly determined positions of agents 1 and 2 are added to the tree, allowing the trajectory to be shaped to meet the requirements. The updated trajectory can be seen in Figure 3d. This process of randomly selecting a point in time, determining the state of the agents and updating their positions is repeated for a user-defined number of times L, to ensure that the trajectory satisfies the STL formula* φ *throughout the time horizon.*

A. The overall algorithm

Here, we provide the main algorithm used to solve the problem at hand. The algorithm is called `MAPS`^{2} (short for '**m**ulti-robot **a**nytime motion **p**lanning under **s**ignal temporal logic **s**pecifications') and consists of the following functions: a function called `GradientDescent()` that addresses equation (7), a function called `SatisfactionVariable()` which calculates the satisfaction variables discussed in Section III-B, and a function called `ValidityDomain()` which calculates the intervals during which a predicate function is active. The algorithm is executed independently by each robot.

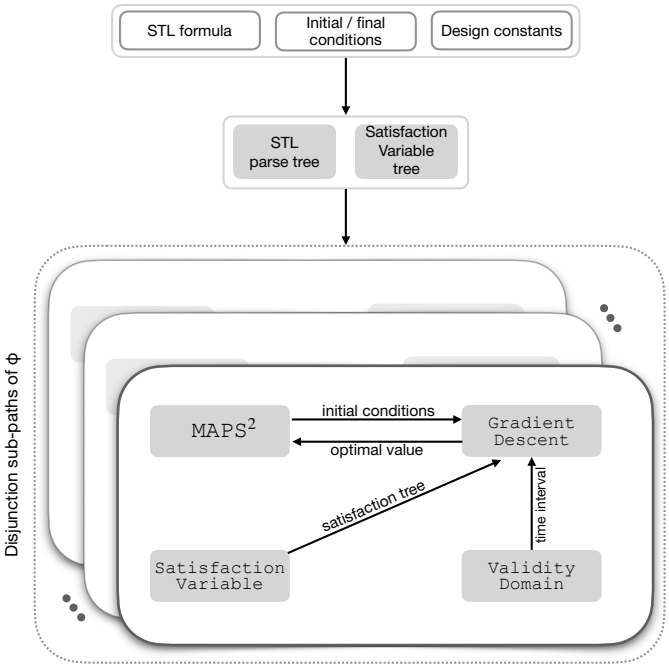


Figure 4: Architecture of the provided algorithm

The architecture of the algorithm is depicted in Figure 4 and proceeds as follows: first, the algorithm starts with an STL formula φ, along with the initial and final conditions. The initial conditions $z_0^i = \{t_0^i, \mathbf{x}_0^i\}$ depend on the robot's initial position and time. The final condition is chosen to be $z_f^i = \{\text{th}(\varphi) + \epsilon, \mathbf{x}_f^i\}$ where $\epsilon > 0$ and $\mathbf{x}_f^i \in \mathbb{R}^{n_i}$ is a random vector. This allows the algorithm to enforce STL tasks at a time instance $\text{th}(\varphi)$. Additionally, all robots initialise a random seed, and determine their neighbours based on the coupled constraints. The algorithm requires a maximum number of nodes, step size and stopping criterion for the optimisation problem.

1) `MAPS`^{2}: The algorithm is presented in Algorithm 1; it starts with an initial trajectory connecting z_0^i and z_0^f (see lines 1-3) and takes a random seed as input. Such a seed allows all robots to pick the same random number over

the time horizon of the formula. It continues by repeatedly sampling a time point, interpolating states, using gradient descent to find a satisfactory solution, and expanding the tree with new vertices until the total number of vertices L is reached, see lines 5-14. In line 7, the `SearchSort()`

Algorithm 1: MAPS[2]

Input: Initial condition $z_0^i = \{t_0^i, \mathbf{x}_0^i\}$, Final condition $z_{\text{f}}^i = \{t_{\text{f}}^i, \mathbf{x}_{\text{f}}^i\}$, Maximum number of nodes L, random seed, step size δ, stopping criterion η

Output: $\mathcal{T}_i$

1 $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z_0^i \cup z_{\text{f}}^i$;
2 $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z_0^i, z_{\text{f}}^i\}$;
3 $\mathcal{T}_i \leftarrow \{\mathcal{V}_i, \mathcal{E}_i\}$;
4 $j \leftarrow 0$;
5 **while** $j \leq L$ *and* $\tau(\textbf{\textit{root}}) \neq +1$ **do**
6 $\quad t^0 \leftarrow$ generate random number in $[t_0^i, t_{\text{f}}^i]$;
7 $\quad$ index $\leftarrow$ `SearchSort(`$\mathcal{V}_i,\ t^0$`)` ;
8 $\quad z_{\text{inter}}^i \leftarrow$ `Interpolate(`$\mathcal{V}_i, index$`)` ;
9 $\quad z_{\text{opt}}^i, \tau \leftarrow$ `GradientDescent(`$z_{inter}^i,\ \delta,\ L',\ \eta$`)` ;
10 $\quad \mathcal{V}_i \leftarrow \mathcal{V}_i \cup z_{\text{opt}}^i$;
11 $\quad \mathcal{E}_i \leftarrow \mathcal{E}_i \setminus \{z_{\text{index}}^i,\ z_{\text{index+1}}^i\}$;
12 $\quad \mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z_{\text{index}}^i,\ z_{\text{opt}}^i\}$;
13 $\quad \mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z_{\text{opt}}^i,\ z_{\text{index+1}}^i\}$;
14 $\quad \mathcal{T}_i \leftarrow \{\mathcal{V}_i, \mathcal{E}_i\}$;
15 $\quad j \leftarrow j + 1$;
16 $\quad$ **if** $j = L$ **then** $j \leftarrow 0,$ and $\forall \mathcal{F}, \tau(\mathcal{F}) = -1$;

function separates the vertices $\mathcal{V}_i$ into two sets based on their time values: one set with time values lower than t^0 (the vertex with the highest time in this set is indexed with 'index'), and another with values greater than t^0 (the vertex with the lowest time in this set is indexed with 'index + 1'). The corresponding vertices are $z_{\text{index}}^i = \{t_{\text{index}}^i, \mathbf{x}_{\text{index}}^i\}$ and $z_{\text{index+1}}^i = \{t_{\text{index+1}}^i, \mathbf{x}_{\text{index+1}}^i\}$. Then, the algorithm linearly interpolates in line 8 via the function `Interpolate()` to obtain the vertex $z_{\text{inter}}^i = \{t^0, \mathbf{x}_{\text{inter}}^i\}$. This is obtained by solving for $\mathbf{x}_{\text{inter}}^i$ element-wise as the solution of

$$\mathbf{x}_{\text{inter}}^i = \Big(\frac{\mathbf{x}_{\text{index+1}}^i - \mathbf{x}_{\text{index}}^i}{t_{\text{index+1}}^i - t_{\text{index}}^i}\Big)(t^0 - t_{\text{index}}^i) + \mathbf{x}_{\text{index}}^i.$$

The vertex z_{inter}^i is the initial condition to solve the optimisation problem (7); and once a solution z_{opt}^i is obtained, it is added to the vertex set $\mathcal{V}_i$ in line 10. The edge set $\mathcal{E}_i$ is reorganised to include z_{opt}^i in lines 11-13. Additionally, as a safeguard, if no solution is found after L iterations , line 16 resets the satisfaction variable of all eventually operators to -1 and begins the search again.

2) `GradientDescent`: The function is presented in Function 2, and as the name suggests, `GradientDescent()` computes the optimal value, z_{opt}, by solving the problem presented in equation (7). This allows the robots to compute vertices that locally satisfy the STL formula. In lines 17-21, we implement the standard gradient descent algorithm with a step size of δ and a stopping criterion of η, as described in Algorithm 9.3 of

Function 2: GradientDescent

Input: $z_{\text{inter}}^i = \{t^0, \mathbf{x}_{\text{inter}}^i\}$, step size δ, maximum iterations L', stopping criterion η

Output: z_{opt}^i, τ

1 Receive neighbour states $\mathbf{x}_{\text{neigh}}^j$ for all $j \in \mathcal{N}_i$;
2 **forall** $\bar{\varphi}$ *in* φ **do**
3 $\quad \text{vd}_{ij}(\bar{\varphi})$'s $\leftarrow$ `ValidityDomain` $(\bar{\varphi}, t^\star, t^0)$;
4 $k \leftarrow 0$;
5 $\lambda_{ij} = 0,\ \forall j$;
6 **case** $t^0 \in \text{vd}_{ij}^F(\bar{\varphi})$ **do**
7 $\quad \lambda_{ij} = 1$;
8 **case** $t^0 \in \text{vd}_{ij}^G(\bar{\varphi})$ **do**
9 $\quad \lambda_{ij} = 1$;
10 **case** $t^0 \in \bigcap_k \text{vd}_{ik}^F(\bar{\varphi})$ **do**
11 $\quad \begin{cases} \lambda_{ij} & = 1 \text{ for any one } j = k \\ \lambda_{ij} & = 0 \text{ otherwise} \end{cases}$
12 **case** $t^0 \in \bigcap_k \text{vd}_{ik}^G(\bar{\varphi})$ **do**
13 $\quad \lambda_{ik} = 1$ for all k;
14 $F^i = \sum_j \lambda_{ij} \max(0, h_{ij})^2$;
15 $\nabla F^i \leftarrow$ `GradientComputation(`$\mathbf{x}_{inter}^i, \mathbf{x}_{neigh}^j$`)` ;
16 $\Delta \mathbf{x}^i = -\nabla F^i$;
17 **while** $F^i \geq \eta$ **do**
18 $\quad \mathbf{x}^i := \mathbf{x}^i + \delta \Delta \mathbf{x}^i$;
19 $\quad$ Receive neighbour states $\mathbf{x}^j$ for all $j \in \mathcal{N}_i$;
20 $\quad \nabla F^i \leftarrow$ `GradientComputation(`$\mathbf{x}^i, \mathbf{x}^j$`)` ;
21 $\quad \Delta \mathbf{x}^i = -\nabla F^i$;
22 $\quad k \leftarrow k + 1$;
23 $\quad$ **if** $k > L'$ **then break**;
24 $z_{\text{opt}}^i = \{t^0, \mathbf{x}^i\}$;
25 **forall** $\bar{\varphi}$ *in* φ **do**
26 $\quad$ **if** $t^0 \in \text{vd}_{ij}(\bar{\varphi})$ **then**
27 $\quad\quad$ **if** $F^i(\mathbf{x}^i) \geq 0$ **then**
28 $\quad\quad\quad$ **node = leaf**$(\bar{\varphi})$;
29 $\quad\quad\quad \tau(\textbf{leaf}) = +1$;
30 $\quad\quad\quad$ **while** ***node*** $\neq$ ***root***$(\bar{\varphi})$ **do**
31 $\quad\quad\quad\quad$ **node = parent(node)**;
32 $\quad\quad\quad\quad \tau(\textbf{node}), t^\star \leftarrow$ `SatisfactionVariable` **(node**,z_{opt}^i**)**;
33 $\quad\quad$ **else** reset $\tau(\bar{\varphi}), t^\star$;
34 **return** $z_{\text{opt}}^i, \tau(\varphi)$

[25]. To evaluate the gradient, which depends on the states of the neighbouring robots, each robot communicates with its neighbours, as demonstrated in lines 1 and 19. This is the only instance of communication in the algorithm. In line 20, the function `GradientComputation()` computes the gradient, either analytically or numerically. Once z_{opt} is determined, the satisfaction variables are updated in Function 3. An additional stopping criterion is implemented in line 23 in case the problem does not converge when there are conflicting predicates at a specific time instance. This occurs,

for example, if $\varphi = \mathcal{F}_{[0,5]}\mathcal{G}_{[0,5]}\mu_1 \wedge \mathcal{G}_{[5,10]}\mu_2$, and there is a conflict between μ_1 and μ_2. In such cases, it becomes necessary for μ_1 to be true exclusively within the interval $[0, 5]$[s], and for μ_2 to hold exclusively within the interval $[5, 10]$[s].

Based on the validity domain, the algorithm determines which predicate functions are active in (6) at every sampled time instance. The Function `ValidityDomain()` in line 3 calculates the validity domains based on Definition 2. Among the set of predicate functions $\{h_{ij} | \forall j \in \mathcal{N}_i\}$ associated with a robot, a binary variable $\lambda_{ij} \in \{0, 1\}$ is assigned to determine whether a predicate function is active or not. It is set to 1 if the predicate is active and 0 otherwise. We distinguish three cases: if the sampled point belongs to the validity domain of a single *eventually* operator and/or a single *always* operator, $\lambda_{ij} = 1$. If the sampled point belongs to the validity domain of multiple *eventually* operators, we activate only one of them, that is, $\lambda_{ij} = 1$ only for one of them. This avoids enforcing conflicting predicates as it is can happen that multiple *eventually* operators may not be satisfied at the same time instance; see lines 6-13.

In lines 25-33, the algorithm updates the satisfaction variable of all paths in the STL formula that impose restrictions on agent i's states. The algorithm goes bottom-up, starting from the **leaf** node to the **root** node. First, it determines if z^i_{opt} is the desired minimum in line 27, and in lines 28-32, the algorithm updates the satisfaction variable of all nodes in the path $\bar{\varphi}$ through the function `SatisfactionVariable()`. If z^i_{opt} is not the desired minimum, then all the satisfaction variables of the path $\bar{\varphi}$ are reset to -1 in line 34. This could result from conflicting predicates at the same time instance.

Function 3: SatisfactionVariable

Input: $\bar{\varphi},\ z^i_{\text{opt}} = \{t^0, \mathbf{x}^i\}$
Output: $\tau, t^\star$

1 **case** $\mathcal{F}_I$ **do**
2 $\quad \tau(\mathcal{F}_I) = +1$;
3 $\quad t^\star = t^0$;
4 $\quad$ **return** $\tau, t^\star$;
5 **case** $\mathcal{G}_I$ **do**
6 $\quad$ **if** $\mathbf{robust}(\mathcal{G}_I) \geq 0$ **then**
7 $\quad\quad \tau(\mathcal{G}_I) = +1$;
8 $\quad$ **return** $\tau, t^\star$
9 **case** $\wedge$ **do**
10 $\quad \tau(\wedge) = +1$;
11 $\quad$ **return** $\tau, t^\star$

3) `SatisfactionVariable`: This function, presented in Function 3, updates the *satisfaction variable tree*, τ. The aforementioned procedure decides if the satisfaction variable corresponding to each node listed is $+1$ (satisfied) or -1 (not yet satisfied). Considering the premise that the predicate is true, as indicated in line 27 of Function 2, we evaluate the satisfaction variable as follows:

- $\mathcal{F}_I$: The satisfaction variable of the eventually operator is updates along with the $t^\star$. This updated $t^\star$ is used to determine the new validity domains, see Example 3 for an illustration of this procedure.

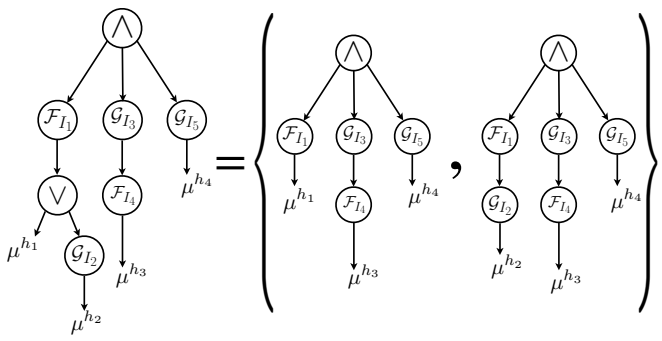


Figure 5: Disjunction representation for disjunctive components using STL parse tree.

- $\mathcal{G}_I$: Unlike the *eventually* operator, determining $\tau(\mathcal{G}_I)$ necessitates the computation of robustness over the entire validity domain of the operator. The function $\mathbf{robust}()$ uses the robust semantics of the STL presented in [2]. Particularly, it samples a user-defined number of points in the interval $\text{vd}^G_{ij}()$ and computes $\inf_{t \in \text{vd}^G_{ij}} h_{ij}(\mathbf{x}^i(t))$. If the robustness is non-negative, indicating satisfaction of the task, the value of $\tau(\mathcal{G}_I)$ is updated to $+1$.
- $\wedge$: This set node returns the satisfaction variable as $+1$ since it does not impose spatial or temporal restrictions.

B. Disjunctions

In our approach, we handle disjunctions as follows. Given an STL formula of the form $\varphi = \bigvee_{i \in \{1,\ldots,K\}} \phi_i$ i.e. $\varphi = \vee(\phi_1, \phi_2, \ldots, \phi_K)$, we split it into K STL formulae and individually solve the planning problem for all $\varphi = \phi_1$, $\varphi = \phi_2$, $\ldots$, $\varphi = \phi_K$. As an example, consider the STL formula (4); we branch into two STL formulae $\varphi = \mathcal{F}_{I_1}\mu_1 \wedge \mathcal{G}_{I_3}\mathcal{F}_{I_4}(\mu_3) \wedge \mathcal{G}_{I_5}(\mu_4)$ and $\varphi = \mathcal{F}_{I_1}\mathcal{G}_{I_2}(\mu_2) \wedge \mathcal{G}_{I_3}\mathcal{F}_{I_4}(\mu_3) \wedge \mathcal{G}_{I_5}(\mu_4)$, as illustrated in Figure 5. Condition $\tau(root) \neq +1$ on line 5 of Algorithm 1 terminates the search once any branch of disjunction is satisfied.

C. Analysis

In this section, we provide arguments for probabilistic completeness of the algorithm along the lines of [26]. Let a trajectory $\mathbf{y}$ be located on the boundary of the set $\mathcal{S}$, the satisfiable set, dividing $\mathcal{W}$ into a feasible set $\mathcal{S}$ and an infeasible set $\mathcal{W} \backslash \mathcal{S}$.

Starting with an initial linear trajectory in the augmented time-space domain, each uniformly sampled time point t^0 corresponds to a position $\mathbf{x}_{\text{inter}}$ either in $\mathcal{S}$ or $\mathcal{W} \backslash \mathcal{S}$. If $\mathbf{x}_{\text{inter}} \in \mathcal{S}$, we leave it unchanged as it meets the requirements. But if $\mathbf{x}_{\text{inter}} \notin \mathcal{S}$, we use gradient descent to reach a point on $\mathbf{y}$, since it lies on the boundary of the constraints' set.

Next, divide the trajectory $\mathbf{y} : [0, \text{th}(\varphi)] \to \mathcal{S}$ into $L + 1$ points $\mathbf{x}_i$, where $0 \leq i \leq L$ and $\mathbf{y}(\text{th}(\varphi)) = \mathbf{x}_f = \mathbf{x}_L$ by dividing the time duration into equal intervals of δ_t. Without loss of generality, assume that the points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are separated by δ_t in time. With $L\delta_t = \text{th}(\varphi)$, the probability of sampling a point in an interval of length δ_t can be calculated as $p = \frac{\delta_t}{\text{th}(\varphi)}$. If $\delta_t << \text{th}(\varphi)$, then $p < 1/2$. Denote the

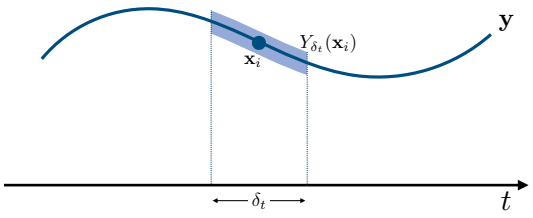


Figure 6: Illustration of $Y_{\delta_t}(\mathbf{x}_i)$

sequential covering class[1] of trajectory $\mathbf{y}$ as $Y_{\delta_t}(\mathbf{x}_i)$. The length of $Y_{\delta_t}(\mathbf{x}_i)$ is δ_t in the time domain and is centered at $\mathbf{x}_i$. See Figure 6 for reference. A trial is counted as successful if we sample a point t^0 within the interval $\delta_t/2$ on either side of $\mathbf{x}_i$, that is, within $Y_{\delta_t}(\mathbf{x}_i)$. If there are L successful trials, the entire trajectory $\mathbf{y}$ is covered, and the motion planning problem is solved. Consider k total samples, where $k \gg L$, and treat this as k Bernoulli trials with success probability p since each sample is independent with only two outcomes. We are now ready to state the following proposition.

Proposition 1. *Let a constant L and probability p such that $p < \frac{1}{2}$. Further, let k represent the number of samples taken by the MAPS^2 algorithm. Then, the probability that MAPS^2 fails to find a path after k samples is at most $\frac{(k-L)p}{(kp-L)^2}$.*

Proof. The probability of not having L successful trials after k samples can be expressed as:

$$\mathbf{P}[X_k \leq L] = \sum_{i=0}^{L-1} \binom{k}{i} p^i (1-p)^{k-i}$$

and according to [27], if $p < \frac{1}{2}$, we can upper bound this probability as:

$$\mathbf{P}[X_k \leq L] \leq \frac{(k-L)p}{(kp-L)^2}.$$

As p and L are fixed and independent of k, the expression $\frac{(k-L)p}{(kp-L)^2}$ approaches 0 with increasing k. Therefore, with uniform sampling, the algorithm MAPS^2 is probabilistically complete. □

The effectiveness of MAPS^2 has been demonstrated through its ability to locate a solution if it exists, which makes it probabilistically complete. Furthermore, every position $\mathbf{x}$ added to the tree is guaranteed to be in $\mathcal{S}$, affirming the soundness of the algorithm.

VII. Simulations

In this section, we present simulations of various scenarios encountered in a multi-robot system. Restrictions are imposed using an STL formula and MAPS^2 is utilised to create trajectories that comply with the STL formula. In the following we consider 4 agents, with $\delta = 0.1$, $\eta = 0.01$ and $L = L' = 100$. The simulations were run on an 8 core Intel® Core™ i7 1.9GHz CPU with 16GB RAM.

[1]Meaning $\mathbf{y} \subset \bigcup_{x=i}^{L} Y_{\delta_t}(\mathbf{x}_i)$

1) Collision avoidance: We begin with a fundamental requirement in multi-robot systems: avoiding collisions. In this scenario, it is assumed that all agents can communicate or sense each other's positions. The following STL formula is used to ensure collision avoidance in the interval 20[s] to 80[s]:

$$\varphi = \mathcal{G}_{[20,80]}(\|x_i - x_j\| \geq 1)$$

where $\{i,j\} \in \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}$. As depicted in Figure 7a, all four agents maintain a distance of at least 1 unit from each other during the interval $[20, 80]$[s]. The maximum computation time by any agent is 0.1143[s].

2) Rendezvous: The next scenario is rendezvous. We use the eventually operator to express this requirement. The STL formula specifies that agents 1 and 3 must approach each other within 1 distance unit during the interval $[40, 60]$[s] and similarly, agents 2 and 4 must meet at a minimum distance of 1 unit during the same interval. The STL formula is:

$$\varphi = \mathcal{F}_{[40,60]}(\|x_1 - x_3\| \leq 1 \wedge \|x_2 - x_4\| \leq 1).$$

As seen in Figure 7b, agents 1 and 3 and agents 2 and 4 approach each other within a distance of 1 unit during the specified interval. It's worth noting that the algorithm randomly selects the specific time $t^\star$ within the continuous interval $[40, 60]$[s] at which the satisfaction occurs. The maximum computation time by any agent is 0.0637[s].

3) Stability: The last task is that of stability, which is represented by the STL formula $\mathcal{F}_{[a_1,b_1]}\mathcal{G}_{[a_2,b_2]}\mu$. This formula requires that μ must always hold within the interval $[t^\star + a_2, t^\star + b_2]$, where $t^\star \in [a_1, b_1]$. This represents stability, as it requires μ to always hold within the interval $[t^\star + a_2, t^\star + b_2]$, despite any transients that may occur in the interval $[a_1, t^\star)$. Figure 7c presents a simulation of the following STL formula:

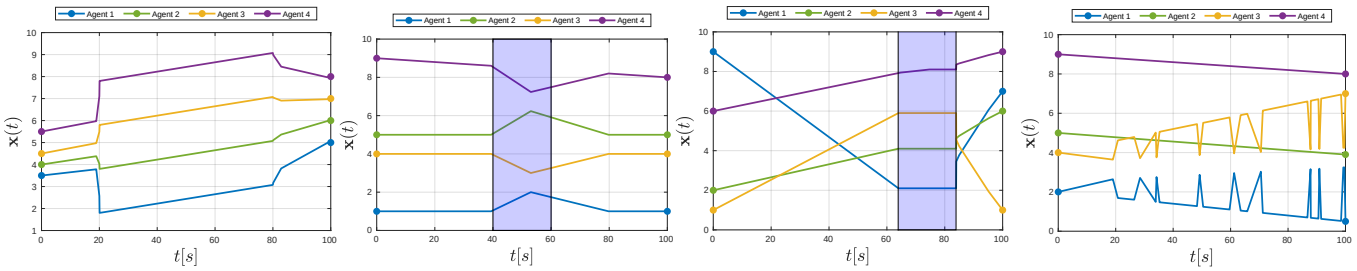
$$\varphi = \mathcal{F}_{[0,100]}\ \mathcal{G}_{[0,20]}\ \Big((1.9 \leq x_1 \leq 2.1) \wedge (3.9 \leq x_2 \leq 4.1) \wedge (5.9 \leq x_3 \leq 6.1) \wedge (7.9 \leq x_4 \leq 8.1)\Big)$$

where $t^\star = 63.97$[s]. The maximum computation time by any agent is 0.0211[s].

4) Recurring tasks: The next scenario is that of recurring tasks. This can be useful when an autonomous vehicle needs to repeatedly survey an area at regular intervals, a bipedal robot needs to plan periodic foot movements, or a ground robot needs to visit a charging station at specified intervals. The STL formula to express such requirements is given by $\mathcal{G}_{[a_1,b_1]}\mathcal{F}_{[a_2,b_2]}\mu$, which reads as 'beginning at a_1[s], μ must be satisfied at some point in the interval $[a_1 + a_2, a_1 + b_2]$[s] and this should be repeated every $[b_2 - a_2]$[s].' A simulation of the following task is shown in Figure 7d:

$$\varphi = \mathcal{G}_{[0,100]}\mathcal{F}_{[0,20]}(\|x_1 - x_3\| \leq 1).$$

Every 20[s], the condition $|x_1 - x_3| \leq 1$ is met. It's worth noting that the specific time $t^\star$ at which satisfaction occurs is randomly chosen by the algorithm. The maximum computation time by any agent is 0.2017[s].



(a) Collision avoidance (b) Rendezvous (c) Stability (d) Recurring tasks

Figure 7: Simulation results of MAPS^2 with four agents.

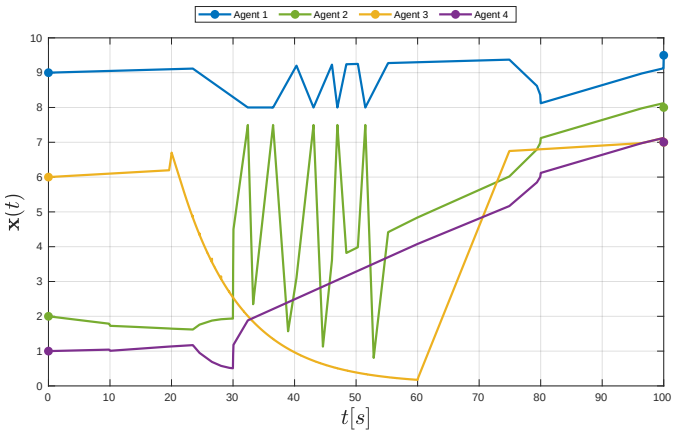


Figure 8: Overall case study

5) Overall case study: In this case study, we demonstrate the application of the aforementioned scenarios by setting up the following tasks:

- Agent 1 is always stays above 8 units.
- Agents 2 and 4 are required to satisfy the predicate $x_2^2 + x_4^2 \leq 2$ within the time interval $[10, 30]$[s].
- Agent 3 is required to track an exponential path within the time interval $[20, 60]$[s].
- Agent 2 is required to repeatedly visit Agent 1 and Agent 3 every 10s within the interval $[30, 50]$[s].
- Agent 1 is required to maintain at least 1 unit distance from the other three agents within the interval $[80, 100]$[s].

The STL formula of the above tasks is as follows:

$$\begin{aligned}\varphi \quad &= (x_1 \geq 8) \ \wedge \ \mathcal{G}_{[10,30]}(x_2^2 + x_4^2 \leq 2) \wedge \\ &\mathcal{G}_{[20,60]}(\|x_3 - 50\exp(-0.1t)\| \leq 0.05) \wedge \\ &\mathcal{G}_{[30,50]}\mathcal{F}_{[0,10]}\Big((\|x_2 - x_1\| \leq 0.5) \wedge (\|x_2 - x_3\| \leq 0.5)\Big) \wedge \\ &\mathcal{F}_{[79.9,80.1]}\mathcal{G}_{[0,20]}\Big((\|x_1 - x_2\| \geq 1) \wedge (\|x_1 - x_3\| \geq 1) \\ &\wedge (\|x_1 - x_4\| \geq 1)\Big)\end{aligned}$$

The parameter L was increased to 1000 and η was decreased to 0.001. In Figure 8, we show the resulting trajectories of each agent generated by MAPS^2 satisfying the above STL formula. The maximum computation time by any agent is 4.611[s].

Remark 1. *To guarantee completeness, our focus in this work is directed towards the planning problem, specifically the generation of trajectories that fulfil a specific criterion, rather than the mechanics of how the agent moves or the precise control techniques used to execute the trajectory. This approach leads to the production of non-smooth trajectories, as seen in the simulations. To address this, we can apply a smoothing procedure to the trajectories using B-Splines, taking into account the velocity and acceleration constraints of the robots, see [28]. Furthermore, to the best of our knowledge, there has been no prior study that tackles the distributed multi-robot STL planning problem under nonlinear, nonconvex coupled constraints; thus a comparison study is not in order.*

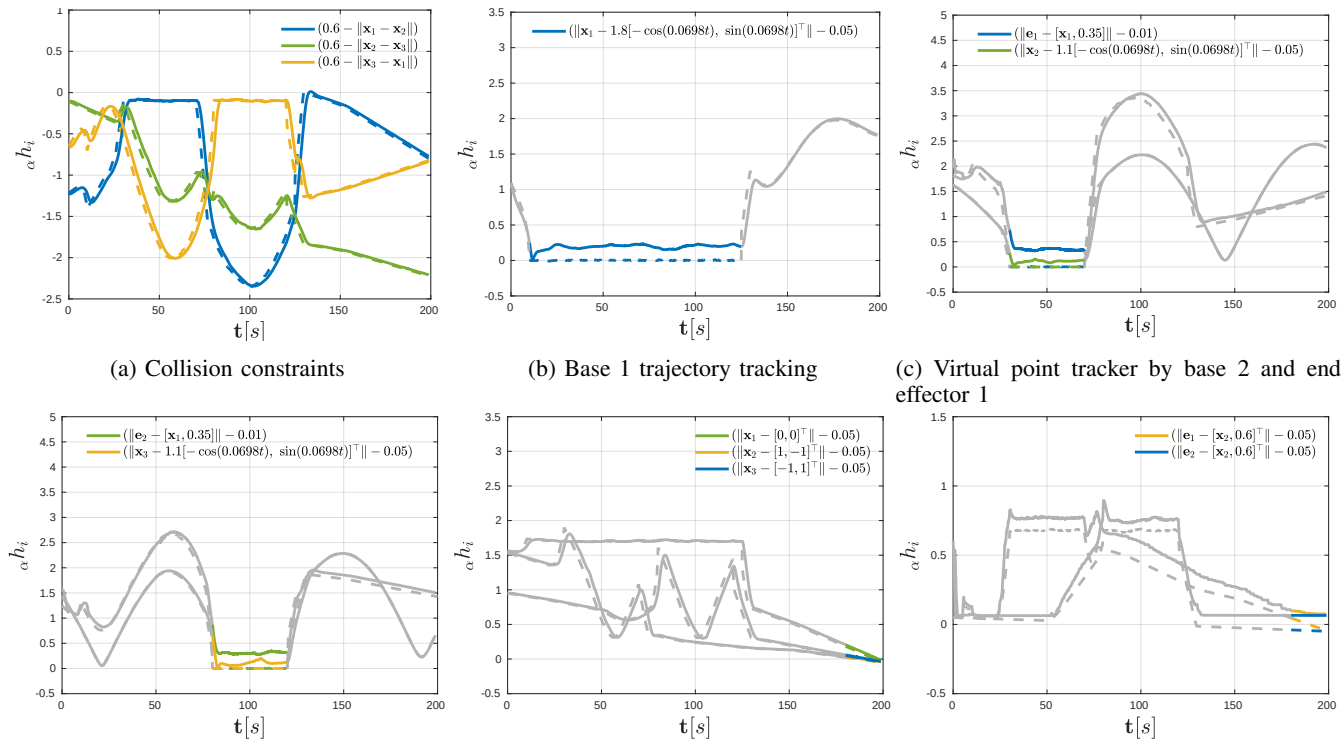
VIII. Experiments

We now present an experimental demonstration of the proposed algorithm. The multi-robot setup involves three robots, as shown in Figure 1, and consists of 3 mobile bases and two 6-DOF manipulator arms. The locations of the three bases are denoted as $\mathbf{x}_1 \in \mathbb{R}^2$, $\mathbf{x}_2 \in \mathbb{R}^2$, and $\mathbf{x}_3 \in \mathbb{R}^2$, respectively. Base 2 and base 3 are equipped with manipulator arms, whose end-effector positions are represented as $\mathbf{e}_1 \in \mathbb{R}^3$ and $\mathbf{e}_2 \in \mathbb{R}^3$, respectively.

The STL formula defining the tasks is the following,

$$\begin{aligned}&\varphi = \|\mathbf{x}_1 - \mathbf{x}_2\| \geq 0.6 \wedge \|\mathbf{x}_2 - \mathbf{x}_3\| \geq 0.6 \wedge \|\mathbf{x}_3 - \mathbf{x}_1\| \geq 0.6 \wedge \\ &\mathcal{G}_{[10,125]}\|\mathbf{x}_1 - 1.8[-\cos 0.0698t, \sin(0.0698t)]^\top\| \leq 0.05 \wedge \\ &\mathcal{G}_{[30,70]}\|\mathbf{e}_1 - [\mathbf{x}_1^\top, 0.35]^\top\| \leq 0.01 \wedge \\ &\mathcal{G}_{[30,70]}\|\mathbf{x}_2 - 1.1[-\cos 0.0698t, \sin(0.0698t)]^\top\| \leq 0.05 \wedge \\ &\mathcal{G}_{[80,120]}\|\mathbf{e}_2 - [\mathbf{x}_1^\top, 0.35]^\top\| \leq 0.01 \wedge \\ &\mathcal{G}_{[80,120]}\|\mathbf{x}_3 - 1.1[-\cos 0.0698t, \sin(0.0698t)]^\top\| \leq 0.05 \wedge \\ &\mathcal{F}_{[180,200]}\|\mathbf{x}_1 - [0,0]^\top\| \leq 0.05 \wedge \\ &\mathcal{F}_{[180,200]}\Big(\|\mathbf{x}_2 - [1,-1]\| \leq 0.05 \wedge \|\mathbf{e}_1 - [\mathbf{x}_2, 0.6]\| \leq 0.05\Big) \wedge \\ &\mathcal{F}_{[180,200]}\Big(\|\mathbf{x}_3 - [-1,1]\| \leq 0.05 \wedge \|\mathbf{e}_2 - [\mathbf{x}_3, 0.6]\| \leq 0.05\Big).\end{aligned}$$

The above task involves collision avoidance constraints that are always active given by the subformula $\bar{\varphi}_1 = (\|\mathbf{x}_1 - \mathbf{x}_2\| \geq 0.6) \wedge (\|\mathbf{x}_2 - \mathbf{x}_3\| \geq 0.6) \wedge (\|\mathbf{x}_3 - \mathbf{x}_1\| \geq 0.6)$. Next, in the duration $[10, 125]$[s], base 1 surveils the arena and follows a circular time varying trajectory given by the subformula $\bar{\varphi}_2 = (\mathcal{G}_{[10,125]}\|\mathbf{x}_1 - c_1(t)\| \leq 0.05)$ where $c_1(t)$ is the circular trajectory. In the duration $[30, 70]$[s],



(a) Collision constraints

(b) Base 1 trajectory tracking

(c) Virtual point tracker by base 2 and end effector 1

(d) Virtual point tracker by base 3 and end effector 2

(e) Desired final position of bases

(f) Desired final position of end effectors

Figure 9: Experimental verification of MAPS^2 with the setup in Figure 1.

end-effector 1 tracks a virtual point 0.35[m] over base 1 to simulate a pick-and-place task, given by the subformula $\bar{\varphi}_3 = \mathcal{G}_{[30,70]}\|\mathbf{e}_1 - [\mathbf{x}_1^\top, 0.35]^\top\| \leq 0.01 \wedge \mathcal{G}_{[30,70]}\|\mathbf{x}_2 - c_2(t)\| \leq 0.05$ where $c_2(t)$ is the circular trajectory. Similarly, in the duration $[80, 120]$[s], end-effector 2 takes over the task to track a virtual point 0.35[m] over base 1, given by the subformula $\bar{\varphi}_4 = \mathcal{G}_{[80,120]}\|\mathbf{e}_2 - [\mathbf{x}_1^\top, 0.35]^\top\| \leq 0.01 \wedge \mathcal{G}_{[80,120]}\|\mathbf{x}_3 - c_2(t)\| \leq 0.05$. Finally, eventually in the duration $[180, 200]$[s], the robots assume a final position given by the subformula $\bar{\varphi}_5 = \mathcal{F}_{[180,200]}\|\mathbf{x}_1 - [0,0]^\top\| \leq 0.05 \wedge \mathcal{F}_{[180,200]}\Big(\|\mathbf{x}_2 - [1,-1]\| \leq 0.05 \wedge \|\mathbf{e}_1 - [\mathbf{x}_2, 0.6]\| \leq 0.05\Big) \wedge \mathcal{F}_{[180,200]}\Big(\|\mathbf{x}_3 - [-1,1]\| \leq 0.05 \wedge \|\mathbf{e}_2 - [\mathbf{x}_3, 0.6]\| \leq 0.05\Big)$.

The results are shown in Figure 9, where the x-axis represents time in seconds, and the y-axis represents the predicate functions defined by (5). The dashed line in the plots represents the predicate functions of the trajectories obtained by solving the optimisation problem (7), while the solid line represents the predicate functions of the actual trajectories by the robots. In the context of (5), negative values indicate task satisfaction. However, due to the lack of an accurate model of the robots and the fact that the optimisation solution converges to the boundary of the constraints, the tracking is imperfect, and we observe slight violations of the formula by the robots in certain cases. Nonetheless, the trajectories generated by the algorithm do not violate the STL formula. The coloured lines represent the functions that lie within the validity domain of the formula. Figure 9a shows that the collision constraint imposed on all 3 bases is not violated, and they maintain a separation of at least 60 cm. In Figure 9b, base 1 tracks a circular trajectory in the interval $[10, 125]$ seconds. In Figures 9c and 9d, the end effectors mounted on top of bases 2 and 3 track a virtual point over the moving base 1 sequentially. In the last 20 seconds, the bases and end effectors move to their desired final positions, as seen in Figures 9e and 9f. The maximum computation time by any robot is 3.611[s]. Figure 10 shows front-view and side-view at different time instances during the experimental run[2].

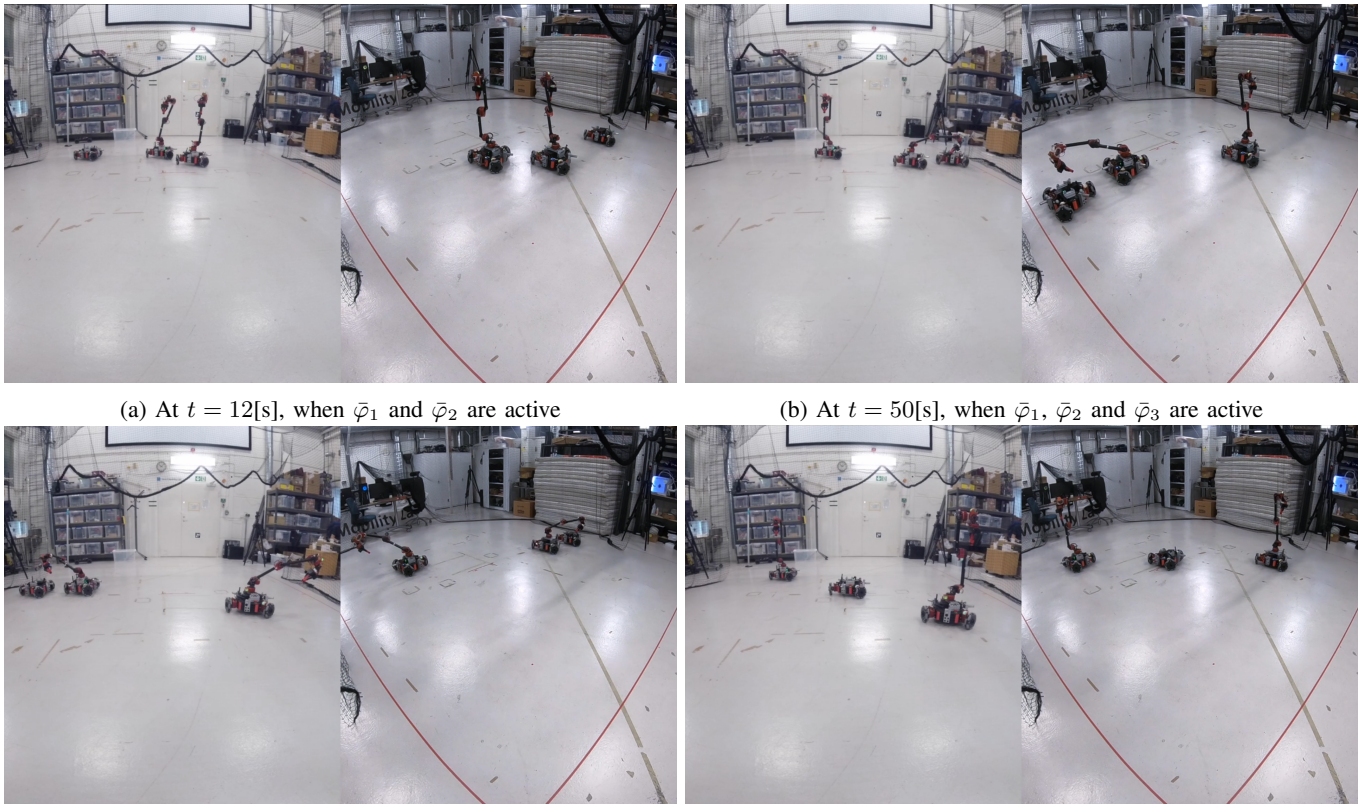
IX. Conclusion

This work proposed MAPS^2, a distributed planner that solves the multi-robot motion-planning problem subject to tasks encoded as STL constraints. By using the notion of validity domain and formulating the optimisation problem as shown in (7), MAPS^2 transforms the spatio-temporal problem into a spatial planning task, for which efficient optimisation algorithms already exist. Task satisfaction is probabilistically guaranteed in a distributed manner by presenting an optimisation problem that necessitates communication only between robots that share coupled constraints. Extensive simulations involving benchmark formulas and experiments involving varied tasks highlight the algorithms functionality. Future work involves incorporating dynamical constraints such as velocity and acceleration limits into the optimisation problem.

References

[1] L. Lamport, "What good is temporal logic?" *Information Processing 83, R. E. A. Mason, ed., Elsevier Publishers*, vol. 83, pp. 657–668, May

[2]The video of the experiments can be found here: https://youtu.be/YkuiPuOerMg



(a) At $t = 12$[s], when $\bar{\varphi}_1$ and $\bar{\varphi}_2$ are active

(b) At $t = 50$[s], when $\bar{\varphi}_1$, $\bar{\varphi}_2$ and $\bar{\varphi}_3$ are active

(c) At $t = 100$[s], when $\bar{\varphi}_1$, $\bar{\varphi}_2$ and $\bar{\varphi}_4$ are active

(d) At $t = 190$[s], when $\bar{\varphi}_1$ and $\bar{\varphi}_5$ are active

Figure 10: Front-view and side-view during experimental run with the setup in Figure 1.

1983. [Online]. Available: https://www.microsoft.com/en-us/research/publication/good-temporal-logic/

[2] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[3] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2021, pp. 432–449.

[4] M. Sewlia, C. K. Verginis, and D. V. Dimarogonas, "Cooperative sampling-based motion planning under signal temporal logic specifications," in *2023 American Control Conference (ACC)*, 2023, pp. 2697–2702.

[5] G. Yang, B. Vang, Z. Serlin, C. Belta, and R. Tron, "Sampling-based motion planning via control barrier functions," in *Proceedings of the 2019 3rd International Conference on Automation, Control and Robots*, 2019, pp. 22–29.

[6] A. I. M. Ayala, S. B. Andersson, and C. Belta, "Temporal logic motion planning in unknown environments," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 5279–5284.

[7] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 2689–2696.

[8] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 4817–4822.

[9] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000510980800455X

[10] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5319–5325.

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

[12] L. Lindemann and D. V. Dimarogonas, "Robust motion planning employing signal temporal logic," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 2950–2955.

[13] C. K. Verginis and D. V. Dimarogonas, "Timed abstractions for distributed cooperative manipulation," *Autonomous Robots*, vol. 42, pp. 781–799, 2018.

[14] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[15] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 115–140, 2019.

[16] 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.

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

[18] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3451–3458, 2022.

[19] L. Lindemann, C. K. Verginis, and D. V. Dimarogonas, "Prescribed performance control for signal temporal logic specifications," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE Press, 2017, p. 2997–3002. [Online]. Available: https://doi.org/10.1109/CDC.2017.8264095

[20] M. Charitidou and D. V. Dimarogonas, "Barrier function-based model predictive control under signal temporal logic specifications," in *2021 European Control Conference (ECC)*, 2021, pp. 734–739.

[21] F. Chen and D. V. Dimarogonas, "Funnel-based cooperative control of leader-follower multi-agent systems under signal temporal logic specifications," in *2022 European Control Conference (ECC)*, 2022, pp. 906–911.

[22] L. Lindemann and D. V. Dimarogonas, "Decentralized robust control of coupled multi-agent systems under local signal temporal logic tasks," in *2018 Annual American Control Conference (ACC)*, 2018, pp. 1567–1573.

[23] Y. Gilpin, V. Kurtz, and H. Lin, "A smooth robustness measure of signal

temporal logic for symbolic control," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 241–246, 2020.

[24] C. Madsen, P. Vaidyanathan, S. Sadraddini, C.-I. Vasile, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta, "Metrics for signal temporal logic formulae," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 1542–1547.

[25] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[26] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, "Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. x–xvi, 2019.

[27] W. Feller, *An Introduction to Probability Theory and Its Applications*. Wiley, January 1968, vol. 1. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{&}path=ASIN/0471257087

[28] D. Lapandić, C. K. Verginis, D. V. Dimarogonas, and B. Wahlberg, "Kinodynamic motion planning via funnel control for underactuated unmanned surface vehicles," *arXiv preprint arXiv:2308.00130*, 2023.