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

Journal Title
XX(X):1–17

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Mayank Sewlia[1], Christos K. Verginis[2] and Dimos V. Dimarogonas[1]

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. Meanwhile, works maximising robustness through optimisation-based methods often suffer from local minima, thus relaxing any completeness arguments due to the NP-hard nature of the problem. Endowed with probabilistic guarantees, MAPS^2 provides an autonomous 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.

1 Introduction

Autonomous robots can 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 by using 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 [Lamport, 1983]. 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 requirements on the behaviour of robots. Once these requirements are established, algorithms are developed to generate satisfying trajectories. Such is the focus of our work.

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 the 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 [Maler and Nickovic, 2004]. STL thus provides a natural and compact way to reason about a robot's motion since it operates in a continuously evolving space-time environment. Additionally, STL is accompanied by a robustness metric which allows us to determine the extent of satisfaction compared to only absolute satisfaction [Donzé, 2013].

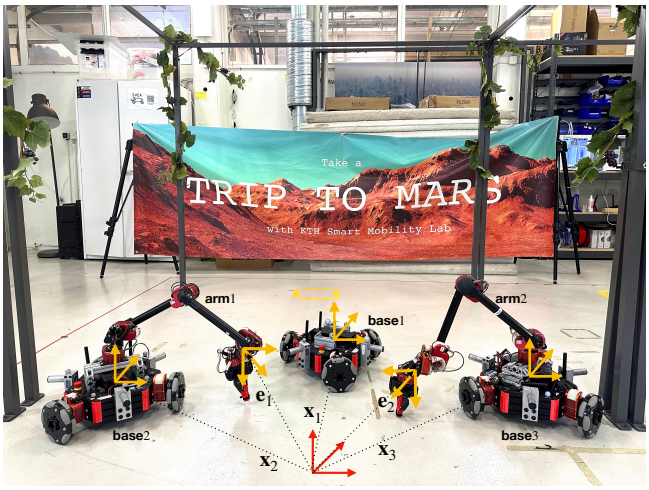


Figure 1. Experimental setup with three mobile bases and two 6-dof manipulators.

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

[1]Division of Decision and Control, School of EECS, KTH Royal Institute of Technology, Stockholm, Sweden.
[2]Division of Signals and Systems, Department of Electrical Engineering, Uppsala University, Uppsala, Sweden.
Acknowledgements: 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.

Corresponding author:
Mayank Sewlia, Division of Decision and Control, School of EECS, KTH Royal Institute of Technology, Stockholm, Sweden.
Email: sewlia@kth.se

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.

In this article, we propose **MAPS**^{2} - **M**ulti-Robot **A**utonomous Motion **P**lanning under **S**ignal Temporal Logic **S**pecifications - to address the multi-robot motion planning problem subject to coupled STL contraints. 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 ensures distributed trajectory generation to satisfy STL formulas that consist of coupled constraints for multiple robots. The article's contributions are summarised in the following attributes:

- The algorithm's effectiveness lies in its ability to distribute STL planning for multiple robots and in providing a mechanism to decouple the STL formula among robots, thereby facilitating the distribution of tasks.
- As opposed to previous work, it covers the entire STL formula and is not limited to a smaller fragment. It reduces conservatism by eliminating the need for approximations of max/min operators and samples in continuous time to avoid abstractions.
- 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 probabilistically complete, meaning that it will find such a trajectory if one exists.

In our prior study [Sewlia et al., 2023], 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 3.2 here). Additionally, we present experimental validation results and introduce a novel STL verification architecture.

The rest of the paper is organised as follows. Section 2 presents the related work. Section 3 presents preliminaries and formulates the problem of this work, Section 4 presents how we decompose the STL formula into temporal and spatial constraints, Section 5 presents the main algorithm MAPS^2 along with analyses of the algorithm, Section 6 presents simulations of various tasks encountered in a robotics problem, while Section 7 presents the experimental validation on a real multi-robot setup. Finally, Section 8 concludes the paper.

2 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 [Yang et al., 2019], 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 [Ayala et al., 2013, Bhatia et al., 2010, Vasile and Belta, 2013, Fainekos et al., 2009], 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 [Wolff et al., 2014], the resulting algorithm has exponential complexity, making it impractical for high-dimensional systems, complex specifications, and long duration tasks. To address this issue, [Kurtz and Lin, 2022] proposes a more efficient encoding for STL to reduce the exponential complexity in binary variables. Additionally, [Lindemann and Dimarogonas, 2017] 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 [Verginis and Dimarogonas, 2018, Kress-Gazit et al., 2009] 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 [Belta and Sadraddini, 2019, Raman et al., 2014, Sadraddini and Belta, 2015]. 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 [Sun et al., 2022] 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 [Kurtz and Lin, 2022]. 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 [Lindemann et al., 2017, Charitidou and Dimarogonas, 2021, Chen and Dimarogonas, 2022, Lindemann and Dimarogonas, 2018]. 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 [Gilpin et al., 2020]. 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. Our current focus is directed towards the planning problem, specifically the generation of trajectories that fulfil STL constraints, rather than the dynamical constraints or the precise control techniques used to execute the trajectory.

Notations: 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.

3 Preliminaries and Problem Formulation

In this section, we start by introducing STL and STL parse tree, followed by the problem formulation.

3.1 Signal Temporal Logic (STL)

Let $\mathbf{x} : \mathbb{R}_+ \to \mathbb{R}^n$ be a continuous-time signal. Signal temporal logic [Maler and Nickovic, 2004] 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:

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

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. [Madsen et al., 2018] . *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. 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 [Sadraddini and Belta, 2015], any STL formula can be written in positive normal form by moving the negation operator to the predicate.

3.2 STL Parse Tree

An STL parse tree is a tree representation of an STL formula [Sewlia et al., 2023]. 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 $\tau : \bar{\varphi} \to \{+1, -1\}$ 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

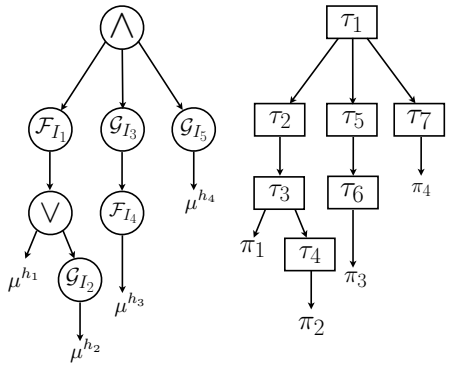


Figure 2. STL parse tree and satisfaction variable tree for the formula in (4).

$\tau = +1$ corresponding to the set node where the subpath of $\bar{\varphi}$ begins. Similarly, $\tau(root(\varphi)) = +1 \Leftrightarrow (\mathbf{x}, t) \models \varphi$ 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* $\tau_7 = +1 \implies (\mathbf{x}, t) \models \mathcal{G}_{I_5}(\mu^{h_4})$.

3.3 Problem Formulation

We consider a team of $\mathbf{N}$ robots, where each robot has state $\mathbf{x}_i \in \mathcal{W}_i \subset \mathbb{R}^{n_i}$, $i \in \{1, \dots, \mathbf{N}\}$ and n_i is the number of degrees of freedom of robot i. The overall state vector is then $\mathbf{x} := [\mathbf{x}_1^\top \ \cdots \ \mathbf{x}_\mathbf{N}^\top]^\top$ evolving in a workspace $\mathcal{W} = \mathcal{W}_1 \times \cdots \times \mathcal{W}_\mathbf{N}$ and we denote by $n = n_1 + \cdots + n_\mathbf{N}$ the number of degrees of freedom of the multi-robot system. We consider the STL formula of the form (1) with a total of K predicates,

$$\mu^{h^{(k)}} = \begin{cases} \top & h^{(k)}(\mathbf{x}(t)) \leq 0 \\ \bot & h^{(k)}(\mathbf{x}(t)) > 0 \end{cases}, \quad k = 1, \dots, K. \tag{5}$$

Before we present the problem statement, we will introduce the multi-robot STL notation and the communication structure of the multi-robot system. In this direction, define a support set for each predicate function $h^{(k)}(\mathbf{x})$ that captures all the robots upon which the predicate function imposes constraints. Define a projection matrix $E_i \in \mathbb{R}^{n \times n_i}$ such that $E_i^\top \mathbf{x} = \mathbf{x}_i$. The matrix E_i takes the form,

$$E_i = \begin{bmatrix} \mathbf{0}_{n_1} \\ \vdots \\ I_{n_i} \\ \vdots \\ \mathbf{0}_{n_\mathbf{N}} \end{bmatrix} \in \mathbb{R}^{n \times n_i},$$

i.e. $E_i v = \big[0, \dots, 0, v, 0 \dots, 0\big]^\top$ inserts any $v \in \mathbb{R}^{n_i}$ at the i-th position of the vector. The support set is then defined for each predicate function $h^{(k)}$ as,

$$\mathbf{S}_k := \Big\{ i \in \mathcal{I} \mid \exists \mathbf{x} \in \mathbb{R}^n, v \in \mathbb{R}^{n_i}, \epsilon > 0 \\ : h^{(k)}(\mathbf{x} + \epsilon E_i v) \neq h^{(k)}(\mathbf{x}) \Big\}.$$

The support set thus captures all robots i for which some perturbation confined to their own state $\mathbf{x}_i$ can change the value of the predicate function $h^{(k)}$. If a predicate function $h^{(k)}$ imposes constraints on the states of multiple robots, i.e., $|\mathbf{S}_k| \geq 2$, then we say that the predicate function is coupled.

Example 2. *For an STL formula* $\varphi = (\|x_1 - x_2\| > 5) \ \wedge (\|x_2 - x_3\| < 2)$, $h^{(1)} = 5 - \|x_1 - x_2\|$ *and* $h^{(2)} = \|x_2 - x_3\| - 2$. *Then* $\mathbf{S}_1 = \{1, 2\}$ *and* $\mathbf{S}_2 = \{2, 3\}$.

Let $K_c \leq K$ denote the number of coupled predicate functions, and let these be indexed as h_j^c, where $j \in \{1, \dots, K_c\}$. For each j, there exists an index $k_j \in \{1, \dots, K\}$ such that $h_j^c = h^{(k_j)}$; let $\mathbf{S}_{k_j} \subseteq \{1, \dots, \mathbf{N}\}$ denote the support set of h_j^c, i.e., the set of robots whose states appear in h_j^c. Each robot $i \in \mathbf{S}_{k_j}$ then uses a local copy $h_{i,j}^c := h_j^c$. This local copy is provided to the robots manually offline prior the start of the algorithm. The robots can also obtain the functions $h_{i,l}^d$ and $h_{i,j}^c$ that are coupled to their own states directly from φ. This could, for example, be done by defining a *regular expression*(regex) pattern and extracting predicate functions that involve the states $\mathbf{x}_i$ [Goyvaerts, 2016].

Next, let L_i be the number of independent predicate functions that involve only the states of robot i, with $\sum_{i=1}^{\mathbf{N}} L_i = K - K_c$. Such predicate functions are indexed as $h_{i,l}^d$, $i \in \{1, \dots, \mathbf{N}\}$ and $l \in \{1, \dots, L_i\}$.

Let Π_{k_j} be the projection that keeps the indicated state components of the robots in $\mathbf{S}_{k_j}$ only, i.e. $\Pi_{k_j}\mathbf{x} := \{E_m^\top \mathbf{x} \mid m \in \mathbf{S}_{k_j}\}$. The predicate function constraints for each robot i are then defined as follows

$$h_{i,l}^d(\mathbf{x}_i) \leq 0 \quad \text{and} \quad h_{i,j}^c(\Pi_{k_j}\mathbf{x}) \leq 0 \tag{6}$$

for all $i \in \{1, \dots, \mathbf{N}\}$, $l \in \{1, \dots, L_i\}$ and $j \in \{1, \dots, K_c | i \in \mathbf{S}_{k_j}\}$. The coupled predicate functions $h_{i,j}^c$ can reflect physical interactions between the robots if the constraint is such, for example, if $h_{i,j}^c$ specifies an obstacle avoidance constraint or an object handover task.

Example 3. *Consider the STL formula,*

$$\varphi = (\|x_1 - x_2\| < 1) \wedge (\|x_2 - x_3\| < 1) \\ (\|x_3 - x_4\| < 1) \wedge (\|x_4 - x_1\| < 1) \\ (\|x_1\| < 1) \wedge (\|x_2 - x_3 - x_4\| < 1) \\ (\|x_2\| < 1) \wedge (\|x_5\| < 1)$$

For the above STL formula, $K = 8$ and $K_c = 5$. The number of independent predicates for each robot are $L_1 = 1$, $L_2 = 1$, $L_3 = L_4 = 0$ and $L_5 = 1$. Table 1 depicts the labelled predicates for the above STL formula. The subscript i in the

Table 1. Labelled predicates for the example STL formula.

$\|x_1 - x_2\| - 1$	$h^c_{1,1},\ h^c_{2,1}$
$\|x_2 - x_3\| - 1$	$h^c_{2,2},\ h^c_{3,1}$
$\|x_3 - x_4\| - 1$	$h^c_{3,2},\ h^c_{4,1}$
$\|x_4 - x_1\| - 1$	$h^c_{4,2},\ h^c_{1,2}$
$\|x_1\| - 1$	$h^d_{1,1}$
$\|x_2 - x_3 - x_4\| - 1$	$h^c_{2,3},\ h^c_{3,3},\ h^c_{4,3}$
$\|x_2\| - 1$	$h^d_{2,1}$
$\|x_5\| - 1$	$h^d_{5,1}$

labels of the predicate functions $h^d_{i,l}$ and $h^c_{i,j}$ specifies the robot responsible for the predicate function.

Now we are ready to define the communication structure of the multi-robot system, which is dictated by a graph. Let the communication graph be given by $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where $\mathbf{V}$ is the set of vertices corresponding to the indices of the robots and $\mathbf{E}$ is the set of edges. In particular, the edge $(i, j) \in \mathbf{E}$ indicates that robot i can communicate with robot j as the subsequent assumption states.

Assumption 2. *If $(i, j) \in \mathbf{E}$, then robots i and j can continuously communicate with each other.*

We further assume that, every coupled predicate function induces an edge, ensuring that all state variables needed to compute the predicate function are locally available for each robot.

Assumption 3. *If there exists $k \in \{1, \dots, K\}$ such that $i, j \in \mathbf{S}_k$, for $i \neq j$, then $(i, j) \in \mathbf{E}$.*

Note that the aforementioned assumption implies that the graph is undirected i.e., $(i, j) \in \mathbf{E}$ implies $(j, i) \in \mathbf{E}$. Additionally, based on $\mathbf{G}$, the neighbourhood set $\mathcal{N}_i$ of a robot i is defined as $\mathcal{N}_i = \{j \in \mathbf{V} \mid (i, j) \in \mathbf{E}\}$. We also assume that $\mathbf{G}$ is static, meaning that no new vertices are added and no edges are created or deleted. With the above assumptions, we are ready to define the distributed information flow:

Definition 2. *An algorithm is called distributed, if it can be executed individually by each robot i (a local version of the algorithm) by using only information from its neighbours $\mathcal{N}_i$.*

This definition of distributed algorithm does not allow for any global information sharing among robots that are not neighbours with each other, thus a central computer cannot evaluate an STL formula. For example, consider the STL formula:

$$\varphi = \Big(\|\mathbf{x}_1 - \mathbf{x}_2\| \leq 1\Big) \wedge \Big(\|\mathbf{x}_2 - \mathbf{x}_3\| \leq 1\Big)$$

where $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ are the states of robot 1, 2, and 3 respectively. Then, Assumption 3 allows for a communication link between robot 1 and robot 2, and between robot 2 and robot 3. A distributed algorithm, in the sense of our work, does not allow for communication between robot 1 and robot 3.
We are now ready to formally state the problem addressed in this paper.

Problem 1. *Given an STL formula φ that specifies tasks on a multi-robot system with $\mathbf{N}$ robots, design a distributed algorithm to find continuous time-varying trajectories $\mathbf{y}_i : [0, \mathrm{th}(\varphi)] \to \mathcal{W}_i$, starting at an initial configuration $\mathbf{y}_i(0) = \mathbf{x}_i(0)$, such that $(\mathbf{y}, t) \models \varphi,\ \forall t \in [0, \mathrm{th}(\varphi)]$ with $\mathbf{y} := [\mathbf{y}_1^\top, \mathbf{y}_2^\top, \dots, \mathbf{y}_\mathbf{N}^\top]^\top$.*

It should be noted that we do not currently address the closed-loop stability of the underlying multi-robot system. Instead, we focus on the trajectory generation aspect and rely on existing low-level control approaches to track the generated trajectories. For more information, see Remark 3. The above problem is addressed assuming that at least one such solution exists. This will help us provide probabilistic completeness guarantees later on. Formally, we state the following assumption:

Assumption 4. *There exists at least one $\mathbf{y}$ such that $(\mathbf{y}, t) \models \varphi$.*

4 STL Formula Decomposition

In this section, we present how to retrieve spatial and temporal constraints from a given STL formula φ.

4.1 Spatial Constraints

In Section 3.3, we provisioned each predicate function $h^{(k)}(\mathbf{x})$ appearing in the STL formula φ over the complete multi-robot system to the corresponding robot i, denoting them as $h^d_{i,l}$ and h^c_{ij}, depending om whether the robot is responsible for an independent or a coupled task.

For robot i, cast the constraints (6) into the cost function F^i as

$$F^i := \sum_{l=1}^{L_i} \frac{1}{2} \max\left(0, h^d_{i,l}\right)^2 + \sum_{\substack{j \in \{1,\dots,K_c\} \\ : i \in \mathbf{S}_j}} \frac{1}{2} \max\left(0, h^c_{i,j}\right)^2 \tag{7}$$

Observe that $F^i : \mathcal{W} \to \mathbb{R}_+$ and $F^i = 0$ if and only if all the constraints in (6) are satisfied. Then, enforcing conditions (6) is equivalent to finding $\mathbf{x}_i$ for a given $\mathbf{x}_j$ $(j \in \mathcal{N}_i)$ such that $F^i = 0$. This problem can be posed as,

$$\min_{\mathbf{x}_i \in \mathcal{W}_i} F^i \tag{8}$$

whose solution $\mathbf{x}_i^\star$ satisfies $F^i(\mathbf{x}^\star) = 0$. In the cost function (7), to reduce computational costs, we only minimise $h^{(k)}(\mathbf{x}(t))$ when $h^{(k)}(\mathbf{x}(t)) > 0$ while leaving $h^{(k)}(\mathbf{x}(t)) \leq 0$ unchanged. This leads us to minimise: $F^i = \max(0, h^{(k)}(\mathbf{x}(t)))$, which results in $F^i = h^{(k)}(\mathbf{x}(t))$ when $h^{(k)}(\mathbf{x}(t)) > 0$ and $F^i = 0$ when $h^{(k)}(\mathbf{x}(t)) \leq 0$. Additionally, squaring the function penalises larger errors more than smaller ones. Other cost functions that enforce $h^{(k)}(\mathbf{x}(t)) \leq 0$ within the validity domain can also be considered. For example, the cost function $F^i := \sum_{l=1}^{L_i} {h^d_{i,l}}^2 + \sum_j {h^c_{i,j}}^2, j \in \{1, \dots, K_c\} : i \in \mathbf{S}_j$ could also be used. However, it was not our first choice, as we aimed to minimise $h^{(k)}(\mathbf{x}(t))$ only when $h^{(k)}(\mathbf{x}(t)) > 0$, whereas this cost function would attempt to minimise $h^{(k)}(\mathbf{x}(t))$ regardless of its sign. Additionally, our formulation is general and works for any type of STL formula as any type of objectives can be encoded in the STL formula, from which,

we can extract the predicate functions and minimise the function F^i.

The solution for finding the global minimum of a nonconvex function is a subject of extensive research. We argue that employing gradient descent with random initialisations is adequate for addressing this problem, particularly since the initialisations are sampled from a compact set, $\mathcal{W}_i$. Furthermore, using the knowledge that the minimum of the function, $F^i(\mathbf{x}) = 0$, acts as a stopping criterion and facilitates the attainment of the desired solution. We direct readers to the seminal work in [Nedic and Ozdaglar, 2009], which presents a distributed gradient descent algorithm for multi-agent systems. Additionally, under certain assumptions, [Daneshmand et al., 2020] demonstrates that gradient descent with a constant step size avoids entrapment at saddle points. Gradient descent is also shown to efficiently manage most reach-avoid constraints without the need for re-initialisation, given that such constraints are expressible using norms. For our application of gradient descent, we utilise the function presented in Function 1. Function 1 implements the gradient descent

Function 1: DistributedOptimisation

Input: $\mathbf{x}_i$, step size δ, maximum iterations L', activation variables λ_{ij}, $k \leftarrow 0$
Output: $\mathbf{x}_i^\star$

1 Receive neighbour states $\mathbf{x}_m$ for all $m \in \mathcal{N}_i$;
2 Compute F^i with weights λ_{ij};
3 $\nabla F^i \leftarrow$ `GradientComputation` $(\mathbf{x}_i^{inter}, \mathbf{x}_j^{neigh})$;
4 $\Delta \mathbf{x}_i = -\nabla F^i$;
5 **while** $F^i > 0$ **do**
6 $\quad \mathbf{x}_i := \mathbf{x}_i + \delta \Delta \mathbf{x}_i$;
7 $\quad$ Receive neighbour states $\mathbf{x}_m$ for all $m \in \mathcal{N}_i$;
8 $\quad \nabla F^i \leftarrow$ `GradientComputation` $(\mathbf{x}_i, \mathbf{x}_j)$;
9 $\quad \Delta \mathbf{x}_i = -\nabla F^i$;
10 $\quad k \leftarrow k + 1$;
11 $\quad$ **if** $k > L'$ **then** `random` $\mathbf{x}_i \in \mathcal{W}_i$ **break**;

algorithm as described in Algorithm 9.3 of [Boyd and Vandenberghe, 2004], utilising initial conditions $\mathbf{x}_i$, step size δ, maximum number of iterations L', and activation variables λ_{ij} as inputs. The activation variables are presented later in Function 3. It returns the optimised states $\mathbf{x}_i^\star$ as output. In line 8, the function `GradientComputation()` computes the gradient, either analytically or numerically. The stopping criterion is met either when a feasible state is determined, indicated by $F^i = 0$, or when the iteration count exceeds L' (line 11), which may occur due to multiple conflicting predicates active within F^i. This situation arises because the algorithm accounts for the possibility that the *eventually* operator may not be satisfied at every sampled point within its validity domain. This occurs, for example, if $\varphi = \mathcal{F}_{[0,5]}\mathcal{G}_{[0,5]}\ (g^{(1)}(\mathbf{x}) \leq \epsilon_1) \wedge \mathcal{G}_{[5,10]}\ (g^{(2)}(\mathbf{x}) \leq \epsilon_2)$, and there is a conflict between $h^{(1)}(\mathbf{x}) := g^{(1)}(\mathbf{x}) - \epsilon_1$ and $h^{(2)}(\mathbf{x}) := g^{(2)}(\mathbf{x}) - \epsilon_2$ (i.e., $\nexists \mathbf{x}_i^\star \in \mathcal{W}_i$ such that $h^{(1)}(\mathbf{x})(\mathbf{x}_i^\star) \leq 0 \wedge h^{(2)}(\mathbf{x})(\mathbf{x}_i^\star) \leq 0$). In such cases, it becomes necessary for $h^{(1)}(\mathbf{x}) \leq 0$ to be true exclusively within the interval $[0, 5]$[s] and for $h^{(2)}(\mathbf{x}) \leq 0$ to be true exclusively within the interval $[5, 10]$[s].

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. 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. The example below shows how to construct the optimisation functions F^i.

Example 4. *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,*

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

Now that spatial constraints are encoded into the optimisation problem, we are ready to encode temporal constraints in the following section, thus completing our STL decomposition into spatial and temporal constraints.

4.2 Temporal Constraints

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

$$\mathrm{vd}(\bar{\varphi}) = \begin{cases} 0, & \text{if } \bar{\varphi} = \mu^h \\ \mathrm{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \neg\bar{\varphi}_1 \\ [a, b], & \text{if } \bar{\varphi} = \mathcal{G}_{[a,b]}\mu^h \\ a \oplus \mathrm{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \mathcal{G}_{[a,b]}\bar{\varphi}_1, \bar{\varphi}_1 \neq \mu^h \\ t^\star + T^\star \oplus \mathrm{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \mathcal{F}_{[a,b]}\bar{\varphi}_1 \end{cases} \tag{9}$$

where $T^\star := \{t \in [a, b] \mid (\mathbf{x}, t) \models \mathcal{F}_{[a,b]}\bar{\varphi}\}$ is a time instant in $[a, b]$ when the state $\mathbf{x}$ evaluated at t of a signal $\mathbf{x}(t)$ satisfies the eventually operator. The variable $t^\star$ is initialised to 0, but takes the value $t^\star = T^\star$ every time $T^\star$ is updated and thus captures the last instance of satisfaction for the eventually operator.

The above definition of $t^\star$ 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 equal to the total number of validity domains. In Definition 3, 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.

Remark 1. *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 $\bar{\varphi} = \mathcal{G}_{[1,10]}\mathcal{G}_{[0,2]}(\cdot)$, 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, $\mathrm{vd}(\mathcal{G}_{[1,10]}\mathcal{G}_{[0,2]}(\cdot)) = \mathrm{th}(\bar{\varphi}) = [1, 12]$.*

Example 5. *Consider the following examples of the validity domain:*

- $\varphi_1 = \mathcal{G}_{[5,10]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$, *then* $\mathrm{vd}(\varphi_1) = [5, 10]$, *which is the interval over which $\mu^{h^{(1)}}$ must hold. Here $\mu^{h^{(1)}}$ is the predicate corresponding to the predicate function $h^{(1)}(\mathbf{x}) = g^{(1)}(\mathbf{x}) - \epsilon_1$.*
- $\varphi_2 = \mathcal{F}_{[5,10]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$, *then $t^\star$ is initialised to* 0, $T^\star \in [5, 10]$ *and* $\mathrm{vd}(\mu^{h^{(1)}}) = 0$. *Therefore,* $\mathrm{vd}(\varphi_2) = T^\star \in [5, 10]$ *is the instance when $\mu^{h^{(1)}}$ must hold.*
- $\varphi_3 = \mathcal{F}_{[5,10]}\mathcal{G}_{[0,2]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$, *then $t^\star$ is initialised to* 0, $T^\star \in [5, 10]$, $\mathrm{vd}(\mathcal{G}_{[0,2]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)) = [0, 2]$. *Therefore,* $\mathrm{vd}(\varphi_3) = 0 + T^\star \oplus [0, 2] = [T^\star, T^\star + 2]$ *is the interval over which $\mu^{h^{(1)}}$ must hold such that φ_3 is satisfied.*
- $\varphi_4 = \mathcal{G}_{[2,10]}\mathcal{F}_{[0,5]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$, *then* $a = 2$ *and* $\mathrm{vd}(\varphi_4) = 2 \oplus \mathrm{vd}(\mathcal{F}_{[0,5]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)) = 2 + 0 + T^\star$ *where* $T^\star \in [0, 5]$. *Suppose* $T^\star = 1$, *then* $\mathrm{vd}(\varphi_4) = 3$ *is the time instance when $\mu^{h^{(1)}}$ must hold. Once $\mu^{h^{(1)}} = \top$, then $t^\star = T^\star$ and the new* $\mathrm{vd}(\varphi_4) = 2 + 1 + T^\star$ *where* $T^\star \in [0, 5]$.
- $\varphi_5 = \mathcal{F}_{[0,100]}\mathcal{G}_{[5,10]}\mathcal{F}_{[0,1]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$, *then* $t^\star = 0$, $T^\star \in [0, 100]$ *and* $\mathrm{vd}(\varphi_5) = T^\star + a \oplus \mathrm{vd}(\mathcal{F}_{[0,1]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1))$. *Suppose* $T^\star = 50$, *then* $\mathrm{vd}(\varphi_5) = 55 \oplus \mathrm{vd}(\mathcal{F}_{[0,1]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1))$ *and so on.*

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

We use the following notational convenience in this work: if a parent node of a leaf node of a path $\bar{\varphi}$ 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 $\bar{\varphi}$ is an *always* operator we denote the corresponding validity domain by $\mathrm{vd}^G()$. The notation $\mathrm{vd}^F()$ indicates that the predicate of the respective leaf node needs to hold at some instance in the said interval, and $\mathrm{vd}^G()$ indicates that the predicate of the respective leaf node needs to hold throughout the interval. The following lemma formalises the relation between the STL formula and its corresponding encoding as described above.

Lemma 1. *Suppose $\mathbf{x}(t) = [\mathbf{x}_1^\top, \mathbf{x}_2^\top, \dots]$ represents the states of all robots, and $\bar{\varphi}_k$ encompasses all subformulas associated with the STL formula φ. Let $\sum_i F^i(\mathbf{x}(t)) = 0$ for all $t \in \bigcup_k \mathrm{vd}(\bar{\varphi}_k)$. Then, it holds that $\mathbf{y}(t) \models \varphi$.*

Proof. The proof follows from the construction of the optimisation function (7) and the validity domain. Notice that if the optimisation problem (8) converges to the desired minima at $F^i(\mathbf{x}) = 0$, then $\mu^{h^d_{i,l}} = \top$ and $\mu^{h^c_{i,j}} = \top$ for all $l \in \{1, \dots, L_i\}$ and $j \in \{1, \dots, K_c\} : i \in \mathbf{S}_j$. Next, by definition, the validity domain is defined for the STL formula and if F^i is minimised during the validity domain, then $\mathbf{y}(t) \models \varphi$.

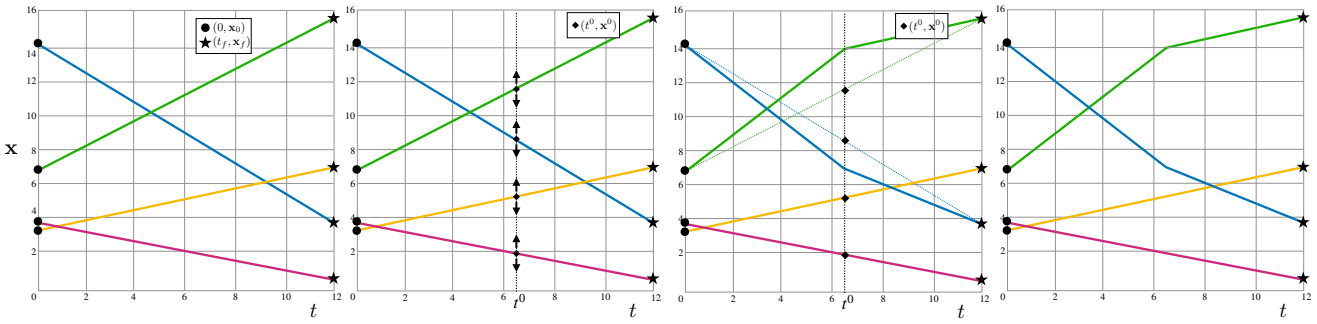
In the next Section, we present how to integrate the validity domain with the optimisation problem in (8), completing thus the spatial and temporal integration.

5 Main Results

In this section, we present the algorithm for generating continuous trajectories that meet the requirements of a given 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 $z \in \mathbb{R}_+ \times \mathcal{W}_i$ is sampled from a space-time plane. Until now, we denoted the states of robot i as $\mathbf{x}_i$, but from here onward, we denote them as $\mathbf{x}^i$.

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 formula $\mathrm{th}(\varphi)$, 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 $z_0^i = \{t_0^i, \mathbf{x}_0^i\}$ and $z_f^i = \{t_f^i, \mathbf{x}_f^i\}$ with a single edge $\mathcal{E}_i = \{(z_0^i, z_f^i)\}$. The initial conditions $z_0^i = \{t_0^i, \mathbf{x}_0^i\}$ depend on the robot's initial position and time. The final conditions are chosen to be $z_f^i = \{\mathrm{th}(\varphi) + \epsilon, \mathbf{x}_f^i\}$ where $\epsilon > 0$ and $\mathbf{x}_f^i \in \mathcal{W}^i$. Let $t_0^i = 0$ and $t_f^i = \mathrm{th}(\varphi) + \epsilon$. The final states $\mathbf{x}_f^i$ can be randomly chosen since the states in the interval $[0, \mathrm{th}(\varphi)]$ will be determined by the algorithm based on the constraints imposed by φ. The algorithm then randomly selects a time instant $t^0 \in [0, \mathrm{th}(\varphi)]$ 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 (8) 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 6. *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*



(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.

(magenta) to be closer than 6 units i.e., for $\epsilon > 0$,

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

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* $\mathcal{E}_i$ *contains only one edge connecting the start and end points. From the initial trajectory, the algorithm randomly selects a point at time instance* t^0 *from the entire time domain and uses linear interpolation to determine the state of each agent at that time. The agents solve (8) 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 (8) 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* $1/2\max(0, h_{i,j}^c)^2$*, and since agent 3 and agent 4 already satisfy the requirements, i.e.,* $h_{i,j}^c < 0$*, 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.*

5.1 MAPS^2

The architecture of the algorithm MAPS^2 (short for '**m**ulti-robot **a**nytime motion **p**lanning under **s**ignal temporal logic **s**pecifications'), is depicted in Figure 4. The algorithm, outlined in Algorithm 2, begins with an initial trajectory connecting z_0^i and z_0^f, along with a random seed and design constants as input (see lines 1-3). The random seed ensures that all robots select the same time instance. The algorithm proceeds by repeatedly sampling a time instance within the interval, interpolating states at the said time instance, applying gradient descent to minimise the function (7), and either adding or discarding the resulting optimal solution. This process is repeated until the total number of vertices,

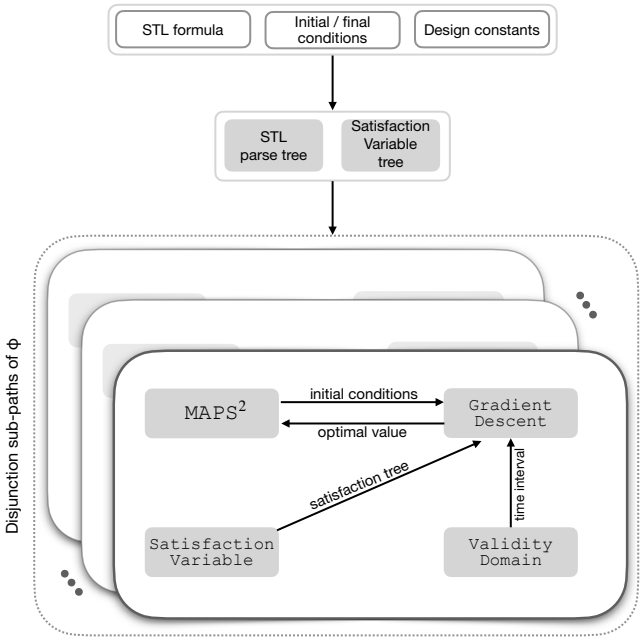


Figure 4. Architecture of the provided algorithm.

L, is reached, see lines 5-14. This is also illustrated in Figure 3.

These steps are implemented as follows: In line 7, the `SearchSort()` 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 $z^i_{\text{index}} = \{t^i_{\text{index}}, \mathbf{x}^i_{\text{index}}\}$ and $z^i_{\text{index}+1} = \{t^i_{\text{index}+1}, \mathbf{x}^i_{\text{index}+1}\}$. Then, the algorithm uses linear interpolation in line 8 via the function `Interpolate()` to obtain the vertex $z^i_{\text{inter}} = \{t^0, \mathbf{x}^i_{\text{inter}}\}$. This is obtained by solving for $\mathbf{x}^i_{\text{inter}}$ element-wise as the solution of

$$\mathbf{x}^i_{\text{inter}} = \left(\frac{\mathbf{x}^i_{\text{index}+1} - \mathbf{x}^i_{\text{index}}}{t^i_{\text{index}+1} - t^i_{\text{index}}}\right)(t^0 - t^i_{\text{index}}) + \mathbf{x}^i_{\text{index}}.$$

The vertex z^i_{inter} is the initial condition to solve the optimisation problem (8); and once a solution z^i_{opt} 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^i_{opt} in lines 11-13.

Moreover, as a safeguard, if a solution remains undiscovered following L iterations, line 16 initiates a reset

Algorithm 2: MAPS^2

Input: Initial condition $z_0^i = \{t_0^i, \mathbf{x}_0^i\}$, Final condition $z_\mathrm{f}^i = \{t_\mathrm{f}^i, \mathbf{x}_\mathrm{f}^i\}$, Maximum number of nodes L, random seed, step size δ, stopping criterion η, satisfaction variables of all subpaths $\bar{\varphi}_k$: $\tau(\bar{\varphi}_k) \leftarrow -1$

Output: $\mathcal{T}_i$

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

procedure. This involves setting the satisfaction variable for all *eventually* operators back to -1 and restarting the search. Since we assume that at least one viable solution always exists (refer to Assumption 4), the absence of a solution occurs solely when an *eventually* operator is satisfied at an impractical instance of time. Such an impractical instance of time affects the solution of the algorithm since there are redundancies in picking the satisfaction instance ($\mathbf{x}(t) \models \mathcal{F}_{[a,b]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$ if $h^{(1)}(\mathbf{x}) := g^{(1)}(\mathbf{x}) - \epsilon_1 \leq 0$ at any single instance in $[a, b]$). By resetting these operators, the algorithm aims to locate a solution under feasible instances.

5.1.1 `GradientDescent:` The function is presented in Function 3 and computes the optimal value, z_opt^i, by solving the problem presented in equation (8). This allows the robots to compute vertices that locally do not violate the STL formula. Once z_opt^i is determined through Function 1, the satisfaction variables are updated in Function 4.

Based on the validity domain, the Function 3 determines which predicate functions are active in (7) at every sampled time instance t^0. The Function `ValidityDomain()` in line 3 calculates the validity domains of each path $\bar{\varphi}$ based on Definition 3. Let K_i be the total number of independent and coupled predicate functions associated with robot i, a binary variable $\lambda_{ij} \in \{0, 1\}, j \in \{1, \ldots, K_i\}$ 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. For example,

- If $\varphi_1 = \mathcal{G}_{[5,10]}(\|x_1 - x_2\| \leq 2)$, then $\lambda_{11} = \lambda_{21} = 1$ whenever $t^0 \in [5, 10]$ and 0 otherwise.
- If $\varphi_2 = \mathcal{F}_{[10,15]}(\|x_3\| \leq 5)$, then $\lambda_{31} = 1$ whenever $t^0 \in [10, 15]$ and 0 otherwise. Once $x_3(t) \models \varphi_2$, $\lambda_{31} = 0\ \forall t$.

Function 3: GradientDescent

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

Output: z_opt^i, τ

1 Receive neighbour states $\mathbf{x}_\mathrm{neigh}^m$ for all $m \in \mathcal{N}_i$;
2 **forall** $\bar{\varphi}$ *in* φ **do**
3 $\mathrm{vd}_{ij}(\bar{\varphi}) \leftarrow$ `ValidityDomain` $(\bar{\varphi})$;
4 $k \leftarrow 0$;
5 $\lambda_{ij} = 0,\ \forall j$;
6 **case** $t^0 \in \mathrm{vd}_{ij}^F(\bar{\varphi})$ **do**
7 $\lambda_{ij} = 1$;
8 **case** $t^0 \in \mathrm{vd}_{ij}^G(\bar{\varphi})$ **do**
9 $\lambda_{ij} = 1$;
10 **case** $t^0 \in \bigcap_k \mathrm{vd}_{ik}^F(\bar{\varphi})$ **do**
11 $\begin{cases} \lambda_{ij} & = 1 \text{ for any one } j = k \\ \lambda_{ij} & = 0 \text{ otherwise} \end{cases}$
12 **case** $t^0 \in \bigcap_k \mathrm{vd}_{ik}^G(\bar{\varphi})$ **do**
13 $\lambda_{ik} = 1$ for all k;
14 $\mathbf{x}_\mathrm{opt}^i \leftarrow$ `DistributedOptimisation` $(\mathbf{x}^i, \delta, L', \lambda_{ij})$;
15 $z_\mathrm{opt}^i = \{t^0, \mathbf{x}_\mathrm{opt}^i\}$;
16 **forall** $\bar{\varphi}$ *in* φ **do**
17 **if** $t^0 \in \mathrm{vd}_{ij}(\bar{\varphi})$ **then**
18 **if** $F^i(\mathbf{x}_{opt}^i) \leq 0$ **then**
19 **node = leaf**$(\bar{\varphi})$;
20 $\tau($**leaf**$) = +1$;
21 **while** ***node*** $\neq$ ***root***$(\bar{\varphi})$ *and* $\tau($***node***$) = +1$ **do**
22 **node = parent(node)**;
23 $\tau($**node**$), t^\star \leftarrow$ `SatisfactionVariable` **(node**,z_opt^i**)**;
24 **else** reset $\tau(\bar{\varphi}), t^\star$;
25 **return** $z_\mathrm{opt}^i, \tau(\varphi)$

The indices i and j in λ_{ij} and vd_{ij} refer to robot i and the jth predicate function associated with robot i, respectively. Here $j \in \{1, \ldots, K_i\}$. 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 at random, that is, $\lambda_{ij} = 1$ only for one of them. This avoids enforcing conflicting predicates as it can happen that multiple *eventually* operators may not be satisfied at the same time instance (For example $\varphi = \mathcal{F}_{[0,1]}(x > 0) \wedge \mathcal{F}_{[0,1]}(x < 0)$); see lines 6-13.

In lines 16-24, the algorithm updates the satisfaction variable of all paths in the STL formula that impose restrictions on robot i's states. The algorithm goes bottom-up, starting from the **leaf** node to the **root** node. First, it determines if z_{opt}^i is the desired minimum (i.e., $F^i(\mathbf{x}_{opt}^i) \leq 0$) in line 18, and in lines 19-23, the algorithm updates the satisfaction variable of all nodes in the path $\bar{\varphi}$ through the function `SatisfactionVariable()`. If z_opt^i is not the

desired minimum, then all the satisfaction variables of the path $\bar{\varphi}$ are reset to -1 in line 25. This could result from conflicting predicates at the same time instance.

Function 4: SatisfactionVariable

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

```
case (node = F_I) do
    τ(F_I) = +1;
    t* = t^0;
    return τ, t*;
case (node = G_I) do
    if robust(G_I) ≥ 0 then
        τ(G_I) = +1;
    return τ, t*
case (node = ∧) do
    τ(∧) = +1;
    return τ, t*
```

5.1.2 `SatisfactionVariable:` This function, presented in Function 4, 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). The discussion of handling disjunction operators is deferred to Section 5.1.3, as they are handled differently. Considering the premise that the predicate is true, as indicated in line 18 of Function 3, we evaluate the satisfaction variable as follows:

- $\mathcal{F}_I$: The satisfaction variable of the eventually operator is updated along with $t^\star = t^0$. This updated $t^\star$ is used to determine the new validity domains in line 3 of Function 3; see Example 3 for an illustration of this procedure.
- $\mathcal{G}_I$: 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 [Maler and Nickovic, 2004]. Particularly, it samples a user-defined number of points in the interval $\text{vd}^G_{ij}()$ and computes $\inf_{t\in\text{vd}^G_{ij}} h^d_{i,l}(\mathbf{x}^i(t))$ or $\inf_{t\in\text{vd}^G_{ij}} h^c_{i,j}(\mathbf{x}^i(t))$. 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.

5.1.3 Branch-and-Pick for Disjunctions: In our approach, we address disjunctions as follows: Given an STL formula of the form $\varphi = \bigvee_{i\in 1,\dots,K} \phi_i$, which can also be represented as $\varphi = \vee(\phi_1, \phi_2, \dots, \phi_K)$, we divide it into K individual STL formulas. The agents then run Algorithm 2 separately for each $\varphi = \phi_i$, where $i \in 1, \dots, K$. For instance, consider the STL formula represented as (4),

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

We branch it into two STL formulas: $\phi_1 = \mathcal{F}_{I_1}\mu_1 \wedge \mathcal{G}_{I_3}\mathcal{F}_{I_4}(\mu_3) \wedge \mathcal{G}_{I_5}(\mu_4)$ and $\phi_2 = \mathcal{F}_{I_1}\mathcal{G}_{I_2}(\mu_2) \wedge \mathcal{G}_{I_3}\mathcal{F}_{I_4}(\mu_3) \wedge \mathcal{G}_{I_5}(\mu_4)$, as illustrated in Figure 5. The search terminates when any branch of the disjunction satisfies the condition $\tau(root) \neq +1$, as specified on line 5 of Algorithm 1. We acknowledge that this naive method of handling disjunctions can result in exponential growth with the addition of more operators. An alternative approach, akin to the branch-and-bound method from optimisation [Morrison et al., 2016], involves evaluating the robustness of each ϕ_i for $i \in 1, \dots, K$ and executing `MAPS`^{2} only for the formulas that show a faster increase in satisfaction. However, this strategy might necessitate a higher level of communication among robots which goes beyond their existing communication network and possibly require a central authority to coordinate task fulfilment. For example, the STL formula

$$\varphi = \mathcal{G}_{[0,5]}(x_1 < 5) \vee (\mathcal{F}_{[0,5]}(|x_2 - x_3| > 2)).$$

comprises disjunction between $\phi_1 = \mathcal{G}_{[0,5]}(x_1 < 5)$ and $\phi_2 = \mathcal{F}_{[0,5]}(|x_2 - x_3| > 2)$. Observe that ϕ_1 requires no inter-robot communication, while ϕ_2 necessitates communication between robots 2 and 3. In the implementation of a method akin to branch-and-bound, we would branch into two formulas, ϕ_1 and ϕ_2, and repeatedly switch between them if we observe the robustness of one formula decaying faster compared to the other. This switching must be performed by a central authority that observes the decay in robustness. If the switching is decided among the robots, then robot 1 of ϕ_1 needs to communicate the robustness decay with the network of robots 2 and 3. This requires robot 1 to establish communication with the network of robots 2 and 3 in order to decide which branch to grow, thereby necessitating communication links where none existed before. Without such a communication link, both ϕ_1 and ϕ_2 would need to be satisfied using the naive approach presented in our work. This motivates our choice to use the naive approach.

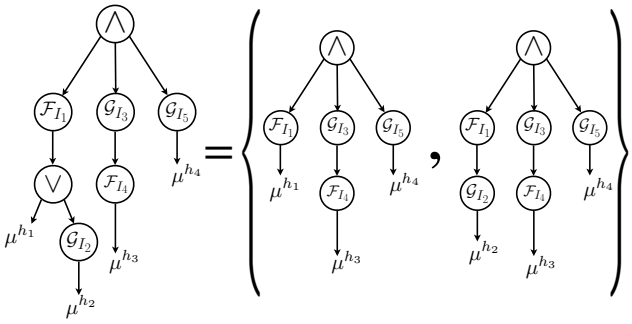


Figure 5. Disjunction representation for disjunctive components using STL parse tree.

5.2 Analysis

In this section, we analyse the proposed algorithm and arrive at proving the probabilistic completeness.

Let the set $\mathcal{S} \subseteq \mathcal{W}$ be a compact set where a trajectory $\mathbf{y} : [0, \text{th}(\varphi)] \to \mathcal{S}$ satisfies the STL formula. Along the lines of [Kleinbort et al., 2019], 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

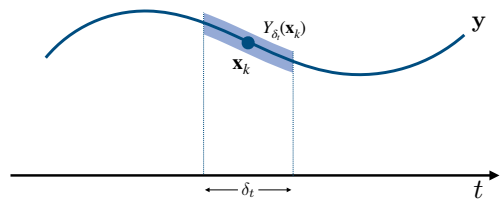


Figure 6. Illustration of $Y_{\delta_t}(\mathbf{x}_k)$.

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 $\mathbf{x}_k$, where $0 \leq k \leq L$ 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 $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ 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 sequential covering class* of trajectory $\mathbf{y}$ as $Y_{\delta_t}(\mathbf{x}_k)$. The length of $Y_{\delta_t}(\mathbf{x}_k)$ is δ_t in the time domain and is centered at $\mathbf{x}_k$. 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 $\mathbf{x}_k$, that is, within $Y_{\delta_t}(\mathbf{x}_k)$. If there are L successful trials, the entire trajectory $\mathbf{y}$ is covered, and the motion planning problem is solved. Consider m total samples, where $m \gg L$, and treat this as m Bernoulli trials with success probability p since each sample is independent with only two outcomes. We are now ready to state the following lemma.

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

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

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

and according to [Feller, 1968], if $p < \frac{1}{2}$, we can upper bound this probability as:

$$\mathbf{P}[X_m \leq L] \leq \frac{(m-L)p}{(mp-L)^2}.$$

As p and L are fixed and independent of m, the expression $\frac{(m-L)p}{(mp-L)^2}$ approaches 0 with as m increases, thus completing the proof.

Next, we present a final lemma which helps us prove the probabilistic completeness of the algorithm.

Lemma 3. *No sampled point $\mathbf{x}_k$ is falsely labelled as satisfying the STL formula φ unless it actually does.*

Proof. The algorithm initiates by setting all satisfaction variables, τ, to -1, as inputs to Algorithm 2. These variables are updated in Function 4 designed for evaluating whether τ meets the satisfaction criteria. The function adjusts τ in accordance with the definition of STL operators presented in Section 3.1, ensuring that updates accurately reflect the satisfaction status. Furthermore, the update to $\tau(\textbf{leaf})$ within Function 3 (referenced at line 20) occurs only when the condition $F^i \leq 0$ is met. This condition indicates that all active predicates are satisfied by definition. Thus, no satisfaction variable is incorrectly updated.

Next, the paper's final result is presented, which states that the probability of the algorithm providing an STL formula satisfying trajectory (if one exists) approaches one as the number of samples tends to infinity. This is a desirable property for sampling-based planners and such algorithms are termed probabilistically complete.

Theorem 1. *Algorithm 2 is probabilistically complete.*

Proof. The proof follows from Lemmas 1, 2, and 3. From Lemma 1 and Lemma 3, we know that every sample added to the trajectory satisfies the STL formula. Thus, what needs to be shown is that the algorithm samples infinitely many times and covers the entire time horizon. From Lemma 2, we know that the probability of covering the entire time horizon is $1 - \mathbf{P}[X_m \leq L]$. Suppose the Algorithm 2 reaches $J = L'$ samples without finding a feasible solution, then it discards J samples as seen in line 16 of Algorithm 2. Given Assumption 4, we have $J < \infty$, and since J is the number of discarded samples, we also have $J \leq m$ where m is the total number of samples sampled so far (including the discarded ones). Thus, the probability of the trajectory satisfying the STL formula is $1 - \frac{((m-J)-L)p}{((m-J)p-L)^2}$, which approaches one as $m \to \infty$. Thus, the algorithm is probabilistically complete.

Remark 2. *Our algorithm can be endowed in a post-processing stage with a module that smoothens the trajectory to avoid large accelerations. However, care needs to be taken since the smoothened paths may no longer satisfy the STL formula. One could also use more sophisticated approaches like B-splines to impose velocity and acceleration limits as shown in [Lapandić et al., 2024].*

Remark 3. *At present, our approach does not incorporate kinematic or dynamic constraints. Incorporation of such constraints could be attempted by either deploying the kinodynamic version of the RRT algorithm [Webb and van den Berg, 2012], or by using an existing low-level controller to track the generated open-loop trajectories. Some examples of such controllers include the Model Predictive Controller [Poignet and Gautier, 2000] and the input constrained Prescribed Performance Controller [Fotiadis and Rovithakis, 2024, Trakas and Bechlioulis, 2023]. This incorporation is by no means straightforward but requires fusion with another type of methodological machinery that goes beyond the scope of the current work. Moreover, such controllers have been developed for a large variety of dynamical systems and hence the proposed algorithm is practical and applicable to a large class of robots.*

*Meaning $\mathbf{y} \subset \bigcup_{k=1}^{L} Y_{\delta_t}(\mathbf{x}_k)$

6 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]

6.0.1 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].

6.0.2 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].

6.0.3 Stability The last task is that of stability, which is represented by the STL formula $\mathcal{F}_{[a_1,b_1]}\mathcal{G}_{[a_2,b_2]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$. This formula requires that $(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$ 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 $(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$ 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].

6.0.4 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 $\mathcal{G}_{[a_1,b_1]}\mathcal{F}_{[a_2,b_2]}(g^{(1)}(\mathbf{x}) \leq \epsilon_1)$, which reads as 'beginning at a_1[s], $g^{(1)}(\mathbf{x}) \leq \epsilon_1$ 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].

In reference to Remark 2, an example of post-processing the trajectories is shown in Figure 8 for the STL formula,

$$\varphi = \mathcal{G}_{[0,100]}\mathcal{F}_{[0,20]}(\|x_1 - x_2\| \leq 1). \tag{10}$$

A 3rd order polynomial was applied using the Savitzky-Golay filter to smoothen the trajectory. Smoothening helps to avoid any large accelerations and sudden velocity changes, though it may come at the cost of potential STL violations.

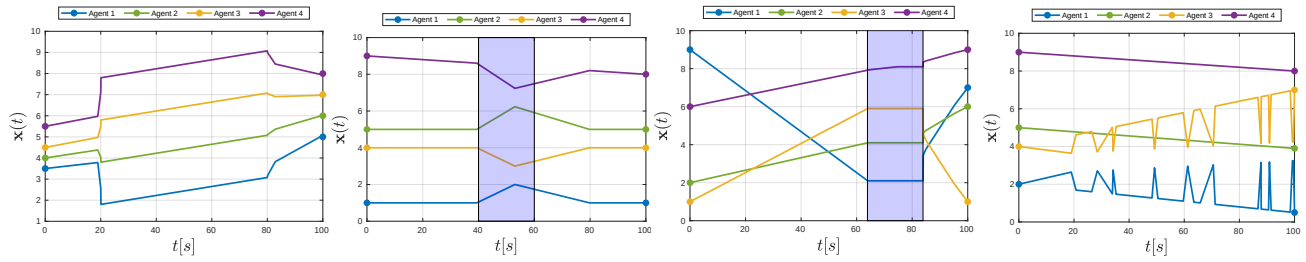
6.0.5 Multi-agent case study In this case study, we design trajectories for a team of 100 agents that exist in a 100×100[m] space and $[0, 100]$[s] time span. The team needs to adhere to the following STL formula,

$$\varphi = \mathcal{G}_{[10,90]}\Big[\|x_i - x_j\| \geq 0.01 \wedge \|x_i - (50,50)\| \leq 5\Big] \tag{11}$$

$\forall i, j \in \{1, 2, \ldots, 100\}$ and $i \neq j$. Note that the above STL formula has 5150 predicates. In the interval $[10, 90]$[s], the STL formula dictates every agent to be at least 0.01[m] apart from every other agent and to be at least 5[m] close to the centre point $(50, 50)$[m]. The simulation results are shown in Figure 10 where the Figures 10a-10c are the trajectories before the start of the algorithm while Figures 10d-10f shows the trajectories at the end of $j = 1000$ iterations, as mentioned in Algorithm 2. The simulation took 17.84[s] to complete without parallelisation. The faster computation can be attributed to the nature of the design of the cost function in (8), which allows for points that do not violate the formula not to be changed. The robustness of the STL formula is shown in Figure 11, a negative robustness signifies task satisfaction. Here, the robustness converges to 0, because the robustness for an *always* operator reflects the worst-case scenario. It is important to note that computing the result for Figure 11 required 12 hours and 10 minutes of computation time since it had to be performed centrally.

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

- Agent 1 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].



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

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

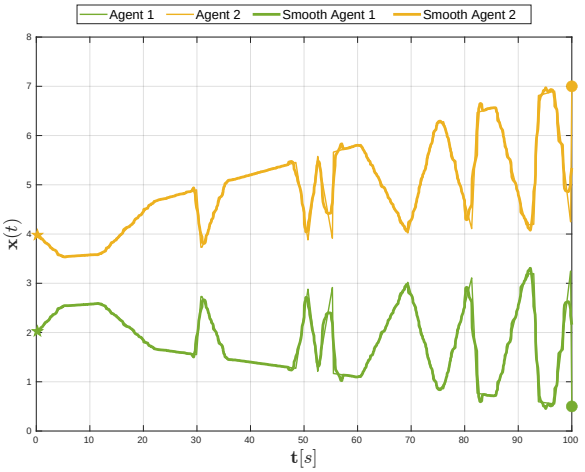


Figure 8. Non-smooth and smooth paths for the formula (10).

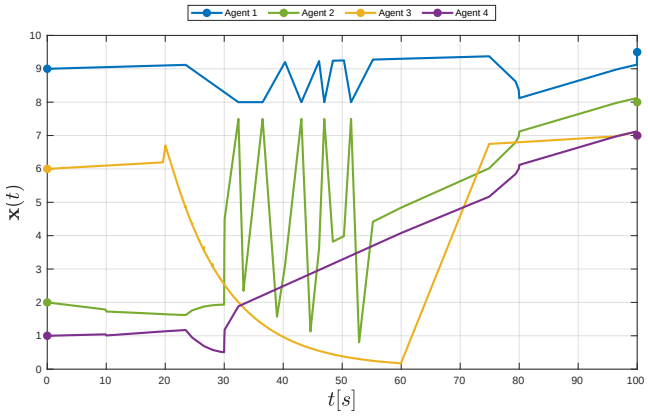


Figure 9. Overall case study.

The STL formula for 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 9, 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].

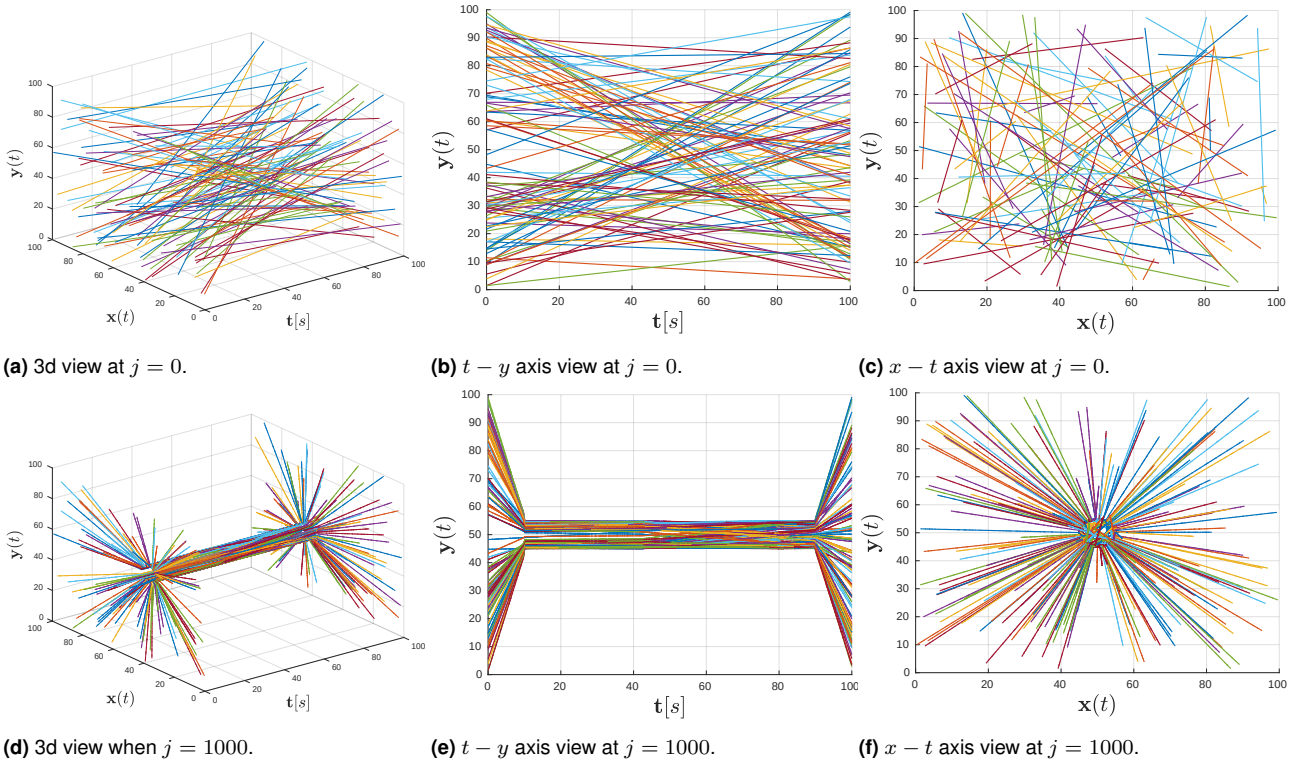
7 Experiments

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], 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 $\bar{\varphi}_5 = \mathcal{F}_{[180,200]}\|\mathbf{x}_1 - [0,0]^\top\| \leq 0.05 \wedge \mathcal{F}_{[180,200]}\Big(\|\mathbf{x}_2 - [1,-1]\| \leq 0.05 \wedge \|\mathbf{e}_1 - [\mathbf{x}_2, 0.6]\| \leq 0.05\Big) \wedge \mathcal{F}_{[180,200]}\Big(\|\mathbf{x}_3 - [-1,1]\| \leq 0.05 \wedge \|\mathbf{e}_2 - [\mathbf{x}_3, 0.6]\| \leq 0.05\Big)$.



(a) 3d view at $j = 0$. **(b)** $t - y$ axis view at $j = 0$. **(c)** $x - t$ axis view at $j = 0$.

(d) 3d view when $j = 1000$. **(e)** $t - y$ axis view at $j = 1000$. **(f)** $x - t$ axis view at $j = 1000$.

Figure 10. Simulation of trajectory generation for 100 agents for the STL formula (11).

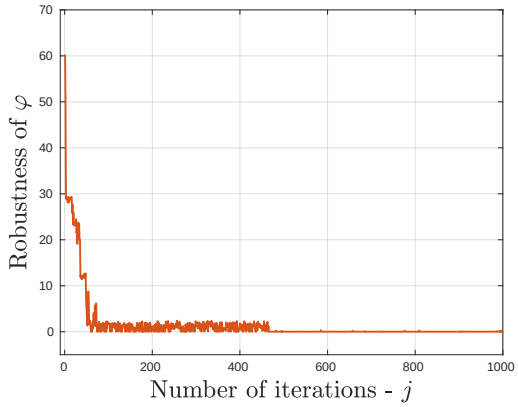


Figure 11. Robustness of the STL formula in (11).

The results are shown in Figure 12, 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 (8), 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 12a 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 12b, base 1 tracks a circular trajectory in the interval $[10, 125]$ seconds. In Figures 12c and 12d, 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 12e and 12f. The maximum computation time by any robot is 3.611[s]. Figure 13 shows front-view and side-view at different time instances during the experimental run[2].

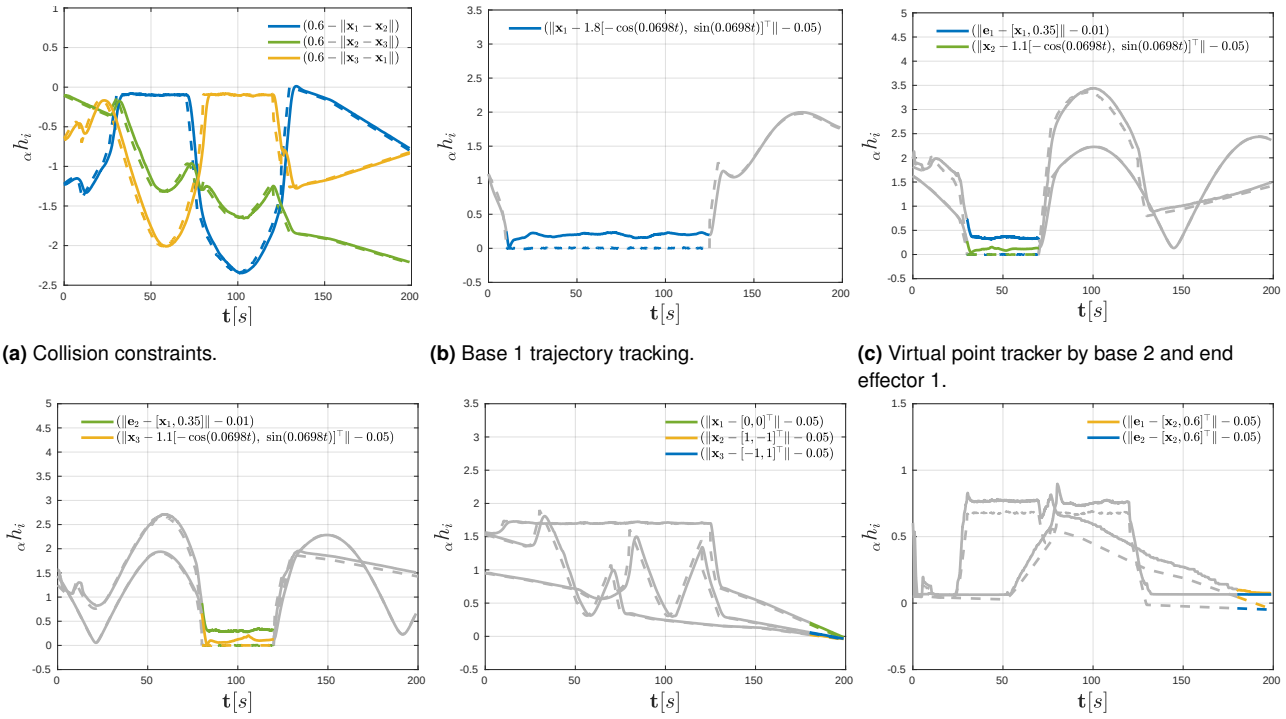
8 Conclusion

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 (8), 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.

Acknowledgements

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.

Notes



(a) Collision constraints.

(b) Base 1 trajectory tracking.

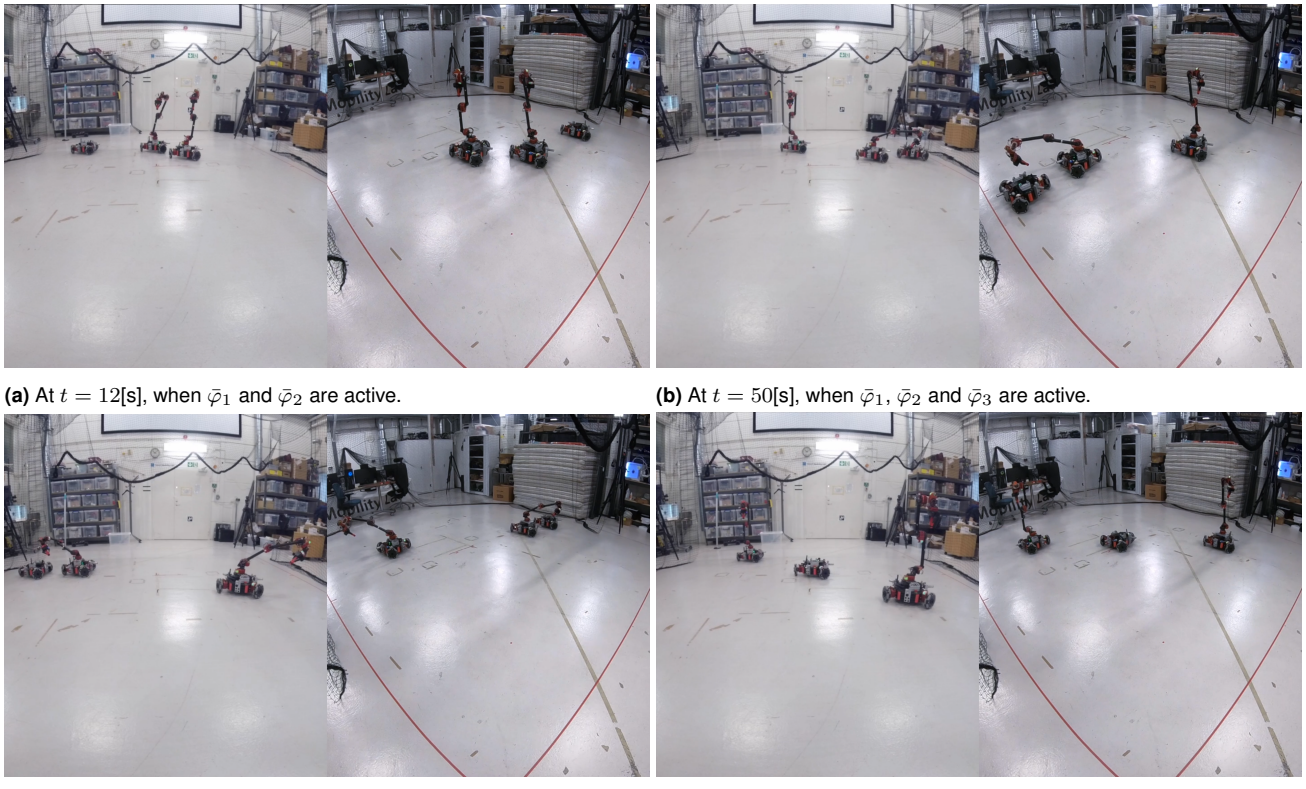
(c) Virtual point tracker by base 2 and end effector 1.

(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 12. Experimental verification of MAPS^2 with the setup in Figure 1.



(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 13. Front-view and side-view during experimental run with the setup in Figure 1.

1. The project code can be found here https://github.com/sewlia/Maps2
2. The video of the experiments can be found here: https://youtu.be/YkuiPuOerMg

References

[Ayala et al., 2013] Ayala, A. M., Andersson, S. B., and Belta, C. (2013). Temporal logic motion planning in unknown environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5279–5284. IEEE.

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

[Bhatia et al., 2010] Bhatia, A., Kavraki, L. E., and Vardi, M. Y. (2010). Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation*, pages 2689–2696.

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

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

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

[Daneshmand et al., 2020] Daneshmand, A., Scutari, G., and Kungurtsev, V. (2020). Second-order guarantees of distributed gradient algorithms. *SIAM Journal on Optimization*, 30(4):3029–3068.

[Donzé, 2013] Donzé, A. (2013). On signal temporal logic. *International Conference on Runtime Verification*, pages 382–383.

[Fainekos et al., 2009] Fainekos, G. E., Girard, A., Kress-Gazit, H., and Pappas, G. J. (2009). Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352.

[Feller, 1968] Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley.

[Fotiadis and Rovithakis, 2024] Fotiadis, F. and Rovithakis, G. A. (2024). Input-constrained prescribed performance control for high-order mimo uncertain nonlinear systems via reference modification. *IEEE Transactions on Automatic Control*, 69(5):3301–3308.

[Gilpin et al., 2020] Gilpin, Y., Kurtz, V., and Lin, H. (2020). A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control Systems Letters*, 5(1):241–246.

[Goyvaerts, 2016] Goyvaerts, J. (2016). Regular expression tutorial - learn how to use regular expressions. https://www.regular-expressions.info/tutorial.html. Archived from the original on 2016-11-01. Retrieved 2016-10-31.

[Kleinbort et al., 2019] Kleinbort, M., Solovey, K., Littlefield, Z., Bekris, K. E., and Halperin, D. (2019). Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters*, 4(2):i–vii.

[Kress-Gazit et al., 2009] Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2009). Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381.

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

[Lamport, 1983] Lamport, L. (1983). What good is temporal logic? *Information Processing 83, R. E. A. Mason, ed., Elsevier Publishers*, 83:657–668.

[Lapandić et al., 2024] Lapandić, D., Verginis, C. K., Dimarogonas, D. V., and Wahlberg, B. (2024). Kinodynamic motion planning via funnel control for underactuated unmanned surface vehicles. *IEEE Transactions on Control Systems Technology*, 32(6):2114–2125.

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

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

[Lindemann et al., 2017] Lindemann, L., Verginis, C. K., and Dimarogonas, D. V. (2017). Prescribed performance control for signal temporal logic specifications. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, page 2997–3002. IEEE Press.

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

[Maler and Nickovic, 2004] Maler, O. and Nickovic, D. (2004). Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer.

[Morrison et al., 2016] Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.

[Nedic and Ozdaglar, 2009] Nedic, A. and Ozdaglar, A. (2009). Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61.

[Poignet and Gautier, 2000] Poignet, P. and Gautier, M. (2000). Nonlinear model predictive control of a robot manipulator. In *6th International Workshop on Advanced Motion Control. Proceedings (Cat. No.00TH8494)*, pages 401–406.

[Raman et al., 2014] Raman, V., Donzé, A., Maasoumy, M., Murray, R. M., Sangiovanni-Vincentelli, A., and Seshia, S. A. (2014). Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87. IEEE.

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

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

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

[Trakas and Bechlioulis, 2023] Trakas, P. S. and Bechlioulis, C. P. (2023). Robust adaptive prescribed performance control for unknown nonlinear systems with input amplitude and rate constraints. *IEEE Control Systems Letters*, 7:1801–1806.

[Vasile and Belta, 2013] Vasile, C. I. and Belta, C. (2013). Sampling-based temporal logic path planning. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4817–4822.

[Verginis and Dimarogonas, 2018] Verginis, C. K. and Dimarogonas, D. V. (2018). Timed abstractions for distributed cooperative manipulation. *Autonomous Robots*, 42:781–799.

[Webb and van den Berg, 2012] Webb, D. J. and van den Berg, J. P. (2012). Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints. *ArXiv*, abs/1205.5088.

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

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