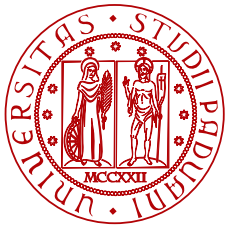


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

MASTER THESIS IN CONTROL SYSTEMS ENGINEERING

Sampling-based Synthesis of Controllers for Multiple Agents under Signal Temporal Logic Specifications

MASTER CANDIDATE

Sara Gomiero

Student ID 2053072

SUPERVISOR

Prof. Angelo Cenedese

University of Padova

KTH SUPERVISORS

Ph.D. Mayank Sewlia

Prof. Dimos Dimarogonas

KTH - Kungliga Tekniska Högskolan

ACADEMIC YEAR
2022/2023

To my family

Abstract

The wide application of robots in industries and society has brought the need to prescribe complex high-level tasks to autonomous agents. Signal Temporal Logic (STL) is a temporal logic that allows to express desired spatio-temporal requirements, while quantifying the satisfaction of the preferences. When planning under STL specifications, the main challenge is to generate trajectories that satisfy the logical formulas and to track those trajectories.

The project proposes a solution for the motion planning problem of multiple autonomous agents, subject to coupled STL specifications. Starting from a scenario where only two agents are involved, a sampling-based algorithm, Coupled STL_RRT*, is designed. The proposed RRT*-based approach builds two trees in the coupled time and state domain in a distributed manner. For each dynamical system, given an initial position, the developed strategy finds a probabilistic optimal trajectory in terms of a cost function that depends on the required control inputs. Before adding new states to the corresponding tree, the algorithm checks if the logical formula is not violated, hence ensuring that the final time-varying trajectory satisfies the spatio-temporal specifications. The dynamics of the autonomous agent is directly taken into account and reachability is exploited to obtain a trajectory that is feasible with respect to the dynamic constraints. The algorithm is then simulated, considering an environment with a static obstacle and different STL requirements, specified by the user.

Finally, the approach is extended to the case of multi-agent systems with more than three agents. As in the previous case, the algorithm builds a spatio-temporal tree for each agent, ensuring that the final trajectory satisfies the STL requirements. The proposed solution is then verified in simulated scenarios, considering 4-agents and 6-agents systems. To conclude, a 3D extension is developed and tested with three agents, assuming a double integrator dynamics both along x and y.

Sommario

L'ampia applicazione dei robot nelle industrie e nella società ha portato alla necessità di prescrivere complessi compiti di alto livello ad agenti autonomi. Signal Temporal Logic (STL) è una logica temporale che consente di esprimere requisiti spazio-temporali e quantificare il livello di soddisfazione delle preferenze. Quando si pianifica considerando specifiche STL, la sfida principale è generare traiettorie che soddisfino le formule logiche e seguire le traiettorie così ottenute.

Il progetto propone una soluzione per il problema di pianificazione del movimento di multipli agenti autonomi, soggetti a specifiche STL accoppiate. Partendo da uno scenario in cui sono coinvolti solo due agenti, un algoritmo basato sul campionamento, Coupled STL_RRT*, è progettato. L'approccio proposto, basato su RRT*, costruisce in modo distribuito due alberi nel dominio del tempo e dello stato accoppiati. Per ogni sistema dinamico, data una posizione iniziale, la strategia sviluppata trova la traiettoria probabilisticamente ottimale in termini di una funzione di costo che dipende dagli input di controllo richiesti. Prima di aggiungere nuovi stati all'albero corrispondente, l'algoritmo controlla se la formula logica non viene violata, assicurando quindi che la traiettoria finale, variabile nel tempo, soddisfi le specifiche spazio-temporali. La dinamica dell'agente autonomo è presa direttamente in considerazione e il concetto di raggiungibilità viene sfruttato per ottenere traiettorie ammissibili rispetto ai vincoli dinamici. L'algoritmo è quindi simulato, considerando un ambiente con un ostacolo statico e diversi requisiti STL, specificati dall'utente.

Infine, l'approccio viene esteso al caso di sistemi multi-agente con più di tre agenti. Come nel caso precedente, l'algoritmo costruisce un albero spazio-temporale per ciascun agente, assicurando che la traiettoria finale soddisfi i requisiti STL. La soluzione proposta è poi verificata in scenari simulati, considerando sistemi con 4 o 6 agenti. Per concludere, un'estensione 3D viene sviluppata e testata con tre agenti, assumendo una dinamica a doppio integratore sia lungo x che lungo y.

Contents

List of Figures

List of Algorithms

List of Acronyms

B-RRT* Bidirectional Rapidly-exploring Random Tree Star

DIS Direction of Increasing Satisfaction

LQR Linear Quadratic Regulation

LTL Linear Temporal Logic

MASs Multi-Agent Systems

MTL Metric Temporal Logic

PPC Prescribed Performance Control

PRM Probabilistic Roadmap

RG-RRT Reachability-Guided Rapidly-exploring Random Tree

RRT Rapidly-exploring Random Tree

RRT* Rapidly-exploring Random Tree Star

RRT*FN Rapidly-exploring Random Tree Star Fixed Nodes

STL Signal Temporal Logic

TL Temporal Logic

Chapter 1

Introduction

1.1 Motivation

In the last few years, multi-agent systems (MASs) have received increasing attention in different studying areas, such as civil engineering and computer science, to solve complex issues. Indeed, the use of multiple robots allows to subdivide sophisticated problems into smaller tasks and brings several advantages compared to single-agent systems. Among the benefits of MASs, it is possible to mention scalability, robustness, increased fault tolerance and better performance. Furthermore, even if many robotic tasks can be executed by single robots, in general multi-agent teams can accomplish these objectives using simpler, less expensive and more flexible robots.

When considering a multi-agent system, the individual tasks are allocated to autonomous entities (the agents), characterized by individual actuation, sensing and decision-making capabilities. Each agent decides on a proper action to solve the task using multiple inputs and exploits its interactions with neighboring agents and with the environment to learn new contexts and actions [1]. The main characteristics of a multi-agent system are:

- autonomy (agents are responsible for their individual tasks);
- complexity (induced by the mechanisms of decision-making, learning, reasoning);
- adaptability (agents activities are adapted to the dynamic changes of the environment);
- concurrency (if tasks are processed in parallel);

- communication (either inter-agent or intra-agent);
- distribution (multi-agent systems are often distributed over a network);
- mobility (agents could be required to move between environments);
- openness (multi-agent systems can dynamically decide upon their participants) [2].

Agents may have collaborative objectives, so that it is necessary to consider coordination, collaboration and connectivity between all the single entities involved in the tasks. When coupled cooperation objectives need to be achieved, it is required to provide communication and real-time coordinated control between all the agents of the system for the execution. The need of cooperation implies that the robots must acquire some knowledge about the state and actions of the other agents, either through sensory perception or explicit communication (see Figure 1.1) [3].

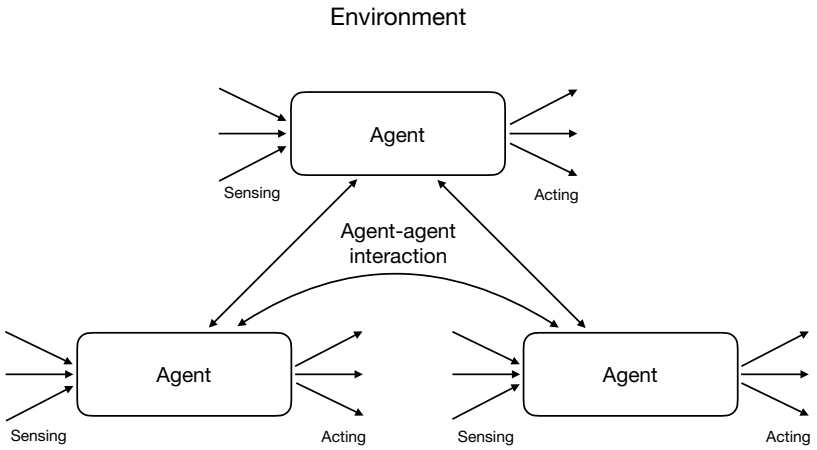
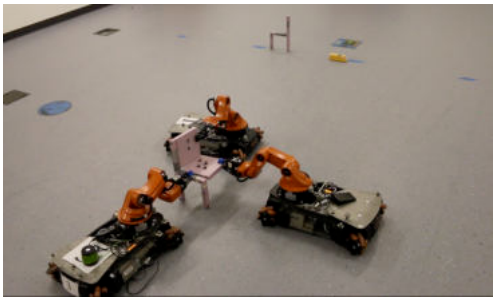


Figure 1.1: Example of multi-agent system

Multi-agent systems have been studied and exploited in several areas, such as robotics, transportation, manufacturing, agriculture (see Figure 1.2). The main application domains of multi-agent systems are ambient intelligence, grid computing, electronic business, computational biology, monitoring and control, resource management, military and manufacturing applications. Many researchers have exploited agent technology to industrial applications such as manufacturing enterprise integration, supply chain management, manufacturing planning, scheduling and control [4].

(a) Assembly

(b) Transportation



(c) Manipulation

(d) Agriculture

Figure 1.2: Examples of application of multi-agent systems

In the last decades, global objectives that have been reached are consensus, formation control, connectivity maintenance. More recently, the goal has become the one of imposing more complex specifications and adding the human intervention during the execution of tasks, thus requiring the capability of controlling robotic agents in real-time.

With the advancement of robotic agents and autonomous vehicles, it has become necessary to define high-level preferences and reach more complex goals. This demand has hence led to the development of efficient and expressive languages, known as temporal logics (TL). These logics provide a mathematically precise language to specify rules and time-constrained requirements on the behavior of robotic systems, offering a great degree of versatility when describing complex planning objectives. More specifically, TL can express traditional robot specifications, such as reaching a goal or avoiding an obstacle, but also more complicated ones, such as sequencing, coverage or temporal ordering of different tasks [5]. Consequently, in recent years, motion planning algorithms have been enhanced with temporal logic specifications. Traditionally, the motion planning problem

consists in finding a sequence of control inputs capable of driving a dynamical system from its initial state to a goal region, while satisfying environment constraints, such as maintaining a certain distance from obstacles or boundaries of the workspace. When complex missions and constraints need to be taken into account, a special form of temporal logic, namely Signal Temporal Logic (STL), can be exploited to establish temporal and spatial rules in robot's trajectories. Through the help of operators in STL formulas, users are able to define minimum or maximum distances from obstacles [6], restrictions on the mutual position between multiple agents, constraints concerning speed and acceleration limits. This combined task and motion planning problem poses computational challenges derived from robot dynamics, collision avoidance, temporal constraints and dependencies between motion trajectories and time requirements. Indeed, a solution satisfying the STL formula φ may not be feasible due to dynamic constraints or, on the other hand, collision-free and dynamically-feasible trajectories may violate the requirements encoded in φ [7].

1.2 Scope and Outline

This project addresses the cooperative motion planning problem of two or more autonomous agents, evolving in time and space, to deliver coupled tasks expressed as signal temporal logic specifications. The contribution is the design of an algorithm, Coupled STL_RRT*, that solves the aforementioned problem, taking into account the dynamics of the two coupled agents. Given the STL formula expressing the desired preferences, the algorithm is guided towards the selection of the probabilistic optimal trajectory for each agent. The choice of the best trajectory, in particular, is influenced by a cost function that depends on the control inputs required to drive the agent through subsequent states. Coupled STL_RRT* is then extended to deal with coupled multi-agent systems.

The thesis is divided as follows:

- Chapter 2 firstly presents notation and preliminaries regarding signal temporal logic. Moreover, a comparison between different temporal logics is provided, with emphasis on the advantages of STL with respect to the other logics. The focus is then on topics related to optimal motion planning algorithms;

- Chapter 3 introduces related works regarding the design of controllers under STL specifications and the solution to multi-agent motion planning;
- Chapter 4 contains the main contribution of the project, i.e. the algorithm designed for the solution of the problem;
- Chapter 5 presents simulations and results obtained by testing the algorithm, initially considering two autonomous agents;
- Chapter 6 regards the extension of the developed algorithm to multi-agent systems and simulations when more than 4 agents are coupled. In addition, a 3D version of the approach is explained and simulated;
- Chapter 7 concludes the thesis, by summarizing the results and highlighting future research directions.

Chapter 2

Background

In this chapter, some general notation and conventions related to Signal Temporal Logic (STL) are presented. More specifically, the focus regards qualitative and quantitative semantics, definitions of operators, horizon, validity domain and robustness of an STL formula. STL is also compared to other temporal logics, highlighting the advantages of the former and the drawbacks of the latters.
The last section of the chapter concerns optimal motion planning algorithms, with emphasis on Rapidly-exploring Random Tree Star (RRT*) approach.

2.1 Notation

Let $\mathbb{R}$ be the set of real numbers, while $\mathbb{R}^n$ is the n-dimensional real vector space. The sets of non-negative and positive real numbers are $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$. The sets of natural numbers and integers are indicated by $\mathbb{N}$ and $\mathbb{Z}$ respectively.
The Boolean domain $\mathbb{B}$ contains two values, true and false. The former is denoted by $\top$, the latter by $\bot$.
Given the interval $I = [a, b]$, the interval $[t + a, t + b]$ is denoted by $t + I$.
If $c \in \mathbb{R}$ and $[a, b] \in \mathbb{R}^2$, the Kronecker sum is defined as $c \oplus [a, b] = [c + a, c + b]$.

2.2 Signal Temporal Logic

STL is a predicate-based logic that takes into account time and space, hence allowing to formulate complex specifications with timing requirements or deadlines and spatial and temporal properties over continuous-time signals.

2.2.1 Syntax

STL is based on a continuous-time signal $\boldsymbol{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ and contains predicates $\mu : \mathbb{R}^n \to \mathbb{B}$ as the atomic elements [8]. The syntax of STL, also known as grammar of STL, is defined as follows:

$$\varphi = \top \,|\, \mu \,|\, \neg\varphi \,\big|\, \mathcal{G}_{[a,b]}\varphi \,\big|\, \mathcal{F}_{[a,b]}\varphi \,\big|\, \varphi_1 \mathcal{U}_{[a,b]}\varphi_2 \,\big|\, \varphi_1 \wedge \varphi_2 \tag{2.1}$$

Above, φ_1, φ_2 are STL formulas, $\neg$ and $\wedge$ are the Boolean operators for negation and conjunction and $\mathcal{U}_{[a,b]}$ encodes the until operator, with $0 \leq a \leq b < \infty$. μ is a predicate of the form $\mu : \mathbb{R}^n \to \mathbb{B}$, defined via a predicate function $h : \mathbb{R}^n \to \mathbb{R}$ as:

$$\mu(\boldsymbol{x}(t)) = \begin{cases} \top & h(\boldsymbol{x}(t)) \geq 0 \\ \bot & h(\boldsymbol{x}(t)) < 0 \end{cases} \tag{2.2}$$

It is also possible to define the disjunction, the eventually and the always operators as $\varphi_1 \vee \varphi_2 \equiv \neg\left(\neg\varphi_1 \wedge \neg\varphi_2\right)$, $\mathcal{F}_{[a,b]}\varphi \equiv \top\mathcal{U}_{[a,b]}\varphi$, and $\mathcal{G}_{[a,b]}\varphi \equiv \neg\mathcal{F}_{[a,b]}\neg\varphi$. The satisfaction relation $(\boldsymbol{x}, t) \models \varphi$ indicates that signal $\boldsymbol{x}$ satisfies φ at time t.

Example 2.1. Consider the predicate function $h(\boldsymbol{x}(t)) := r - \|\boldsymbol{x}(t)\|$. Clearly, the corresponding predicate $\mu(\boldsymbol{x}(t)) = \top$ if and only if $\|\boldsymbol{x}(t)\| \leq r$. For a given trajectory $\boldsymbol{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ as in Figure 2.1, it follows that $\mu(\boldsymbol{x}(t)) = \top$ if and only if $\boldsymbol{x}(t)$ is within the ball of radius r centered at the origin.

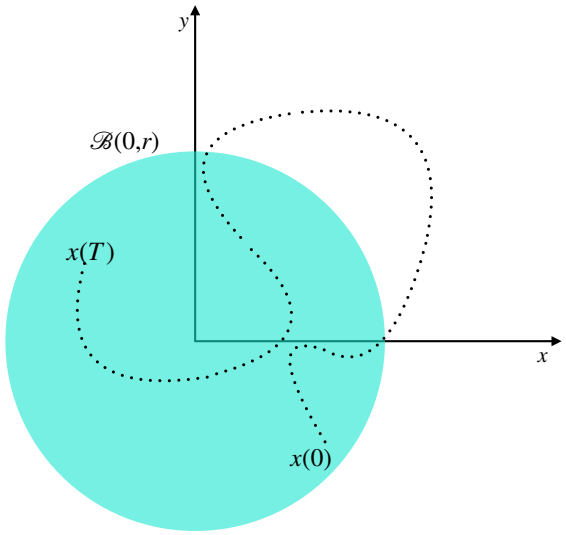


Figure 2.1: Example of predicate function

Example 2.2. Consider a given environment, characterized by the presence of obstacles. The desire of keeping a minimum and maximum distance to every obstacle can be written in STL as: $\varphi = \mathcal{G}\left(\text{dist}(\sigma, \mathcal{O}) - D_{\min} \wedge D_{\max} - \text{dist}(\sigma, \mathcal{O})\right)$, in which σ is a signal representing a path, $\text{dist}(\sigma, \mathcal{O})$ returns the Euclidean distance between a path σ and the closest obstacle $o \in \mathcal{O}$ and $\text{dist}(\sigma, \mathcal{O}) - D_{\min}$, $D_{\max} - \text{dist}(\sigma, \mathcal{O})$ are predicate functions $h_{\min}$ and $h_{\max}$, respectively. The corresponding predicates, $\mu_{\min}$ and $\mu_{\max}$, are evaluated as true if $\text{dist}(\sigma, \mathcal{O}) \geq D_{\min}$ and $\text{dist}(\sigma, \mathcal{O}) \leq D_{\max}$. In turn, the formula φ is evaluated as true if both $\mu_{\min}$ and $\mu_{\max}$ hold true for all times.

Qualitative semantics

Given the interval $I = [a, b]$, the STL qualitative semantics is defined recursively as follows:

$$
\begin{aligned}
(\boldsymbol{x}, t) &\models \mu && \Leftrightarrow h(\boldsymbol{x}(t)) \geq 0 \\
(\boldsymbol{x}, t) &\models \neg\varphi && \Leftrightarrow \neg((\boldsymbol{x}, t) \models \varphi) \\
(\boldsymbol{x}, t) &\models \varphi_1 \wedge \varphi_2 && \Leftrightarrow (\boldsymbol{x}, t) \models \varphi_1 \wedge (\boldsymbol{x}, t) \models \varphi_2 \\
(\boldsymbol{x}, t) &\models \varphi_1 \vee \varphi_2 && \Leftrightarrow (\boldsymbol{x}, t) \models \varphi_1 \vee (\boldsymbol{x}, t) \models \varphi_2 \\
(\boldsymbol{x}, t) &\models \mathcal{F}_I \varphi && \Leftrightarrow \exists t_1 \in t + I \text{ s.t. } (\boldsymbol{x}, t_1) \models \varphi \\
(\boldsymbol{x}, t) &\models \mathcal{G}_I \varphi && \Leftrightarrow \forall t_1 \in t + I, (\boldsymbol{x}, t_1) \models \varphi \\
(\boldsymbol{x}, t) &\models \varphi_1 \mathcal{U}_I \varphi_2 && \Leftrightarrow \exists t_1 \in t + I \text{ s.t. } (\boldsymbol{x}, t_1) \models \varphi_2 \wedge \forall t_2 \in [t, t_1], (\boldsymbol{x}, t_2) \models \varphi_1
\end{aligned}
$$

Time horizon

The time horizon of an STL formula is defined as:

$$
\text{th}(\varphi) = \begin{cases}
0 & \text{if } \varphi = \mu \\
\text{th}\left(\varphi_1\right) & \text{if } \varphi = \neg\varphi_1 \\
\max\left\{\text{th}\left(\varphi_1\right), \text{th}\left(\varphi_2\right)\right\} & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \text{ or } \varphi = \varphi_1 \vee \varphi_2 \\
b + \max\left\{\text{th}\left(\varphi_1\right), \text{th}\left(\varphi_2\right)\right\} & \text{if } \varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \\
b + \text{th}\left(\varphi_1\right) & \text{if } \varphi = \mathcal{G}_{[a,b]}\varphi_1 \text{ or } \varphi = \mathcal{F}_{[a,b]}\varphi_1
\end{cases} \tag{2.3}
$$

In this thesis, only time bounded temporal operators will be taken into account, i.e. $\text{th}(\varphi) < \infty$. The choice results to be not restrictive when considering real robots, given that in many robotic tasks an unbounded formula may not be feasible.

Validity domain

The validity domain $\text{vd}(\varphi)$ of an STL formula φ is recursively defined as:

$$\text{vd}(\varphi) = \begin{cases} 0, & \text{if } \varphi = \mu \\ \text{vd}\left(\varphi_1\right), & \text{if } \varphi = \neg\varphi_1 \\ [a, b], & \text{if } \varphi = \mathcal{G}_{[a,b]}\mu \\ a \oplus \text{vd}\left(\varphi_1\right), & \text{if } \varphi = \mathcal{G}_{[a,b]}\varphi_1, \varphi_1 \neq \mu \\ t^* + I^* \oplus \text{vd}\left(\varphi_1\right), & \text{if } \varphi = \mathcal{F}_{[a,b]}\varphi_1 \end{cases} \tag{2.4}$$

where $I^* \in [a, b]$ and $t^* = \left\{t \mid \varphi = \mathcal{F}_{[a,b]}\varphi_1\right\}$ is a variable whose initial value is 0, changing over time to save the last instance of satisfaction for the eventually operator [9]. If a path contains only predicates, then $\text{vd}\left(\varphi\right) = \text{th}(\varphi)$.

Example 2.3. Consider the following examples for the evaluation of the validity domain:

- $\varphi_1 = \mathcal{G}_{[3,8]}\mu$, then $\text{vd}\left(\varphi_1\right) = [3, 8]$. Hence the predicate μ must hold during all the interval of definition of the always operator, in order to have φ_1 satisfied;
- $\varphi_2 = \mathcal{F}_{[5,10]}\mu$, then $t^* = 0, I^* \in [5, 10]$ and $\text{vd}\left(\mu\right) = 0$. Therefore, $\text{vd}\left(\varphi_2\right) = I^* \in [5, 10]$ is the instance when μ is required to hold;
- $\varphi_3 = \mathcal{F}_{[5,10]}\mathcal{G}_{[0,3]}\mu$, then $t^* = 0, I^* \in [5, 10]$, $\text{vd}\left(\mathcal{G}_{[0,3]}\mu\right) = [0, 3]$. Therefore, $\text{vd}\left(\varphi_3\right) = 0 + I^* \oplus [0, 3] = [I^*, I^* + 3]$ is the interval over which μ needs to hold such that φ_3 is satisfied;
- $\varphi_4 = \mathcal{G}_{[2,10]}\mathcal{F}_{[0,5]}\mu$, then $a = 2$ and $\text{vd}\left(\varphi_4\right) = 2 \oplus \text{vd}\left(\mathcal{F}_{[0,5]}\mu\right) = 2 + 0 + I^*$ where $I^* \in [0, 5]$. For instance, if $I^* = 1$, then $\text{vd}\left(\varphi_4\right) = 3$ is the time instance when μ needs to hold. Once $\mu = \top$, then $t^* = I^*$ and the new $\text{vd}\left(\varphi_4\right) = 2 + 1 + I^*$ where $I^* \in [0, 5]$.

2.2.2 Robust Semantics

In addition to qualitative semantics, authors have suggested many forms of quantitative semantics, also known as robust semantics. This semantics allows to quantify the degree of satisfaction of the STL formula, i.e. given a signal $\boldsymbol{x}$ it determines how robustly $\boldsymbol{x}$ satisfies an STL formula φ at time t. For this reason, this semantics can be usefully exploited for the control of dynamical systems

under STL specifications, giving an upper bound on noise that can be added to the signal $\boldsymbol{x}$ before the Boolean evaluation of the semantics changes [8]. More specifically, the quantitative STL semantics returns a real value representing the distance to satisfaction or violation of the considered STL formula and it is defined as follows:

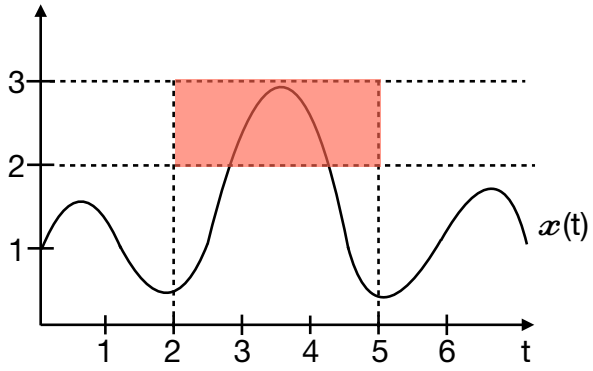
$$
\begin{aligned}
\rho\left(h(\boldsymbol{x}(t)) \geq 0, \boldsymbol{x}, t\right) &= h(\boldsymbol{x}(t)) \\
\rho\left(\neg\varphi, \boldsymbol{x}, t\right) &= -\rho(\varphi, \boldsymbol{x}, t) \\
\rho\left(\varphi_1 \wedge \varphi_2, \boldsymbol{x}, t\right) &= \min\left(\rho\left(\varphi_1, \boldsymbol{x}, t\right), \rho\left(\varphi_2, \boldsymbol{x}, t\right)\right) \\
\rho\left(\mathcal{G}_I\varphi, \boldsymbol{x}, t\right) &= \inf_{t_1 \in t+I} \rho\left(\varphi, \boldsymbol{x}, t_1\right) \\
\rho\left(\mathcal{F}_I\varphi, \boldsymbol{x}, t\right) &= \sup_{t_1 \in t+I} \rho\left(\varphi, \boldsymbol{x}, t_1\right) \\
\rho\left(\varphi_1 \mathcal{U}_I \varphi_2, \boldsymbol{x}, t\right) &= \sup_{t_1 \in t+I} \min\left(\rho\left(\varphi_2, \boldsymbol{x}, t_1\right), \inf_{t_2 \in (t, t_1)} \rho\left(\varphi_1, \boldsymbol{x}, t_2\right)\right)
\end{aligned}
$$

A signal $\boldsymbol{x}$ satisfies an STL formula φ at time t if and only if $\rho(\varphi, \boldsymbol{x}, t) \geq 0$. Clearly, the greater the value of ρ, the more robust the signal is. In other words, the signal not only satisfies the STL formula, but it is also robust to eventual perturbations. On the other hand, if the value of ρ is positive but close to 0, in the next time instants it could happen to violate the constraints encoded in the STL formula.

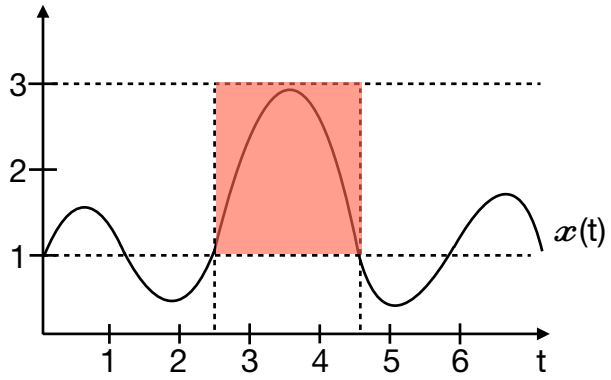
Example 2.4. Consider a continuous-time signal $\boldsymbol{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ and the three STL formulas

$$
\begin{aligned}
\varphi_1 &:= \mathcal{F}_{[2,5]}(2 \leq \boldsymbol{x} \leq 3) \\
\varphi_2 &:= \mathcal{G}_{[2.5,4.6]}(1 \leq \boldsymbol{x} \leq 3) \\
\varphi_3 &:= \mathcal{G}_{[2,5]}(0.4 \leq \boldsymbol{x} \leq 2.5).
\end{aligned}
$$

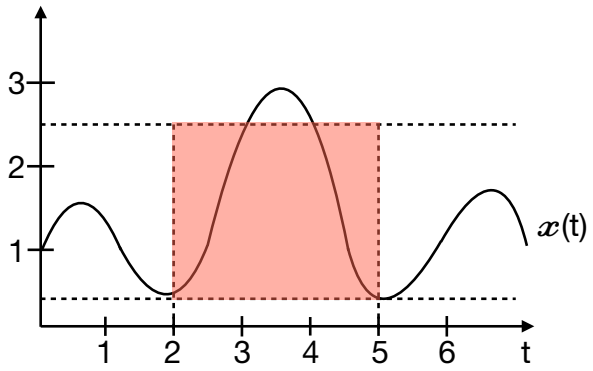
From Figure 2.2, it can be seen that φ_1 and φ_2 are both satisfied by $\boldsymbol{x}$ at time 0, namely $(\boldsymbol{x}, 0) \models \varphi_1$ and $(\boldsymbol{x}, 0) \models \varphi_2$. However, the formula φ_1 is satisfied more robustly than φ_2 since $\rho(\varphi_1, \boldsymbol{x}, 0) > \rho(\varphi_2, \boldsymbol{x}, 0)$. Indeed only a small noise added to $\boldsymbol{x}(t)$ at $t = 4$ will lead to $(\boldsymbol{x}, 0) \not\models \varphi_2$, while a small noise will not affect the satisfaction of φ_1. The formula φ_3, instead, is not satisfied, since $\boldsymbol{x}$ does not result to be between 0.4 and 2.5 at every instant in the time interval [2, 5].



(a) $\varphi_1 := \mathcal{F}_{[2,5]}(2 \leq \boldsymbol{x} \leq 3)$ is satisfied by $\boldsymbol{x}$



(b) $\varphi_2 := \mathcal{G}_{[2.5,4.6]}(1 \leq \boldsymbol{x} \leq 3)$ is satisfied by $\boldsymbol{x}$



(c) $\varphi_3 := \mathcal{G}_{[2,5]}(0.4 \leq \boldsymbol{x} \leq 2.5)$ is not satisfied by $\boldsymbol{x}$

Figure 2.2: Three examples regarding robust semantics of STL

Example 2.5. Consider again the specification of Example 2.2. Suppose it is desired that a robot stays 3 m away from every obstacle during its exploration task, written in STL as $\psi = \mathcal{G}\varphi$, and $\varphi = \text{dist}(\sigma, \mathcal{O}) - 3$. Let $\sigma(\alpha)$ be a path such that (i) $\text{dist}(\sigma(0), \mathcal{O}) = 5$, (ii) $\text{dist}(\sigma(0.3), \mathcal{O}) = 2$ and (iii) $\text{dist}(\sigma(1), \mathcal{O}) = 0.5$. In

each individual case, the robustness of the path with respect to φ is calculated as $\rho(\varphi, \sigma, \alpha) = h(\sigma(\alpha))$, and results in (i) $\rho(\varphi, \sigma, 0) = 2$, (ii) $\rho(\varphi, \sigma, 0.3) = -1$ and (iii) $\rho(\varphi, \sigma, 1) = -4.5$ [6]. Finally, the robustness of the path with respect to the formula ψ is calculated as follows:

$$\begin{aligned}\rho(\psi, \sigma) &= \min_{\alpha \in [0,1]} \rho(\varphi, \sigma, \alpha) \\ &= \min(\rho(\varphi, \sigma, 0), \rho(\varphi, \sigma, 0.3), \rho(\varphi, \sigma, 1)) \\ &= -4.5\end{aligned}$$

2.2.3 STL Parse Tree

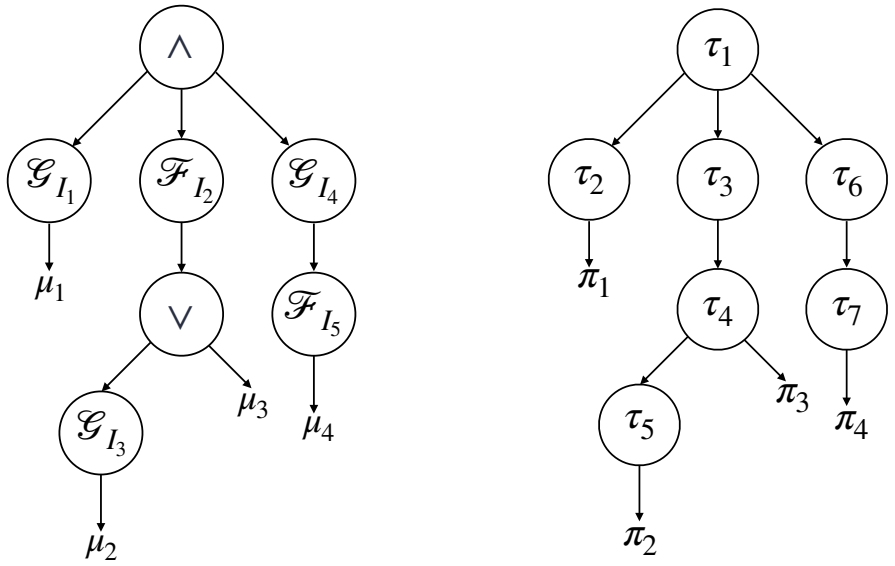
An STL formula can be easily represented as a tree, called STL Parse Tree. Each node represents either a temporal operator (always, eventually), a logical operator (conjunction, disjunction, negation) or a predicate (μ). In particular, leaf nodes constitute the predicate nodes of the tree, while the set nodes are operators and are accompanied by a satisfaction variable $\tau \in \{-1, +1\}$. The satisfaction variable tree has the same structure as the parse tree; each set node is associated with a satisfaction variable τ_i, while each leaf node is associated with a predicate variable π_i. A path is a formula from a root node to a leaf node; a sub-path, instead, is a path from a set node to a leaf node. A signal $\boldsymbol{x}$ satisfies a sub-path if the set node, corresponding to the beginning of the path, has $\tau = +1$ [9].

Example 2.6. Consider the following STL formula:

$$\varphi = \mathcal{G}_{I_1}\left(\mu_1\right) \wedge \mathcal{F}_{I_2}\left(\mathcal{G}_{I_3}\left(\mu_2\right) \vee \mu_3\right) \wedge \mathcal{G}_{I_4}\mathcal{F}_{I_5}\left(\mu_4\right)$$

The STL parse tree and the satisfaction variable tree for the STL formula are shown in Figure 2.3. Suppose $\tau_3 = +1 \implies \boldsymbol{x} \models \mathcal{F}_{I_2}\left(\mathcal{G}_{I_3}\left(\mu_2\right) \vee \mu_3\right)$. Moreover, if $\tau_2 = +1 \implies \boldsymbol{x} \models \mathcal{G}_{I_1}\left(\mu_1\right)$. Hence the following equivalence holds:

$$(\boldsymbol{x}, t) \models \varphi \Leftrightarrow \rho(\varphi, \boldsymbol{x}, t) \geq 0 \Leftrightarrow \tau_6 = +1 \Leftrightarrow \tau(\text{root}) = \tau_1 = +1.$$



(a) STL parse tree

(b) Satisfaction variable tree

Figure 2.3: Example of STL parse tree and satisfaction variable tree

2.3 Control under Temporal Logic

The need of autonomous systems, able to address complex and time-constrained tasks in dynamic environments, has led to the development of efficient and expressive languages to describe objectives under strict deadlines and encode such specifications for control. Sophisticated and complicated tasks, such as sequential or reactive ones, including qualitative and quantitative spatio-temporal properties, are typically formulated in temporal logics. In the last years, a great amount of temporal logics, including Linear Temporal Logic (LTL), Metric Temporal Logic (MTL) and Signal Temporal Logic (STL), has been considered and exploited in control problems.

It is possible to distinguish between two categories of methods for temporal logic control: automata-based and optimization-based. In the first case, the system must be finitely abstracted and the temporal logic specifications are represented through an automaton. Even though promising results have been achieved exploiting this kind of approach, automata-based solutions are generally very computationally expensive. In the second case, instead, the definition of quantitative semantics for temporal logics is exploited, in order to compute a real-value (robustness metric) that measures how strongly a specification is satisfied or vi-

olated. As a consequence, the solution of the control problem coincides with the solution of an optimization problem, whose objective is the one of maximizing the robustness [10].

In the next two sections, different temporal logics are compared: the focus is on the drawbacks of LTL and MTL and the advantages of STL specifications.

2.3.1 Drawbacks of Linear and Metric Temporal Logic

Temporal logics can be naturally exploited to express traditional robot specifications, such as reaching a goal or avoiding an obstacle. To obtain satisfaction of MTL specifications, abstraction-based approaches are required. As already presented before, these methods are characterized by the translation of the temporal logic specification into an automata representation and by the abstraction of the system and the environment into a finite transition diagram. Based on these abstractions, algorithms are derived for verification and synthesis of discrete controllers, that drive the system to satisfy the specification. Despite their success in the correct-by-construction design of controllers, abstraction-based approaches could be affected by ”state explosion” issues. Indeed both the synthesis and abstraction algorithms scale at least exponentially with the dimension of the discretized configuration space (curse of dimensionality), leading to impracticable application of such approaches for large-scale multi-agent systems [11].

In order to provide computational solutions to motion planning problems under LTL specifications, it is necessary to operate in discrete time. For example, authors in [5] construct discrete abstractions of robot motion, generate discrete plans satisfying the temporal logic formula and finally translate the latter to continuous trajectories using hybrid control. Critical to this approach is providing formal guarantees ensuring that, if the discrete plan satisfies the temporal logic formula, then the continuous motion also satisfies the exact same formula.

Another drawback of standard temporal logics concerns the fact that they only involve qualitative temporal operators and cannot deal with quantitative temporal requirements. Indeed, LTL specifications require only the order of events that should be executed by the system, neglecting temporal distance between

them. MTL and STL, compared with LTL, quantitatively take time into account. In this way, more complex specifications, including specific time requirements, can be formulated. In fact, MTL and STL allow to impose time-constrained tasks, such as strict deadlines [12]. Typical examples of objectives, expressed through STL or MTL formulas, are:

- Maximal distance between an event and its reaction, e.g. event A is followed by event B within 5 time units;
- Exact distance between events, e.g. event A is followed by event B in exactly 3 time units;
- Minimal distance between events, e.g. two consecutive events A are at least 7 time units apart;
- Periodicity, e.g. event A occurs regularly with a period of 5 time units [13].

Approaches such as MTL, however, usually define tasks over finite-state spaces, requiring the abstraction of the underlying continuous -time and -space systems to discrete one. As explained before, this abstraction could lead to possible loss of information and risk of ”state explosion” [9]. Moreover, MTL and LTL are proposition-based logics, hence resulting to be less expressive than STL, which is instead a predicate-based logic.

Using STL avoids the aforementioned drawbacks and allows to express specifications in continuous time and space. The advantages of such approach are discussed below more in detail.

2.3.2 Advantages of Signal Temporal Logic

STL is a predicate-based logic interpreted over continuous-time signals and allows to impose desired spatial and temporal properties. STL specifications can be exploited to specify tasks that systems may or may not satisfy, based on logic operators (negation, disjunction, conjunction) and temporal operators (eventually, always, until). Additionally, it is possible to include combinations of surveillance (“Visit regions A and B every 8 seconds, while agents are in triangular formation”) and safety (“Stay at least 4 meters away from region B in the time interval [10 30] seconds”).

Furthermore, contrary to LTL and MTL, STL offers tools to evaluate how much the task is satisfied or not over a continuous-time signal, avoiding the abstraction of the system dynamics. As already presented in Section 2.2.2, one of the main advantages relies on the fact that STL permits to introduce robustness notions, i.e. to specify how severely a given specification must be satisfied. The idea, hence, is to be able to access how much of the specification is satisfied and not just whether it is satisfied. Such quantitative semantics has been successfully used for implementing optimization-based synthesis and efficiently assessing satisfaction [14]. The quantitative semantics provides a useful tool for the control of dynamical systems under STL specifications, giving an upper bound on noise that can be added to the signal x before the evaluation of the semantics changes, namely before the satisfied formula becomes violated or vice versa. The robustness degree then gives a measure of how much a signal can be perturbed before changing the truth value of the specification [8].

2.4 Optimal Motion Planning

Given a starting position A and a goal position B, motion planning algorithms aim at finding a trajectory from A to B, such that the robot avoids obstacles along the path and satisfies constraints on the workspace. However, a feasible solution is not sufficient if its quality is also important. For instance, in many industrial applications, the users are interested in solution paths with the minimum time to execute, so as to achieve the highest productivity. This requirement results in the problem of optimal motion planning, i.e. computing motion plans that are of the minimum cost with respect to a given cost functional, such as the length of a path or the consumed energy [15].

In many applications, given the high dimensionality of the search space, the geometric properties of the obstacles, the cost to be optimized, the dynamic and kinematic model of the robot, online computation may be not feasible [16]. This has led to the development of two branches of research in the area of motion planning: on-line planning and off-line planning. In the former case, the trajectory is computed during motion; these kinds of planners are used if obstacles are discovered while the robot is moving and the environment is changing as time passes. In the latter case, instead, the trajectory is obtained before motion begins. In many industrial applications, where tasks are repeated in static envi-

ronments, off-line planners are widely used.
In order to simplify the study of STL constraints for the coupled agents, this thesis considers a known workspace with static obstacles. The two probabilistic optimal trajectories are computed off-line: when the maximum number of iterations is reached, the algorithm returns the sequence of optimal control inputs (with corresponding time intervals) that allows each agent to follow the probabilistic optimal trajectory.

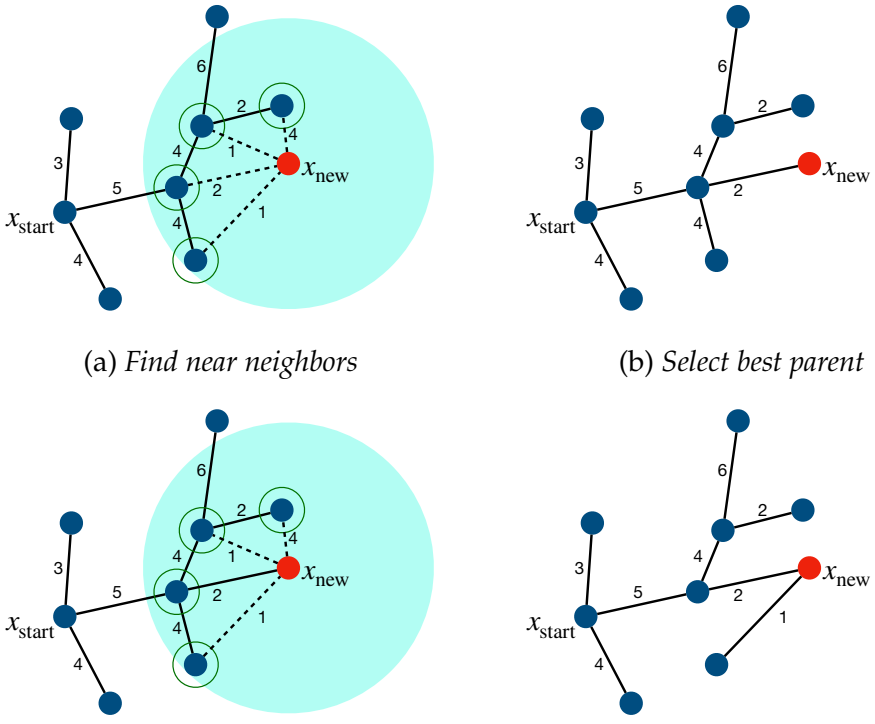
2.4.1 RRT*

Even when dynamics is not considered, motion planning is PSPACE-hard. Complete algorithms, providing solutions if they exist, scale exponentially with the number of degrees of freedom of the robots, resulting in computational hardness. Hence, practical approaches relax completeness guarantee, in order to avoid the issues deriving from the curse of dimensionality. In the last years, sampling-based planners have become a powerful tool for the solution of path and motion planning problems. With respect to grid-based algorithms such as A* and D* or potential field approaches, sampling-based planning methods are characterized by low computational cost, applicability to large scale problems and probabilistic completeness [17].

One of the most used sampling-based planners is Rapidly-exploring Random Tree (RRT), proposed by LaValle and Kuffner in [18]. The basic idea of the simple RRT algorithm is to sample random states and extend the tree toward them. In each iteration, a random state is selected from the metric space. Then, the nearest neighbor is found in the current tree, according to a predefined metric. At this point, a control u is computed, in order to move from the selected closest vertex toward the random point. Finally, the new edge and the new vertex are inserted, resulting hence in a tree that explores the state space in a uniform and rapid way.

However, basic RRT algorithm usually finds a feasible solution very quickly, with no guarantee on the quality of the solution, given that it does not use any metric to measure the motion optimality between subsequent states. To solve the issue, authors in [19] have proposed the optimal version of RRT, called RRT*, and showed its globally asymptotically optimality. The two key additions, with respect to the standard RRT, are the near neighbor search and the rewiring pro-

cedure (see Figure 2.4). For what regards the first one, it finds the best parent vertex to the new node, considering a circle of radius r. Inside this region, in particular, the parent is selected in order to obtain the minimum cost path from the initial node to the new one, instead of taking the nearest node as parent by default, as it happens in RRT. Concerning the second feature of RRT*, neighbors of the new connected vertex are considered. More specifically, if the path to a neighbor vertex, considering the new node as parent, has a cost that is lower with respect to the path passing through the current parent of the neighbor vertex, then the neighbor vertex is rewired. This means that the edge between the neighbor node and its parent is removed, while a new edge is created, selecting the new added node as its new parent.



(a) *Find near neighbors* (b) *Select best parent*

(c) *Check cost for rewiring* (d) *Rewired tree*

Figure 2.4: Near neighbor search and rewiring operation in RRT*

In recent years, a large variety of RRT*-based approaches has been proposed. Among them, it is possible to cite RRT*FN (deals with the problem of high memory consumption, allowing a limited number of nodes in the tree), B-RRT* (generates two trees simultaneously, one from the start position and one from

the goal position) and Adapted RRT* (solves optimal path planning for car-like robots, dealing with non-holonomic or kinodynamic constraints). For a more complete survey on RRT*-based approaches, it is possible to consult [17].

The pseudo-code of the RRT* algorithm is reported in Algorithm 1 [19]. $G = (V, E)$ is the tree that is built incrementally by the algorithm, where V is the set of nodes and E is the set of edges.

Algorithm 1: RRT*

$V \leftarrow \{x_{\text{start}}\}$;
$E \leftarrow \emptyset$;
for $k = 1, ..., K$ **do**
  $G \leftarrow (V, E)$;
  $x_{\text{rand}} \leftarrow$ `SampleFree`;
  $x_{\text{nearest}} \leftarrow$ `Nearest` (G, x_{rand});
  $x_{\text{new}} \leftarrow$ `Steer` $(x_{\text{nearest}}, x_{\text{rand}})$;
  if `NoCollision` $(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow$ `Near` (G, x_{new}, r);
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
    $c_{\text{min}} \leftarrow$ `Cost` $(x_{\text{nearest}}) + c($ `Line` $(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do**
      if `NoCollision` $(x_{\text{near}}, x_{\text{new}}) \wedge$ `Cost` $(x_{\text{near}}) + c($ `Line` $(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
        $c_{\text{min}} \leftarrow$ `Cost` $(x_{\text{near}}) + c($ `Line` $(x_{\text{near}}, x_{\text{new}}))$;
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do**
      if `NoCollision` $(x_{\text{new}}, x_{\text{near}}) \wedge$ `Cost` $(x_{\text{new}}) + c($ `Line` $(x_{\text{new}}, x_{\text{near}})) <$ `Cost` (x_{near}) **then**
        $x_{\text{parent}} \leftarrow$ `Parent` (x_{near});
        $E \leftarrow (E \backslash \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G = (V, E)$;

Chapter 3

Related Work

3.1 Synthesis of Controllers under Signal Temporal Logic

Given a system subject to constraints expressed through temporal logic formulas, the satisfaction of the specifications has been checked in the past years using formal verification techniques. Recently, the need for computationally efficient control methods under temporal logic tasks has become more apparent, in particular when multi-agent systems are considered. Existing approaches, such as model checking, only examine if the given tasks are satisfied by the system and increase exponentially with the dimension of the model. The computational burdens of model checking can be relaxed using sampling-based planners, control barrier functions and prescribed performance control.

3.1.1 Sampling-based Methods

Sampling-based methods have helped the improvement and progress in robotic motion planning and control, also facing high-dimensional domains. Most of sampling-based methods exploit Rapidly-exploring Random Trees (RRT) or Probabilistic Roadmaps (PRM) for temporal logic synthesis. The two algorithms presented in the next section solve the motion planning problem under STL specifications, extending RRT to include the STL formula.

STLcoRRT

In sampling-based planners, the free space is modeled using graphs, composed by nodes and edges. Control problems, such as motion planning problem, can be hence reduced to a graph-search problem (find a path in the graph). This sampling procedure is exploited by authors in [9], in order to develop a distributed algorithm for cooperative motion planning of two coupled agents under STL specifications. In particular, the proposed algorithm builds incrementally a spatio-temporal tree for each agent, by sampling points in the coupled state and time domain of the agents. Before adding a new edge, it checks if it satisfies certain parts of the given task. By doing so, the obtained time-varying trajectories, represented by the two spatio-temporal trees, satisfy the given task.

The algorithm, STLcoRRT, is a variation of the standard Rapidly-exploring Random Tree algorithm. While the latter samples a point $\boldsymbol{x}_{\text{samp}}$ from $\mathbb{R}^n$, finds its nearest neighbor in the existing tree and creates a new vertex by drawing an edge of predefined length [20], the key element of the former is the incorporation of time in the sampling process, leading to a time-augmented sampling method. The generated tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, consequently, is a spatio-temporal tree. Each vertex in the set $\mathcal{V} = \{z_1, z_2, \ldots\}$ is given by $z_i = (t_i, \boldsymbol{x}_i) \in \mathcal{Z} \subset \mathbb{R}_+ \times \mathbb{R}^n$. Clearly, the time associated with a new sample vertex must be greater than the time associated with the nearest node of the tree; in other words, every new sample z_{new} must be ahead in time with respect to z_{nearest}. Moreover, differently from the original sampling-based algorithm, the obstacle collision checking is substituted with a procedure that checks upon satisfaction or violation of the STL formula for each added edge.

More specifically, STLcoRRT algorithm includes two components: the sampling-based procedure to build a tree and the verification of the satisfaction of the STL formula. Each sample corresponds to a $(n+1)$-dimensional point $\boldsymbol{z} = (t, \boldsymbol{x})$; if it is feasible with respect to the STL formula φ, it is added as a vertex to the tree.

Considering the point of view of agent i, after the initialization of the tree, agent i requests the updated tree of agent j. Then, the sampling procedure is implemented, through three standard functions taken from the original RRT algorithm. At this point, two scenarios arise: either agent i samples in a time

region where agent j has already explored, or it samples in a time region where agent j is yet to explore. Depending on the situation, different functions are exploited to decide the satisfiability of the STL formula. Both the agents simultaneously run STLcoRRT and communicate with the other agent after every edge addition. The final module is the verification of the STL formula based on the sampled points.

The proposed algorithm solves the cooperative motion planning problem under the entire fragment of signal temporal logic for continuous -time and -state systems, without resorting to discretization techniques. However, dynamics of the two agents and explicit control design are not taken into account.

Algorithm based on Direction of Increasing Satisfaction

The dynamics of the system is directly taken into account, instead, by authors in [14], where a different sampling-based approach is proposed. The key is to guide and influence the selection of samples using a quantitative measure of how well and robustly the specification is satisfied by the best path in the current tree of samples. This allows to converge to a path that indeed satisfies the task expressed as an STL specification with maximum robustness. By defining the Direction of Increasing Satisfaction (DIS), it is possible to find the most promising direction of exploration to improve the robustness of the STL formula, given a partial trajectory. In particular, two functions are exploited when conjunctions are taken into account: the choice function decides which of the two input formulae yields the largest robustness gain for their conjunction; the blending function combines the two directions computed for the two sub-formulae of the conjunction and should give priority to the one returned by the choice function. Relying on quantitative semantics of the logic, it is possible to guide sampling.

Given a dynamical system $\mathcal{R} = \left(f, X, U, \boldsymbol{x}_{\text{init}}\right)$ and an STL specification, the aim is to find a control policy u such that the robustness degree of the state trajectory, originating in $\boldsymbol{x}_{\text{init}}$ under u, is maximized with respect to the STL specification. The algorithm is based on RRT* and generates a tree of state-formula pairs. States are generated in a random way, while STL formulas associated with them indicate the progress towards the satisfaction of the overall STL specification. To obtain maximal satisfaction, biased sampling and guided steering are exploited.

More specifically, to guide the steering of the system, authors take into account a random convex combination between states that have been randomly generated and states along the DIS. Finally, an online monitoring algorithm for STL formulas is developed.

After the initialization, in each iteration, the algorithm randomly samples a time and state and attempts to add to the existing tree a new vertex that minimizes a convex combination of random sampling and moving along the DIS. Once the new node is added, near vertices are considered and eventually rewiring is performed, in order to take into account the best path to reach vertices.

With respect to STLcoRRT, this sampling-based algorithm considers only a single agent and linear predicates. On the other hand, dynamics of the system and robustness of the STL specification are taken into account. As it will be explained in Chapter 4, the proposed solution for the motion planning problem of coupled agents addressed in this thesis will exploit the advantages of both algorithms, allowing to drive the multi-agent system through probabilistic optimal trajectories.

3.1.2 Control Barrier Functions

Control barrier functions were introduced as a tool to formally prove safety of hybrid systems. Intuitively, an environment is safe if dangerous things do not happen. Given the locus where damaging things could happen, it is possible to exploit invariance to guarantee safety properties of the system. Indeed, if a set is invariant, any trajectory starting inside it will never reach the complement of the set, i.e. the unsafe one [21]. The concept of safety is of extreme importance when considering autonomous systems, especially the ones operating in unknown and unstructured environments and cooperating with humans. For control systems and based on the notion of barrier functions, control barrier functions have first been presented to guarantee the existence of a control law that renders a desired safe set forward invariant. They are related to Control Lyapunov Functions, but instead of stability they guarantee that the trajectories of a system remain in a pre-defined forward invariant set [22]. In order to encode STL tasks into a control barrier function, it is necessary to design the temporal properties of the control barrier function in order to account for the STL semantics.

Consider a dynamical system of the form:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}) + g(\boldsymbol{x})\,\boldsymbol{u}, \boldsymbol{x}(t_0) := \boldsymbol{x}_0 \tag{3.1}$$

where f and g are locally Lipschitz continuous functions. Let $\mathfrak{b} : \mathfrak{D} \times [t_0, t_1] \to \mathbb{R}$ be a continuously differentiable function, where $\mathfrak{D} \subseteq \mathbb{R}^n$, and define the set $\mathfrak{C}(t) := \{\boldsymbol{x} \in \mathfrak{D} \mid \mathfrak{b}(\boldsymbol{x}, t) \geq 0\}$.

The continuously differentiable function $\mathfrak{b}(\boldsymbol{x}, t)$ is said to be a candidate control barrier function if for each $\boldsymbol{x}_0 \in \mathfrak{C}(t_0)$ there exists an absolutely continuous function $\boldsymbol{x} : [t_0, t_1] \to \mathbb{R}^n$ with $\boldsymbol{x}(t_0) := \boldsymbol{x}_0$, such that $\boldsymbol{x}(t) \in \mathfrak{C}(t)$ for all $t \in [t_0, t_1]$.

A candidate control barrier function $\mathfrak{b}(\boldsymbol{x}, t)$ is a valid control barrier function for (3.1) if there exists a locally Lipschitz continuous class $\mathcal{K}$ function α such that, for all $(\boldsymbol{x}, t) \in \mathfrak{C}(t) \times [t_0, t_1]$,

$$\sup_{\boldsymbol{u} \in \mathcal{U}} \frac{\partial \mathfrak{b}(\boldsymbol{x}, t)^T}{\partial \boldsymbol{x}}(f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}) + \frac{\partial \mathfrak{b}(\boldsymbol{x}, t)}{\partial t} \geq -\alpha(\mathfrak{b}(\boldsymbol{x}, t)) \tag{3.2}$$

Recall that a class $\mathcal{K}$ function $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a continuous and strictly increasing function with $\alpha(0) = 0$. An extended class $\mathcal{K}$ function is a function $\alpha : \mathbb{R} \to \mathbb{R}$ that is again continuous and strictly increasing with $\alpha(0) = 0$.

If $\boldsymbol{u}$ is locally Lipschitz continuous in $\boldsymbol{x}$ and piecewise continuous in t and it belongs to the set of control inputs S_u that guarantee the satisfaction of formula 3.2, then the set $\mathfrak{C}(t)$ is forward invariant for the control law $\boldsymbol{u}(\boldsymbol{x}, t)$ if $\mathfrak{b}(\boldsymbol{x}, t)$ is a valid control barrier function. This means that, considering a trajectory starting inside $\mathfrak{C}$, i.e. $\boldsymbol{x}_0 \in \mathfrak{C}(t_0)$, $\boldsymbol{x}(t)$ will remain inside $\mathfrak{C}(t)$ for all $t \in [t_0, t_1]$. In applications, such as safety control, this leads to the fact that the trajectory of the dynamical system remains inside the desired safe region.

Authors in [23] have shown that time-varying control barrier functions can be used to satisfy STL formulas for single-agent systems, by leveraging the temporal properties of $\mathfrak{b}(\boldsymbol{x}, t)$. Their main contribution is a control strategy that offers a good trade-off between computational efficiency and expressivity of the STL fragment under consideration.

In a first step, time-varying control barrier functions are defined; in a second step, they are used to satisfy STL tasks. The temporal properties of the control barrier functions hence need to be designed in a way that accounts for the STL semantics. The main idea is described in the next example.

Example 3.1. Consider the formula $\varphi := \mathcal{F}_{[5,15]}\left(\|\boldsymbol{x} - [10 \quad 0]^T\| \leq 5\right)$ and let $t_0 := 0$ and $t_1 := 15$. Note that the corresponding predicate function is $h(\boldsymbol{x}) := 5 - \|\boldsymbol{x} - [10 \quad 0]^T\|$. Also consider, without loss of generality, an initial condition $x(0) := [0 \quad 0]^T$. For the candidate control barrier function $\mathfrak{b}(\boldsymbol{x}, t) := \gamma(t) - \left\|\boldsymbol{x} - [10 \quad 0]^T\right\|$ with $\gamma(t) := -\frac{5}{15}t + 10$, it holds that $\mathfrak{b}(\boldsymbol{x}(0), 0) = 0$ and hence $\boldsymbol{x}(0) \in \mathfrak{C}(0)$. If there exists a control law $\boldsymbol{u}(\boldsymbol{x}, t)$ so that the solution to 3.1 satisfies $\boldsymbol{b}(\boldsymbol{x}(t), t) \geq 0$ for all $t \in [t_0, t_1]$, e.g. when $\mathfrak{b}(\boldsymbol{x}, t)$ is a valid control barrier function, then $(\boldsymbol{x}, 0) \models \varphi$ follows. Note therefore that $\gamma(t_1) = 5$ and hence $\mathfrak{b}(\boldsymbol{x}, t_1) = 5 - \left\|\boldsymbol{x} - [10 \quad 0]^T\right\| = h(\boldsymbol{x})$. This means that $\mathfrak{b}(\boldsymbol{x}(t_1), t_1) \geq 0$ implies that $\left\|\boldsymbol{x}(t_1) - [10 \quad 0]^T\right\| \leq 5$, which yields $(\boldsymbol{x}, 0) \models \varphi$.

From the example above, it results that $\mathfrak{b}(\boldsymbol{x}, t)$ is associated with the predicate function $h(\boldsymbol{x})$ and guarantees the satisfaction of the STL formula. However, in order to add conjunctions, a smooth under-approximation of the min-operator is used. In the general case, the control barrier function $\mathfrak{b}(\boldsymbol{x}, t)$ has the form:

$$\mathfrak{b}(\boldsymbol{x}, t) := -\ln\left(\sum_{i=1}^{p} \exp\left(-\mathfrak{b}_i(\boldsymbol{x}, t)\right)\right) \tag{3.3}$$

where p is the number of functions $\mathfrak{b}_i(\boldsymbol{x}, t)$, each corresponding to either an always, eventually, or until operator with time interval $[a_i, b_i]$. To avoid conservatism of the control barrier function when a large number of tasks is considered, it is possible to deactivate the functions $\mathfrak{b}_i(\boldsymbol{x}, t)$ when the corresponding temporal operators are satisfied (at $t = b_i$, namely the right extreme of the interval of definition of the operator itself). The control barrier function of the exponential form is hence piecewise continuous in t and characterized by a hybrid time domain.

Let T_i denote the time interval at which $\mathfrak{b}_i(\boldsymbol{x}, t)$ contributes to $\mathfrak{b}(\boldsymbol{x}, t)$. The deactivation can be encoded by exploiting the integer functions $o_i(t)$ defined as:

$$o_i(t) = \begin{cases} 1, & t \in T_i \\ 0, & t \notin T_i \end{cases} \tag{3.4}$$

As presented by authors in [24], the modified barrier function can be written as:

$$\mathfrak{b}(\boldsymbol{x}, t) = -\ln\left(\sum_{i=1}^{p} o_i(t)\exp\left(-\mathfrak{b}_i(\boldsymbol{x}, t)\right)\right) \tag{3.5}$$

In [25], the results presented above are extended to multi-agent systems. More specifically, authors propose a decentralized control barrier function-based feedback control law for continuous-time multi-agent systems under a set of STL tasks. Time-varying control barrier functions are hence expanded to systems with discontinuous control inputs; the control law is finally obtained by solving a computationally tractable convex quadratic program.

Control barrier functions can be constructed to account for a fragment of STL tasks and also maximize the robustness by which the STL specification is satisfied. However, on the other hand, it is also true that these functions are designed such that they under-approximate the predicate functions of the STL task. Moreover, they can deal with only a small fragment of STL specifications, without taking into account disjunctions.

3.1.3 Prescribed Performance Control

Prescribed Performance Control (PPC) is a funnel-based feedback control strategy, that constrains a generic tracking error $\boldsymbol{e} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^n$ to a time-varying funnel, specifically designed by the user. By prescribed performance, authors in [26] mean that the output tracking error converges to a predefined residual set, with convergence rate no less than a specified value and maximum overshoot less than some constant. They also prove that, to solve the prescribed performance tracking problem of a dynamical system, it is sufficient to achieve bounded states of a novel system, involving both non linearities of the original one and the desired performance characteristics.

PPC can be applied to dynamical systems, in order to impose a desired transient behavior of the system trajectories and satisfy STL specifications. Authors in [27] consider a non-linear system subject to a subset of STL specifications and translate this constrained control problem into a PPC one, ensuring satisfaction of the temporal formulas.

Let $\boldsymbol{e}(t) := \boldsymbol{x}(t) - \boldsymbol{x}_d(t)$ be the tracking error, defined as the difference between the achieved trajectory and the desired one. To prescribe transient and steady-state behavior of this error, it is possible to define two functions, i.e. the performance function γ and the transformation function S.

Firstly, a performance function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{>0}$ is a continuously differentiable, bounded, positive and non-increasing function, given by:

$$\gamma(t) := (\gamma_0 - \gamma_\infty) \exp(-lt) + \gamma_\infty \tag{3.6}$$

where $\gamma_0, \gamma_\infty \in \mathbb{R}_{>0}$ with $\gamma_0 \geq \gamma_\infty$ and $l \in \mathbb{R}_{\geq 0}$ [25].

The task is to synthesize a feedback control law such that, given $-\gamma_i(0) < e_i(0) < M\gamma_i(0)$ for all $i \in \{1, \ldots, n\}$, the errors e_i satisfy the same chain of inequalities $\forall t \in \mathbb{R}_{\geq 0}$:

$$-\gamma_i(t) < e_i(t) < M\gamma_i(t) \tag{3.7}$$

with $0 \leq M \leq 1$ and γ_i being a performance function as defined in 3.6.

Then, given the normalized error $\xi_i(t) := \frac{e_i(t)}{\gamma_i(t)}$, it is possible to define the transformation function $S : (-1, M) \to \mathbb{R}$ as:

$$S(\xi) := \ln\left(-\frac{\xi + 1}{\xi - M}\right) \tag{3.8}$$

Dividing 3.7 by γ_i and applying the transformation function S results in an unconstrained control problem $-\infty < S(\xi_i(t)) < \infty$ with the transformed error $\epsilon_i(t) := S(\xi_i(t))$. If the latter is bounded for all $t \geq 0$, then e_i satisfies 3.7.

To better understand how the performance function $\gamma(t)$ can be exploited to impose a transient behavior on the tracking error $\boldsymbol{e}(t)$, it is possible to observe the evolution of the performance functions $-\gamma(t)$ and $M\gamma(t)$ in Figure 3.1. Notice that the scalar error $e_i(t)$ is indeed contained in the funnel defined by $M := 0.8$ and $\gamma_i(t) := (\gamma_\infty - \gamma_0) \exp(-lt) + \gamma_0$ with $\gamma_\infty := 10$, $\gamma_0 := 1$, and $l := 0.2$.

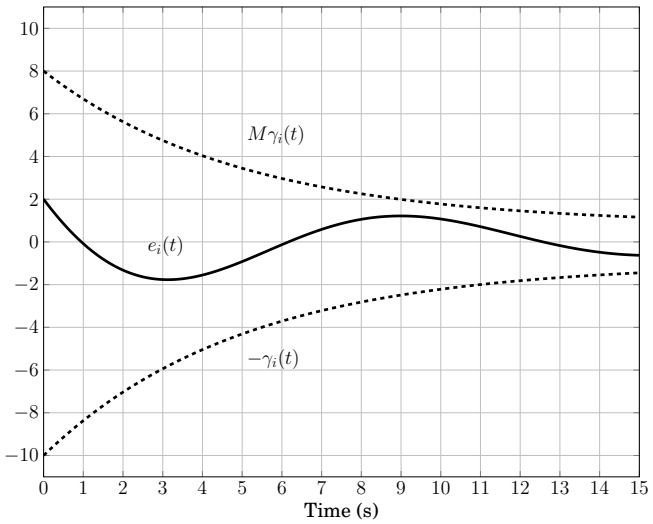


Figure 3.1: Evolution of a scalar error contained in a funnel

At this point the objective is to synthesize a continuous feedback control law $\boldsymbol{u}(\boldsymbol{x}, t)$ for atomic STL formulas of the kind:

$$\psi := \top \mid \mu \mid \neg\mu \mid \psi_1 \wedge \psi_2$$
$$\varphi := G_{[a,b]}\psi \mid F_{[a,b]}\psi$$

STL formulas as φ are known as atomic temporal formulas, due to use of eventually and always operators that makes t appear explicitly in the robustness metric $\rho(\varphi, \boldsymbol{x}, t)$. Instead, STL specifications as ψ are known as non-temporal formulas, i.e. Boolean formulas.

The approach is to leverage the funnel control idea presented before and replace the tracking error, that is supposed to evolve within the funnel, by the robustness semantics of the STL specification [8]. Considering the STL formulas defined above, denote with $\rho(\psi, \boldsymbol{x}, t)$ the robustness metric associated with each non-temporal formula ψ, assumed to be concave or convex. The main idea of PPC is to achieve satisfaction of the temporal formula φ by controlling the evolution of $\rho(\psi, \boldsymbol{x}, t)$ in time, such that it stays bounded between two prescribed curves, that identify a funnel, related to the always or eventually operators. When considering the always operator, for example, the lower curve must remain at or above 0 during all the interval $[a, b]$ to ensure that $\rho(\psi, \boldsymbol{x}, t) \geq 0$ and therefore satisfaction of the task φ as $\rho(\varphi, \boldsymbol{x}, t) = \min_{t\in[a,b]} \rho(\psi, \boldsymbol{x}, t) \geq 0$ [28]. More specifically,

the two boundaries for $\rho(\psi, \boldsymbol{x}, t)$ are defined by a curve $\gamma(t) \in \mathbb{R}$ and a parameter $\rho_{max} \in \mathbb{R}$, chosen so that the task φ will be satisfied if $\gamma(t) < \rho(\psi, \boldsymbol{x}, t) < \rho_{\max}$ holds for all $t \in [a, b]$.

Under some assumptions, the satisfaction is achieved by the control law:

$$\boldsymbol{u}_{\varphi}(\boldsymbol{x}, t) = \epsilon(\boldsymbol{x}, t) g^{\mathrm{T}}(\boldsymbol{x}) \frac{\partial \rho(\psi, \boldsymbol{x}, t)}{\partial \boldsymbol{x}} \tag{3.9}$$

where ϵ is the transformation error defined before. To compute the derivative of $\rho(\psi, \boldsymbol{x}, t)$, authors in [27] use a differentiable under-approximation for conjunctions of propositions.

Funnel-based control applied to systems under STL specifications results to be computationally tractable and robust. However, this approach is characterized by loss of optimality guarantees, such as provided by optimization or learning-based approaches. Moreover, it allows to consider only fragments of an STL formula and it could lead to possible infinity control, if the robust semantics is close to the boundaries of the funnel.

3.2 Kinodynamic Planning Algorithms using RRT*

In the last years, RRT* has been exploited to solve the optimal path planning problem, that consists in finding the shortest and smooth path between two positions, avoiding eventual obstacles. The standard RRT* algorithm connects pair of states (nodes of the tree) using straight lines, that are infeasible for kinodynamic systems, due to differential constraints.

Kinodynamic planning consists in driving a robot from a start state to a goal state or region, avoiding obstacles and obeying kinematic and dynamic constraints, that express the relationship between a robot's control and its motion. The found trajectories hence need not only to lie in the free space but also to be feasible with respect to the model of the robot's dynamics [29]. The constraints are usually expressed by means of differential equations that govern the state of the robot. The two main issues to be addressed when solving a kinodynamic planning problem with sampling-based algorithms are the following ones:

1. How to evaluate distance between states under kinodynamic constraints;

2. How to drive the robot through unvisited states.

In many cases, the objective is not only to find a feasible solution, but also to take into account the quality of it, with the aim of driving the system through an optimal path. Algorithms are hence designed to find optimal trajectories with respect to a cost functional that depends on both the state and the input. This cost is usually selected in order to minimize time and energy, maximize safety or some combinations of them. However, for the most general formulations of robot dynamics, driving the robot exactly from a start to a target state may be computationally expensive, requiring tools from nonlinear optimal control.

Authors in [30] have proposed a variant of RRT under differential constraints and proved its probabilistic completeness. Specifically, under the assumption of Lipschitz continuous functions in the system dynamics, the approach expands the tree by propagating random control for random durations.

To deal with differential constraints, Kinodynamic-RRT* has been proposed by authors in [31]. This extension of RRT* is able to handle systems described by differential constraints. Given a system described by a differential equation, the optimal kinodynamic motion planning problem is solved: the objective is to find a control u such that the corresponding trajectory satisfies the constraints, avoids obstacles, reaches the goal region and minimizes a cost functional J. More specifically, the tree is initialized with the initial state. In each iteration, one random sample is collected and the tree is expanded towards this sample, following the procedure:

1. Extension of nearest vertex toward the sample. The former is selected according to a distance function and the control and trajectory are computed (x_{new} steers the nearest vertex towards the sample and its final state is denoted with z_{new});

2. If there is no collision, z_{new} is added and its parent is selected considering neighbor nodes and taking the one that can be steered to z_{new} with minimum cost;

3. Finally, rewiring of the tree is performed. Again, near vertices of z_{new} are considered and z_{new} is set as parent of a near vertex if it can be steered towards z_{near} with a trajectory whose cost is lower than the current cost of z_{near}.

Authors in [32] instead have proposed a different variant of RRT*, LQR-RRT*, that allows to derive automatically the distance metric and the node extension method required by RRT*. These two heuristics are obtained applying linear quadratic regulation (LQR), without requiring domain-specific design choices. The steps required by the algorithm are the same followed by the standard RRT*; the differences regard how near and nearest nodes are evaluated and how steering towards the random sample is obtained. More specifically, `LQRNearest` function returns the nearest node in term of the cost-to-go function S, solution of the Algebraic Riccati equation related to the matrices of the linearized system. In a similar way, `LQRNear` returns the set of vertices within a certain distance from the considered state, evaluated through the LQR cost function. Finally, `LQRSteer` function connects two states using the LQR policy calculated by linearizing about the current state.

The optimal motion planning for robots with linear differential constraints has been solved by researchers in [33], by using a fixed-final-state-free-final-time controller, able to connect any two states in an exact and optimal way, where the cost function is expressed as a trade-off between the duration and the control effort. The proposed algorithm works similarly to the standard RRT*, including indeed a sampling procedure, a search of the nearest node in terms of a non-Euclidean distance and a rewiring of the tree. However, two main differences arise: the found trajectory arrives exactly at a goal state (not goal region as in most of RRT* cases) and the sampled point is connected directly to the tree, instead of the final point of a partial trajectory using a steering function as in the standard RRT*.

In this thesis, the designed algorithm inherits the idea of using reachability to build the tree, as presented by authors in [34]. They have proposed the Reachability-Guided Rapidly-exploring Random Tree (RG-RRT) as a planning strategy for general systems subject to differential constraints. This procedure directly considers the limitations of the system dynamics to shape the Voronoi bias, decreasing the sensitivity to the distance metric and not requiring a system-specific metric heuristic. This results in a tree that expands efficiently, taking into account system dynamics and considering the reachable region of the state space of each node. $R_{\Delta t}(z_0)$, specifically, is defined as the set of all points that can be achieved from z_0 in finite time Δt, according to the state equation and the

set of available control inputs U. Once the initial node and its corresponding reachable set are added, a random sample is drawn. If the closest reachable point is closer to the sample than the closest node of the tree, then both this reachable point and the corresponding parent node are returned. Otherwise, the sample is discarded and a new one is drawn: this results in throwing away samples for which the nearest node is closer than its reachable set.

Chapter 4

Methodologies

The following chapter addresses the cooperative motion planning problem of coupled agents and presents the developed solution. Firstly, the problem formulation is reported; secondly, the algorithm is explained from a general point of view; finally, all the involved functions are defined step by step.

4.1 Problem Formulation

The problem addressed in this thesis is the cooperative motion planning problem of two autonomous agents, subject to a coupled task expressed as an STL formula of the form 2.1.

Assume that the two dynamical systems evolve in $\mathbb{R}^n$ and that they are described by $\mathcal{R}_i = \left(f_i, X_i, U_i, \boldsymbol{x}_{\text{init}i}\right)$, $i = 1, 2$, where $X_i \subseteq \mathbb{R}^n$ is the state space, $U_i \subseteq \mathbb{R}^m$ is the control space, $f_i : X_i \times U_i \to X_i$ is a Lipschitz continuous function, and $\boldsymbol{x}_{\text{init}i}$ is the initial state of system i. The system behavior (for both agents) is given by:

$$\mathcal{R}_i : \dot{\boldsymbol{x}}_i = f_i(\boldsymbol{x}_i(t), \boldsymbol{u}_i(t)), \boldsymbol{x}_i(0) = \boldsymbol{x}_{\text{init}i}, i = 1, 2 \tag{4.1}$$

Denote with $\boldsymbol{x}_i\left[\boldsymbol{x}_{\text{init}i}, \boldsymbol{u}_i\right]$ the state trajectory originating at $\boldsymbol{x}_{\text{init}i}$ obtained by implementing the control policy $\boldsymbol{u}_i$.
The system $\mathcal{R}_i$ is said to satisfy an STL specification φ under a control policy $\boldsymbol{u}_i$ if the state trajectory starting at $\boldsymbol{x}_{\text{init}i}$ and obtained through the application of $\boldsymbol{u}_i$ satisfies φ, namely $\boldsymbol{x}_i\left[x_{\text{init}i}, \boldsymbol{u}_i\right] \models \varphi$.

The aim of the project is, however, not only to come up with a feasible trajectory

with respect to the environment and the STL specification, but to find the optimal trajectory for each agent. In other words, the returned trajectory, with the associated control sequence, must be such that it avoids the obstacles, satisfies the STL specification and minimizes a cost function.
Hence, the problem to be addressed can be formulated as follows:

Problem 4.1. Given two dynamical systems $\mathcal{R}_1, \mathcal{R}_2$ subject to a task expressed as an STL formula φ, find control policies $\boldsymbol{u}_1^*$ and $\boldsymbol{u}_2^*$ with time domain $[0, T]$ such that the costs of the state trajectories, originating respectively in $\boldsymbol{x}_{\text{init}_1}$ and $\boldsymbol{x}_{\text{init}_2}$ under $\boldsymbol{u}_1^*$ and $\boldsymbol{u}_2^*$ and satisfying the STL specification, are minimized, i.e.

- $\boldsymbol{x}_i \left[\boldsymbol{x}_{\text{init}i}, \boldsymbol{u}_i^*\right] \in X_{\text{free}}$ for all $t \in [0, T]$, $i = 1, 2$;
- $\boldsymbol{x}_i \left[\boldsymbol{x}_{\text{init}i}, \boldsymbol{u}_i^*\right] \models \varphi$, $i = 1, 2$;
- $\boldsymbol{u}_i^* = \underset{\boldsymbol{u}_i \in \mathcal{U}_i}{\text{argmin}} \ \text{cost}\left(\boldsymbol{x}_i \left[\boldsymbol{x}_{\text{init}i}, \boldsymbol{u}_i\right]\right), i = 1, 2.$

The proposed solution for the cooperative optimal motion planning problem under STL specifications is explained in detail in the next section.

4.2 Coupled STL_RRT*

The designed approach is inspired by the standard RRT* algorithm and by the work in [9]. It derives, for each agent, the control sequence u^* associated with a probabilistic optimal trajectory satisfying the STL formula φ. With reference to RRT*, time is added in the sampling process, thus resulting in the construction of spatio-temporal trees. On the other hand, the differences with respect to the solution proposed in [9] are the incorporation of dynamic constraints, exploiting reachability, the best parent search and rewiring procedure. These additions allow to obtain time-varying trajectories that are probabilistic optimal, feasible with respect to the dynamics and satisfying φ.

The proposed algorithm, Coupled STL_RRT*, presents three main components: the sampling-based procedure to expand the tree, the STL verification to check upon the satisfaction of the STL formula, and the rewiring procedure to obtain minimum-cost trajectories. Each sample corresponds to a point $z = (t, x) \in \mathcal{Z} \subset$

$\mathbb{R}_+ \times \mathbb{R}$; the nearest point in the tree is firstly considered and reachable states are computed from it. Before adding new points to the tree, the best parent is selected and the STL verification procedure checks if the potential new nodes do not violate the active part of the STL formula and if the corresponding trajectories are feasible as well. Finally, the rewiring step modifies the connections between the nodes to ensure that the total path from the root to each vertex is the one with minimum cost. When the algorithm terminates, the generated trajectories are guaranteed to satisfy the STL formula and to be both feasible and probabilistic optimal. The different steps are highlighted in Figure 4.1.

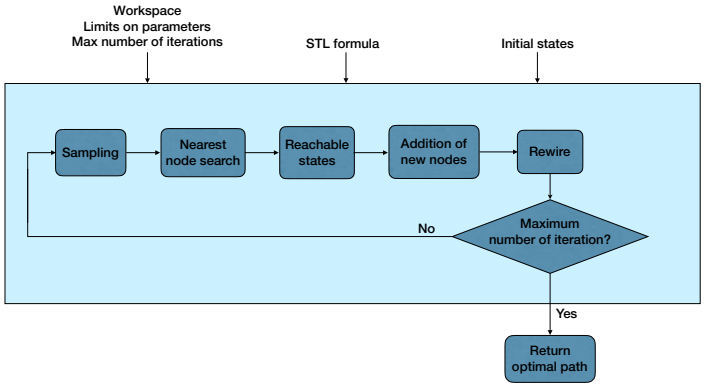


Figure 4.1: Main steps of Coupled STL_RRT*

4.2.1 Overall Algorithm

For the discussion below, index i is used to refer to the agent that is currently running the pseudo-code and index j for the other agent. From the point of view of agent i, once the tree is initialized with the initial state $z_{\text{init}i}$ (lines 1-3), in each iteration the following steps are performed:

1. Sample a random state z_{rand_i}. Check for collision with the obstacle: if it occurs, keep sampling a new state, maintaining the same time variable, until the random sample does not lie in the obstacle region (lines 5-7);

2. Find the closest node z_{nearest_i} from the nodes of the current tree. If the nearest nodes of the two agents share the same time variable, go to step 3, given that the two trees are synchronized. Otherwise, keep the smallest time variable t_{min}. Considering the other agent, update the nearest node, by choosing between nodes with time variable t_{min} (lines 8-9);

3. Compute the set of reachable states from z_{nearest_i} (line 10);

4. Add new nodes $z_{\text{new}i}$ to the tree, depending on the time variable of the sample and of the reachable nodes. Before adding a new node, select the best parent, check if the corresponding trajectory does not collide with the obstacle and if it satisfies the active STL formula (lines 11-15);

5. Rewire the tree, considering neighbor nodes $z_{\text{near}i}$ in the tree that are reachable from $z_{\text{new}i}$. If the path through $z_{\text{new}i}$ has cost lower than the current one, update the parent of $z_{\text{near}i}$ with $z_{\text{new}i}$. Before rewiring the tree, check for collision and satisfaction of the active STL formula (line 18).

The pseudo-code of the algorithm, from the point of view of agent i, is reported in Algorithm 2, while the block diagram can be seen in Figure 4.2.

Algorithm 2: Coupled STL_RRT*: Agent i perspective

Input: $(f_i, X_i, U_i, x_{\text{init}i})$ = dynamical system
φ = STL formula
obs = obstacle to be avoided
$t_{\max}, v_{\max}, y_{\max}, u_{\max}, N_{\max}$ = limits on parameters
Output: u_i = optimal control sequence

1 $T_i \leftarrow (V_i = \emptyset, E_i = \emptyset)$;
2 $z_{\text{init}i} \leftarrow (0, x_{\text{init}i})$;
3 $V_i \leftarrow$ `InsertNode` $(z_{\text{init}i}, V_i)$;
4 **for** $i = 1$ **to** $i = N_{\max}$ **do**
5 $z_{\text{rand}i} \leftarrow$ `RandomState` ();
6 **while** ! `NoPointCollision` $(z_{\text{rand}i}, obs)$ **do**
7 $z_{\text{rand}i} \leftarrow$ `RandomState` ();
8 $z_{\text{nearest}i} \leftarrow$ `Nearest` $(z_{\text{rand}i}, V_i)$;
9 $z_{\text{nearest}i} \leftarrow$ `Synchro` $(z_{\text{nearest}i}, z_{\text{nearest}j})$;
10 $R_i \leftarrow$ `ReachableSet` $(z_{\text{nearest}i})$;
11 $newNodes_i \leftarrow$ `ComputeNn` (R_i) ;
12 $newNodes_i \leftarrow$ `BestParent` (V_i) ;
13 **if** $t_{\text{new}_i} \in I_\varphi$ **then**
14 $newNodes_i \leftarrow$ `CheckPointSTL` $(newNodes_i, \varphi)$;
15 $newNodes_i \leftarrow$ `CheckTrajSTL` $(newNodes_i, \varphi)$;
16 **foreach** $z_{\text{new}i} \in newNodes_i$ **do**
17 $V_i \leftarrow$ `InsertNode` $(z_{\text{new}i}, V_i)$;
18 `Rewire` $(z_{\text{new}i})$;
19 $path_i \leftarrow$ `FindOptPath` (V_i);
20 $u_i \leftarrow$ `FindOptU` $(path_i)$;
21 **return** (u_i);

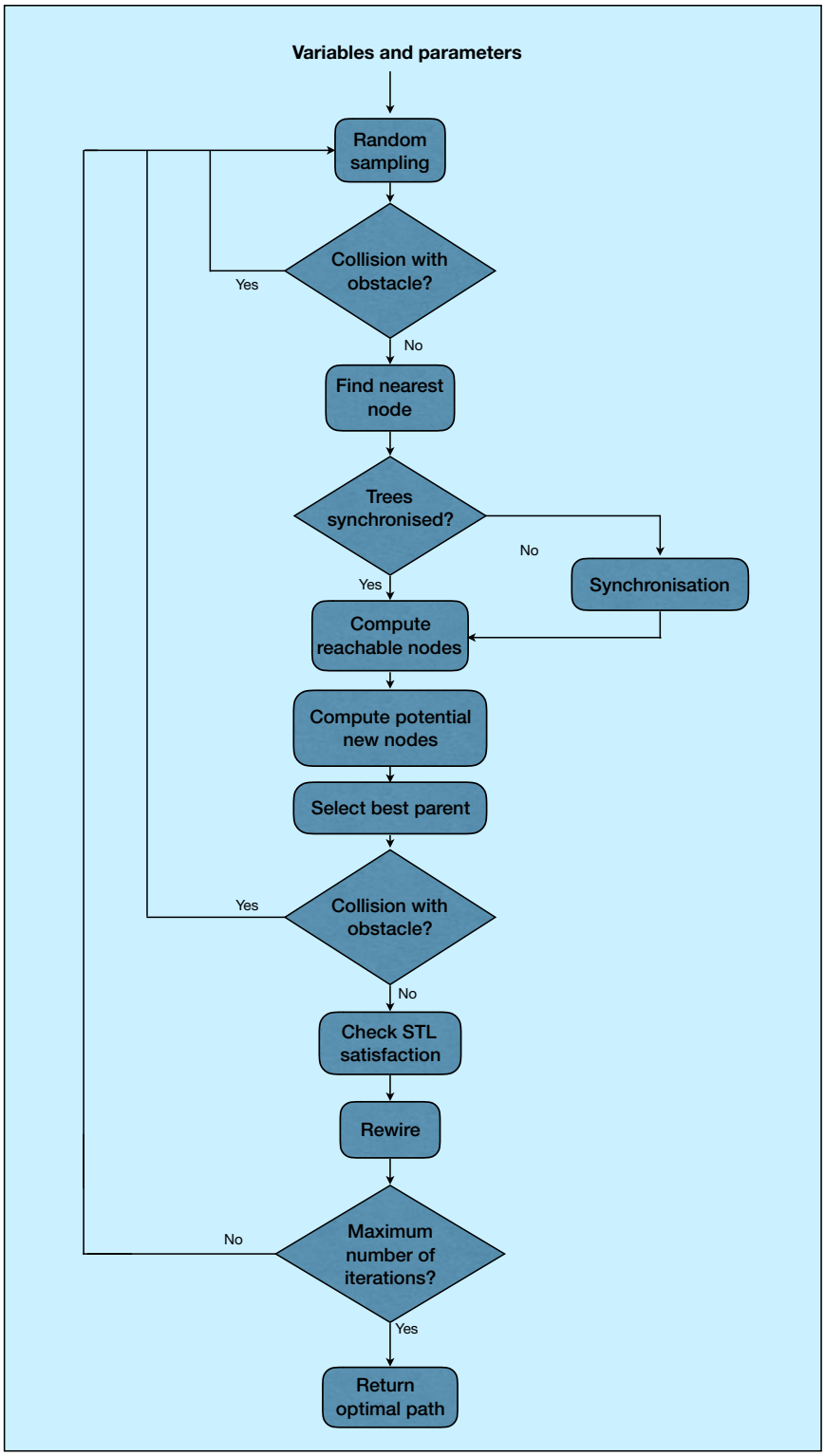


Figure 4.2: Block diagram of Coupled STL_RRT*

4.2.2 Definition of Variables

In this work, a double integrator dynamics is considered. Recall that, given the initial state $x(0)$, the initial velocity $\dot{x}(0)$ and the control input u, the state equation and the velocity equation result to be the following ones:

$$\begin{aligned} x(t) &= x(0) + \dot{x}(0)t + \tfrac{1}{2}ut^2 \\ \dot{x}(t) &= \dot{x}(0) + ut \end{aligned} \tag{4.2}$$

For each agent, the algorithm builds a tree, $T = (V, E)$, consisting of a set of vertices $\{z_i\}$ connected by edges. Each agent is identified by a `struct` (either a_1 or a_2), in which the initial state, the random sample, the new nodes to be added are saved, iteration after iteration. In turn, each node is associated with a `struct` containing the following parameters:

- $coord$: the first one represents the time variable, the second one defines the state;
- vel: velocity of the dynamical system at that node;
- $parent$: parent of the node;
- u: control to be applied to reach the node starting from its parent;
- $cost$: cost to reach the node starting from the root of the tree;
- pos: position of the node in the array containing all the vertices of the tree.

4.2.3 STL Specifications

In this thesis, STL formulas mainly express minimum or maximum distance between the two agents or between the agents and the environment. Formulas that have been taken into account contain the always and the eventually operators, also nested to obtain complex specifications. Note that the until operator $\varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ is equivalent to $\mathcal{F}_{[t_1]}\varphi_2 \wedge \mathcal{G}_{[a,t_1]}\varphi_1$, with $t_1 \in (a, b]$. The two agents construct their trees simultaneously, checking if no collision happens with the obstacle and if the STL formula is satisfied in its validity domain. The possible cases that are taken into account are single operators and nested formulas. In order to consider all the possibilities and to be able to adapt the algorithm to different types of STL formulas, each STL specification is defined using a `struct` containing the following parameters:

- $d_{\min}$: minimum distance between agents or minimum value of the state of one agent;
- $d_{\max}$: maximum distance between agents or maximum value of the state of one agent;
- t_i: initial instant of the validity domain of single operator formulas;
- t_f: final instant of the validity domain of single operator formulas;
- $type$: operator in the STL formula (0 if always, 1 if eventually);
- $ineq$: type of inequality (0 if $\geq$, 1 if $\leq$);
- $agent$: agent involved in the specification (0 if both agents, 1 if agent 1, 2 if agent 2);
- $flag\_ev$: Boolean variable accounting for the satisfaction of the eventually operator. If true, the eventually has been satisfied and so it is not necessary to check for it in the next iterations;
- $flag\_nes$: Boolean variable indicating if the STL formula is nested. Once the nested formula is satisfied, it is set to be false;
- val_dom: validity domain of the formula. The interval is initialized, depending on the type of the STL specification, as explained in Section 2.2.1.

In order to deal with nested formulas, a further `struct` is defined, containing the parameters:

- $type$: type of nested formula (0 if $\mathcal{G}(\mathcal{F})$, 1 if $\mathcal{F}(\mathcal{G})$);
- $outer$: interval of definition of the external operator in the nested formula;
- $inner$: interval of definition of the internal operator in the nested formula.

Single operators

The simplest STL formulas are given by single operators, concerning either one agent or both of them. Within this class of specifications, it is possible to distinguish three sub-cases: single predicate (1), always operator (2) or eventually operator (3).

An example of (1) is $\varphi = |x_1 - x_2| > 3$, requiring the agents to be 3 units apart at all time instants. The two agents build their trees, ensuring that every added node is at least 3 units away from all the nodes of the other agent with same time variable. By proceeding in this way, any final trajectory will satisfy φ.

The formula $\varphi = \mathcal{G}_{[3,5]} |x_1 - x_2| > 4$ is an example of the use of the always operator (2). In particular, it requires that in the interval $[3, 5]$ every added vertex is at least 4 units apart with respect to the other agent. When the two agents need to expand the tree in $[3, 5]$, namely when the always operator is active, it is necessary to check that the distance between the agents is at least 4, for each possible pair of incident nodes. If the constraint is not satisfied, the potential new nodes are not added to the trees.

Finally, an example of case (3) is given by formula $\varphi = \mathcal{F}_{[5,7]} |x_1 - x_2| > 10$, requiring the agents to be 10 units apart at any instant in the interval $[5, 7]$. Differently from the always operator, even if a pair of new nodes $(z_{\text{new}_1}, z_{\text{new}_2})$, with time variable $t_{\text{new}} \in [5, 7]$, does not satisfy φ, $(z_{\text{new}_1}, z_{\text{new}_2})$ are added to the corresponding trees, because the satisfaction could occur for the subsequent vertices. As soon as two added nodes do not violate the specification, they are taken as new roots for the expansion of the trees, in order to ensure that the final trajectories will go through them, hence satisfying the STL formula.

Nested formulas

Operators can be nested into STL formulas, allowing to define specifications such as $\varphi = \mathcal{G}_{[0,10]}\mathcal{F}_{[1,3]} |x_1 - x_2| > 8$ (1) or $\varphi = \mathcal{F}_{[0,10]}\mathcal{G}_{[1,2]} |x_1 - x_2| < 5$ (2).

In the first example, the formula requires to satisfy $|x_1 - x_2| > 8$ every 2 seconds in the interval $[1, 13]$ in which the STL formula is active. If, for example, the predicate is true at $t = 2.5$, then it is necessary to have $|x_1 - x_2| > 8$ also at any instant in the time interval $[3.5, 5.5]$. Hence, the time interval for the evaluation of the predicate keeps being updated, until the nested formula becomes inactive.

In the second case, it is required to satisfy $|x_1 - x_2| < 5$ for one second in the time interval $[1, 12]$. If the predicate is true at $t = 1.5$, the nested STL formula results to be satisfied if the predicate keeps being true in the interval $[2.5, 3.5]$.

4.2.4 Definition of Functions

InsertNode

The function `InsertNode` adds the node z_i into the array containing all the nodes of the current tree. Before the insertion, all the parameters associated to the vertex must be properly set. In the case of the initial node, in particular, velocity, control input, parent, cost and time are set to 0, the position in the array is set to 1 and the state is set depending on the initial configuration of the agent.

RandomState

Once the trees of the two agents are initialized, the algorithm starts to build incrementally the RRT* structures. In each iteration, through the function `RandomState`, two random samples are drawn. First of all, the time variable, that is the same for both agents, is randomly generated. Then, the state variables are randomly drawn as well. If the sample lies inside the obstacle region, the time variable is kept and the state is again randomly generated. Hence, the function `RandomState` is called until both samples do not collide with the obstacle.

Nearest

Given the random sample $z_{\text{rand}\,i} = (t_{\text{rand}\,i}, x_{\text{rand}\,i})$, the function `Nearest` finds the nearest node $z_{\text{nearest}\,i} = (t_{\text{nearest}\,i}, x_{\text{nearest}\,i})$ to it in the current tree of the agent. More specifically, the function evaluates the distance between the sample and each node of the tree, computing the Euclidean metric. It is important to underline the fact that the nearest node must be back in time with respect to the random sample, given that tree is required to propagate forward in time. In other words the condition $t_{\text{nearest}\,i} < t_{\text{rand}\,i}$ must be satisfied. In order to be sure that the random sample is ahead in time with respect to the closest node in the tree, the distance is set to ∞ if the considered vertex has time variable greater than the one of the sample. In Figure 4.3, it is possible to observe the distinction between two groups of nodes in the current tree. Given the random sample, its distance from the red vertices is set to ∞, due to the fact that they are ahead in time. Instead, the distance between the sample and the light blue nodes is computed using the Euclidean metric. Among all these values, the function finally selects the minimum one.

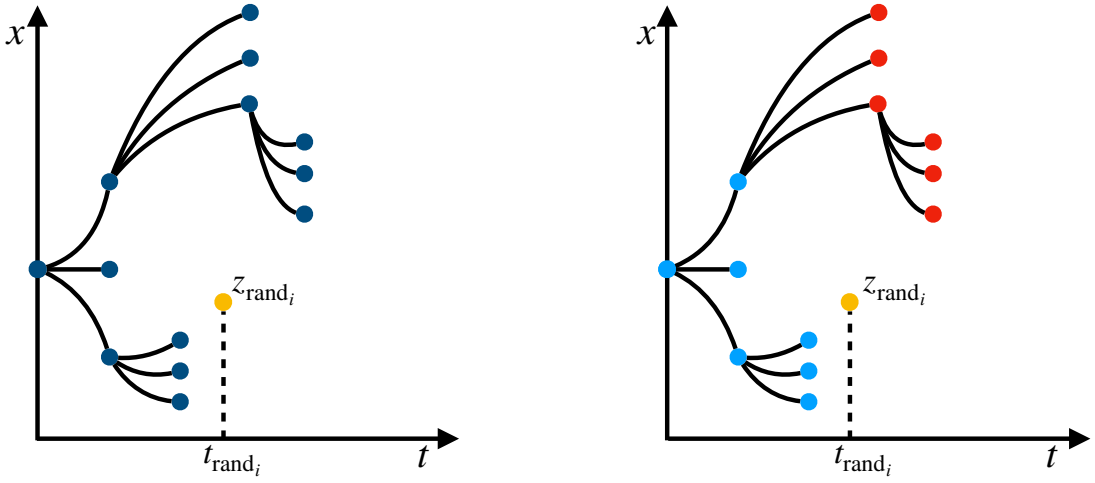


Figure 4.3: Example of the evaluation of the distance between random sample and nodes in the current tree: for red nodes, distance is set to ∞; for light blue nodes, the Euclidean distance is computed.

Synchro

The function `Synchro` is necessary to synchronize the expansion of the two trees. Once the two nearest nodes $z_{\text{nearest}i}$ and $z_{\text{nearest}j}$ are found, indeed, two possible cases arise. If the two nearest nodes share the same time variable, the agents are already synchronized and the algorithm goes further. Otherwise, if the two time variables of the nearest nodes do not coincide, i.e. $t_{\text{nearest}i} > t_{\text{nearest}j}$ or vice versa, the algorithm proceeds as follows:

1. Take the smallest time variable between $t_{\text{nearest}i}$ and $t_{\text{nearest}j}$, for example $t_{\text{nearest}j}$. For simplicity, call it $t_{\min}$;
2. Consider agent i and update the nearest node. This is done by evaluating again the Euclidean distance between the nodes in the current tree and the random sample, but restricting the set of candidate nearest nodes to the set of vertices with time variable equal to $t_{\min}$;
3. Pick the updated nearest node $z_{\text{nearest}i}$ as the node with time variable $t_{\min}$ that is closest to the random sample $z_{\text{rand}i}$.

This procedure ensures that the two trees grow simultaneously, adding nodes at the same time instants, and facilitates the evaluation of the STL specifica-

tion. An example of the two cases (trees already synchronized and trees not synchronized) is shown in Figure 4.4.

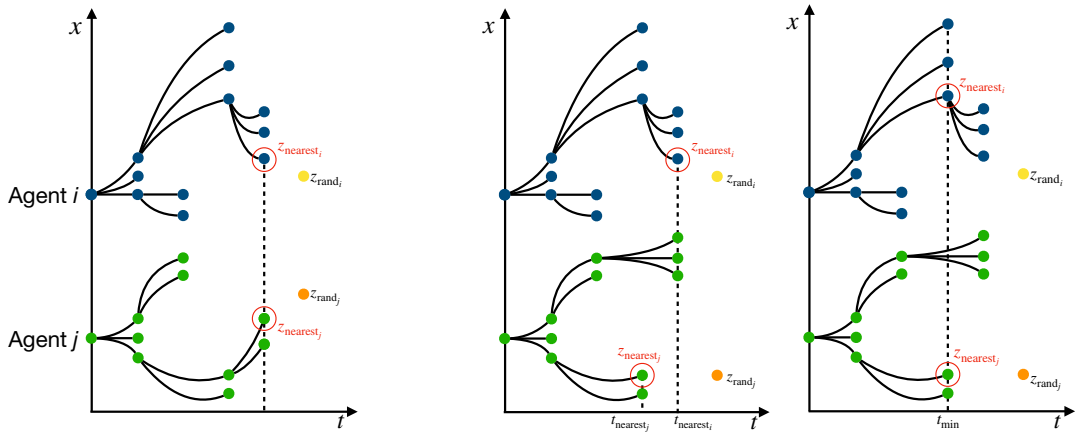


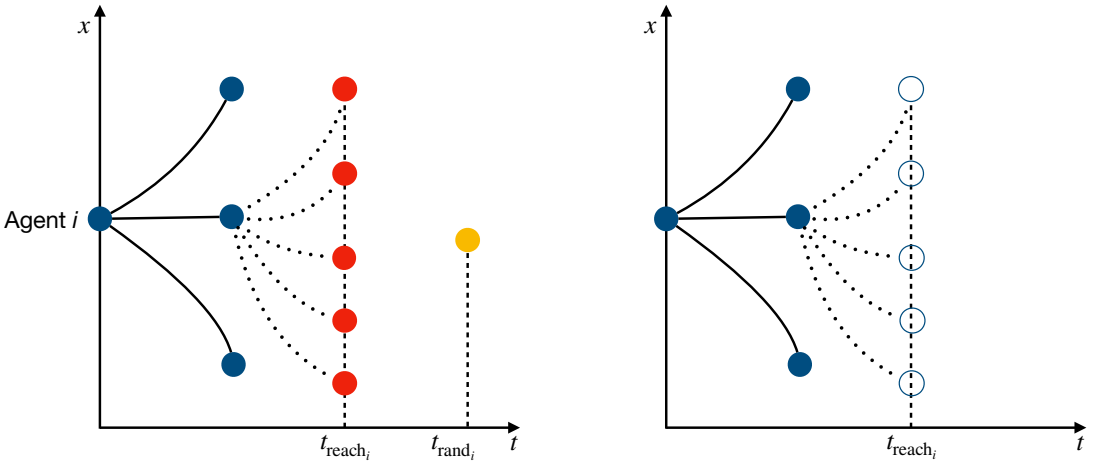
Figure 4.4: Example of synchronization of nearest nodes: trees on the left are already synchronized, trees on the right require synchronization

ReachableSet

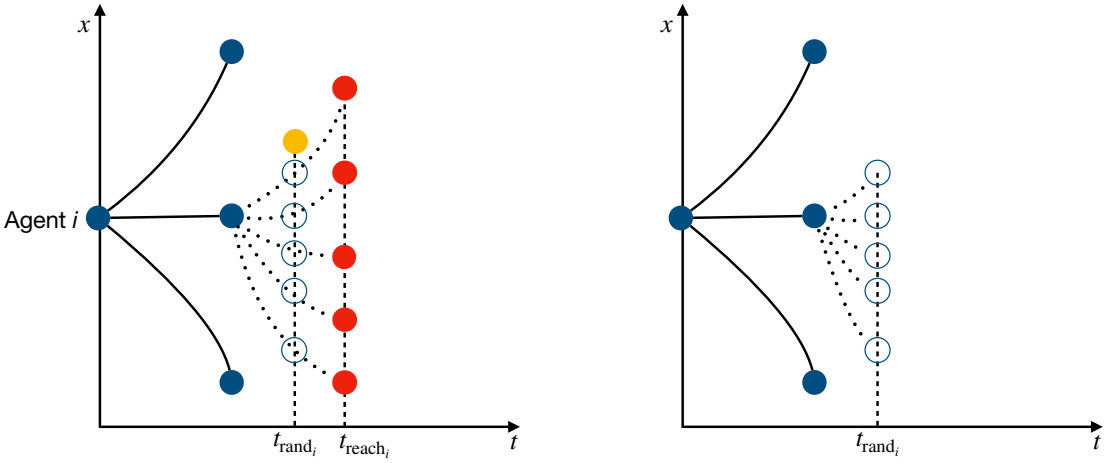
In the standard RRT* algorithm, to move from the selected nearest node x_{nearest} toward the sampled point x_{samp}, an edge of length $step$ is drawn in the direction of x_{samp}. This results in a new node x_{new}, added to the tree together with the edge. The standard RRT* algorithm, hence, connects states using straight lines, that are infeasible for kinodynamic systems, due to differential constraints. In this work, to expand the tree of agent i taking into account the dynamics of the system, reachability is exploited. More specifically, the function `ReachableSet` computes the reachable states starting from $z_{\text{nearest}i}$. They are obtained by applying a constant control input u in the interval $[-u_{\max}\ u_{\max}]$, for a fixed time Δt, randomly generated in each iteration. By discretizing the interval $[-u_{\max}\ u_{\max}]$, the reachable states are evaluated and saved only if they satisfy the limits on the environment and on the velocity. If, for at least one agent, the reachable set from z_{nearest} is empty, the algorithm continues to the next iteration. Moreover, notice that the reachable nodes of agent i share the same time variable not only with each other, but also with the reachable nodes of the other agent, given that the nearest nodes are synchronized and Δt is the same for both autonomous systems.

ComputeNn

Based on the reachable nodes returned by the function `ReachableSet`, potential nodes to be added to the tree are evaluated. The new candidate vertices are obtained by calling the function `ComputeNn`. Depending on the time variable of the random sample and the time variable of the reachable nodes, two cases arise, as depicted in Figure 4.5.



(a) $t_{\text{reach}_i} < t_{\text{rand}_i}$



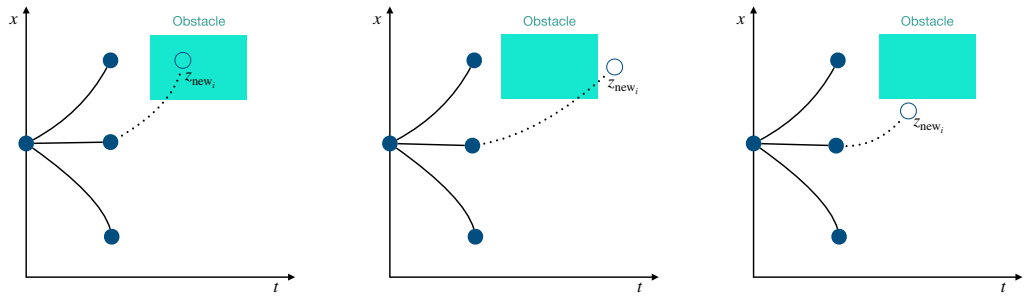
(b) $t_{\text{rand}_i} < t_{\text{reach}_i}$

Figure 4.5: Example of evaluation of potential new nodes to be added to the tree

In the first case (see Figure 4.5a), the time variable $t_{\text{reach}\,i}$ of the reachable nodes is smaller than the time variable $t_{\text{rand}\,i}$ of the random sample. All reachable nodes are hence saved as potential nodes to be added to the tree. In the second case (see Figure 4.5b), instead, the time variable of the random sample is smaller than the

time variable of the reachable nodes. Therefore, for each reachable node $z_{\text{reach}\,i}$, $t_{\text{rand}\,i}$ is used to save as potential new node the one with time variable $t_{\text{rand}\,i}$, taken along the trajectory that connects $z_{\text{nearest}\,i}$ with the considered reachable node $z_{\text{reach}\,i}$.

At this point, for each potential new node $z_{\text{new}\,i}$, the algorithm checks for possible collisions with the obstacle region, not only considering the single vertex but also the whole trajectory. It could happen, indeed, that $z_{\text{new}\,i}$ lies outside the obstacle region, but part of the trajectory collides. By interpolating the parabola between $z_{\text{nearest}\,i}$ and $z_{\text{new}\,i}$, it is possible to check if no collision occurs. If $z_{\text{new}\,i}$ is already inside the obstacle region, it is directly discarded (Figure 4.6a). $z_{\text{new}\,i}$ is also not considered as potential new node if at least one point along the trajectory collides with the obstacle region (Figure 4.6b). Instead, if no collision occurs, as shown in Figure 4.6c, the node is kept.



(a) z_{new_i} collision (b) Trajectory collision (c) No collision

Figure 4.6: Examples of collision and no collision with the obstacle

BestParent

As in the standard RRT* algorithm, new nodes are not connected directly to the nearest node, but to the best vertex in the current tree. To this aim, for each potential new node, the function `BestParent` selects as parent the vertex that brings to the minimum cost path from the initial node to the new one. More specifically, the function iterates considering all nodes in the tree that are within a radius r centered on the potential new node. Initially, the parent of the potential new node is set to be the nearest one; then, if through a node z_i of the current tree it is possible to reach the new node with a path that has lower cost, the parent is updated to be z_i and also the cost, velocity and control are

changed, depending on z_i.

CheckPointSTL

At this point, it is necessary to verify the satisfaction of the STL specification before expanding the tree. To this aim, the algorithm calls the function `CheckPointSTL`, that checks, for each potential new node $z_{\text{new}i} = (t_{\text{new}i}, x_{\text{new}i})$, if it satisfies the STL specification φ. The function is exploited only if $t_{\text{new}i}$ is within the validity domain of the STL formula; otherwise, the procedure is skipped. In the former case, two possibilities arise depending on the specification.

For STL formulas that express preferences related to only one agent, for example $\varphi := \mathcal{G}_I(1 \leq x_i \leq 3)$ (1) or $\varphi := \mathcal{F}_I(2 \leq x_i \leq 4)$ (2), the procedure is the same applied to check for eventual collisions with the obstacle. However, the function operates differently if the operator is the always or the eventually one (see Algorithm 3).

Algorithm 3: CheckPointSTL: single agent involved

Input: $newNodes_i$ = array of potential new nodes of agent i
φ = STL formula
Output: n_i = updated array of potential new nodes of agent i

```
1  n_i ← ∅;
2  if φ = G_I(μ) then
3      foreach z_newi ∈ newNodes_i do
4          if μ = ⊤ then
5              n_i ← Insert (z_newi);
6  if φ = F_I(μ) then
7      foreach z_newi ∈ newNodes_i do
8          if μ = ⊤ then
9              n_i ← z_newi;
10             flag_ev ← ⊤;
11             break;
12     if flag_ev = ⊥ then
13         n_i ← newNodes_i;
14 return n_i;
```

In the case of the always operator (1), given $z_{\text{new}\,i}$, the function evaluates the STL formula considering the coordinates of the new point and, if the specification is not satisfied, the point is discarded (it will not be added to the tree of agent i).

On the other hand, in the case of the eventually (2), even if $z_{\text{new}\,i}$ does not satisfy the specification, it is not discarded: the predicate could be true in the subsequent iterations, when evaluating it for one of the children nodes of $z_{\text{new}\,i}$. If, instead, $z_{\text{new}\,i}$ satisfies the STL formula, it is saved as unique node to be added to the tree of agent i and selected as new root for the expansion of the tree. In this way, it is possible to ensure that every final trajectory will satisfy the eventually operator, forcing the presence of $z_{\text{new}\,i}$ in it. Moreover, the Boolean variable $flag\_ev$ is set to be true, given that the STL specification is satisfied by at least $z_{\text{new}\,i}$.

The second possibility regards STL specifications that involve both agents, such as $\varphi := \mathcal{G}_I(|x_i - x_j| \leq 3)$ or $\varphi := \mathcal{F}_I(|x_i - x_j| \geq 4)$. When the STL specification is coupled, the verification of the satisfaction requires to consider also the incident nodes of the other agent. These nodes are the ones that share the same time variable of the potential new nodes of agent i and are extracted from the tree of agent j by interpolation (see the example in Figure 4.7). Again, it is necessary to distinguish the scenario of the always operator (1) from the one of the eventually operator (2), as highlighted in Algorithm 4.

In case (1), the function `CheckPointSTL` iterates considering all the nodes in $newNodes_i$ and in $newNodes_j$. More specifically, for each potential new node $z_{\text{new}\,i}$ of agent i, it is necessary to consider each possible new node $z_{\text{new}\,j}$ of agent j. In order to ensure that the formula is satisfied for each choice of the two agents, from $newNodes_i$ the function eliminates the nodes that violate the STL specification with at least one node in $newNodes_j$. By proceeding in this way, the two updated arrays $newNodes_i$ and $newNodes_j$ contain new nodes for the two agents that satisfy the coupled STL formula, whatever is the choice of the nodes in the final trajectories. The same procedure is applied also iterating through each incident node $z_{\text{inc}\,j}$ of agent j.

When the STL formula contains the eventually operator, instead, two cases arise. As seen before for a single agent, if all the possible pairs of nodes ($z_{\text{new}\,i}$, $z_{\text{new}\,j}$)

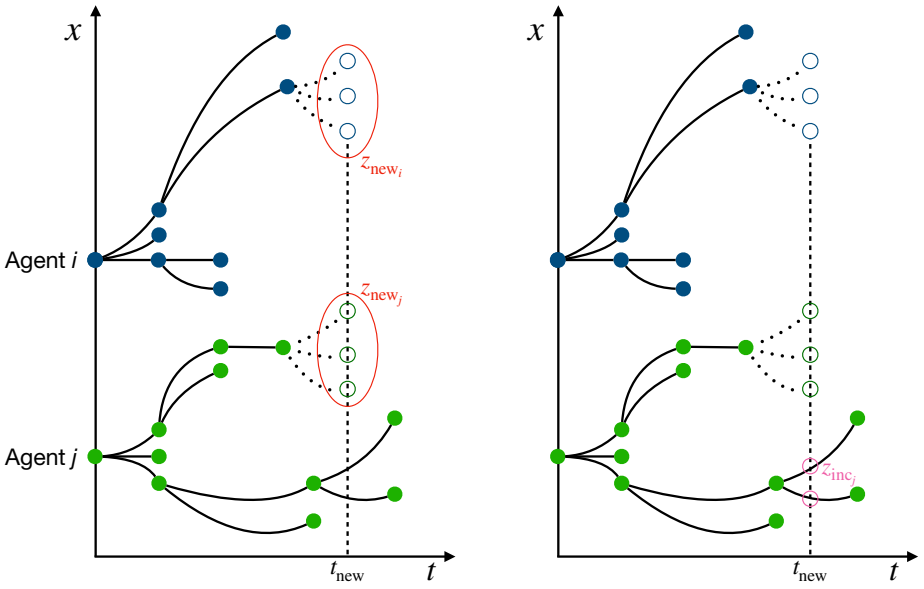


Figure 4.7: Example of incident nodes of agent j to the potential new nodes of agent i

or ($z_{\text{new}\,i}$, $z_{\text{inc}\,j}$) do not satisfy the specification, the two arrays of potential new nodes are not modified and no point is discarded. Indeed, the STL formula could be satisfied by two children nodes in the next iterations. On the other hand, as soon as a pair ($z_{\text{new}\,i}$, $z_{\text{new}\,j}$) or ($z_{\text{new}\,i}$, $z_{\text{inc}\,j}$) satisfies the specification, the two satisfying nodes are returned as unique nodes to be added to the trees of the agents. Both of them will be set as new roots for the future expansion of the trees, in order to ensure that the final trajectories will pass through them and hence satisfy the eventually operator. Moreover, the Boolean variable $flag\_ev$ is set to be true.

CheckTrajSTL

The last function that is called before the expansion of the tree is `CheckTrajSTL`, which checks if the nodes in $newNodes_i$ satisfy the active STL specification along their trajectories. Clearly, the verification is performed only considering the time instants within the validity domain of the STL formula. Again, two cases arise, depending on the type of the preferences, as already seen in the explanation of `CheckPointSTL`.

If the STL specification regards only one agent, it is necessary to interpolate the

Algorithm 4: CheckPointSTL: both agents involved

Input: $newNodes_i$ = array of potential new nodes of agent i
$newNodes_j$ = array of potential new nodes of agent j
φ = STL formula
Output: n_i = updated array of potential new nodes of agent i
n_j = updated array of potential new nodes of agent j

```
1  n_i ← ∅, n_j ← ∅;
2  inc_j ← IncidentPoint (T_j, newNodes_i);
3  if φ = G_I(μ) then
4      flag = ⊤;
5      foreach z_new i ∈ newNodes_i do
6          foreach z_new j ∈ {newNodes_j} ∪ {inc_j} do
7              if μ = ⊥ then
8                  flag = ⊥;
9                  break;
10         if flag = ⊤ then
11             n_i ← Insert (z_new i);
12     n_j ← newNodes_j;
13 if φ = F_I(μ) then
14     foreach z_new i ∈ newNodes_i do
15         foreach z_new j ∈ {newNodes_j} ∪ {inc_j} do
16             if μ = ⊤ then
17                 n_i ← z_new i, n_j ← z_new j;
18                 flag_ev ← ⊤;
19                 break;
20     if flag_ev = ⊥ then
21         n_i ← newNodes_i, n_j ← newNodes_j;
22 return n_i, n_j;
```

trajectory that connects each $z_{\text{new}\,i}$ to its parent. The function, again, operates differently depending on the operator.

In the case of the always operator, if there is no satisfaction for at least one point along the trajectory, then $z_{\text{new}\,i}$ is discarded from the array of potential new nodes and it will not be added as new node to the tree.

On the other hand, in the case of the eventually operator, more than one scenario emerges. If $flag\_ev$ is true, this means that the eventually operator has been

already satisfied for at least one node. If so, there is no need of checking upon the satisfaction of the specification also along the trajectory and the array of potential new nodes is not modified. Instead, if the STL formula is still not satisfied, the function interpolates along the trajectory up to $z_{\text{new}\,i}$. Two cases arise in turn, as seen also for the function `CheckPointSTL`. When, for each potential new node, the STL specification is not satisfied along the trajectory (this implies that it is necessary to satisfy the specification in the subsequent iterations), the array $newNodes_i$ is just returned without any modification. If, alternatively, a point along the trajectory of $z_{\text{new}\,i}$ satisfies the specification, then $z_{\text{new}\,i}$ is returned as unique new point to be added and $flag\_ev$ is set to be true. The pseudo-code of the function, when a single agent is involved, is reported in Algorithm 5.

In the second case, when the STL formula specifies preferences on both agents, the procedure needs to consider all the combinations between each possible node in $newNodes_i$ and each possible node in $newNodes_j$. Incident nodes of agent j are also taken into account (see Algorithm 6).

In particular, when the always operator is considered, `CheckTrajSTL` updates the arrays $newNodes_i$ and $newNodes_j$, by eliminating from $newNodes_i$ the nodes $z_{\text{new}\,i}$ that violate the STL formula along the trajectory with at least one node of $newNodes_j$ or one incident node of agent j. In order to check upon the satisfaction of the STL specification along the trajectory, it is necessary to interpolate between each potential new node and its parent, taking into account the potential new nodes and the incident nodes of agent j. An example of the procedure is reported in Figure 4.8. More specifically, starting from a potential new node $z_{\text{new}\,i}$ and an incident node $z_{\text{inc}\,j}$, the corresponding trajectories are interpolated going backwards through the parent nodes in the tree of agent j. In the first step, time is interpolated between the time variable of the parent of $z_{\text{inc}\,j}$ (called t_{start}) and the time variable of $z_{\text{inc}\,j}$ (called t_{final}). In the second step, t_{final} becomes the time variable of the parent of $z_{\text{inc}\,j}$, while t_{start} is set as the time variable of the parent of the parent of $z_{\text{inc}\,j}$. The interpolation is performed in this way, until t_{start} becomes the time variable of the parent of $z_{\text{new}\,i}$.

In the case of the eventually operator, instead, there is more than one possibility, as presented when the STL formula involves only one agent. Indeed, if the specification is still not satisfied, for each combination ($z_{\text{new}\,i}$, $z_{\text{new}\,j}$) and ($z_{\text{new}\,i}$, $z_{\text{inc}\,j}$),

Algorithm 5: CheckTrajSTL: single agent involved

Input: $newNodes_i$ = array of potential new nodes of agent i
φ = STL formula
Output: n_i = updated array of potential new nodes of agent i

```
1  n_i ← ∅;
2  if φ = G_I(μ) then
3      foreach z_newi ∈ newNodes_i do
4          for t ∈ [t_par_i, t_new_i] do
5              z_interi ← Interpolate (z_newi);
6              if μ = ⊥ then
7                  flag = ⊥;
8                  break;
9          if flag = ⊤ then
10             n_i ← Insert (z_newi);
11 if φ = F_I(μ) then
12     if flag_ev = ⊤ then
13         n_i ← newNodes_i;
14     else
15         foreach z_newi ∈ newNodes_i do
16             for t ∈ [t_par_i, t_new_i] do
17                 z_interi ← Interpolate (z_newi);
18                 if μ = ⊤ then
19                     n_i ← z_newi;
20                     flag_ev ← ⊤;
21                     break;
22         if flag_ev = ⊥ then
23             n_i ← newNodes_i;
24 return n_i;
```

interpolation is performed along the corresponding trajectories. If, considering $z_{\text{new}\,i}$ and $z_{\text{new}\,j}$, two interpolated points satisfy the STL formula, $z_{\text{new}\,i}$ and $z_{\text{new}\,j}$ are saved as unique new nodes and $flag\_ev$ is set to be true. Otherwise, if the eventually operator still remains not satisfied, the two arrays $newNodes_i$ and $newNodes_j$ are not modified. This happens also if the formula containing the eventually operator has been already satisfied.

Algorithm 6: CheckTrajSTL: both agents involved

Input: $newNodes_i$ = array of potential new nodes of agent i
$newNodes_j$ = array of potential new nodes of agent j
φ = STL formula

Output: n_i = updated array of potential new nodes of agent i
n_j = updated array of potential new nodes of agent j

```
1  n_i ← ∅, n_j ← ∅;
2  inc_j ← IncidentPoint (T_j, newNodes_i);
3  if φ = G_I(μ) then
4      foreach z_new_i ∈ newNodes_i do
5          foreach z_new_j ∈ {newNodes_j} ∪ {inc_j} do
6              for t ∈ [t_par_i, t_new_i] do
7                  z_inter_i ← Interpolate (z_new_i);
8                  z_inter_j ← Interpolate (z_new_j);
9                  if μ = ⊥ then
10                     flag = ⊥;
11                     break;
12         if flag = ⊤ then
13             n_i ← Insert (z_new_i);
14     n_j ← newNodes_j;
15 if φ = F_I(μ) then
16     if flag_ev = ⊤ then
17         n_i ← newNodes_i, n_j ← newNodes_j;
18     else
19         foreach z_new_i ∈ newNodes_i do
20             foreach z_new_j ∈ {newNodes_j} ∪ {inc_j} do
21                 for t ∈ [t_par_i, t_new_i] do
22                     z_inter_i ← Interpolate (z_new_i);
23                     z_inter_j ← Interpolate (z_new_j);
24                     if μ = ⊤ then
25                         n_i ← z_new_i, n_j ← z_new_j;
26                         flag_ev ← ⊤;
27                         break;
28         if flag_ev = ⊥ then
29             n_i ← newNodes_i, n_j ← newNodes_j;
30 return n_i, n_j;
```

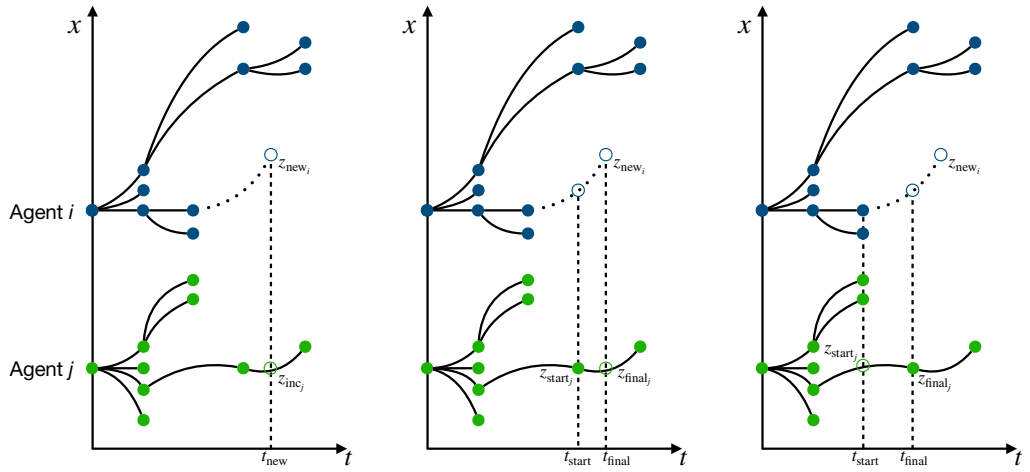


Figure 4.8: Example of interpolation considering a potential new node of agent i and an incident node of agent j

Rewire

Once a new node $z_{\text{new}\,i}$ is added to the tree of agent i, the function `Rewire` is used to rewire the tree. As already presented in Section 2.4.1, the rewiring procedure allows to find probabilistic optimal trajectories, by minimizing the path between the beginning of the tree and the nodes. Considering the newly added node $z_{\text{new}\,i}$, the function searches within a given radius in the current tree. If a neighbor node to $z_{\text{new}\,i}$ (call it $z_{\text{near}\,i}$), ahead in time with respect to $z_{\text{new}\,i}$, can be reached from $z_{\text{new}\,i}$ and the cost is lower than the current cost, the parent of $z_{\text{near}\,i}$ is updated to be $z_{\text{new}\,i}$. The pseudo-code of the rewiring procedure is reported in Algorithm 7.

In the proposed algorithm it is possible to distinguish three steps within the function `Rewire`:

1. Check if the node $z_{\text{near}\,i}$ can be reached, starting from $z_{\text{new}\,i}$, and if the obtained cost is lower than the current one (lines 4-5);
2. Check if the rewired trajectory does not collide with the obstacle and does not exit from the environment (line 6);
3. If necessary, check if the STL formula is satisfied along the rewired trajectory (lines 7-9).

Algorithm 7: Rewiring

Input: $z_{\text{new}i}$ = new node added to the tree of agent i

1 **foreach** $z_i \in V_i$ **do**
2 **if** `Dist` $(z_{\text{new}i}, z_i) < r$ **then**
3 $t = t_i - t_{\text{new}i}$;
4 $[flag, ctrl]$ = `CheckReachability` $(z_{\text{new}i}, z_i)$;
5 **if** $flag = \top \wedge z_{i\text{cost}} > z_{\text{new}i\text{cost}} +$ `Abs` $(ctrl)\cdot t$ **then**
6 **if** `NoTrajColl` $(z_{\text{new}i}, z_i, obs) \wedge$ `NoTrajExit` $(z_{\text{new}i}, z_i)$ **then**
7 $flag\_rew$ = `CheckRewireSTL` $(z_{\text{new}i}, z_i)$;
8 **if** $flag\_rew = \bot \wedge \varphi = \mathcal{G}_I(\mu)$ **then**
9 `Continue`;
10 $z_{i\text{parent}} \leftarrow z_{\text{new}i}$;
11 $z_{i\text{cost}} = z_{\text{new}i\text{cost}} +$ `Abs` $(ctrl)\cdot t$;
12 $z_{i\text{vel}} \leftarrow z_{\text{new}i\text{vel}} + ctrl \cdot t$;
13 $z_{i\text{u}} \leftarrow ctrl$;

If all the steps are passed, the parameters of $z_{\text{near}i}$ (cost, control, velocity and parent) are updated according to $z_{\text{new}i}$ (lines 10-13) and the rewired trajectory is plotted.

For what regards step 1, given the double integrator dynamics and the coordinates of $z_{\text{near}i}$ and $z_{\text{new}i}$, it is possible to compute the required control u, in order to drive the agent between the two nodes. If the resulting value is within the interval $[-u_{\max}\ u_{\max}]$, then $z_{\text{near}i}$ is actually reachable, starting from $z_{\text{new}i}$. If so, the function checks upon the feasibility of the trajectory with respect to the environment. Finally, the STL formula is evaluated along the rewired trajectory. Again, the procedure differs, depending on whether the specification concerns only one of the agents or both and depending on the type of operator.

When only one agent is involved and the STL formula contains the always operator, it is necessary to interpolate the rewired trajectory and check upon the satisfaction in the time instants within the validity domain of the formula itself. If there is at least one violation, the rewiring between $z_{\text{new}i}$ and $z_{\text{near}i}$ will not be performed.
Instead, in the case of the eventually operator, rewiring is performed, also in the case of no satisfaction, given that the STL formula could be satisfied in the

subsequent iterations.

The procedure is more complicated when both agents are involved. Starting from the always operator, consider the rewired trajectory of agent i between $z_{\text{new}\,i}$ and $z_{\text{near}\,i}$. While the previous trajectory, having $z_{\text{near}\,i}$ as final point, was satisfying the specification for construction, the rewired one could violate the formula with the corresponding trajectories of agent j. Given that the final trajectories will travel backwards through the trees, the nodes of the other agent that need to be considered are the ones sharing the time variable with $z_{\text{near}\,i}$, namely the incident ones to $z_{\text{near}\,i}$. For each of them, the function interpolates the trajectories of agent i and agent j between the same time instants, up to $z_{\text{new}\,i}$ (the procedure is the same one of the function `CheckTrajSTL`). If no violation occurs, then the rewired trajectory is plotted and the parameters of $z_{\text{near}\,i}$ are updated. An example of the described procedure is reported in Figure 4.9.

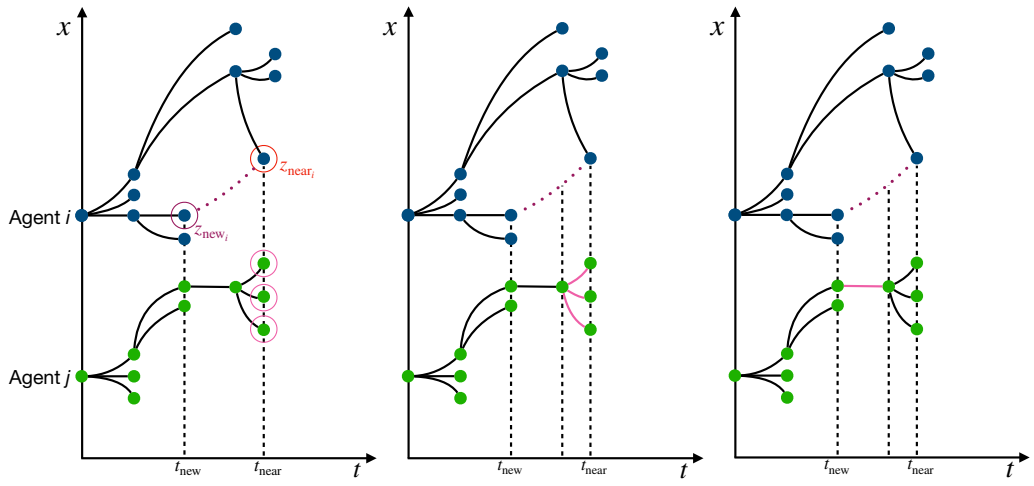


Figure 4.9: Trajectories to be interpolated in the rewiring procedure

Chapter 5

Simulations and Discussion

This chapter presents simulations carried out in Matlab to test the designed algorithm. The focus is on STL fragments of the kind $\varphi = \mu \mid \mathcal{G}_{[a,b]}\varphi \mid \mathcal{F}_{[a,b]}\varphi \mid \varphi_1 \wedge \varphi_2$.

5.1 Parameters

As already highlighted in Algorithm 2, the inputs of Coupled STL_RRT* are the following ones:

- $(f_i, X_i, U_i, x_{\text{init}_i})$ = dynamical system;
- φ = STL formula;
- obs = obstacle to be avoided;
- $t_{\max}$ = limit on the time;
- $v_{\max}$ = limit on the velocity;
- $y_{\max}$ = limit on the environment;
- $u_{\max}$ = limit on the control input;
- $N_{\max}$ = maximum number of iterations.

For what regards the simulations, double integrator dynamics is considered. Consequently, the maximum control input that can be applied to the two autonomous agents represents the maximum acceleration, expressed in m/s^2.
All the parameters of the STL formula (see Section 4.2.3) are set by the user, depending on the desired preferences. It is possible to take into account formulas of the kind $\varphi = \mathcal{F}_I \, |x_1 - x_2| \lesseqgtr d \wedge \mathcal{G}_I \, |x_1 - x_2| \lesseqgtr d$, by saving each STL fragment

of the conjunction into an array. It is important to underline the fact that it could be possible to consider also predicates that do not express only maximum or minimum distances between the agents, as soon as the corresponding predicate functions can be evaluated.
The obstacle to be avoided, instead, is static and it is represented by a rectangle, whose coordinates can be arbitrarily changed by the user. It could be also possible to define a more complex environment with various obstacles, defining an array to contain all of them.

5.2 Simulation Results

The algorithm has been tested setting the aforementioned parameters as follows:

- $x_{\text{init}_1} = 3\ m$, $x_{\text{init}_2} = -3\ m$;
- obs = rectangle (left lower corner = (3.7, 4), length = 3 m, height = 1.8 m);
- $t_{\max} = 30\ s$;
- $v_{\max} = 1\ m/s$;
- $y_{\max} = 6\ m$;
- $u_{\max} = 1.25\ m/s^2$;
- $N_{\max} = 500$.

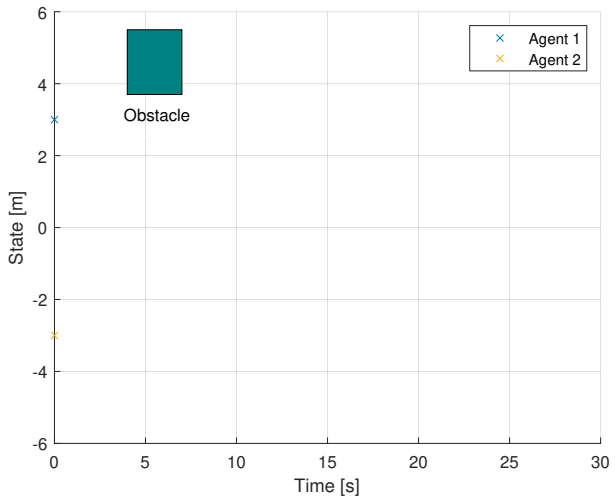


Figure 5.1: Environment and initial positions of the agents

In the next subsections, the results obtained with different STL formulas are presented. In all the cases, the tree of agent 1 is the blue one, while the vertices and edges of agent 2 are yellow. The rewired trajectories are coloured in green and the returned probabilistic optimal trajectories, satisfying the STL formula, are highlighted in red.

5.2.1 Single Predicate Formula

The first STL formula that has been taken into account contains a single predicate μ, expressing a desired minimum or maximum distance between the two autonomous agents. More specifically, the preference is encoded as $\varphi = |x_1 - x_2| > 4$, requiring the agents to be 4 units apart at all times. Agent 1 and agent 2 build their trees ensuring that every added vertex is at least 4 units away from all incident nodes of the other agent. A new node is added only if it satisfies the specification, ensuring that the final trajectories will satisfy φ as well.

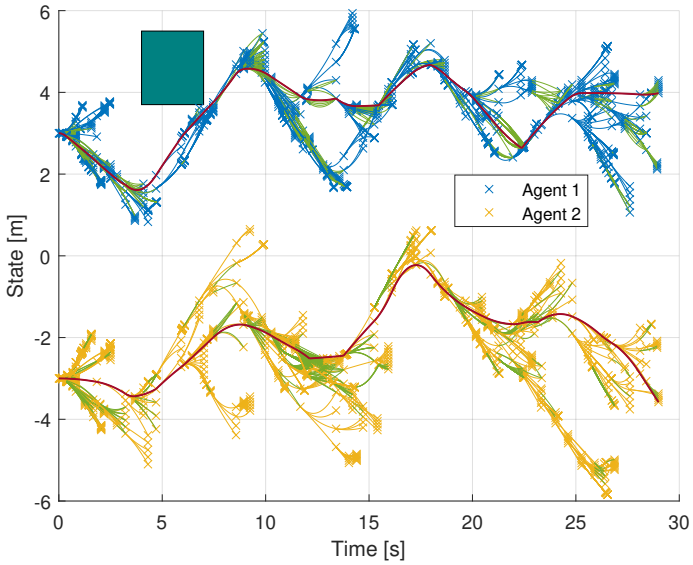
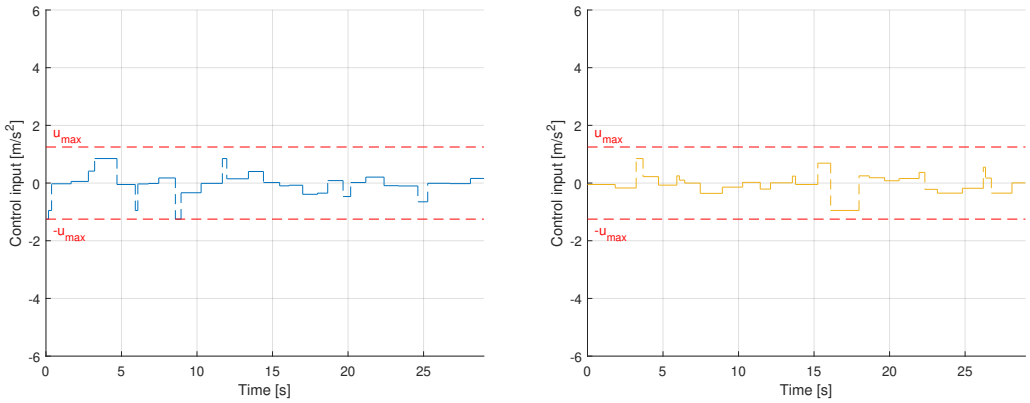


Figure 5.2: $\varphi = |x_1 - x_2| > 4$

The obtained probabilistic optimal trajectories are highlighted in Figure 5.2, while the optimal sequences of control inputs are plotted in Figure 5.3.



(a) Agent 1: optimal control sequence (b) Agent 2: optimal control sequence

Figure 5.3: Optimal control inputs for $\varphi = |x_1 - x_2| > 4$

5.2.2 Always Operator

The second formula that has been simulated contains the always operator, expressing a desired distance between the two agents in a specific time interval.

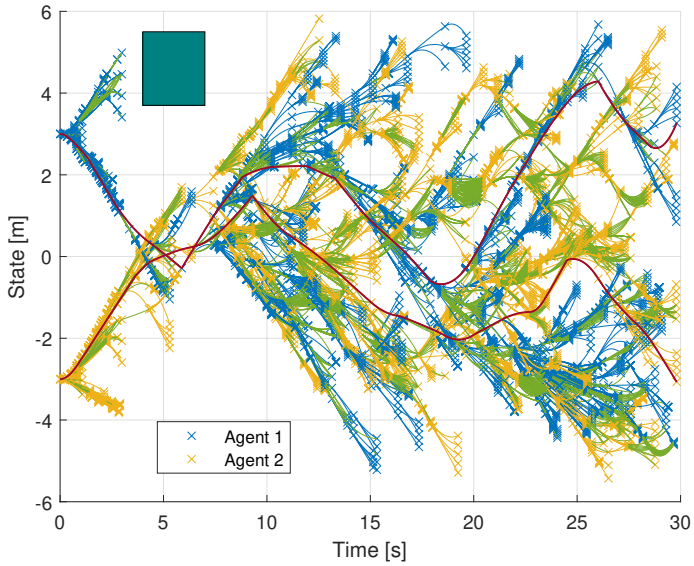
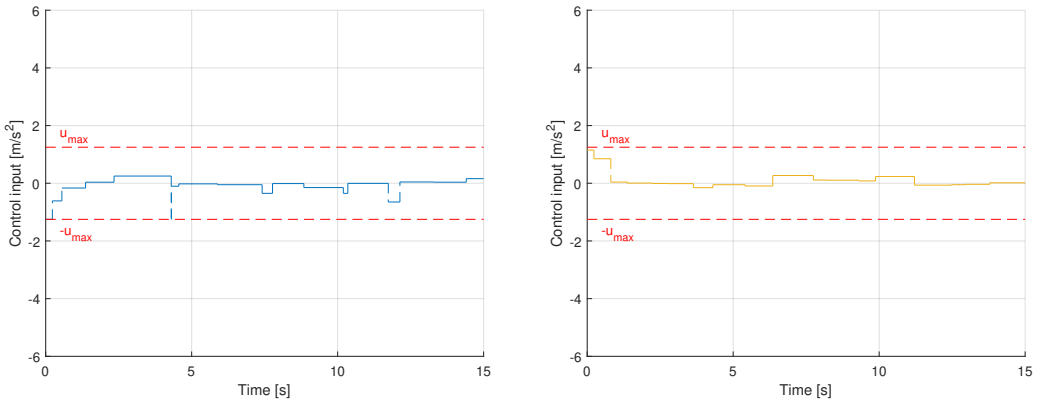


Figure 5.4: $\varphi = \mathcal{G}_{[3,8]} |x_1 - x_2| < 2$

Figure 5.4 shows the evolution of the two trees and the probabilistic optimal trajectories when the STL formula is $\varphi = \mathcal{G}_{[3,8]} |x_1 - x_2| < 2$. It requires that in the interval $[3, 8]$ every added vertex is less than 2 units apart with respect to the other agent. When the two agents need to expand the tree in $[3, 8]$, the

algorithm checks that the distance between the autonomous agents is less than 2 units, for each possible pair of incident nodes. If the constraint is not satisfied, the potential new nodes are not added to the trees. Notice that outside the validity domain of the always operator the distance could be greater than 2 units, as it happens in the first instants (initial distance is indeed equal to 6 units).



(a) Agent 1: optimal control sequence (b) Agent 2: optimal control sequence

Figure 5.5: Optimal control inputs for $\varphi = \mathcal{G}_{[3,8]} |x_1 - x_2| < 2$

5.2.3 Eventually Operator

As third case, the algorithm has been tested considering an STL specification containing the eventually operator. More specifically, the preference is given by $\varphi = \mathcal{F}_{[3,7]} |x_1 - x_2| > 5$, requiring the agents to be 5 units apart at any instant in the interval $[3, 7]$. Differently from the always operator, even if a pair of new nodes (z_{new_1}, z_{new_2}) with time variable $t_{\text{new}} \in [3, 7]$ does not satisfy φ, (z_{new_1}, z_{new_2}) are added to the corresponding trees, because the satisfaction could occur for the children nodes. As soon as two added nodes do not violate the specification, they are taken as new roots for the expansion of the trees, in order to ensure that the final trajectories will go through them, hence satisfying the STL formula. In this specific case (see Figure 5.6), the first two nodes that satisfy the STL formula are circled in red. The common time variable is $t = 4.5244\ s$, while the two state variables are $x_1 = 2.6003\ m$ and $x_2 = -2.4099\ m$.

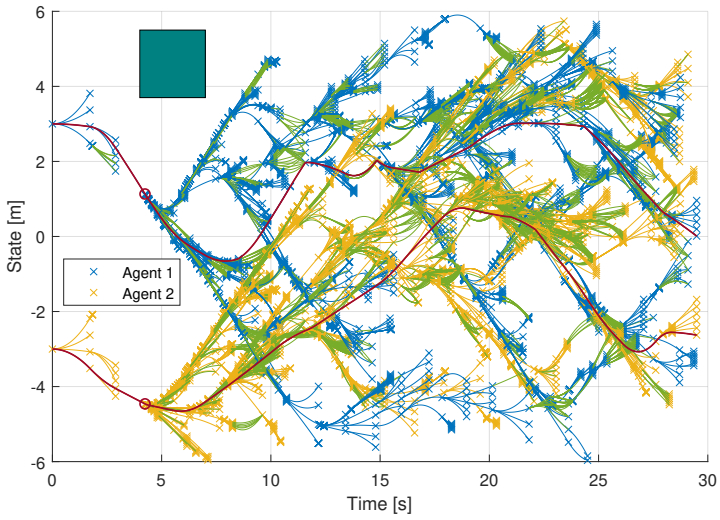
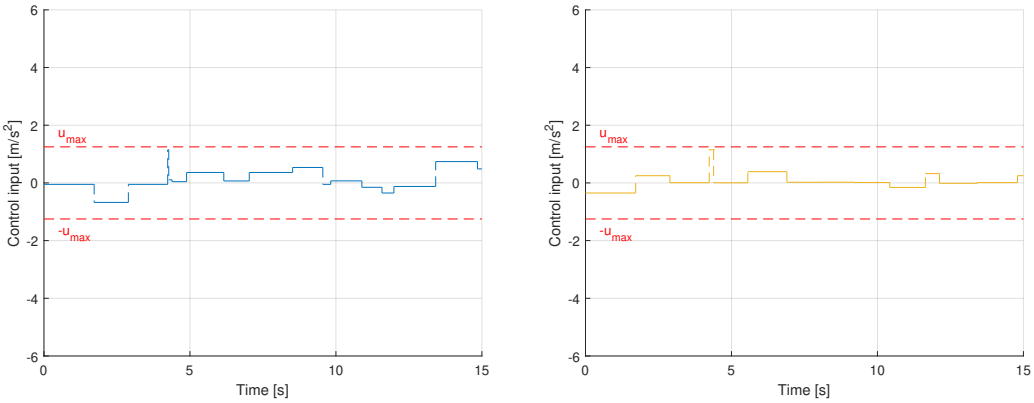


Figure 5.6: $\varphi = \mathcal{F}_{[3,7]} |x_1 - x_2| > 5$



(a) Agent 1: optimal control sequence (b) Agent 2: optimal control sequence

Figure 5.7: Optimal control inputs for $\varphi = \mathcal{F}_{[3,7]} |x_1 - x_2| > 5$

5.2.4 Nested Formulas

In the fourth case, nested formulas have been simulated. Operators can be nested into STL formulas, allowing to define specifications such as $\varphi = \mathcal{G}_{I_G}\mathcal{F}_{I_F} |x_1 - x_2| \lesseqgtr d$ (1) or $\varphi = \mathcal{F}_{I_F}\mathcal{G}_{I_G} |x_1 - x_2| \lesseqgtr d$ (2).

For the first type of nested formulas, $\varphi = \mathcal{G}_{[0,6]}\mathcal{F}_{[1,3]} |x_1 - x_2| > 4$ has been taken into account. The formula requires to satisfy $|x_1 - x_2| > 4$ every 2 seconds in the interval $[1, 9]$ in which the STL formula is active. The time interval for the evaluation of the predicate keeps being updated, until the nested formula

becomes inactive.

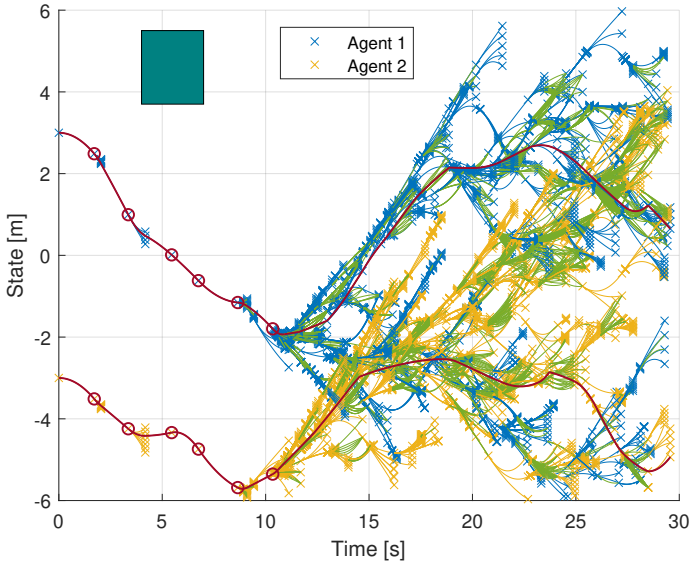
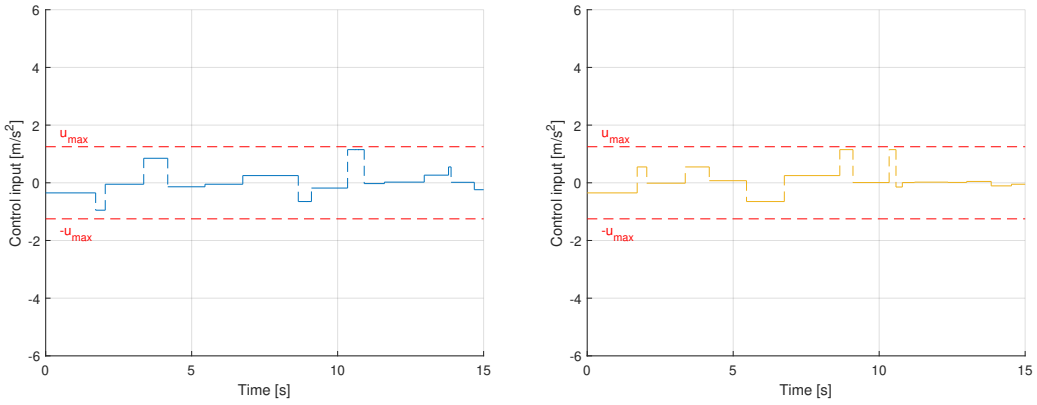


Figure 5.8: $\varphi = \mathcal{G}_{[0,6]}\mathcal{F}_{[1,3]}\,|x_1 - x_2| > 4$



(a) Agent 1: optimal control sequence (b) Agent 2: optimal control sequence

Figure 5.9: Optimal control inputs for $\varphi = \mathcal{G}_{[0,6]}\mathcal{F}_{[1,3]}\,|x_1 - x_2| > 4$

In Figure 5.8, the vertices that satisfy the eventually operator are circled in red. They are then used as new roots for the expansion of the trees, in order to ensure that the final trajectories will pass through them. In particular, the selected pair of nodes are the following ones:

- $t = 1.7118\ s$, $x_1 = 2.4872\ m$ and $x_2 = \text{-}3.5128\ m$;
- $t = 3.3582\ s$, $x_1 = 0.9950\ m$ and $x_2 = \text{-}4.2435\ m$;

- t = 5.4563 s, x_1 = 0.0119 m and x_2 = -4.3372 m;
- t = 6.7514 s, x_1 = -0.6186 m and x_2 = -4.7413 m;
- t = 8.6519 s, x_1 = -1.1539 m and x_2 = -5.6827 m;
- t = 10.3414 s, x_1 = -1.7973 m and x_2 = -5.3531 m;

For the second type of nested formulas, instead, $\varphi = \mathcal{F}_{[0,10]}\mathcal{G}_{[1,5]}\,|x_1 - x_2| < 3$ has been tested. The preference requires to satisfy $|x_1 - x_2| < 3$ for four seconds in the time interval $[1, 15]$.

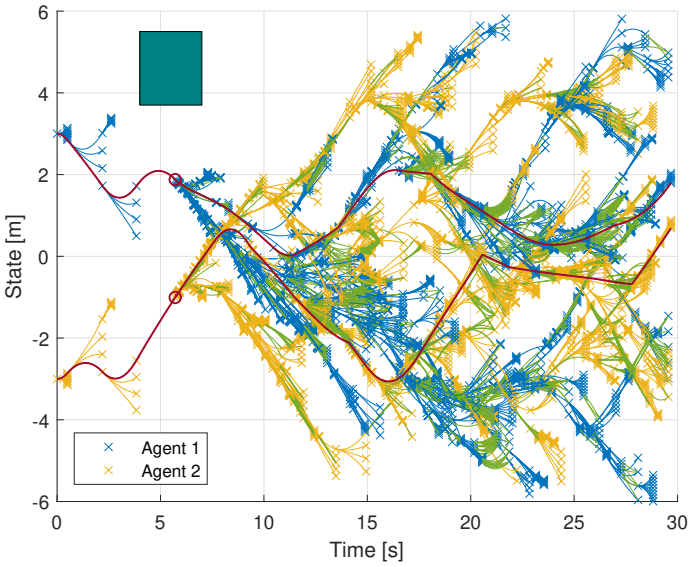
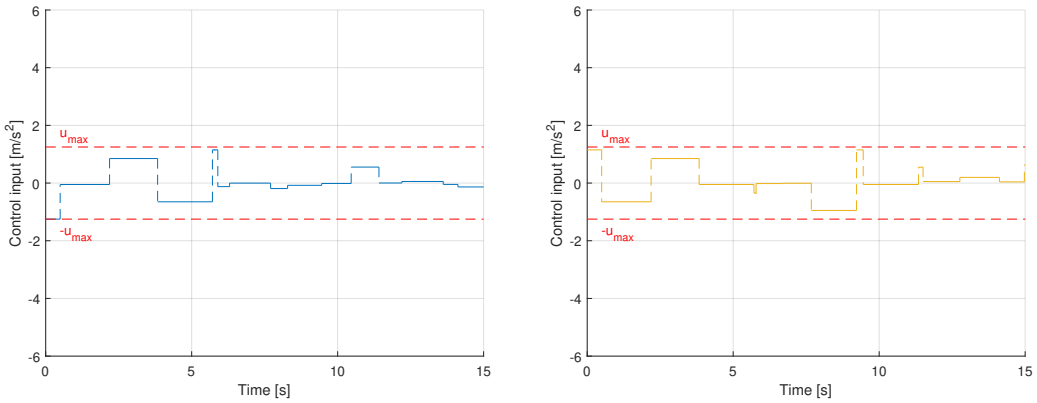


Figure 5.10: $\varphi = \mathcal{F}_{[0,10]}\mathcal{G}_{[1,5]}\,|x_1 - x_2| < 3$

From Figure 5.10, it is possible to observe that the first time instant in [0, 10] in which the predicate $|x_1 - x_2| < 3$ is verified is t = 5.7112 s. The corresponding state variables in the trees have values x_1 = 1.8772 m and x_2 = -1.0060 m. Starting from them, the algorithm imposes to have a maximum distance of 3 units between the two agents in the next 4 seconds, at least in the interval [5.7112 9.7112] seconds. In this way, the nested formula is satisfied and then deactivated. For t > 10 s, instead, the trees are allowed to have vertices at distance greater than 3 units.
The optimal sequences of control inputs are reported in Figure 5.11.



(a) Agent 1: optimal control sequence (b) Agent 2: optimal control sequence

Figure 5.11: Optimal control inputs for $\varphi = \mathcal{F}_{[0,10]}\mathcal{G}_{[1,5]} |x_1 - x_2| < 3$

5.2.5 Conjunctions

In order to express even more complex specifications, STL formulas containing conjunctions have been simulated. In particular, consider $\varphi = \mathcal{G}_{[0,10]}(x_1 > 0) \wedge \mathcal{G}_{[0,6]} |x_1 - x_2| > 3$. Agent 1 is required to sample only in the set $x_1 > 0$ in [0, 10], while the two agents need to keep a minimum distance equal to 3 units in the interval [0, 6] (see Figure 5.12).

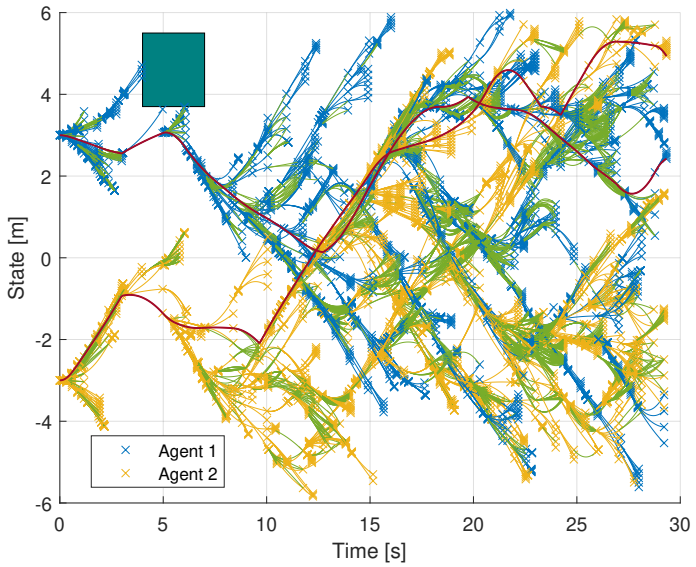
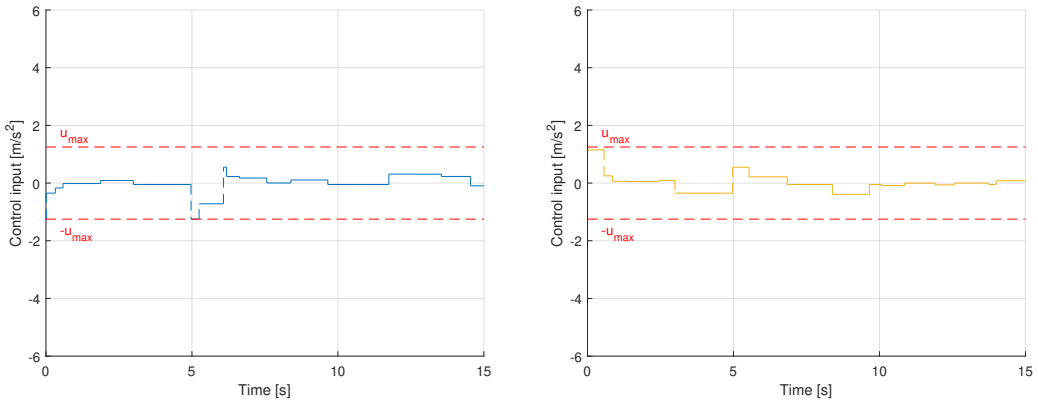


Figure 5.12: $\varphi = \mathcal{G}_{[0,10]}(x_1 > 0) \wedge \mathcal{G}_{[0,6]} |x_1 - x_2| > 3$



(a) Agent 1: optimal control sequence (b) Agent 2: optimal control sequence

Figure 5.13: Optimal control inputs for $\varphi = \mathcal{G}_{[0,10]}(x_1 > 0) \wedge \mathcal{G}_{[0,6]} |x_1 - x_2| > 3$

5.2.6 Combination of Nested Operators and Conjunctions

The last example contains conjunctions of nested formulas and single operators, expressing preferences related to a single agent and to both as well.

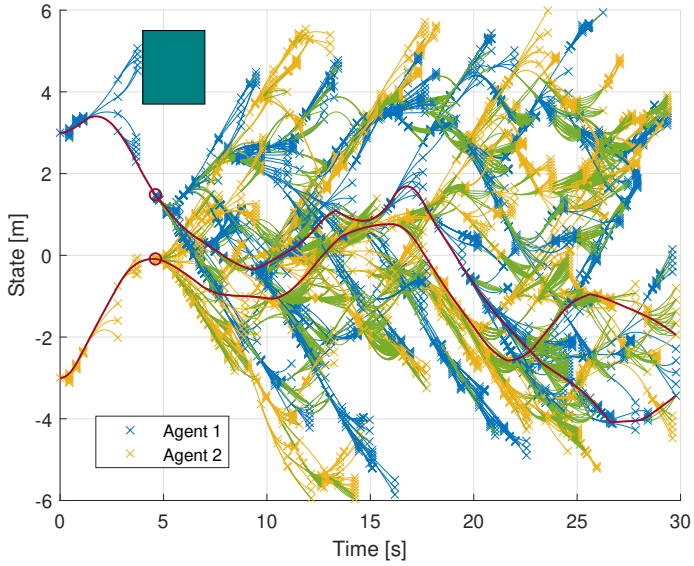
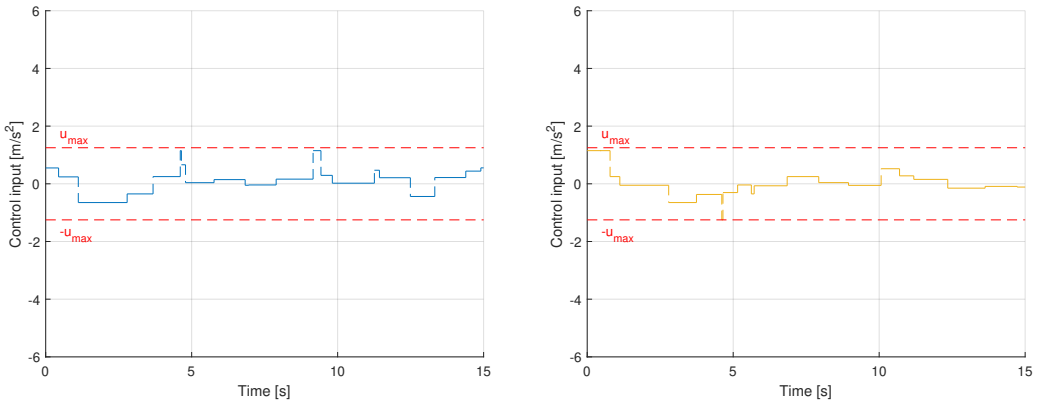


Figure 5.14: $\varphi = \mathcal{G}_{[0,5]}(x_1 > 0) \wedge \mathcal{G}_{[0,5]}(x_2 < 0) \wedge \mathcal{F}_{[4,10]}\mathcal{G}_{[1,5]} |x_1 - x_2| < 2$

$\varphi = \mathcal{G}_{[0,5]}(x_1 > 0) \wedge \mathcal{G}_{[0,5]}(x_2 < 0) \wedge \mathcal{F}_{[4,10]}\mathcal{G}_{[1,5]} |x_1 - x_2| < 2$ is the formula that has been simulated. Agent 1 is required to sample only in the set $x_1 > 0$ in the interval $[0, 5]$, agent 2 is required to sample only in the set $x_2 < 0$ in $[0, 5]$ and the distance between them must be less than 2 units for 4 seconds in the time

interval [5, 15]. As already seen in the second example of the nested formulas, as soon as two nodes satisfy the predicate $|x_1 - x_2| < 2$, the algorithm imposes to keep a distance less than 2 units also in the next 4 seconds.



(a) Agent 1: optimal control sequence (b) Agent 2: optimal control sequence

Figure 5.15: Optimal control inputs for $\varphi = \mathcal{G}_{[0,5]}(x_1 > 0) \wedge \mathcal{G}_{[0,5]}(x_2 < 0) \wedge \mathcal{F}_{[4,10]}\mathcal{G}_{[1,5]} |x_1 - x_2| < 2$

Chapter 6

Multi-agent Extension

This chapter extends the algorithm Coupled STL_RRT* to multiple agents. The work is divided as follows: Section 6.1 presents the problem formulation; Section 6.2 discusses the extensions and changes of Coupled STL_RRT* to deal with multi-agent systems; Section 6.3 contains simulations; Section 6.4 exposes the 3D extension and its test on a 3-agents system.

6.1 Problem Formulation

The problem to be addressed in this chapter is the cooperative motion planning problem of multiple agents, subject to coupled STL tasks.

A multi-agent system can be represented by an undirected graph $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$, where each node is an agent and each edge represents the communication between two agents. Let $|\mathbf{V}| = N$ and φ be the STL formula encoding preferences for the multi-agent system. When multiple agents are involved, STL specifications could include coupled constraints between agents, such as $\mu = |x_2 - x_3| < 3$. Therefore, let M be the set of all predicates in the STL formula φ, and $\mathbf{V}' = \{v_i \in \mathbf{V} \mid x_i \text{ appears in any predicate of } \varphi\}$ be the set of all nodes whose states appear in the predicates of the STL formula φ. Furthermore, let $\mathbf{E}' = \{\{v_i, v_j\} \in \mathbf{E} \mid x_i, x_j \text{ appear in a single predicate } \mu \in M\}$. Then the graph $\mathbf{G}' = \{\mathbf{V}', \mathbf{E}'\}$ is the task dependency graph, a sub-graph of the connection graph [34]. Clearly, agents that have coupled tasks are neighbours in both $\mathbf{G}$ and $\mathbf{G}'$ (see Figure 6.1 for an example).

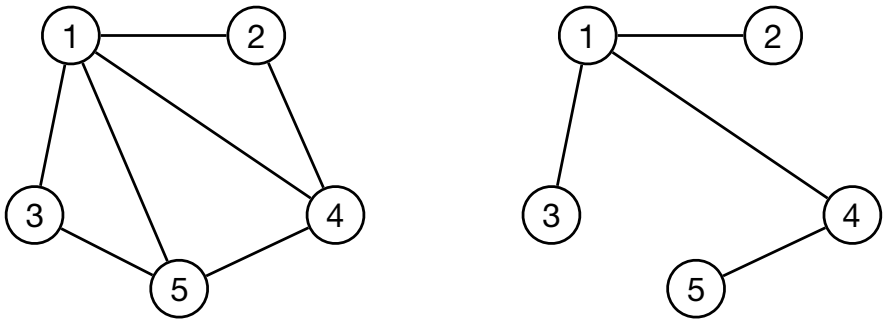


Figure 6.1: Example of connection graph (left) and task dependency graph (right)

At this point, given a multi-agent system represented by a graph and an STL formula φ expressing coupled constraints between the agents, the problem to be solved is an extension of Problem 4.1 and can be formulated as follows:

Problem 6.1. Given a multi-agent system with N agents, each of them represented by the dynamical system $\mathcal{R}_i$, $i = 1, ..., N$, and an STL formula φ, find control policies $\boldsymbol{u}_i^*$ with time domain $[0, T]$ such that the costs of the state trajectories, originating in $\boldsymbol{x}_{\text{init}_i}$ under $\boldsymbol{u}_i^*$ and satisfying the STL specification, are minimized, i.e.

- $\boldsymbol{x}_i \left[\boldsymbol{x}_{\text{init}i}, \boldsymbol{u}_i^*\right] \in X_{\text{free}}$ for all $t \in [0, T]$, $i = 1, ..., N$;
- $\boldsymbol{x}_i \left[\boldsymbol{x}_{\text{init}i}, \boldsymbol{u}_i^*\right] \models \varphi$, $i = 1, ..., N$;
- $\boldsymbol{u}_i^* = \underset{\boldsymbol{u}_i \in \mathcal{U}_i}{\text{argmin}} \ \text{cost}\left(\boldsymbol{x}_i \left[\boldsymbol{x}_{\text{init}i}, \boldsymbol{u}_i\right]\right)$, $i = 1, ..., N$.

6.2 MA-Coupled STL_RRT*

This section provides the proposed solution for the cooperative optimal motion planning problem of multi-agent systems under STL specifications. The designed approach is an extension of the algorithm Coupled STL_RRT*, presented in Section 4.2. Again, three are the main components: the sampling-based procedure to expand the tree, the STL verification to check upon the satisfaction of

Algorithm 8: MA-Coupled STL_RRT*: Agent i perspective

Input: $(f_i, X_i, U_i, x_{\text{init}i})$ = dynamical system
φ = STL formula
obs = obstacle to be avoided
$t_{\max}, v_{\max}, y_{\max}, u_{\max}, N_{\max}$ = limits on parameters
Output: u_i = optimal control sequence

$T_i \leftarrow (V_i = \emptyset, E_i = \emptyset)$;
$z_{\text{init}i} \leftarrow (0, x_{\text{init}i})$;
$V_i \leftarrow$ `InsertNode` $(z_{\text{init}i}, V_i)$;
for $i = 1$ **to** $i = N_{\max}$ **do**
  $z_{\text{rand}i} \leftarrow$ `RandomState` ();
  while ! `NoPointCollision` $(z_{\text{rand}i}, obs)$ **do**
    $z_{\text{rand}i} \leftarrow$ `RandomState` ();
  $z_{\text{nearest}i} \leftarrow$ `Nearest` $(z_{\text{rand}i}, V_i)$;
  $z_{\text{nearest}i} \leftarrow$ `Synchro` $(z_{\text{nearest}i})$;
  $R_i \leftarrow$ `ReachableSet` $(z_{\text{nearest}i})$;
  $newNodes_i \leftarrow$ `ComputeNn` (R_i) ;
  $newNodes_i \leftarrow$ `BestParent` (V_i) ;
  if $t_{\text{new}} \in I_\varphi$ **then**
    $newNodes_i \leftarrow$ `CheckPointSTL` $(newNodes_i, \varphi)$;
  $newNodes_i \leftarrow$ `CheckTrajSTL` $(newNodes_i, \varphi)$;
  foreach $z_{\text{new}i} \in newNodes_i$ **do**
    $V_i \leftarrow$ `InsertNode` $(z_{\text{new}i}, V_i)$;
    `Rewire` $(z_{\text{new}i})$;
$path_i \leftarrow$ `FindOptPath` (V_i);
$u_i \leftarrow$ `FindOptU` $(path_i)$;
return (u_i);

the STL formula, and the rewiring procedure to obtain minimum-cost trajectories. Among the multiple agents, let i be the one running the algorithm. The pseudo-code is reported in Algorithm 8.

As in the 2-agents scenario, the algorithm builds for every agent a tree, $T = (V, E)$, consisting of a set of vertices $\{z_i\}$ connected by edges. Each agent is identified by a `struct` a_i, $i = 1, ..., N$; they are all saved into the `struct` a, where the first position contains a_1, the second a_2 and so on. For what regards the STL specifications, instead, one addition is required. Indeed, in the 2-agents case, predicates encode constraints related to either one of the agents or both of them. When multiple agents are taken into account, STL specifications express coupled constraints between agents. Consequently, for each STL fragment in φ, it

is necessary to indicate the involved entities, by defining the parameters $agent_1$ and $agent_2$. Note that, in the general case, STL tasks could encode preferences related to more than two agents, such as $\varphi = \mathcal{G}_{[2,6]} |x_1 + x_2 + x_3| < 5$. However, in this work, the straightforward extension of Coupled STL_RRT* considers only coupled constraints.

Highlighted lines in Algorithm 8 are the ones that have been changed with respect to the case in which only two agents are involved. For what regards the function `Synchro`, when multiple agents are considered, it is necessary to select the minimum time variable between all the nearest nodes. At this point, all the agents (excluded the one whose z_{nearest} has the minimum t_{nearest}) must be synchronized, following the procedure explained in Chapter 4. Concerning `CheckPointSTL` and `CheckTrajSTL`, instead, the main change is related to the fact that the algorithm is required to extract the two involved agents from the considered STL fragment. Hence, for each predicate in φ, the parameters $agent_1$ and $agent_2$ are consulted, in order to check upon the verification of the STL specification taking into account only the implicated agents.

6.3 Simulations

MA-Coupled STL_RRT* has been tested considering different multi-agent systems, starting from 4 agents and then increasing the number of autonomous entities and the complexity of the STL formula. In all the simulations, the inputs of the algorithm have been set as follows:

- obs = rectangle (left lower corner = (4, 7), length = 3 m, height = 2 m)
- $t_{\text{max}} = 15\ s$;
- $v_{\text{max}} = 1\ m/s$;
- $y_{\text{max}} = 10\ m$;
- $u_{\text{max}} = 1.25\ m/s^2$;
- $N_{\text{max}} = 200$.

Depending on the number of agents, instead, the initial positions x_{init_i} have been allocated differently.

6.3.1 Simulations with 4 Autonomous Agents

The first tests that have been performed take into account a multi-agent system with 4 autonomous agents. The initial positions of the agents have been set as follows: $x_{\text{init}_1} = 6\ m$, $x_{\text{init}_2} = 2\ m$, $x_{\text{init}_3} = -2\ m$, $x_{\text{init}_4} = -6\ m$.
The STL specification that has been simulated is $\varphi = \mathcal{G}_{[2,6]} |x_1 - x_2| < 5 \wedge \mathcal{G}_{[0,6]} |x_1 - x_4| > 8 \wedge \mathcal{F}_{[0,7]} |x_1 - x_3| < 7 \wedge \mathcal{F}_{[3,10]} |x_3 - x_4| > 4$. Agents 1 and 2 are required to be less than 5 units apart at every instant in the interval $[2, 6]$; agents 1 and 4 are required to be at least 8 units apart at every instant in the interval $[0, 6]$; agents 1 and 3 are required to be less than 7 units apart at any instant in the interval $[0, 7]$; agents 3 and 4 are required to be at least 4 units apart at any instant in the interval $[3, 10]$. When the specification includes the eventually operator, the nodes that satisfy it are circled in red and taken as new roots for the future expansion of the trees, as already explained in Chapter 4.

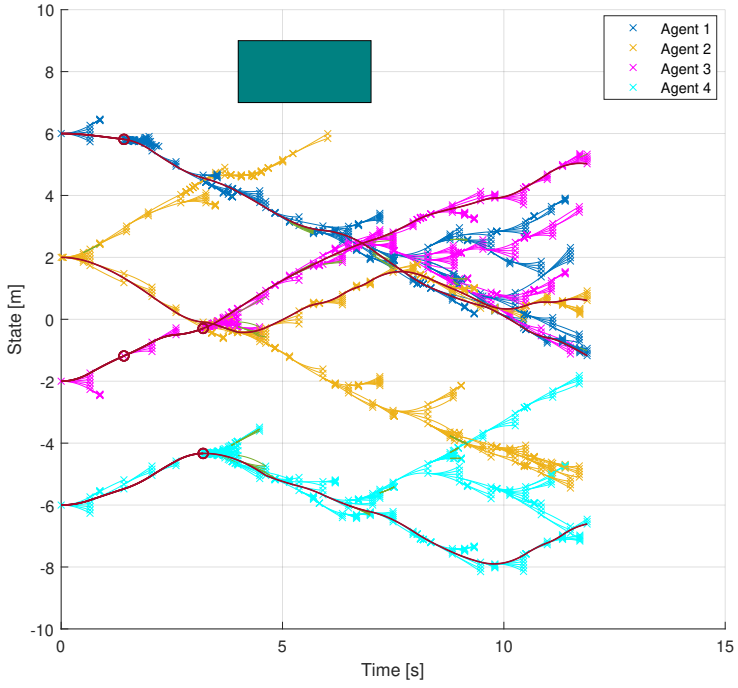


Figure 6.2: $\varphi = \mathcal{G}_{[2,6]} |x_1 - x_2| < 5 \wedge \mathcal{G}_{[0,6]} |x_1 - x_4| > 8 \wedge \mathcal{F}_{[0,7]} |x_1 - x_3| < 7 \wedge \mathcal{F}_{[3,10]} |x_3 - x_4| > 4$

Maintaining the same multi-agent system, nested STL formulas have been tested. In particular, the considered specification is $\varphi = \mathcal{F}_{[0,6]} |x_1 - x_2| > 5 \wedge \mathcal{F}_{[2,8]}\mathcal{G}_{[0,2]} |x_1 - x_3| < 7 \wedge \mathcal{G}_{[3,6]} |x_3 - x_4| > 2$. For what regards the nested formula related to agents 1 and 3, it is required to satisfy $|x_1 - x_3| < 7$ for 2

seconds in the time interval $[2, 10]$. The resulting trajectories are highlighted in Figure 6.3.

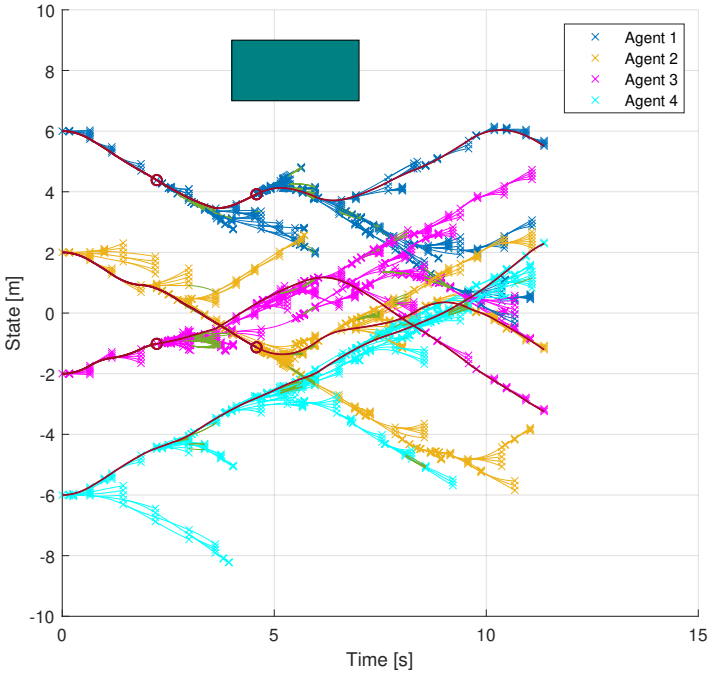


Figure 6.3: $\varphi = \mathcal{F}_{[0,6]} |x_1 - x_2| > 5 \wedge \mathcal{F}_{[2,8]}\mathcal{G}_{[0,2]} |x_1 - x_3| < 7 \wedge \mathcal{G}_{[3,6]} |x_3 - x_4| > 2$

6.3.2 Simulations with 6 Autonomous Agents

When the multi-agent system has been extended to include 6 agents, the initial positions of the agents have been set as follows: $x_{\text{init}_1} = 7\,m$, $x_{\text{init}_2} = 4\,m$, $x_{\text{init}_3} = 1\,m$, $x_{\text{init}_4} = -1\,m$, $x_{\text{init}_5} = -4\,m$, $x_{\text{init}_6} = -7\,m$.
The first simulated STL formula contains 5 coupled predicates between the agents: $\varphi = \mathcal{G}_{[2,5]} |x_1 - x_2| < 3 \wedge \mathcal{F}_{[0,7]} |x_2 - x_3| > 4 \wedge \mathcal{G}_{[0,3]} |x_3 - x_4| < 3 \wedge \mathcal{F}_{[3,6]} |x_4 - x_5| < 2 \wedge \mathcal{G}_{[4,7]} |x_5 - x_6| < 3$. More specifically, agents 1 and 2 are required to be less than 3 units apart at every instant in the interval $[2, 5]$; agents 2 and 3 are required to be at least 4 units apart at any instant in the interval $[0, 7]$; agents 3 and 4 are required to be less than 3 units apart at every instant in the interval $[0, 3]$; agents 4 and 5 are required to be less than 2 units apart at any instant in the interval $[3, 6]$; agents 5 and 6 are required to be less than 3 units apart at every instant in the interval $[4, 7]$ (see Figure 6.4).
The second simulated STL formula is $\varphi = \mathcal{F}_{[5,10]} |x_1 - x_5| < 10 \wedge \mathcal{G}_{[0,5]} |x_1 - x_6| > 7 \wedge \mathcal{F}_{[2,5]} |x_5 - x_4| > 2 \wedge \mathcal{G}_{[8,11]} |x_2 - x_4| < 6 \wedge \mathcal{F}_{[0,5]}\mathcal{G}_{[0,2]} |x_2 - x_3| > 1$. With

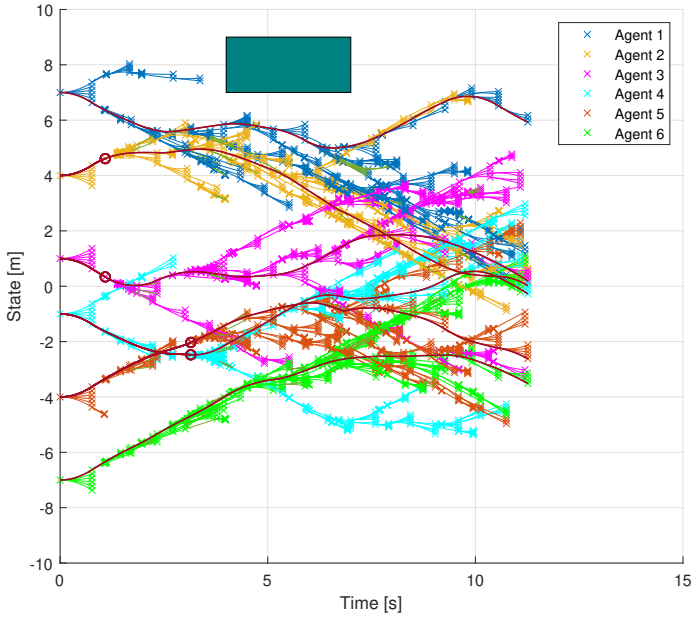


Figure 6.4: $\varphi = \mathcal{G}_{[2,5]} |x_1 - x_2| < 3 \wedge \mathcal{F}_{[0,7]} |x_2 - x_3| > 4 \wedge \mathcal{G}_{[0,3]} |x_3 - x_4| < 3 \wedge \mathcal{F}_{[3,6]} |x_4 - x_5| < 2 \wedge \mathcal{G}_{[4,7]} |x_5 - x_6| < 3$

respect to the previous case, where only single operators are considered, nested formulas have been added (Figure 6.5).

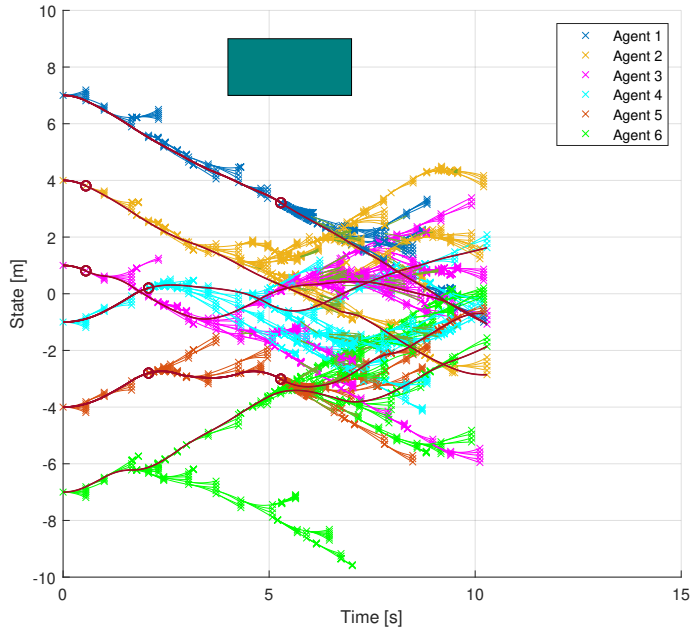


Figure 6.5: $\varphi = \mathcal{F}_{[5,10]} |x_1 - x_5| < 10 \wedge \mathcal{G}_{[0,5]} |x_1 - x_6| > 7 \wedge \mathcal{F}_{[2,5]} |x_5 - x_4| > 2 \wedge \mathcal{G}_{[8,11]} |x_2 - x_4| < 6 \wedge \mathcal{F}_{[0,5]}\mathcal{G}_{[0,2]} |x_2 - x_3| > 1$

6.4 3D Extension

The algorithm MA-Coupled STL_RRT* has been extended to deal with the control of autonomous agents, whose state is specified as a 2-dimensional vector, having as components the x and the y position. Also in this case, the robotic systems are supposed to evolve in time and space following a double integrator dynamics.

For each agent, the algorithm builds a tree, $T = (V, E)$, consisting of a set of vertices $\{z_i\}$ connected by edges. With respect to the two-dimensional case, where only the position along x is controlled, the 3D strategy requires as control input a 2-dimensional vector, whose first component is u_x, namely the acceleration along x, and the second one is u_y, namely the acceleration along y. The velocity vector, moreover, contains both v_x and v_y, obtained respectively with the equations $v_x(t) = v_x(0) + u_x \cdot t$ and $v_y(t) = v_y(0) + u_y \cdot t$. Consequently, each node is associated with a `struct` containing the following parameters:

- $coord$: a 3-dimensional vector, with components given by the x position, the y position and the time variable associated to the state;
- vel: a 2-dimensional vector, with components given by the velocity of the dynamical system at that node along x and y;
- $parent$: parent of the node;
- u: a 2-dimensional vector, with components given by the control to be applied along x and y to reach the node starting from its parent;
- $cost$: cost to reach the node starting from the root of the tree;
- pos: position of the node in the array containing all the vertices of the tree.

The strategy comprehends three main components, as in the case of the previously proposed approaches: the sampling-based procedure to expand the tree, the STL verification to check upon the satisfaction of the STL formula, and the rewiring procedure to obtain minimum-cost trajectories. The functions involved are the same presented in Algorithm 2 and 8, extended to deal with 3 dimensions. In particular, when the reachable states are computed, the algorithm considers all the combinations of control inputs u_x and u_y, discretized in the same interval [-u_{max} u_{max}].

6.4.1 Simulations

The 3D extension has been simulated by setting the following parameters:

- $(x_{\text{init}_1} = -1\ m,\ y_{\text{init}_1} = 1\ m)$, $(x_{\text{init}_2} = 0\ m,\ y_{\text{init}_2} = -1\ m)$, $(x_{\text{init}_3} = 1\ m,\ y_{\text{init}_3} = 1\ m)$;
- obs = rectangle (left lower corner = (1.5, 1.5), length = 0.4 m, height = 0.4 m);
- $t_{\max} = 60\ s$;
- $x_{\max} = y_{\max} = 2\ m$;
- $v_{x_{\max}} = v_{y_{\max}} = 0.3\ m/s$;
- $u_{x_{\max}} = u_{y_{\max}} = 0.5\ m/s^2$;
- $N_{\max} = 200$.

The STL formula taken into account is $\varphi = |x_1 - x_2| > 0.6 \wedge |x_1 - x_3| > 0.6 \wedge |x_3 - x_2| > 0.6$. More specifically, it requires the three autonomous agents to maintain a minimum distance of 0.6 units during all the motion. The results are highlighted in Figure 6.6.

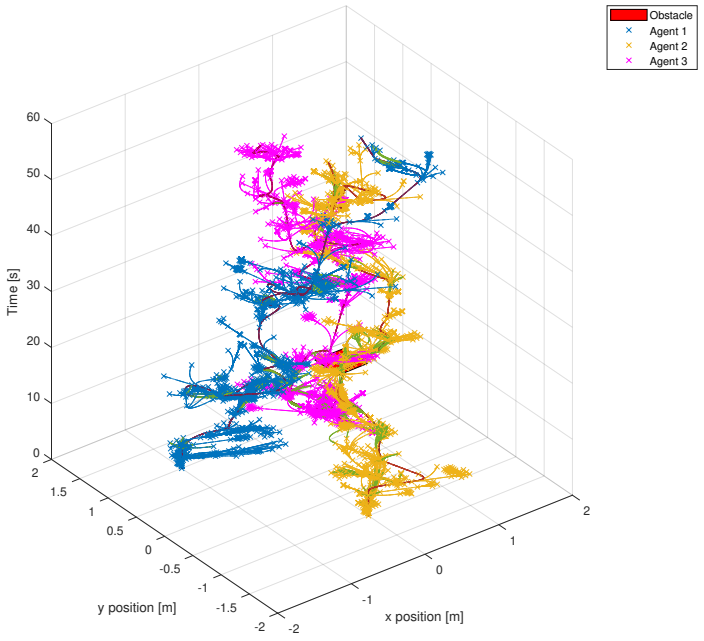


Figure 6.6: $\varphi = |x_1 - x_2| > 0.6 \wedge |x_1 - x_3| > 0.6 \wedge |x_3 - x_2| > 0.6$

Chapter 7

Conclusions and Future Research Directions

This chapter summarizes the thesis and presents possible research directions for future work.

7.1 Conclusions

In this thesis, a multi-agent motion planning approach has been developed, aiming at finding probabilistic optimal trajectories for multiple autonomous agents under STL tasks.

Firstly, in Chapter 4 the problem has been solved taking into account only two agents, subject to a coupled STL specification. The design algorithm, Coupled STL_RRT*, samples points in the coupled state space and builds two spatio-temporal trees, one for each agents, in a distributed manner. The procedure is based on RRT* and exploits reachability to add states that are feasible with respect to the dynamics of the agents. The approach has been simulated considering different types of STL formulas, proving its efficiency and capability of returning probabilistic optimal trajectories satisfying the STL constraints (Chapter 5).

In Chapter 6 the algorithm has been extended to multi-agent systems comprehending more than three agents. Again, three are the main components of MA-Coupled STL_RRT*: the sampling-based procedure to expand the tree,

the STL verification to check upon the satisfaction of an STL formula, and the rewiring procedure to obtain minimum-cost trajectories. Simulations have been performed, considering coupled multi-agent systems with 4 and 6 agents respectively. Although the computational time increases when more and more autonomous agents are considered, the algorithm is still able to find probabilistic optimal trajectories that do not violate the STL requirements selected by the user. The chapter presents also the 3D extension of MA-Coupled STL_RRT*, simulated considering three autonomous agents that are controlled along x and y, taking into account a double integrator dynamics for both components.

7.2 Future Research Directions

The approach developed in this project concentrates on autonomous agents whose dynamics is supposed to be the one of a double integrator. In the general case, however, dynamical systems are described by non linear differential equations, with n-dimensional state spaces. The proposed algorithm, hence, could be extended to deal with different type of autonomous agents and dynamic constraints, exploiting again reachability to move the agents towards feasible states.

The exposition in Chapter 4 and 6 assumes that the agents move in an environment with only static obstacles, represented by rectangles. Indeed, this thesis proposes an offline approach for supervision and random exploration of defined environments, where the position of the obstacles is known beforehand. However, in real-world warehouse settings and autonomous guidance applications, environment changes might occur. In these cases, the workspace is usually a partially-known environment with moving non-static obstacles (such as human beings or other agents), whose movements are typically unpredictable. In order to account for dynamic obstacles and unexplored regions, hence, a possible improvement could be the change to an online approach, amenable to real-time computation. As a consequence, robots could be able to adapt their trajectories while exploring, if dynamic obstacles appear along their paths.

The thesis addresses the multi-agent motion planning problem under STL specifications that are supposed to be feasible. However, it could happen that the user specifies STL constraints that are not achievable. In these cases, the robustness metric could be exploited to find solutions that minimize the violation of

the logical formula.

To conclude, in numerous industrial applications, humans provide high-level tasks to robots. These requirements could change, requiring to update and adapt the STL specifications in real-time. Motion planning algorithms, hence, need to be able to re-plan trajectories in order to satisfy the changing tasks while autonomous agents are operating. To adapt to changing environmental and contextual conditions, it becomes necessary to complement reactive behaviours with replanning, by integrating sensory information [7]. This leads to the need of developing decentralized reactive online re-plan methods, capable of coordinating motion among agents, avoiding collisions and satisfying the assigned requirements.

References

[1] A. Dorri, S. S. Kanhere, and R. Jurdak. "Multi-Agent Systems: A Survey". In: *IEEE Access* 6 (2018). Conference Name: IEEE Access. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2831228.

[2] M. Oprea. "Applications of Multi-Agent Systems". In: *Information Technology*. Ed. by Ricardo Reis. IFIP International Federation for Information Processing. Boston, MA: Springer US, 2004, pp. 239–270. ISBN: 978-1-4020-8159-0. DOI: 10.1007/1-4020-8159-6_9.

[3] L. Chaimowicz et al. "An architecture for tightly coupled multi-robot cooperation". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164). Vol. 3. ISSN: 1050-4729. May 2001, 2992–2997 vol.3. DOI: 10.1109/ROBOT.2001.933076.

[4] W. Shen and D. H. Norrie. "Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey". In: *Knowledge and Information Systems* 1.2 (May 1, 1999), pp. 129–156. ISSN: 0219-3116. DOI: 10.1007/BF03325096. URL: https://doi.org/10.1007/BF03325096.

[5] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. "Temporal Logic Motion Planning for Mobile Robots". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. Proceedings of the 2005 IEEE International Conference on Robotics and Automation. ISSN: 1050-4729. Apr. 2005, pp. 2020–2025. DOI: 10.1109/ROBOT.2005.1570410.

[6] F. S. Barbosa et al. "Guiding Autonomous Exploration With Signal Temporal Logic". In: *IEEE Robotics and Automation Letters* 4.4 (Oct. 2019). Conference Name: IEEE Robotics and Automation Letters, pp. 3332–3339. ISSN: 2377-3766. DOI: 10.1109/LRA.2019.2926669.

[7] E. Plaku and S. Karaman. "Motion Planning with Temporal-logic Specifications: Progress and Challenges". In: 2016.

[8] L. Lindemann. "Planning and Control of Multi-Agent Systems under Signal Temporal Logic Specifications". In: (2020). Publisher: KTH Royal Institute of Technology. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-281428.

[9] M. Sewlia. "Cooperative Manipulation and Motion Planning Under Signal Temporal Logic Specifications". In: (2023). Publisher: KTH Royal Institute of Technology. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-325288.

[10] I. Haghighi et al. "Control from Signal Temporal Logic Specifications with Smooth Cumulative Quantitative Semantics". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019 IEEE 58th Conference on Decision and Control (CDC). ISSN: 2576-2370. Dec. 2019, pp. 4361–4366. DOI: 10.1109/CDC40024.2019.9029429.

[11] Z. Liu et al. "Distributed communication-aware motion planning for multi-agent systems from STL and SpaTeL specifications". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017 IEEE 56th Annual Conference on Decision and Control (CDC). Dec. 2017, pp. 4452–4457. DOI: 10.1109/CDC.2017.8264316.

[12] L. Lindemann et al. "Coupled Multi-Robot Systems Under Linear Temporal Logic and Signal Temporal Logic Tasks". In: *IEEE Transactions on Control Systems Technology* 29.2 (Mar. 2021). Conference Name: IEEE Transactions on Control Systems Technology, pp. 858–865. ISSN: 1558-0865. DOI: 10.1109/TCST.2019.2955628.

[13] R. Koymans. "Specifying real-time properties with metric temporal logic". In: *Real-Time Systems* 2.4 (Nov. 1, 1990), pp. 255–299. ISSN: 1573-1383. DOI: 10.1007/BF01995674. URL: https://doi.org/10.1007/BF01995674.

[14] C. I. Vasile, V. Raman, and S. Karaman. "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Sept. 2017, pp. 3840–3847. DOI: 10.1109/IROS.2017.8206235.

[15] Y. Yang, J. Pan, and W. Wan. "Survey of optimal motion planning". In: *IET Cyber-Systems and Robotics* 1.1 (2019), pp. 13–19. ISSN: 2631-6315. DOI: 10.1049/iet-csr.2018.0003. URL: https://onlinelibrary.wiley.com/doi/abs/10.1049/iet-csr.2018.0003.

[16] Z. Shiller. "Off-Line and On-Line Trajectory Planning | SpringerLink". In: 2015. URL: https://link.springer.com/chapter/10.1007/978-3-319-14705-5_2.

[17] I. Noreen, A. Khan, and Z. Habib. "Optimal path planning using RRT* based approaches: a survey and future directions". In: *International Journal of Advanced Computer Science and Applications* 7.11 (2016).

[18] S. M. LaValle and J. J. Kuffner Jr. "Randomized kinodynamic planning". In: *The international journal of robotics research* 20.5 (2001), pp. 378–400.

[19] S. Karaman and E. Frazzoli. "Sampling-based Algorithms for Optimal Motion Planning". In: arXiv:1105.1186. arXiv, May 5, 2011. DOI: 10.48550/arXiv.1105.1186. arXiv: 1105.1186[cs]. URL: http://arxiv.org/abs/1105.1186.

[20] S. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).

[21] A. D. Ames et al. "Control Barrier Functions: Theory and Applications". In: *2019 18th European Control Conference (ECC)*. 2019 18th European Control Conference (ECC). June 2019, pp. 3420–3431. DOI: 10.23919/ECC.2019.8796030.

[22] G. Yang, C. Belta, and R. Tron. "Continuous-time Signal Temporal Logic Planning with Control Barrier Functions". In: *2020 American Control Conference (ACC)*. 2020 American Control Conference (ACC). ISSN: 2378-5861. July 2020, pp. 4612–4618. DOI: 10.23919/ACC45564.2020.9147387.

[23] L. Lindemann and D. V. Dimarogonas. "Control Barrier Functions for Signal Temporal Logic Tasks". In: *IEEE Control Systems Letters* 3.1 (Jan. 2019). Conference Name: IEEE Control Systems Letters, pp. 96–101. ISSN: 2475-1456. DOI: 10.1109/LCSYS.2018.2853182.

[24] M. Charitidou and D. V. Dimarogonas. "Barrier Function-based Model Predictive Control under Signal Temporal Logic Specifications". In: *2021 European Control Conference (ECC)*. 2021 European Control Conference (ECC). June 2021, pp. 734–739. DOI: 10.23919/ECC54610.2021.9655231.

[25] L. Lindemann and D. V. Dimarogonas. "Decentralized Control Barrier Functions for Coupled Multi-Agent Systems under Signal Temporal Logic Tasks". In: *2019 18th European Control Conference (ECC)*. 2019 18th European Control Conference (ECC). June 2019, pp. 89–94. DOI: 10.23919/ECC.2019.8796109.

[26] C. P. Bechlioulis and G. A. Rovithakis. "Robust approximation free prescribed performance control". In: *2011 19th Mediterranean Conference on Control & Automation (MED)*. 2011 19th Mediterranean Conference on Control & Automation (MED). June 2011, pp. 521–526. DOI: 10.1109/MED.2011.5983022.

[27] 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)*. 2017 IEEE 56th Annual Conference on Decision and Control (CDC). Dec. 2017, pp. 2997–3002. DOI: 10.1109/CDC.2017.8264095.

[28] P. Varnai and D. V. Dimarogonas. "Prescribed Performance Control Guided Policy Improvement for Satisfying Signal Temporal Logic Tasks". In: *2019 American Control Conference (ACC)*. 2019 American Control Conference (ACC). ISSN: 2378-5861. July 2019, pp. 286–291. DOI: 10.23919/ACC.2019.8814999.

[29] B. Donald et al. "Kinodynamic motion planning". In: *Journal of the ACM* 40.5 (Nov. 1, 1993), pp. 1048–1066. ISSN: 0004-5411. DOI: 10.1145/174147.174150. URL: https://dl.acm.org/doi/10.1145/174147.174150 (visited on 06/12/2023).

[30] M. Kleinbort et al. "Probabilistic Completeness of RRT for Geometric and Kinodynamic Planning With Forward Propagation". In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019). Conference Name: IEEE Robotics and Automation Letters, pp. i–vii. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2888947.

[31] S. Karaman and E. Frazzoli. "Optimal kinodynamic motion planning using incremental sampling-based methods". In: *49th IEEE Conference on Decision and Control (CDC)*. 49th IEEE Conference on Decision and Control (CDC). ISSN: 0191-2216. Dec. 2010, pp. 7681–7687. DOI: 10.1109/CDC.2010.5717430.

[32] A. Perez et al. “LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012 IEEE International Conference on Robotics and Automation. ISSN: 1050-4729. May 2012, pp. 2537–2542. DOI: 10.1109/ICRA.2012.6225177.

[33] D. J. Webb and J. van den Berg. “Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints”. In: arXiv:1205.5088. arXiv, May 23, 2012. DOI: 10.48550/arXiv.1205.5088. arXiv: 1205.5088[cs]. URL: http://arxiv.org/abs/1205.5088.

[34] A. Shkolnik, M. Walter, and R. Tedrake. “Reachability-guided sampling for planning under differential constraints”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009 IEEE International Conference on Robotics and Automation. ISSN: 1050-4729. May 2009, pp. 2859–2865. DOI: 10.1109/ROBOT.2009.5152874.

Acknowledgments

I would like to express my gratitude to my supervisor, Prof. Angelo Cenedese, for his support and willingness throughout my thesis. Additionally, all the work wouldn't have been possible without the guidance of my co-supervisors during the Erasmus experience in Stockholm, Prof. Dimos Dimarogonas and Ph.D. Mayank Sewlia. In particular, thank you Mayank for your time, insights, precious help and collaboration.
I would like to extend my thanks to all the beautiful people I met in Sweden, who made my stay unforgettable. Thank you Marta, Marleen and Viktoria for your friendship and support starting from the first days, Bara for the best memories in the library, Michele (x2), Federico, Giacomo, Simone for the coffee breaks and the company at the department. I am also grateful to all my friends of "Tomorrow", from Glenda to Vittoria, Ines, Tommaso, Giulio, Diogo, Ladina and Maxi.
During all the time I spent abroad, I constantly felt the support and closeness of my friends of the master degree. In particular, I would like to express my gratitude to Giulia, who has been a faithful colleague and friend during all my studies. We have shared happiness, goals, rewarding, anxiety and I am sure that the past five years wouldn't have been the same without you.
I have to sincerely thank my closest friends Giorgia, Valentina, Margherita, Asya, Sara, always by my side, and the "Survivers". But especially thank you Lucia, for the life we have been sharing since 2003. Your friendship and support have been a certainty and I am grateful for everything we experienced together.
Finally, I would like to extend my thanks to my family. Thank you mom and dad for your love and sacrifices, Sofia and Tommaso for bearing me since 2001 and 2004 and being the funniest brothers I could ask for: it was good to be back with you. Thanks also to my grandmother Mima and my grandfather Giuseppe for the constant encouragement.