

Path Planning and Control for Multi-Manipulator Systems under Spatio-Temporal Constraints

Mayank Sewlia

KTH Royal Institute of Technology

For my mum and dad

Abstract

Cooperative manipulation, where multiple robotic manipulators coordinate to transport an object, presents significant challenges due to the physical coupling between manipulators, the complexity of motion planning in constrained environments, and the need for precise control without full system knowledge. These challenges are further exacerbated when tasks impose not only spatial constraints but also strict timing requirements, as specified through signal temporal logic. Under such complex requirements, cooperative manipulation remains a relatively underexplored area. These shortcomings motivate the present thesis. Broadly, by blending tools from control theory, formal methods, and motion planning, this thesis aims to deepen our understanding of how to accomplish such complex tasks within the cooperative manipulation setting. Specifically, we develop controllers and planning algorithms that enable the specification, planning, and execution of cooperative robotic manipulation under such tasks.

We approach this problem in three parts. In the first part, we develop control algorithms that take task specifications in the form of signal temporal logic and encode them through funnel-based formulations. Our main results include performing cooperative manipulation under signal temporal logic tasks with both shared task knowledge among all agents and task knowledge limited to a single agent. To achieve this, we design a prescribed performance control scheme that ensures the boundedness of errors within the funnels, thereby enforcing task satisfaction. We present both experimental and simulation validations of the proposed algorithms.

In the second part, we shift focus from designing controllers to developing planning algorithms for satisfying signal temporal logic specifications. This shift provides additional flexibility, allowing us to broaden the class of specifications considered. Our main results include the development of a cooperative sampling-based planning algorithm for two autonomous agents, which is then extended to multiple agents through a distributed optimisation approach. In both approaches, we perform sampling in both spatial and temporal domains to search for trajectories that satisfy signal temporal logic formulas. We present several examples to demonstrate the performance of the algorithms, along with experimental validation and proofs of probabilistic completeness.

In the final part of the thesis, we address cooperative manipulation through constrained environments. We integrate the control and planning components through a trajectory optimisation framework that continues to address signal temporal logic tasks under environmental constraints. The main contributions include decoupling the planning of mobile bases from that of the manipulator arms, and developing an inverse kinematics algorithm combined with a control design strategy to track the resulting joint-space trajectories while avoiding obstacles. We present simulation results to validate the effectiveness of the proposed approach.

Sammanfattning

Kooperativ manipulation, då flera robotmanipulatorer samarbetar för att transportera ett objekt, innebär betydande tekniska utmaningar, till följd av växelverkan mellan manipulatorerna, komplexiteten i att planera rörelser i begränsade miljöer samt behovet av exakt koordinering trots ofullständig kunskap om systemet. Utmaningarna ökar ytterligare när uppgifterna inte bara innebär rumsliga krav utan även inkluderar strikta tidsbegränsningar. Under sådana komplexa krav är kooperativ manipulation fortfarande ett relativt outforskat område. Bristen på kunskap motiverar föreliggande avhandling. Genom att kombinera verktyg från reglerteori, formella metoder samt rörelseplanering syftar avhandlingen till att fördjupa vår förståelse av hur man utför ovan nämnda komplexa uppgifter inom ramen för kooperativ manipulation. Specifikt strävar vi efter att utveckla styrsystem och planeringsalgoritmer som möjliggör specifikation, planering och exekvering av kooperativ robotmanipulation för dessa uppgifter.

Vi angriper problemet i tre separata delar av avhandlingen. I den första delen utvecklar vi styralgoritmer som behandlar uppgiftsspecifikationer, givna i form av Signal Temporal Logic (STL), och kodar dem genom trätbaserade formuleringar. Våra huvudsakliga resultat inkluderar kooperativ manipulation under STL-uppgifter, både med delad uppgiftsinformation, mellan alla agenter, och med uppgiftsinformation begränsad till en enskild agent. För att uppnå detta utformar vi ett Prescribed Performance Control (PPC)-system som säkerställer att felen inom traktarna förblir begränsade och därmed garanterar uppgiftens tillfredsställelse. Vi presenterar både experimentella och simuleringsbaserade valideringar av de föreslagna algoritmerna.

I den andra delen skiftar vi fokus från design av styrsystem till utveckling av planeringsalgoritmer för att uppfylla STL-specifikationer. Denna förskjutning ger ytterligare flexibilitet och möjliggör en bredare hantering av olika typer av STL-formler. Våra huvudsakliga resultat inkluderar utvecklingen av en kooperativ samplingsbaserad planeringsalgoritm för två autonoma agenter (STLcoRRT), som därefter utökas till flera agenter genom en distribuerad optimeringsmetod (MAPS²). I båda metoderna samplas både rumsliga och tidsmässiga domäner för att hitta STL-tillfredsställelse banor. Vi presenterar flera exempel som visar algoritmernas prestanda, samt experimentell validering och matematiska bevis av probabilistisk fullständighet.

I den sista delen av avhandlingen behandlar vi kooperativ manipulation genom begränsade miljöer. Vi integrerar styr- och planeringskomponenterna genom ett ramverk för banoptimering som fortsätter att hantera STL-uppgifter under miljöbegränsningar. De viktigaste bidragen inkluderar frikopplingen av planeringen av mobila baser från planeringen av manipulatorarmarna samt utvecklingen av en algoritm för invers kinematik, i kombination med en kontrollstrategi för att följa de resulterande banorna i ledrummet samtidigt som hinder undviks. Vi presenterar simuleringsresultat som validerar effektiviteten i det föreslagna tillvägagångssättet.

Acknowledgement

First and foremost, I owe my utmost gratitude to Prof. Dimos Dimarogonas for giving me this opportunity and for his unwavering support and guidance throughout my PhD journey. I thoroughly enjoyed working with you and learning from you—both in technical matters and in the managerial aspects of life. I am truly thankful for your spontaneous “Are you stuck?” visits to my office, which regularly rescued me from difficult moments. Next, I would like to extend my sincere gratitude to my co-supervisor, Prof. Christos Verginis, for taking me by the hand at the start of my PhD and teaching me how to write papers and respond to criticism. I will always cherish our conference outings and the joy of exploring new cities together. I hope we never forget the desert safari.

I owe my sincere thanks to my officemates for creating a warm and supportive environment, and for always being so joyous—thank you, Pedro, Miguel, and Amir. I am deeply grateful for our friendship, Rijad. I will always be indebted to you for the things you taught me, the random trivia, and the company you kept during difficult times. You’ve been the best travel companion one could ask for. I truly admire your love for control theory and your passion for building a community. I am also immensely thankful to Victor for all the badminton and gym sessions we powered through together. I hope you run many more marathons in the future. I genuinely admire your love for mathematics and your commitment to rigour. Pedro, you have been a true source of motivation for me in academic work, coding, and leadership. I’ve always appreciated your brutally honest, no-nonsense advice on important life decisions. David, DCS will always feel slightly incomplete without you. The hikes, the apple moos, the beers, the lillebrors, the grilling—every activity with you was many times more enjoyable. I’m truly grateful for your support and friendship. Inês, your laugh echoing through the corridors of DCS will be deeply missed. The joy you bring to those around you is incredibly contagious. Thank you again, Miguel, for always helping me when I’m stuck and for patiently answering my questions. Congratulations in advance! Thank you, Alice, for the wonderful lunch discussions, sailing tips, truths of life, and tennis sessions. I will always remember and smile at your ‘as one does’ remark. Thank you, Rebecka, for all the goodies from Strömsnäsbruk. To Robert, thank you for the inspiration in running and I will always admire your discipline towards causes you care about. To Dženan, thank you for all the chess sessions. Thanks to Elis for showing me how to live an adventurous

life—I hope to see you conquering many more mountains. Thank you, Adrian, for all the technical help over the years and for bringing order to DCS. Thank you, Adam, for the climbing inspiration and for the amazing trip to Gällnö—one day I’ll convince you that “all you need is RGB.” Grattis on your licentiate! Thank you, Enno, for introducing me to so many new cycling races—discussing the Tour de France with you is always a joy. Thank you, Kamil, for instilling in me the importance of theory. I truly admire your passion for math and movies. Thank you, Tommy, for all the gym sessions and weekend lunches we shared. One day, you’ll be brave enough to do leg day with me. Thank you Jacob, for teaching me some useful optimisation tricks, ‘oh well’! Thank you Dongjae, for all the fruitful discussions, I really enjoyed picking your brain. I’m also thankful to Alana, Alejandro, Andrea, Angela, Braghadeesh, Daniele, Elisa, Farhad, Frank, Frédéric, Gregorio, Ingvar, Jacopo, Javad, Jiabao, Mahsa, Mani, Margarita, Maria, Mina, Nana, Peter, Robin, Sara, Shankar, Siyuan, Vandana, Victor, Yassir, Ying, Yuchao and Zifan for creating such an amazing atmosphere in the division.

I would also like to express my gratitude to my rowing teammates—Emma, Graham, Joe, Ólavur, Sonja, and Stan. Being on the water with you all is one of the most joyous memories I will take back with me. Thanks to my friends from Bangalore—Abhishek, Arjun, Glen, and Sharath—for their unwavering support and friendship. Thank you, Prahe, for standing by me through every high and low, with unwavering love, patience, and strength. Finally, I thank my mom and dad for their unconditional love and support.

Acronyms

ADMM	Alternating Direction Method of Multipliers
CTL	Computation Tree Logic
DOF	Degrees of Freedom
EA	Edge Approval
IK	Inverse Kinematics
LTL	Linear Temporal Logic
MAPS²	Multi-Robot Anytime Motion Planning under Signal Temporal Logic Specifications
MILP	Mixed Integer Linear Programming
MITL	Metric Interval Temporal Logic
MPC	Model Predictive Control
MTL	Metric Temporal Logic
PD	Proportional-Derivative
PPC	Prescribed Performance Control
RRT	Rapidly-exploring Random Tree
SNOPT	Sparse Nonlinear OPTimiser
STL	Signal Temporal Logic
STLcoRRT	Signal Temporal Logic Cooperative Rapidly-exploring Random Tree
VD	Validity Domain

List of Figures

1.1	The robotics problem.	4
1.2	Examples of certain STL formulas.	10
1.3	Visualisation of a single Franka Emika Panda [1] arm in DRAKE [2] . . .	13
2.1	Visualisation of frames of two arms rigidly grasping an object	26
2.2	An illustration of the switching function (2.21)	33
2.3	Design of performance functions for the respective STL formulas	35
2.4	Evolution of the object's position $p_o(t)$	44
2.5	Evolution of the Euler angles of the object $\eta_o(t)$	44
2.6	Control input to the robots $u(t)$	45
2.7	Evolution of $\bar{h}_i(x_o, t)$ in blue and performance functions $\gamma_i(t)$ in red for $i = 1, \dots, 6$	45
2.8	Evolution of normalised velocity errors $\xi_v(t)$	46
2.9	Two mobile manipulators grasping an object.	49
2.10	Setup of 3 YouBots grasping an object.	56
2.11	Evolution of (2.51) along with the corresponding funnels.	57
2.12	Leader control input from (2.50).	58
2.13	Evolution of x_o and x_{od}	58
2.14	Estimated effort by follower 1 using (2.46).	59
2.15	Snapshots of the simulation environment at various timestamps.	61
3.1	STL parse tree and satisfaction variable tree for the formula in (3.2). . .	68
3.2	Examples of RRT algorithm.	70
3.3	Two graphs representing different communication topologies.	71
3.4	TIAGo Mobile Manipulator developed by PAL Robotics.	73
3.5	FutureAgent illustration (i) Left: Agent i adds an edge where Agent j has no incident edges (lines 1-2), (ii) Center: Agent j samples until it finds a vertex incident to Agent i 's new edge (lines 3-6), (iii) Right: Agent i interpolates to create a vertex pain $\{z_{\text{inter}}^i, z_{\text{new}}^j\}$ (line 7).	78
3.6	PastAgent illustration: Agent i samples z_{new}^i and interpolates in line 7 to find all incident Agent j samples.	79
3.7	Trees for the formula $\phi = \top$	84
3.8	Trees for the formula $\phi = \mu = x_1 - x_2 > 5$	85
3.9	Trees for the formula $\phi = \mathcal{G}_{[4,6]} x_1 - x_2 < 2$	86

3.10	Trees for the formula $\phi = \mathcal{F}_{[4,6]} x_1 - x_2 > 8$.	87
3.11	Trees for the formula $\phi = \mathcal{G}_{[2,8]}(x_1 < 1) \wedge \mathcal{G}_{[4,6]} x_1 - x_2 < 2$.	88
3.12	Two examples runs of $\phi = \mathcal{F}_{[4,6]}(x_1 - x_2 > 8) \vee \mathcal{G}_{[6,8]} x_1 - x_2 < 2$.	89
3.13	Two examples runs of $\phi = \mathcal{G}_{[0,10]}\mathcal{F}_{[1,3]}(x_1 - x_2 > 8)$.	90
3.14	Experimental setup with three mobile bases and two 6-dof manipulators	91
3.15	Illustration of the proposed algorithm.	98
3.16	Illustration of the proposed algorithm.	99
3.17	Architecture of the provided algorithm	101
3.18	Disjunction representation for disjunctive components using STL parse tree.	106
3.19	Illustration of $Y_{\delta_t}(\mathbf{x}_i)$	107
3.20	Collision Avoidance	110
3.21	Rendezvous	111
3.22	Stability	112
3.23	Recurring Tasks	113
3.24	Non-smooth and smooth paths for the formula (3.13)	114
3.25	Overall Case Study	114
3.26	3D Simulation of trajectory generation for 100 agents for the STL formula (3.14).	116
3.27	$t - y$ axis simulation of trajectory generation for 100 agents for the STL formula (3.14).	117
3.28	$x - y$ axis simulation of trajectory generation for 100 agents for the STL formula (3.14).	118
3.29	Robustness of the STL formula in (3.14)	119
3.30	Collision constraints	119
3.31	Base 1 trajectory tracking	120
3.32	Virtual point tracker by base 2 and end effector 1	120
3.33	Virtual point tracker by base 3 and end effector 2	121
3.34	Desired final position of basess	121
3.35	Desired final position of end effectors	122
3.36	At $t = 12[s]$, when $\bar{\varphi}_1$ and $\bar{\varphi}_2$ are active	122
3.37	At $t = 50[s]$, when $\bar{\varphi}_1$, $\bar{\varphi}_2$ and $\bar{\varphi}_3$ are active	123
3.38	At $t = 100[s]$, when $\bar{\varphi}_1$, $\bar{\varphi}_2$ and $\bar{\varphi}_4$ are active	123
3.39	At $t = 190[s]$, when $\bar{\varphi}_1$ and $\bar{\varphi}_5$ are active	124
4.1	Visualisation of frames of two panda arms rigidly grasping an object(we consider panda arms developed by Franka Emika [3]).	129
4.2	Proposed solution architecture.	130
4.3	Setup of 3 panda arms developed by Franka Emika grabbing an object.	135
4.4	The workspace with obstacles shown in red, desired waypoints in green, and robots in white.	136
4.5	Obstacles and its approximation in the workspace $\mathcal{W}$.	138
4.6	Desired trajectory from MAPS ² along with the solution to (4.8)	139
4.9	Snapshots of a simulation run.	142

Contents

Acknowledgement	v
Acronyms	vii
List of Figures	viii
Contents	1
1 Introduction	3
1.1 Thesis Outline	4
1.2 Temporal Logics	6
1.3 Robotic Manipulation	11
2 Control for Multi-Manipulator Systems	21
2.1 Introduction	22
2.2 Preliminaries	24
2.3 Cooperative Object Manipulation under STL Constraints	31
2.4 Leader-Follower Cooperative Manipulation under STL Constraints .	47
2.5 Conclusion	59
3 Geometric Path-Planning for Multi-Manipulator Systems	63
3.1 Introduction	64
3.2 Preliminaries	66
3.3 Cooperative Sampling-Based Motion Planning under STL Constraints	72
3.4 Multi-Robot Autonomous Motion Planning under STL Constraints .	85
3.5 Conclusion	125
4 Trajectory Optimisation for Multi-Manipulator Systems	127
4.1 Introduction	127
4.2 Problem Formulation	128
4.3 Trajectory Planning and Control for Cooperative Manipulators in Constrained Environments	130
4.4 Simulations	134

4.5 Conclusion	137
5 Conclusion and Future Directions	143
5.1 Future Directions	144
References	147

Chapter 1

Introduction

Natural language, while expressive, is often imprecise and prone to misunderstandings. Consider the sentence “I’ll be there soon.” While it is commonly used in English, the word “soon” is vague, and even “there” lacks clarity—it could mean arriving in a few minutes at place A, a few hours at place B, or even later in the day at place C, depending on the context. To address such limitations, more precise and unambiguous language structures, such as logical formulas, are often employed. For example, asking whether the vacuum cleaner is in Room A becomes a simple yes/no (i.e., $\top$ or $\perp$) question. This ambiguity is especially problematic in time-critical situations, where clarity is essential. Moreover, in robotics applications where timing is critical, it is not only important whether an event has occurred, but also when it occurred. This motivates the use of temporal logics, which enable precise, verifiable task specifications that include timing constraints. One such formal language is Signal Temporal Logic (STL), which is used throughout this thesis. Once a task is formally specified using STL, the next challenge is translating this specification into low-level robot commands. One realisation of this pipeline is illustrated in Figure 1.1. The process begins with the *Requirements* block, which captures the desired behaviour of the robot. From there, a *Motion Planner* generates guidance (i.e., feasible trajectories) that satisfies these requirements. A *Controller* then ensures that the robot follows this plan by generating appropriate commands for the robot’s actuators. To monitor and correct execution in the physical world, *Sensors* are used to provide feedback.

Importantly, the robot need not be a single vacuum cleaner or industrial arm—it could also be a team of robots working cooperatively to achieve a shared objective. In manipulation tasks, this leads to cooperative manipulation, where multiple robotic arms coordinate to lift and transport an object. This brings us to the central question of the thesis:

how do we control and plan for a multi-manipulator system such that it satisfies a given signal temporal logic specification?

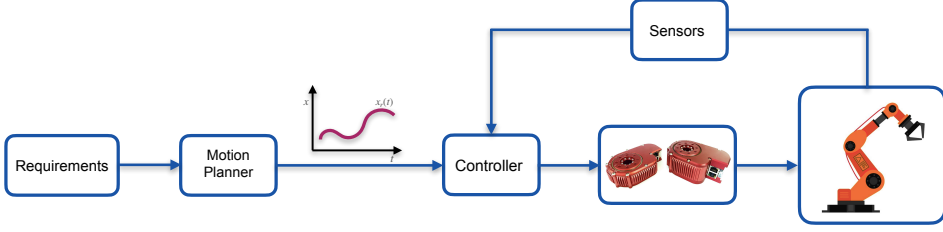


Figure 1.1: The robotics problem.

In other words, we seek to develop a framework that allows us to specify spatio-temporal tasks, along with providing methodologies and guidelines for executing these tasks both at a higher level, such as planning, and at a lower level, such as actuation commands.

1.1 Thesis Outline

This thesis is structured into three main chapters: Chapter 2, Chapter 3, and Chapter 4. Each chapter begins with an introduction and a technical literature review, followed by problem formulation, main results, and concludes with simulations and a summary.

Chapter 2: This chapter addresses the cooperative manipulation problem under STL task specifications for systems with uncertain dynamics, and proposes two complementary control approaches. The first contribution is the design of a decentralized control mechanism that guarantees satisfaction of STL-encoded tasks by translating them into time-varying funnels and enforcing compliance using the Prescribed Performance Control (PPC) methodology. This approach requires no model knowledge of the agents or the object, no force/torque sensing at grasp points, and avoids approximations in STL robustness metrics, allowing for richer task specifications. The second contribution introduces a leader-follower control framework, where a designated leader tracks a desired object trajectory while the followers ensure cooperative transport, also without requiring full dynamic knowledge. Simulation results are provided to validate both frameworks. To the best of our knowledge, this is the first work addressing cooperative manipulation under STL constraints with uncertain dynamics. The exposition is based on:

P1. Mayank Sewlia, Christos K. Verginis, and Dimos V. Dimarogonas. Cooperative object manipulation under signal temporal logic tasks and uncertain dynamics. *IEEE Robotics and Automation Letters*, 7(4), pp.11561-11568. 2022.

P2. Mayank Sewlia, Christos K. Verginis, and Dimos V. Dimarogonas. Cooperative sampling-based motion planning under signal temporal logic specifications. In *2023 American Control Conference (ACC)* (pp. 2697-2702). IEEE. May 2023.

Chapter 3: This chapter approaches the problem of cooperative motion planning under STL specifications in two parts. The first part contributed to the literature by developing a computationally-efficient algorithm that solves, in a distributed manner, the cooperative motion-planning problem under the entire fragment of signal temporal logic for continuous-time and -state systems, without resorting to discretisation techniques. Among many places, coupled autonomous-agents find application in warehouses where a load is too heavy for a single manipulator and thus require cooperation between two manipulators to move the load. Multiple-agents can then form teams of coupled-agents to move multiple loads at once.

In the second part we extend this framework to multiple agents through a distributed optimisation-based algorithm. The main contributions of this part are as follows. The algorithm effectively distributes STL planning across multiple robots by introducing a mechanism to decouple the STL formula among agents, thereby enabling decentralised task allocation. Unlike previous work, it supports the full STL syntax and avoids conservatism by eliminating the need for approximations of max/min operators, while sampling directly in continuous time to avoid abstractions. It also incorporates a broad class of coupled constraints—both linear and nonlinear—within the distributed optimisation framework, allowing it to handle diverse tasks such as pick-and-place operations and time-varying behaviours like trajectory tracking. Extensive simulation and hardware experiments are presented to demonstrate the execution of complex tasks using MAPS². Additionally, the algorithm is sound, ensuring that it produces STL-satisfying trajectories, and probabilistically complete, meaning it will find a solution if one exists. The results from this chapter are based on:

P3. Mayank Sewlia, Christos K. Verginis, and Dimos V. Dimarogonas. MAPS²: Multi-Robot Autonomous Motion Planning under Signal Temporal Logic Specifications. *arXiv preprint arXiv:2309.05632*. 2023.

P4. Mayank Sewlia, Christos K. Verginis, and Dimos V. Dimarogonas. Leader-follower cooperative manipulation under spatio-temporal constraints. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 10312-10317). IEEE. October 2024.

Chapter 4: This chapter builds on the results of Chapters 2 and 3 to address the problem of cooperative manipulation through constrained environments. The setup consists of multiple mobile manipulators transporting an object according to an STL specification. The main contribution lies in proposing a novel trajectory optimisation algorithm for the mobile bases and using its output as a warm start

for an inverse kinematics optimisation problem. The novelty of the approach is in enabling cooperative manipulation without the need for force-torque measurements at the grasp points, while allowing for reconfiguration based on environmental constraints. We also provide several physics-based, realistic simulations to validate the proposed solution for manipulation in constrained environments. This chapter is based on the following work:

P5. Mayank Sewlia, Christos K. Verginis, and Dimos V. Dimarogonas. Trajectory Planning and Control for Cooperative Manipulators in Constrained Environments. *To be submitted*.

Author's Contributions

The sequence of authors for each of the papers included in this thesis reflects their respective contributions, with the first author making the most significant contribution.

In **P1**, **P2**, **P3**, **P4**, and **P5**, the first author proposed the work and the theoretical framework, and took the primary role in writing the manuscript and collecting the results. The remaining authors actively contributed to formulating the solution strategy, validating the results, proposing additional considerations, and critically reviewing and editing the manuscript.

Work Outside of the Scope of the Thesis

We now list other work completed during our doctoral studies that is not included in this thesis. The following works are largely unrelated to the main theme of the thesis.

P6. Rufus Cheuk Yin Wong, Mayank Sewlia, Adrian Wiltz, and Dimos V. Dimarogonas. Generating and optimizing topologically distinct guesses for mobile Manipulator path planning. *arXiv preprint arXiv:2410.20635*. 2024.

P7. Fei Chen, Mayank Sewlia, and Dimos V. Dimarogonas. Cooperative control of heterogeneous multi-agent systems under spatiotemporal constraints. *Annual Reviews in Control*, 57, p.100946. 2024.

P8. Dženan Lapandić, Mayank Sewlia, Bo Wahlberg and Dimos V. Dimarogonas. Receding horizon kinodynamic motion planning for dynamic environments. *To be submitted*.

1.2 Temporal Logics

“ The existence of what is when it is ... is necessary.
”

Aristotle

In the above quote, Aristotle suggests that knowing what exists is not enough, it is also necessary to know when it exists. The idea of incorporating time into logical reasoning dates back to antiquity [4]. Time is a fundamental concept for reasoning about events in both computational and physical systems. Many critical properties of such systems are inherently temporal—they concern what happens, when it happens, and in what order. To formally specify and analyse these time-dependent behaviours, temporal logics (the logic of time) have emerged as powerful and versatile tools. Temporal logic allows us to reason not only about what is, but also about what was, what will be, what has always been, and what will always be.

Fundamental Concepts of Temporal Logics

Temporal logics are a family of formal systems designed to reason about propositions whose truth value may change over time. They extend classical logic by explicitly incorporating the dimension of time. The fundamental building blocks of temporal logics include propositions, which are declarative statements that can be either true or false, and a model of time, which can be discrete or continuous [5].

Such a model of time is specified by a set of temporal operators, that allows one to assert how the truth value of a proposition evolves over time. The most common temporal operators include: $\mathcal{G}$ (always or globally), $\mathcal{F}$ (finally or eventually), $\mathcal{X}$ (neXt) and $\mathcal{U}$ (Until). The semantics of temporal logic define how the truth of formulas is determined. Often, this is done using the concept of a timeline, which is a sequence of states representing the progression of time. For instance, $\mathcal{F}p$ is true in a state if there exists a future state in some path where p is true, while $\mathcal{G}p$ is true if p holds in all future states along all paths (in branching time) or along the single path (in linear time).

This distinction between linear and branching time is fundamental in temporal logic. Linear time assumes a single, linear progression of time, akin to one path extending into the future. This model suits deterministic systems where the sequence of events unfolds in a fixed order. An example of such a logic is Linear Temporal Logic (LTL). In contrast, branching time allows for multiple possible futures from a single point in time, represented as a tree-like structure. This is useful for modelling non-deterministic systems where different choices or events can lead to different outcomes. An example of such a logic is Computation Tree Logic (CTL).

There are also two principal models of time used in temporal logic: the instant model and the interval model.

- Instants are individual points in time with no duration. They are understood using a binary precedence relation ($\prec$) to express ordering.
- Intervals, in contrast, represent durations between two instants—a start and an end. The interval model supports not only precedence ($\prec$) but also overlap (O) and inclusion ($\subseteq$) relations, enabling richer temporal expressions. This model is more expressive and can also describe instants by using ‘point intervals,’ where the start and end instants are the same.

Instant-based logics, such as LTL and CTL, allow reasoning about the truth of propositions at specific points in time. For example,

- The LTL formula $\mathcal{G}(\text{request} \rightarrow \mathcal{F}\text{grant})$ means “every request is eventually granted.”
- The CTL formula $\mathcal{A}[\mathcal{G} \text{ safe} \rightarrow \mathcal{F} \text{ recovered}]$ means “for all paths, if from now on the system is safe, then eventually it recovers.”

Interval-based logics, such as Metric Temporal Logic (MTL) and Signal Temporal Logic (STL), reason over durations of time. For instance,

- The MTL formula $\mathcal{G}_{[0,5]}(\text{request} \rightarrow \text{grant})$ means “every request is eventually granted within 5 time units.”
- The STL formula $\mathcal{G}_{[0,5]}(|x| < 1)$ means “the absolute value of signal x should remain less than 1 for the next 5 time units.”

Another key distinction is between propositional and predicate-based logics. Propositional logic uses simple, atomic statements that are either true or false. Predicate logic allows for more expressive reasoning using variables and quantifiers. This enables not only qualitative evaluation but also quantification over objects or time. For example,

- In propositional logic, “it is raining” is represented simply as a proposition p that could be true or false.
- In predicate logic, we write $\text{Raining}(t)$, which gives a value indicating how heavily it is raining. If $\text{Raining}(t) > 0$, then it is raining, and the value reflects the intensity of the rain.

While instant-based logics like LTL and CTL are widely used in discrete systems, reasoning about systems that evolve continuously over time — such as physical processes or robot motion — requires logics that can handle real-valued signals and time intervals. Signal Temporal Logic (STL) builds on these ideas and provides a natural framework for specifying behaviours over continuous trajectories.

Signal Temporal Logic

STL allows for the expression of both qualitative and quantitative requirements of systems in continuous domains [6]. Additionally, STL provides a natural and compact way to reason about a robot’s motion as it exists in a continuously evolving space-time environment.

To understand STL, we first need to clarify what it means to *specify* a requirement in logic. A formula in STL is a mathematical statement that expresses some desired behaviour of a system over time—like “the robot never collides” or “if a signal becomes high, it eventually drops back down.” STL formulas are built using

a formal syntax (rules for writing expressions), and their meaning is interpreted using semantics (rules for evaluating whether a formula is true over a given signal). The process of determining whether a formula is satisfied by a signal at a point in time is called the satisfaction relation.

Let $\mathbf{x} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^n$ be a continuous-time signal, the formal syntax of STL is,

$$\varphi = \top \mid \mu \mid \neg\varphi \mid \mathcal{G}_{[a,b]}\varphi \mid \mathcal{F}_{[a,b]}\varphi \mid \varphi_1 \mathcal{U}_{[a,b]}\varphi_2 \mid \varphi_1 \wedge \varphi_2 \quad (1.1)$$

where φ_1, φ_2 are STL formulas and $\mathcal{U}_{[a,b]}$ encodes the *until* operator, with $0 \leq a \leq b < \infty$; μ is a predicate of the form $\mu : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{B}$ defined via a predicate function $h : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ as

$$\mu = \begin{cases} \top & h(\mathbf{x}, t) \geq 0 \\ \perp & h(\mathbf{x}, t) < 0 \end{cases}. \quad (1.2)$$

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

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

We also define the following operators

- *disjunction* : $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- *eventually* : $\mathcal{F}_{[a,b]}\varphi \equiv \top \mathcal{U}_{[a,b]}\varphi$
- *always* : $\mathcal{G}_{[a,b]}\varphi \equiv \neg\mathcal{F}_{[a,b]}\neg\varphi$

One of STL's key technical features is its robust semantics, also called quantitative semantics. Instead of just true or false, we can now assign a real-valued robustness score $\rho(\varphi, \mathbf{x}, t)$ that indicates how well the signal $\mathbf{x}$ satisfies the formula φ . Informally, $\rho > 0$ means satisfaction (the larger, the more margin), and $\rho < 0$ means violation (the more negative, the more severe the violation). For example, if $\varphi = (\mathbf{x} < 5)$, we can define $\rho(\mathbf{x} < 5, \mathbf{x}, t) = 5 - \mathbf{x}(0)$ which is the difference between the threshold and the actual value at time 0. More formally, the robust semantics

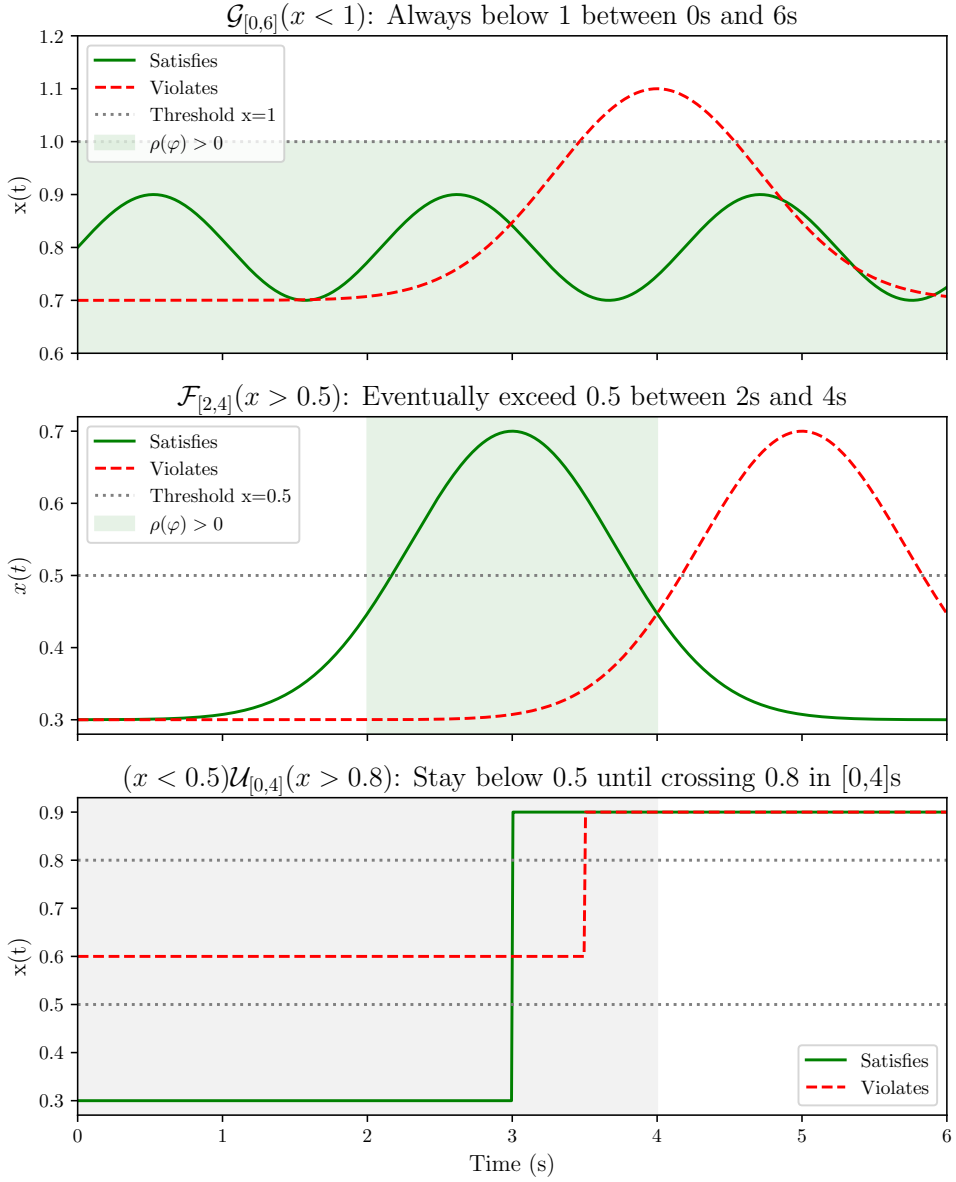


Figure 1.2: Examples of certain STL formulas.

$(\mathbf{x}, t) \models \varphi$ are defined as follows:

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

A signal $\mathbf{x}$ satisfies an STL formula φ at time t if and only if $\rho(\varphi, \mathbf{x}, t) \geq 0$. Robust semantics add a rich layer to STL, enabling use in optimisation and control. For instance, one can use a numerical optimiser to search for a system trajectory that minimizes the robustness of a safety property, effectively finding the most dangerous scenario that nearly violates the property (useful for testing and falsification). Or in controller synthesis, one can maximise robustness to ensure a wide safety margin.

In Figure 1.2, examples of signals that satisfy or violate specific STL formulas are shown. In the top plot, the green signal satisfies the formula $\varphi = \mathcal{G}_{[0,6]}(x < 1)$ because it remains below 1 throughout the interval from 0 to 6 seconds. The red signal violates the formula by exceeding 1 within this interval. In the middle plot, the green signal satisfies the formula $\varphi = \mathcal{F}_{[2,4]}(x > 0.5)$ because it eventually exceeds 0.5 within the interval from 2 to 4 seconds, as indicated by the green shaded region. The red signal fails to do so and therefore violates the formula. In the bottom plot, the green signal satisfies the formula $\varphi = (x < 0.5)\mathcal{U}_{[0,4]}(x > 0.8)$ since it stays below 0.5 until it exceeds 0.8 within the interval from 0 to 4 seconds. In contrast, the red signal violates the formula because it rises above 0.5 before reaching 0.8.

In conclusion, temporal logic offers a precise and unambiguous way to specify desired system behaviour, with applications in model checking, formal verification, and control synthesis. Despite these advantages, it comes with certain limitations. The formal syntax and semantics can be intricate, and since specifications are written by humans, they are inherently constrained by human understanding especially when expressing complex behaviours. Moreover, there is often a gap between the abstract specifications and the concrete details of a system's implementation, such as its kinematics or dynamics. Bridging this gap carefully is essential to ensure that verification results hold meaning in practice.

1.3 Robotic Manipulation

Newton's first law of motion states that a body at rest remains at rest, and a body in motion continues in motion unless acted upon by an external force—unless, in other words, it is manipulated. This idea of manipulation underlies many of our

everyday actions: birds manipulate the air around them with their wings; we lift a cup of coffee to take a sip; we strike a cricket ball with a bat. When such manipulation is performed by a robot or mechanical system, it falls under the field of robotic manipulation.

Robotic manipulation focuses on designing and controlling robotic systems capable of interacting with and altering their environment. This includes basic tasks like grasping, lifting, moving, and placing objects, as well as more complex ones like assembly and disassembly. The field traces back to the 1950s, when the first industrial robots were developed. One of the earliest robots, *Unimate*, created by George Devol in 1951 [7], was a hydraulically actuated manipulator with five degrees of freedom. In [8], Devol remarked:

The routine job of transporting one article after another from a supply point to a delivery point has not heretofore been met with flexible programming.

This may be one of the earliest definitions of a pick-and-place task—a fundamental problem still actively researched today, even after 75 years. While robotic manipulation encompasses far more than pick-and-place, this task serves as a useful starting point for understanding its core ideas.

Consider the robotic arm shown in Figure 1.3, which is a 7-degree-of-freedom manipulator developed by Franka Robotics. Like most modern manipulators, it consists of a series of rigid links connected by joints. These joints house motors that can be controlled either by position or torque, enabling precise movement of the end-effector, denoted by $\{E\}$, to perform a desired task. The configuration of the arm is described by a set of generalised coordinates q_i 's, representing the joint angles. For revolute joints, these angles lie within the interval $q_i \in [0, 2\pi)$. Joints are numbered sequentially from 1 at the base to n at the end-effector. Each rigid link i connects joint $i - 1$ to joint i , as illustrated in Figure 1.3. The end-effector forms the final link in this kinematic chain.

Determining the end-effector position and orientation in space i.e. ${}^W X^E \in SE(3)$, relative to a world frame $\{W\}$ fixed at the robot's base, is known as forward kinematics. Mathematically, forward kinematics is a function $f : Q \rightarrow SE(3)$ that maps joint space Q to the Special Euclidean group in 3 dimension $SE(3)$, which describes the position and orientation of the end-effector in 3D space. The forward kinematics map is defined as,

$${}^W X^E = f(q) \tag{1.3}$$

where $q = (q_i)$ for $i \in \{1, \dots, n\}$. The joint angles $q_i \in \mathbb{S}^1$ are associated with the unit circle in a plane, denoted by the space $\mathbb{S}^1$. The joint space Q is then the Cartesian product of all such individual joint spaces. The forward kinematics map is unique i.e., each set of joint angles corresponds to a single end-effector position and orientation. For a 6 degrees of freedom manipulator, if an additional constraint such as the elbow configuration (e.g., up or down) is specified, the inverse map can also be unique. However, for manipulators with $n > 6$, there are more joints than

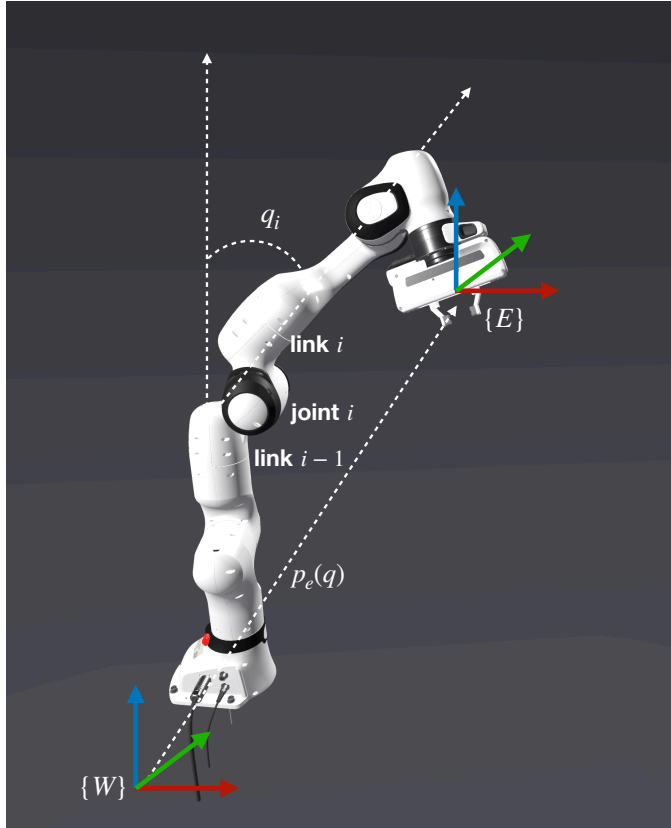


Figure 1.3: Visualisation of a single Franka Emika Panda [1] arm in DRAKE [2]

the number of Cartesian coordinates and the inverse problem is no longer unique. In such redundant manipulators, multiple joint configurations can produce the same end-effector pose. This is known as the inverse kinematics problem, which is often ill-posed in these cases due to the redundancy in the system.

The relationship between joint velocities $\dot{q}$ and end-effector velocity v is slightly more involved, as the mapping between the joint space Q and $SE(3)$ is non-linear [9]. The instantaneous spatial velocity of the end-effector is given by,

$$v = J(q)\dot{q}. \quad (1.4)$$

Here $J(q) \in \mathbb{R}^{6 \times n}$ is the manipulator Jacobian. The end-effector velocity v consists of translation velocity $\mathbf{v} \in \mathbb{R}^3$ and angular velocity $\omega \in \mathbb{R}^3$. Note that the time derivative of the generalised positions q does not always correspond to the generalised velocities, this holds true only when all joints are revolute joints.

To control the actions performed by a robot, we need to define how the robot evolves over time and how external actions affect its states. Such a mathematical

model describing the dynamic behaviour of the robot is derived below. There are multiple ways to approach this; my favourite is using the Principle of Least Action to derive Lagrange's equation of motion, and subsequently deriving the general robot arm dynamics through this equation [10].

Consider a process where energy is converted from one form to another. Introduce a quantity that evolves during this energy conversion process and denote it as the generalised path $q(t)$. Assuming no external energy interacts with this process, energy transforms from one form to another along the path $q(t)$. The Lagrangian is defined as the difference between these two forms of energy. It is a function of time t , path q and its rate of change $\dot{q}$, with units in Joules:

$$\mathcal{L}(t, q, \dot{q}) = \text{First form of energy} - \text{Second form of energy} \quad (1.5)$$

To set up the Principle of Least Action, we postulate that a moving body has two forms of energy: kinetic energy due to its velocity and potential energy due to interaction with other bodies. Define the action $\mathcal{S}$ as the magnitude of the integral of the Lagrangian over the path $q(t)$ [11],

$$\mathcal{S}(t, q, \dot{q}) = \left| \int_{t_0}^{t_1} \mathcal{L}(t, q, \dot{q}) dt \right| \quad (1.6)$$

The Principle of Least Action states that among all possible paths available, nature selects the one that minimises the action $\mathcal{S}$. This principle was first proposed by Pierre Louis Maupertuis, a student of Johann Bernoulli, in 1744. Maupertuis questioned Fermat's assertion that nature minimises time and asked: why should nature care only about time? He proposed a new quantity called action, defined as mass $\times$ velocity $\times$ distance. If a particle or wave travels in segments, the total action is the sum of the actions over each segment. The path followed by the particle or wave, he argued, is the one that minimises this total action. In [12], he wrote,

This action is the true expense of Nature, which she manages to make as small as possible...

Unfortunately, Maupertuis was attacked, mocked, and ignored for his proposal. His formulation lacked mathematical rigour and gave no clear reason why nature should minimise this specific quantity. Fortunately, Leonhard Euler defended Maupertuis and took the principle seriously [13]. Euler proposed two conditions under which the Principle of Least Action could hold: (1) total energy must be conserved, and (2) energy must be constant across all possible paths between two points. However, a general and rigorous formulation came later, in 1750, by Joseph-Louis Lagrange [14]. This led to the foundational equations of variational mechanics now known as the Euler–Lagrange equations.

Assume we know the Lagrangian and action of the system. and let $q(t)$ be the minimising path. Introduce a small perturbation $\delta q(t)$ to the path $q(t)$ resulting in

$\tilde{q}(t)$ as,

$$\tilde{q}(t) = q(t) + \epsilon \delta q(t) \quad (1.7)$$

where ϵ is a small parameter. The perturbed Lagrangian is,

$$\mathcal{L}(t, \tilde{q}, \dot{\tilde{q}}) = \mathcal{L}(t, q + \epsilon \delta q, \dot{q} + \epsilon \dot{\delta q}) \quad (1.8)$$

This Lagrangian can be related to the original Lagrangian as,

$$\mathcal{L}(t, \tilde{q}, \dot{\tilde{q}}) = \mathcal{L}(t, q, \dot{q}) + \epsilon \left(\delta q \frac{\partial \mathcal{L}}{\partial q} + \dot{\delta q} \frac{\partial \mathcal{L}}{\partial \dot{q}} \right) + \mathcal{O}(\epsilon^2) \quad (1.9)$$

where the term $\mathcal{O}(\epsilon^2)$ indicates all additional terms multiplied by ϵ^2 and higher powers of ϵ . The difference in action of the perturbed and the original path is,

$$\mathcal{S}(\tilde{y}) - \mathcal{S}(y) = \epsilon \left[\int_{t_0}^{t_1} \left(\delta q \frac{\partial \mathcal{L}}{\partial q} + \dot{\delta q} \frac{\partial \mathcal{L}}{\partial \dot{q}} \right) dt \right] + \mathcal{O}(\epsilon^2) \quad (1.10)$$

The integral term is called the first variation of the action and is denoted by $\delta \mathcal{S}$.

$$\delta \mathcal{S}(\delta q, q) = \int_{t_0}^{t_1} \left(\delta q \frac{\partial \mathcal{L}}{\partial q} + \dot{\delta q} \frac{\partial \mathcal{L}}{\partial \dot{q}} \right) dt \quad (1.11)$$

Since path $q(t)$ minimises action, the difference $\mathcal{S}(\tilde{y}) - \mathcal{S}(y)$ must be positive for all choices of δq , this occurs only when the first variation is 0,

$$\begin{aligned} \delta \mathcal{S}(\delta q, q) &= 0 \\ \int_{t_0}^{t_1} \left(\delta q \frac{\partial \mathcal{L}}{\partial q} + \dot{\delta q} \frac{\partial \mathcal{L}}{\partial \dot{q}} \right) dt &= 0 \end{aligned}$$

Using integration by parts, this becomes,

$$\begin{aligned} \int_{t_0}^{t_1} \left(\delta q \frac{\partial \mathcal{L}}{\partial q} + \dot{\delta q} \frac{\partial \mathcal{L}}{\partial \dot{q}} \right) dt &= 0 \\ \int_{t_0}^{t_1} \delta q \frac{\partial \mathcal{L}}{\partial q} dt + \left[\delta q \frac{\partial \mathcal{L}}{\partial \dot{q}} \right]_{t_0}^{t_1} - \int_{t_0}^{t_1} \delta q \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) dt &= 0 \end{aligned}$$

Since paths start and end at the same point, we have $\delta q(t_0) = \delta q(t_1) = 0$. The above equation simplifies to,

$$\int_{t_0}^{t_1} \frac{\partial \mathcal{L}}{\partial q} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) dt = 0$$

resulting in the Euler-Lagrange equation,

$$\frac{\partial \mathcal{L}}{\partial q} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) = 0 \quad (1.12)$$

The Euler-Lagrange equation only holds for conservative systems. If external (and non-conservative) forces are present, it becomes,

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) - \frac{\partial \mathcal{L}}{\partial q} = \tau \quad (1.13)$$

where τ is the generalized force acting on the system. The above equation is the Lagrange equation of motion. Rearranging the above equation gives,

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) = \frac{\partial \mathcal{L}}{\partial q} + \tau$$

which is a restatement of Newton's law in generalised coordinates,

$$\frac{d}{dt}(\text{momentum}) = \text{applied force}.$$

We will derive the equations of motions for a robotic manipulator using the Lagrangian method. The Lagrangian $\mathcal{L}$ is defined as the difference between the system's kinetic energy T and potential energy V , both expressed in generalised coordinates,

$$\mathcal{L}(q, \dot{q}) = T(q, \dot{q}) - V(q). \quad (1.14)$$

To compute the Lagrangian, we first determine the kinetic and potential energy of each link in the manipulator. In this regard, consider a rigid body of mass m , with a coordinate frame fixed at its center of mass. Let (p, R) denote the trajectory of the body in an inertial frame. The total kinetic energy of the body is the sum of its translational and rotational components,

$$\begin{aligned} T &= \frac{1}{2} m \|\dot{p}\|^2 + \frac{1}{2} \omega^\top \mathcal{I} \omega \\ &= \frac{1}{2} (v^B)^\top \begin{bmatrix} mI & 0 \\ 0 & \mathcal{I} \end{bmatrix} v^B \\ &=: \frac{1}{2} (v^B)^\top \mathcal{M} (v^B). \end{aligned}$$

Here $\mathcal{M}$ is the generalised inertia matrix of the body expressed in body frame and v^B is the body frame spatial velocity. The kinetic energy of the i th link is then,

$$T_i(q, \dot{q}) = \frac{1}{2} (v_i^B)^\top \mathcal{M}_i (v_i^B) = \frac{1}{2} \dot{q}^\top J_i^\top \mathcal{M}_i J_i \dot{q} \quad (1.15)$$

where $\mathcal{M}_i$ is the generalised inertia matrix for the i th link and J_i is the link Jacobian. The total kinetic energy of the manipulator is the sum over all links,

$$T(q, \dot{q}) = \sum_{i=1}^n T_i(q, \dot{q}) = \frac{1}{2} \dot{q}^\top M(q) \dot{q} \quad (1.16)$$

where $M(q) \in \mathbb{R}^{n \times n}$ is the manipulator inertia matrix defined as,

$$M(q) = \sum_{i=1}^n J_i^\top \mathcal{M}_i J_i.$$

The potential energy of the i th link is given by,

$$V_i(q) = m_i g h_i(q)$$

where h_i is the height of the center of mass of the i th link, and g is the gravitational acceleration. The total potential energy of the manipulator is the sum over all links,

$$V(q) = \sum_{i=1}^n V_i(q) = \sum_{i=1}^n m_i g h_i(q).$$

We now have everything needed to write the Lagrangian of the manipulator (1.14),

$$L(q, \dot{q}) = \frac{1}{2} \dot{q}^\top M(q) \dot{q} - V(q)$$

To compute the dynamics of the open-chain manipulator, let $q \in \mathbb{R}^n$ denote joint angles of the manipulator. Expressing the kinetic energy as a sum over matrix elements, the Lagrangian becomes,

$$L(q, \dot{q}) = \frac{1}{2} \sum_{i,j=1}^n M_{ij}(q) \dot{q}_i \dot{q}_j - V(q)$$

Substituting this into (1.13) gives,

$$\frac{d}{dt} \frac{\partial}{\partial \dot{q}_i} \left(\frac{1}{2} \sum_{i,j=1}^n M_{ij}(q) \dot{q}_i \dot{q}_j - V(q) \right) - \frac{\partial}{\partial q_i} \left(\frac{1}{2} \sum_{i,j=1}^n M_{ij}(q) \dot{q}_i \dot{q}_j - V(q) \right) = \tau_i$$

Each term is computed as,

$$\begin{aligned} \frac{d}{dt} \frac{\partial}{\partial \dot{q}_i} \left(\frac{1}{2} \sum_{i,j=1}^n M_{ij}(q) \dot{q}_i \dot{q}_j \right) &= \frac{d}{dt} \left(\sum_{j=1}^n M_{ij}(q) \dot{q}_j \right) = \sum_{j=1}^n \left(M_{ij} \ddot{q}_j + \dot{M}_{ij} \dot{q}_j \right) \\ \frac{\partial}{\partial q_i} \left(\frac{1}{2} \sum_{i,j=1}^n M_{ij}(q) \dot{q}_i \dot{q}_j - V(q) \right) &= \frac{1}{2} \sum_{j,k=1}^n \frac{\partial M_{kj}}{\partial q_i} \dot{q}_k \dot{q}_j - \frac{\partial V}{\partial q_i} \end{aligned}$$

Substituting in the Lagrange's equation and expressing $\dot{M}_{ij}$ using partial derivatives yields,

$$\sum_{j=1}^n M_{ij} \ddot{q}_j + \sum_{j,k=1}^n \left(\frac{\partial M_{ij}}{\partial q_k} \dot{q}_j \dot{q}_k - \frac{1}{2} \frac{\partial M_{kj}}{\partial q_i} \dot{q}_k \dot{q}_j \right) + \frac{\partial V}{\partial q_i}(q) = \tau_i, \quad i = 1, \dots, n$$

Rearranging the above equation gives,

$$\sum_{j=1}^n M_{ij}(q)\ddot{q}_j + \sum_{j,k=1}^n \Gamma_{ijk}\dot{q}_j\dot{q}_k + \frac{\partial V}{\partial q_i}(q) = \tau_i, \quad i = 1, \dots, n \quad (1.17)$$

where Γ_{ijk} are called the Christoffel symbols corresponding to the inertia matrix $M(q)$ and given by,

$$\Gamma_{ijk} = \frac{1}{2} \left(\frac{\partial M_{ij}(q)}{\partial q_k} + \frac{\partial M_{ik}(q)}{\partial q_j} - \frac{\partial M_{kj}(q)}{\partial q_i} \right).$$

Equation (1.17) is a second-order ordinary differential equation in terms of manipulator joint variables q . The terms represent,

- Inertial forces dependent on joint accelerations $\ddot{q}$,
- Centrifugal and Coriolis forces being quadratic in joint velocities $\dot{q}$,
- Potential forces dependent on joint positions q ,
- External forces τ acting on the manipulator.

For convenience, we can extract the gravitational forces $g(q)$ from τ_i and define the Coriolis matrix using,

$$C_{ij}(q) = \sum_{k=1}^n \Gamma_{ijk}\dot{q}_k = \frac{1}{2} \sum_{k=1}^n \left(\frac{\partial M_{ij}(q)}{\partial q_k} + \frac{\partial M_{ik}(q)}{\partial q_j} - \frac{\partial M_{kj}(q)}{\partial q_i} \right) \dot{q}_k$$

This leads to the standard vector form of the manipulator dynamics,

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau. \quad (1.18)$$

The following properties hold true for the equations of motion (1.18),

- The inertia matrix $M(q)$ is symmetric and positive definite, i.e., $M(q) = M^\top(q)$ and $\dot{q}^\top M(q)\dot{q} > 0$ for all $\dot{q} \neq 0$.
- $\dot{M} - 2C \in \mathbb{R}^{n \times n}$ is a skew-symmetric matrix as,

$$\begin{aligned} (\dot{M} - 2C)_{ij} &= \dot{M}_{ij}(q) - 2C_{ij}(q, \dot{q}) \\ &= \sum_{k=1}^n \left(\frac{\partial M_{ij}}{\partial q_k} \dot{q}_k - \frac{\partial M_{ij}}{\partial q_k} \dot{q}_k - \frac{\partial M_{ik}}{\partial q_j} \dot{q}_k + \frac{\partial M_{kj}}{\partial q_i} \dot{q}_k \right) \\ &= \sum_{k=1}^n \left(\frac{\partial M_{kj}}{\partial q_i} \dot{q}_k - \frac{\partial M_{ik}}{\partial q_j} \dot{q}_k \right) \end{aligned}$$

and switching the i th and j th indices shows $(\dot{M} - 2C)^\top = -(\dot{M} - 2C)$ i.e., this matrix is skew-symmetric.

An alternative way to derive the dynamics is by summing all forces acting on the robot's rigid bodies and applying Newton's second law. While more intuitive, this method requires explicit knowledge of all forces and accelerations. In contrast, the Lagrangian formulation has the advantage of relying solely on energy terms—kinetic and potential. See [15] for a comprehensive introduction to robotic manipulation.

Chapter 2

Control for Multi-Manipulator Systems

Having introduced robotic manipulation and STL in the previous chapter, we are now ready to use this machinery and apply on the more challenging cooperative manipulation under STL task problem. This chapter extends the discussion to the cooperative setting where multiple robots work together to manipulate a shared object. We will design low-level control laws that ensure the object satisfies the tasks defined on it using the STL formula and provide experimental validation of the proposed ideas. We begin by formulating the equations of motion for a cooperative manipulation system and present a brief but comprehensive review of the existing literature. We then introduce STL as a formal framework to express complex, time-bound tasks on the object being manipulated.

In the first part of the chapter, we consider a setting where all robots are aware of the STL task defined on the object. We use the Prescribed Performance Control (PPC) framework to ensure that the object’s state evolves within a predefined time-varying funnel, which corresponds to satisfying the STL formula. The control strategy is decentralised and does not rely on knowledge of the agents’ or object’s dynamics. Each robot computes its control input independently based on its own state and the task specification. This approach is validated through experiments with two manipulators handling an object cooperatively.

In the second part, we relax the assumption of shared task knowledge. Only one robot—the leader—knows the STL task, while the others act as followers. The control objective is to satisfy the STL specification despite this asymmetry. We model the system in a leader-follower framework and design a stiffness-based controller for the followers to maintain object posture, while the leader uses a high-gain prescribed performance controller to guide the object trajectory. The method does not require knowledge of the object’s dynamics and is evaluated in simulation using KUKA YouBots.

The chapter develops two related but distinct control strategies for STL-guided

cooperative manipulation—one with shared task knowledge and one without—and provides both theoretical formulation and practical validation for each.

This chapter is based on [16] and [17].

2.1 Introduction

The cooperative manipulation problem involves controlling multiple robots to jointly manipulate a shared object—typically one that cannot be handled by a single robot alone. This collaboration offers several advantages, including load sharing and the ability to manipulate heavier or more complex payloads. However, it also introduces challenges such as coordinating motion between agents, distributing forces evenly, performing decentralised planning, and translating high-level task specifications for the object into low-level control actions at each robot’s joints. Cooperative manipulation systems find applications in human-robot interaction, aerial payload transport, manipulation in cluttered environments, and industrial automation [18].

Unlike the related literature, which mainly considers the tracking/regulation of the manipulated object, we are here interested in more complex tasks over time, such as “never take the object to dangerous regions” or “keep moving the object from region A to B within a predefined time interval”, which must be executed via the control actions of the agents. Such complex tasks can be expressed by temporal logics. Examples include Linear Temporal Logic (LTL) [19], Metric-interval Temporal Logic (MITL) [20] and Signal Temporal Logic (STL); LTL and MITL use automata representations of the task and require discrete abstractions of the underlying continuous systems; STL, on the other hand, introduced in [6], is a formalism used to specify properties of dense-time, real-valued signals. It allows for inclusion of time and state bounds of a continuous-time system and hence a useful formalism to model complex tasks in robotics [21]. Moreover, STL offers a degree of space robustness, allowing us to measure the extent of satisfaction of a signal against a given formula [22].

Many works consider STL tasks for multi-agent systems [23], but without addressing the special case of cooperative manipulation schemes. The latter exhibit unique challenges due to their complex dynamics and the interaction of the robotic agents through their grasping points with the object. On the other hand, decentralized control of cooperative manipulators is a widely-studied problem in the literature [18, 24–28]. However, most related works study the problem of trajectory tracking for the object, possibly with simultaneous force regulation and state constraints, such as collision avoidance. On the contrary, we focus on the cooperative manipulation of an object such that it satisfies a more general task given by an STL formula. Works such as [29] and [19] use LTL to achieve multi-robot planning. However there are certain inherent differences between Linear Temporal Logic (LTL) and Signal Temporal Logic (STL). STL is defined over continuous space and time, and is accompanied by a robustness metric signifying the degree of satisfaction of a formula. In contrast, LTL is a proposition based logic evaluated

over infinite sequence of states without time constraints. Temporal logics were also used in [20] for cooperative manipulation schemes in the form of MITL constraints, resorting however to workspace discretization and abstraction techniques, which we avoid in this work. MITL is a proposition based logic that allows timing constraints on temporal operators but fails to define behaviour of continuous signals in space and time. In [30], a high-level and a low-level task planner are used to satisfy a metric temporal logic specification but are limited to manipulation with a single robotic arm. Compared to [31], which studies prescribed performance control for STL tasks, we consider a richer STL fragment, provide a model-free controller, and deal with conjunctions in a non-hybrid control framework.

In this chapter, we address two problems. The first, presented in Section 2.3, is the cooperative manipulation of an object under STL tasks with shared task knowledge among all agents. The second, discussed in Section 2.4, involves cooperative manipulation under STL tasks with hierarchical and limited task knowledge distributed among the agents. This latter setting is more challenging and interesting, as it allows some agents to assume greater responsibility in fulfilling the STL task. A leader-follower framework naturally arises in such scenarios. For example, a leader agent may (i) possess more powerful actuators capable of lifting heavier payloads, (ii) be equipped with a greater number of degrees of freedom, (iii) have broader knowledge of the environment and obstacles, or (iv) have exclusive access to the task specifications— which is the focus of second part. The premise of the second part is thus to explore the cooperative manipulation of an object when only a subset of the agents are aware of the desired tasks associated with the object. For instance, a task could involve picking and placing an object within specific timing constraints. We refer to these tasks as spatio-temporal tasks, which impose both spatial and temporal requirements. By allocating the task knowledge to a specific subset of agents, we enable the remaining agents to dedicate themselves to other secondary but important tasks, such as helping to carry the payload. In our approach, the follower agents are responsible for lifting the object and compensating for its gravity, whereas the leader agent is charged with manipulating the object according to specified requirements. Nevertheless, we proceed under the assumption that there is no knowledge of the dynamical terms of the robots and the object, and we abstain from using force measurements to identify any external forces exerted by the leader agents. The control challenge is divided into designing a controller for the follower agents to compensate for the object’s gravitational forces and for the leader agent to guide the object along a desired trajectory. We consider mobile manipulators as they allow for larger workspace accessibility and enable the utilisation of redundant degrees of freedom to perform secondary tasks for the mobile bases.

The current literature can be reviewed in three categories that contribute to our discussions later: adaptive control with gravity compensation, compliant control of cooperative manipulators, and the enforcement of spatio-temporal constraints within compliant manipulators. A foundational approach in robotics has been the parameterisation of unknown system dynamics coupled with adaptive control.

Studies spanning from the early 1980s, such as [32], [33], and [34], to more recent contributions like [35], illustrate the evolution of adaptive control for robotics manipulators. However, parameterising the entire dynamics does come with the cost of increased computational demands and poor performance. An alternative is to only parameterise the gravity terms and use a Proportional-Derivative (PD) controller, such as [36] presented for a single manipulator setting. Works such as [37] and [38] study compliant control for manipulators both in a single-agent and a multi-agent setting. However, such methods require and make use of the knowledge of the centrifugal and Coriolis terms of the system. Impedance control of manipulators, studied in [39] and [40], shapes the impedance behaviour of a system, often involving either the acceleration measurements or the measurements of the external forces. This could be circumvented by designing a force estimator [41] or treating the forces as disturbances and designing a disturbance observer [42–44], still requiring, however, the knowledge of the dynamical terms. Closely related problems were studied in [45], [46] and [47]. In [45], the authors used an inverse dynamics controller to design an impedance control scheme along with an estimation law to estimate the leader’s intention. In [46], an impedance controller and a wrench estimator are designed to capture the human’s intention in transporting the object. The assumption of knowing the inertial and Coriolis terms of agents and object is made in both the above works such that the controller is able to shape the dynamics in a satisfactory manner. In [47], the authors study the cooperative manipulation problem using adaptive control, where all the robots work cooperatively to track the reference trajectory. The dynamics of the system are parameterised, and redundancy resolution is discussed.

2.2 Preliminaries

In this section, we first present the equations of motion for cooperating manipulators grasping an object. We then introduce the Prescribed Performance Control (PPC) framework, a control strategy that ensures certain error dynamics remain within user-defined funnels, thereby providing stability guarantees. Next, we describe the STL fragment considered in this work, and finally, we show how STL tasks can be encoded within the PPC framework. This formulation allows us to guarantee STL task satisfaction by ensuring stability in the sense defined by PPC.

Equations of Motion for Cooperative Manipulation

The manipulator dynamics derived in the previous chapter (Equation (1.18)) take the form,

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau \quad (2.1)$$

where τ denotes the joint torques, and M, C, g are dynamical terms. This formulation is referred to as the joint-space dynamics, as the states are defined in joint space and have dimension n , corresponding to the number of joints. However, it is

often more convenient to define the dynamics in end-effector space (or task space) to reason with respect to an inertial frame. For instance, when a desired trajectory is specified in $SE(3)$, it is computationally more efficient to define the tracking error in task space and directly design a controller there—rather than converting the $SE(3)$ objective into joint-space coordinates and relying on inverse kinematics to obtain the corresponding joint angles.

Let $f : Q \rightarrow \mathbb{R}^p$ be a smooth and invertible mapping between the joint-space variables $q \in Q$ and the task-space variable $x \in \mathbb{R}^p$ where p is the dimension of the task-space. For now let $n = p$, for the case $p < 6$ we only get a partial parameterisation of $SE(3)$ and if $p > 6$ we get a redundant parameterisation of $SE(3)$.

Rewriting the dynamics in terms of the task-space variable $x \in \mathbb{R}^p$,

$$\dot{x} = \bar{J}(q)\dot{q}, \quad \bar{J}(q) = \frac{\partial f}{\partial q} \in \mathbb{R}^{p \times n}.$$

Here $\bar{J}$ is the Jacobian of the mapping $f : Q \rightarrow \mathbb{R}^p$ and not the manipulator Jacobian that we encountered in differential kinematics of equation (1.4). The joint velocities and accelerations can be written in terms of the task-space variables as,

$$\dot{q} = \bar{J}^{-1}\dot{x}, \quad \ddot{q} = \bar{J}^{-1}\ddot{x} + \frac{d}{dt}\bar{J}^{-1}\dot{x}.$$

Substituting this in (2.1) gives us the task-space dynamics,

$$\bar{M}(q)\ddot{x} + \bar{C}(q, \dot{q})\dot{x} + \bar{g}(q) = u \quad (2.2)$$

where,

$$\begin{aligned} \bar{M} &= \bar{J}^{-\top} M(q) \bar{J}^{-1}, \\ \bar{C} &= \bar{J}^{-\top} \left(C \bar{J}^{-1} + M \frac{d}{dt} \bar{J}^{-1} \right), \\ \bar{g} &= \bar{J}^{-\top} g(q), \\ u &= \bar{J}^{-\top} \tau. \end{aligned}$$

Here, we assumed that the manipulator does not interact with objects or the environment, and that u consists only of the forces at the end-effector resulting from the joint torques. To model interaction with the environment, we apply an external wrench $\lambda \in \mathbb{R}^p$ at the end-effector. The wrench λ is a 6D vector representing the external force and torque acting on the end-effector. The resulting dynamics can then be written as:

$$\bar{M}(q)\ddot{x} + \bar{C}(q, \dot{q})\dot{x} + \bar{g}(q) = u - \lambda. \quad (2.3)$$

From this point on, we assume $p = 6$ i.e. the task space is fully described in $SE(3)$. To derive the equations of motion for the cooperative manipulation system, we first present the dynamics of the agents (or manipulators), then the dynamics of the

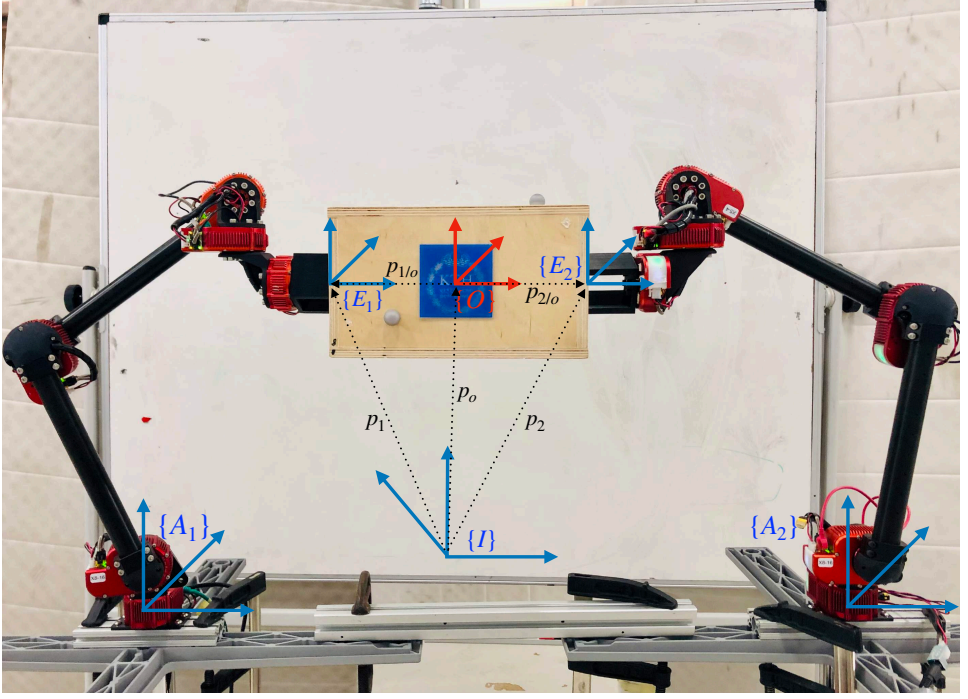


Figure 2.1: Visualisation of frames of two arms rigidly grasping an object

unactuated object, and finally the model of the coupled agent–object system. To set up the formulation, consider a workspace in $\mathbb{R}^3$ with N robotic agents rigidly grasping an object, as illustrated in Figure 2.1. The reference frame of the i th agent is denoted by $\{A_i\}$, the end-effector frame by $\{E_i\}$, and the object frame by $\{O\}$. Here $i = 1, 2, \dots, N$ is the index of each agent.

Agent Dynamics: Let $q_i \in \mathbb{R}^{n_i}$ denote the joint angles of agent i , where n_i is the number of joints. Let $x_i = [p_i^\top \ \eta_i^\top]^\top \in \mathbb{R}^3 \times \mathbb{T}^3$ be the pose of the end-effector of agent i , where $p_i \in \mathbb{R}^3$ is the 3D position of the origin of the frame $\{E_i\}$ and $\eta_i \in \mathbb{T}^3$ represents the Euler angles of the frame $\{E_i\}$. Moreover, let $v_i \in \mathbb{R}^6$ denote the spatial velocity of the frame $\{E_i\}$, composed of translational and rotational components: $[\dot{p}_i^\top \ \omega_i^\top]^\top$ where $\omega_i \in \mathbb{R}^3$ is the angular velocity of the frame $\{E_i\}$. To write the agent dynamics in a more physically meaningful form, we now use the manipulator Jacobian. Unlike Equation (2.3), which is abstract and dependent on the parameterisation of the task space, the manipulator Jacobian is defined through differential kinematics (1.4) and maps joint velocities to the end-effector twist i.e., $v_i = J_i(q_i)\dot{q}_i$ where $J_i \in \mathbb{R}^{n_i} \times \mathbb{R}^{6 \times n_i}$. This representation is independent of coordinate parameterisation and directly reflects the physical motion of the end-effector. With a slight abuse of notation, adopting the manipulator Jacobian formulation,

the dynamics of the i th agent can be expressed as [48],

$$\bar{M}_i(q_i)\dot{v}_i + \bar{C}_i(q_i, \dot{q}_i)\dot{v}_i + \bar{g}_i(q_i) + \bar{w}_i(q_i, \dot{q}_i, t) = u_i - \lambda_i \quad (2.4)$$

where $\bar{M}_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{6 \times 6}$ is the task-space positive definite inertia matrix, $\bar{C}_i : \mathbb{R}^{2n_i} \rightarrow \mathbb{R}^{6 \times 6}$ the task-space Coriolis matrix, $\bar{g}_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^6$ the task-space gravity vector, $\bar{w}_i : \mathbb{R}^{2n_i} \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^6$ represents model uncertainties and bounded external disturbances, $u_i \in \mathbb{R}^6$ is the task-space input wrench and $\lambda_i \in \mathbb{R}^6$ is the generalized force vector that agent i exerts on the object. The vector fields $\bar{M}_i(\cdot)$, $\bar{C}_i(\cdot)$, $\bar{g}_i(\cdot)$, are continuously differentiable, and we further assume that $\bar{w}_i(q_i, \dot{q}_i, t)$ is continuous in q_i and $\dot{q}_i$ for each fixed $t \geq 0$, and uniformly bounded in t for each fixed $(q_i, \dot{q}_i) \in \mathbb{R}^{2n_i}$. Nevertheless, in this work we assume that all the dynamic terms $\bar{M}_i(\cdot)$, $\bar{C}_i(\cdot)$, $\bar{g}_i(\cdot)$, $\bar{w}_i(\cdot)$ are *unknown* and cannot be used in the control design. Similarly, we do not employ any force-related sensor unit and hence λ_i is also unknown to agent i . The only information that agent i has access to is its joint variables $(q_i, \dot{q}_i)$, from which it can derive its end-effector's pose (p_i, η_i) and velocity v_i through the forward kinematics. The dynamics (2.4) can be written in vector form as:

$$\bar{M}(q)\dot{v} + \bar{C}(q, \dot{q})v + \bar{g}(q) + \bar{w}(q, \dot{q}, t) = u - \lambda, \quad (2.5)$$

where,

- $v = [v_1^\top, \dots, v_N^\top]^\top \in \mathbb{R}^{6N}$,
- $\bar{M} = \text{diag}\{\bar{M}_i\} \in \mathbb{R}^{6N \times 6N}$,
- $\bar{C} = \text{diag}\{\bar{C}_i\} \in \mathbb{R}^{6N \times 6N}$,
- $\bar{g} = [\bar{g}_1^\top, \dots, \bar{g}_N^\top]^\top \in \mathbb{R}^{6N}$,
- $\bar{w} = [\bar{w}_1^\top, \dots, \bar{w}_N^\top]^\top \in \mathbb{R}^{6N}$,
- $u = [u_1^\top, \dots, u_N^\top]^\top \in \mathbb{R}^{6N}$, and,
- $\lambda = [\lambda_1^\top, \dots, \lambda_N^\top]^\top \in \mathbb{R}^{6N}$.

In this formulation, we assume that the agents operate away from their kinematic singularities, where $J_i(q_i)$ loses rank.

Object Dynamics: The pose and velocity of the frame $\{O\}$ attached to the object's center of mass are denoted by $x_o = [p_o^\top \ \eta_o^\top]^\top \in \mathbb{R}^3 \times \mathbb{T}^3$ and $v_o = [\dot{p}_o^\top \ \omega_o^\top]^\top \in \mathbb{R}^6$, respectively, where $p_o \in \mathbb{R}^3$, $\eta_o \in \mathbb{T}^3$, and $\omega_o \in \mathbb{R}^3$ are the position, Euler-angle orientation, and angular velocity of the object's center of mass. The Euler-angles are $\eta_o = [\phi_o, \theta_o, \psi_o]^\top \in \mathbb{T}^3$. The dynamics of the object are given by

$$\dot{x}_o = J_{o_r}^{-1}(x_o)v_o, \quad (2.6a)$$

$$M_o(x_o)\dot{v}_o + C_o(x_o, v_o)v_o + g_o(x_o) + w_o(t) = \lambda_o \quad (2.6b)$$

where $M_o : \mathbb{R}^3 \times \mathbb{T}^3 \rightarrow \mathbb{R}^{6 \times 6}$ is the positive-definite inertia matrix, $C_o : \mathbb{R}^3 \times \mathbb{T}^3 \times \mathbb{R}^6 \rightarrow \mathbb{R}^{6 \times 6}$ is the Coriolis matrix, $g_o : \mathbb{R}^3 \times \mathbb{T}^3 \rightarrow \mathbb{R}^6$ is the gravity vector and $w_o : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^6$ is a bounded vector of external disturbances. Additionally, $J_{o_r} : \mathbb{T}^3 \rightarrow \mathbb{R}^{6 \times 6}$ is the object representation Jacobian and $\lambda_o \in \mathbb{R}^6$ is the force vector acting on the object's center of mass. The object representation Jacobian J_{o_r} is a function of the object's Euler angles η_o and is defined as,

$$J_{o_r} = \text{diag}\{I_3, J_{\eta_o}\}, \text{ where } J_{\eta_o} = \begin{bmatrix} 1 & \sin(\phi_o) \tan(\theta_o) & \cos(\phi_o) \tan(\theta_o) \\ 0 & \cos(\phi_o) & -\sin(\theta_o) \\ 0 & \frac{\sin(\phi_o)}{\cos(\theta_o)} & \frac{\cos(\phi_o)}{\cos(\theta_o)} \end{bmatrix}$$

and is not well defined for $\theta = \pm \frac{\pi}{2}$ and is referred to as a representation singularity. The terms $M_o(\cdot)$, $C_o(\cdot)$, $g_o(\cdot)$ are continuously differentiable and we assume that $w_o(t)$ is bounded. The matrix J_{o_r} and its inverse are well-defined when the object's pitch angle θ_o (second entry of η_o) evolves in the set $(-\frac{\pi}{2}, \frac{\pi}{2})$, which is assumed to be the case. Additionally and similarly to the agents, the object's dynamic terms $M_o(\cdot)$, $C_o(\cdot)$, $g_o(\cdot)$, $w_o(\cdot)$ and force λ_o are considered to be unknown and cannot be used in the control design.

Coupled Agent-Object Dynamics: In this analysis, we assume rigid grasps, meaning there is no relative translational or rotational motion between the manipulators' end-effectors and the object. This assumption is justified when the manipulated object is relatively small compared to the robot's grasping capability, or when the static and dynamic friction forces at the grasp interfaces are sufficiently large to prevent slippage. We adopt this approach because it allows us to focus explicitly on the more challenging aspect of control design after grasp acquisition, rather than addressing the complexities involved in determining optimal grasp configurations and strategies.

In view of Figure 2.1, the pose of end-effector frame $\{E_i\}$ and the object center of mass frame $\{O\}$ are related as,

$$p_i = p_o + p_{i/o}, \quad \eta_i = \eta_o + \eta_{i/o} \quad (2.7)$$

where p_i is the position of i th end effector in an inertial frame, p_o is the position of the object center of mass in an inertial frame, and $p_{i/o}$ is the position of the end-effector frame $\{E_i\}$ with respect to the object frame $\{O\}$ in the same inertial frame. The vector $p_{i/o}$ is constant and depends on the grasp configuration. The same holds for the orientation of the end-effector frame $\{E_i\}$ with respect to the object frame $\{O\}$. The velocity of the end-effector frame $\{E_i\}$ can be obtained by differentiating the equation (2.7),

$$v_i = J_{o_i}(q_i)v_o \quad (2.8)$$

where $J_{o_i} : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^6$ is the object-to- i th agent Jacobian, which is defined as,

$$J_{o_i} = \begin{bmatrix} I_3 & S(-p_{i/o}) \\ 0_3 & I_3 \end{bmatrix}$$

and is always full rank. Moreover from (2.8) one obtains,

$$\dot{v}_i = J_{o_i}(q_i)\dot{v}_o + \dot{J}_{o_i}(q_i)v_o.$$

Observe that the angular velocity of end-effector is equal to that of the object due to grasp rigidity, i.e., $\omega_i = \omega_o$. The kineto-static duality between the end-effector and object frames implies that the generalized forces acting on the end-effector frame $\{E_i\}$ are equal to those acting on the object frame $\{O\}$, i.e.,

$$\lambda_o = G\lambda, \quad (2.9)$$

where $G := G(q) : \mathbb{R}^{\sum_{i=1}^N n_i} \rightarrow \mathbb{R}^{6 \times 6N}$ is the full row-rank grasp matrix defined as,

$$G(q) = [J_{o_1}^\top(q_1) \quad J_{o_2}^\top(q_2) \quad \cdots \quad J_{o_N}^\top(q_N)].$$

Substituting the dynamics (2.6) and (2.5) in equation (2.9) results in,

$$M_o(x_o)\dot{v}_o + C_o(x_o, v_o)v_o + g_o(x_o) + w_o(t) = G \left(u - \bar{M}(q)\dot{v} - \bar{C}(q, \dot{q})v - \bar{g}(q) - \bar{w}(q, \dot{q}, t) \right). \quad (2.10)$$

The end-effector velocity (2.8) in vector form is,

$$v = G^\top v_o \in \mathbb{R}^{6N},$$

and the end-effector acceleration is,

$$\dot{v} = \dot{G}^\top v_o + G^\top \dot{v}_o \in \mathbb{R}^{6N}.$$

Substituting these terms in (2.10) gives us the coupled agent-object dynamics,

$$(M_o + G\bar{M}G^\top)\dot{v}_o + (C_o + G\bar{M}\dot{G}^\top + GCG^\top)v_o + (g_o + G\bar{g}) + (w_o + G\bar{w}) = Gu$$

and the overall system dynamics can be expressed as,

$$\tilde{M}\dot{v}_o + \tilde{C}v_o + \tilde{g} + \tilde{w} = Gu \quad (2.11)$$

where

$$\tilde{M} = M_o + G\bar{M}G^\top \quad (2.12)$$

$$\tilde{C} = C_o + G\bar{M}\dot{G}^\top + GCG^\top \quad (2.13)$$

$$\tilde{g} = g_o + G\bar{g} \quad (2.14)$$

$$\tilde{w} = w_o + G\bar{w}. \quad (2.15)$$

Prescribed Performance Control

The control design in this chapter relies on the framework of Prescribed Performance Control (PPC) [49] to satisfy STL tasks for a cooperatively manipulated object. PPC ensures that a tracking error $e(t)$ evolves strictly within a predefined, time-varying region, thereby enforcing both transient and steady-state performance specifications. This region is defined by a performance function $\gamma(t)$, which shapes the desired evolution of the error over time.

Definition 1 *A performance function $\gamma(t)$ is a continuously differentiable, bounded, strictly positive and decreasing function of time. An example function we consider in this work is, $\gamma(t) = (\gamma_0 - \gamma_\infty) \exp(-lt) + \gamma_\infty$ where $\gamma_0, \gamma_\infty \in \mathbb{R}_{>0}$ with $\gamma_0 \geq \gamma_\infty$ and $l \in \mathbb{R}_{\geq 0}$.*

More specifically, the constant $\gamma_0 = \gamma(0)$ is selected such that $\gamma_0 > |e(0)|$, and the constant $\gamma_\infty = \lim_{t \rightarrow \infty} \gamma(t)$ represents the maximum allowable size of the tracking error $e(t)$ at steady state, which may be set arbitrarily small to a value reflecting the resolution of the measurement device, thus achieving practical convergence of $e(t)$ to zero. Moreover, the decreasing rate of $\gamma(t)$, which is affected by the constant l in this case, introduces a lower bound on the required speed of convergence of $e(t)$. Therefore, the appropriate selection of the performance function imposes performance characteristics on the tracking error $e(t)$. To implement PPC, we follow the following procedure,

- Define a performance function $\gamma(t)$ that describes the desired evolution of the tracking error $e(t)$ over time. The performance function is selected such that it is strictly positive, bounded, and decreasing.
- Transform the error function to an unconstrained control problem by normalizing it with respect to the performance function. This transformation makes sure that boundedness of the transformed error enforces the satisfaction of the constraints.
- Design a gradient-based control law that guarantees the boundedness of the transformed error signal, thereby ensuring that the original constraints are satisfied.

Signal Temporal Logic Fragment

In Section 1.2, we introduced the general form of STL formula, their semantics, and associated robustness metrics. In this chapter, to enable control design that can reason about and guarantee the satisfaction of STL tasks, we focus on a smaller yet expressive fragment of STL. The fragment considered is,

$$\psi ::= \top \mid \mu \mid \neg\mu \tag{2.16a}$$

$$\phi ::= \psi \mid \mathcal{G}_{[a,b]}\psi \mid \mathcal{F}_{[a,b]}\psi \mid \psi_1 \mathcal{U}_{[a,b]}\psi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \tag{2.16b}$$

where ψ, ψ_1 and ψ_2 are formulae of class ψ given in (2.16a), ϕ_1, ϕ_2 are formulae of class ϕ given in (2.16b) and $a, b \in \mathbb{R}_{\geq 0}$ with $a \leq b$. The following temporal operators still can be defined from (1.1); *eventually*: $\mathcal{F}_{[a,b]}\psi \equiv \top \mathcal{U}_{[a,b]}\psi$, *always*: $\mathcal{G}_{[a,b]}\psi \equiv \neg \mathcal{F}_{[a,b]}\neg\psi$ and *disjunction*: $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$.

Through (2.16), we can still define constraints such as ‘Let the signal $\mathbf{x}$ be ϵ -close to point A in the entire time interval $[5, 10]$ s and let the signal $\mathbf{x}$ be ϵ -close to point B sometime in the time interval $[10, 15]$ s’, with the following formula,

$$\phi = \mathcal{G}_{[5,10]}(\|\mathbf{x} - A\| \leq \epsilon) \wedge \mathcal{F}_{[10,15]}(\|\mathbf{x} - B\| \leq \epsilon)$$

for some $\epsilon > 0$.

2.3 Cooperative Object Manipulation under STL Constraints

Problem Formulation

The problem at hand is the design of a decentralised control policy for u_i , $i \in \{1, 2, \dots, N\}$, in (2.4), such that the object satisfies a user-defined STL task. More formally:

Problem 1 *Consider a system of N agents rigidly grasping an object with unknown coupled dynamics (2.11). Let an STL formula ϕ (as in (2.16)) over the object trajectory $x_o(t)$. Design a decentralised control policy for u_i , $i \in \{1, \dots, N\}$, in (2.4), that leads to the satisfaction of ϕ , i.e., $(x_o, t) \models \phi$.*

Solution Approach

Our approach to solving the above problem proceeds as follows. First, we reformulate the STL satisfaction problem as a state evolution problem constrained within a predefined, time-varying funnel. We then employ the PPC methodology to ensure that the system remains within this funnel, thereby guaranteeing satisfaction of the STL task. The proposed control framework is fully decentralised, meaning that each agent computes its control input independently, without requiring communication with other agents. It operates without any knowledge of the dynamics of the robotic agents or the object, and does not rely on estimation or approximation models. Moreover, it does not require force or torque sensors at the agent-object grasping interfaces. We also avoid the use of approximations in STL robustness metrics and consider a richer STL fragment than those addressed in prior work.

Encoding STL Tasks in the PPC Framework

As seen in Section 1.2, an STL formula ϕ consists of temporal operators along with predicates μ (see (2.16)). The predicates are defined based on the predicate

function, denoted by $p(x_o, t)$ i.e.,

$$\mu = \begin{cases} \top & p(x_o, t) \geq 0 \\ \perp & p(x_o, t) < 0 \end{cases}. \quad (2.17)$$

We assume the predicate functions belong to C^1 , that is, they are continuously differentiable. We exploit the PPC formulation by appropriately mapping each predicate functions $p(x_o, t)$ to a corresponding C^1 functions $h(x_o, t)$, such that $\mu = \top$ if and only if $-\gamma_h(t) < h(x_o(t), t) < \gamma_h(t)$, where $\gamma_h(t)$ is a user-defined performance function. That is, predicate satisfaction is equivalent to the containment of $h(x_o(t), t)$ within the interval $(-\gamma_h(t), \gamma_h(t))$.

We now describe in more detail the construction of the performance and predicate functions for the fragment of the form (2.16b). The STL formula under consideration can be expressed as,

$$\phi = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_R, \text{ where each } \phi_i \text{ is a conjunction of the form,} \quad (2.18)$$

$$\phi_i = \phi_{i,1} \wedge \phi_{i,2} \wedge \cdots \wedge \phi_{i,M_i}, \text{ and each } \phi_{i,j} \text{ is of the form,} \quad (2.19)$$

$$\psi \mid \mathcal{G}_{[a,b]}\psi \mid \mathcal{F}_{[a,b]}\psi, \quad (2.20)$$

for all $j \in \{1, \dots, M_i\}$, $i \in \{1, \dots, R\}$ for positive constants M_i and R . And ψ is a formula of the form (2.16a). Here, R represents the number of disjunction components and M_i represents the number of conjunction components corresponding to each i -th disjunction component. The until operator $\psi_1 \mathcal{U}_{[a,b]} \psi_2$ is equivalent to the conjunction $\mathcal{F}_{[t_1]}\psi_2 \wedge \mathcal{G}_{[a,t_1]}\psi_1$, with $t_1 \in (a, b]$, and is hence incorporated in the aforementioned formulation.

We consider a bottom-up approach and consider first a formula $\phi_{i,j}$ of the form (2.20). For each $\phi_{i,j}$, $i \in \{1, \dots, R\}$, $j \in \{1, \dots, M_i\}$, if $\phi_{i,j} = \mathcal{G}_{[a_{i,j}, b_{i,j}]}\psi_{i,j}$ or $\phi_{i,j} = \mathcal{F}_{[a_{i,j}, b_{i,j}]}\psi_{i,j}$, then we opt to activate $\phi_{i,j}$ only in the specific time interval $[a_{i,j}, b_{i,j}]$. More specifically, we define smooth switching functions $\beta_{i,j}(t)$ such that

$$\beta_{i,j}(t) = \begin{cases} 0, & t < t_{i,j}^* - \delta \\ 1, & t \in [t_{i,j}^*, b_{i,j}] \\ 0, & t > b_{i,j} + \delta, \end{cases} \quad (2.21)$$

where $\delta > 0$ is the “rise and fall” time, and we set $\beta_{i,j}(t) = 1$ if $\phi_{i,j} = \psi_{i,j}$ (i.e., there are no temporal constraints). See Figure 2.2 for an illustration of the switching function

$$\beta_{i,j}(t) = \frac{1}{1 + \exp(-30(t + 0.2))} - \frac{1}{1 + \exp(-30(t - 5.2))}$$

corresponding to the formula $\phi = \mathcal{G}_{[0,5]}\mu$. In the above function, $t_{i,j}^* = a_{i,j} = 0$ and $\delta = 0.4$.

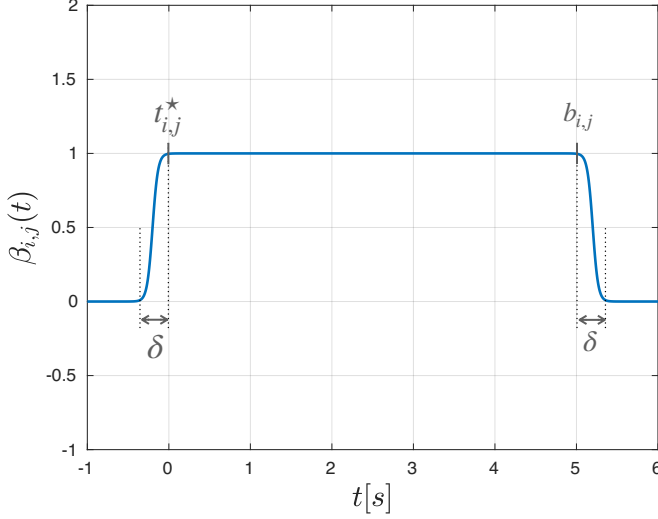


Figure 2.2: An illustration of the switching function (2.21)

The time instants $t_{i,j}^* \in \mathbb{R}_{\geq 0}$ are chosen a priori as

$$t_{i,j}^* \in \begin{cases} [a_{i,j}] & \text{if } \phi_{i,j} = \mathcal{G}_{[a_{i,j}, b_{i,j}]} \psi \\ [a_{i,j}, b_{i,j}] & \text{if } \phi_{i,j} = \mathcal{F}_{[a_{i,j}, b_{i,j}]} \psi. \end{cases}$$

In the experiments, $t_{i,j}^*$ for the eventually($\mathcal{F}$) operator is chosen randomly in the interval $(a_{i,j}, b_{i,j}]$. Next, let the predicates $\mu_{i,j}$ and the respective predicate functions $p_{i,j}(x_o, t)$ that correspond to $\psi_{i,j}$ of the subformulae (2.20). We map $p_{i,j}(x_o, t)$ to C^1 functions $h_{i,j}(x_o, t)$ such that

$$\mu_{i,j} = \begin{cases} \top, & -\gamma_{h_{i,j}}(t) < \bar{h}_{i,j}(x_o, t) < \gamma_{h_{i,j}}(t), \\ \perp, & \text{otherwise} \end{cases}$$

which implies

$$p_{i,j}(x(t), t) \geq 0 \Leftrightarrow -\gamma_{h_{i,j}}(t) < \bar{h}_{i,j}(x_o(t), t) < \gamma_{h_{i,j}}(t), \quad (2.22)$$

where

$$\bar{h}_{i,j}(x_o, t) = \beta_{i,j}(t) h_{i,j}(x_o, t). \quad (2.23)$$

Here, $\gamma_{h_{i,j}}(t)$ are appropriately chosen performance functions of the form $\gamma_{h_{i,j}}(t) = (\gamma_{h_{i,j}}^0 - \gamma_{h_{i,j}}^\infty) \exp(-l_{i,j} t) + \gamma_{h_{i,j}}^\infty$ (see Section 2.2), and the incorporation of $\beta_{i,j}$ in $\bar{h}_{i,j}$ accommodates the local activation of the temporal subformulae of the form $\mathcal{G}_{[a_{i,j}, b_{i,j}]} \psi_{i,j}$, $\mathcal{F}_{[a_{i,j}, b_{i,j}]} \psi_{i,j}$; intuitively, we aim to force $h_{i,j}(x_o, t)$ to evolve inside the

funnels prescribed by $\gamma_{h_{i,j}}(t)$, such that an appropriate choice of $\gamma_{h_{i,j}}(t)$ will yield the satisfaction of the STL task. Apart from achieving satisfaction of the predicates, the PPC formulation allows for greater control over the rate of convergence and robustness of $h_{i,j}(x_o, t)$, since the latter are explicitly shaped by the user-defined performance functions $\gamma_{h_{i,j}}(t)$. These functions also accommodate the temporal constraints imposed by the operators $\mathcal{G}_{[a_{i,j}, b_{i,j}]}, \mathcal{F}_{[a_{i,j}, b_{i,j}]}$. For instance, let $\phi_{i,j} = \mathcal{G}_{[5,10]}(\|x_o(t)\| < \epsilon)$. Then, $p_{i,j}(x_o, t) = \epsilon - \|x_o\|$, $h_{i,j}(x_o, t) = \|x_o\|$, and we choose $\gamma_{h_{i,j}}$ such that $\gamma_{h_{i,j}}(5) = \epsilon$, implying $\gamma_{h_{i,j}}(t) < \epsilon$, for all $t > 5$. Note that, subformulae of the form $\mathcal{G}_{[a_{i,j}, b_{i,j}]} \psi$ and ψ must be satisfied initially, i.e., at $t = a_{i,j}$ and $t = 0$, respectively. This is formalised in the next assumption:

Assumption 1 *For every $\phi_{i,j} = \psi$, it holds that $p_{i,j}(x_o(0), 0) \geq 0$, and for every $\phi_{i,j} = \mathcal{G}_{[a_{i,j}, b_{i,j}]} \psi$, it holds that $p_{i,j}(x_o(a_{i,j}), 0) \geq 0$, $j \in \{1, \dots, M_i\}$, $i \in \{1, \dots, R\}$.*

Consider the following examples of STL formulas and their corresponding performance functions, as illustrated in in Figure 2.3.

- $\phi = |x| < 2$. Here, the error function is $e(x) = x$, and the performance functions that lead to the satisfaction of ϕ are $\gamma(t) = 2$ and $\gamma(t) = -2$.
- $\phi = \mathcal{G}_{[5,10]}((x^2 - 2) < 0.1)$. In this case, the error function is $e(x) = x^2 - 2$ and the performance functions are designed such that $\gamma(5) = 0.1$ and $\gamma(10) = 0.1$. This ensures that if $e(x)$ remains within the funnel bounds, the STL formula is satisfied.
- $\phi = \mathcal{F}_{[5,10]}((x^2 - 2) < 0.1)$. Again, the error function is $e(x) = x^2 - 2$ but now the performance function is designed such that $\gamma(t^*) = 0.1$ where $t^* \in [5, 10]$.

Conjunctions:

Consider now the formula $\phi_i = \phi_{i,1} \wedge \phi_{i,2} \wedge \dots \wedge \phi_{i,M_i}$ where each $\phi_{i,j}$ is of the form (2.20), $i \in \{1, \dots, R\}$. Let $\bar{h}_{i,j}(x_o, t)$ be the predicate function corresponding to each $\phi_{i,j}$, constructed as in (2.23). We accommodate the aforementioned conjunction by defining the vector $\bar{h}_i(x_o, t) = [\bar{h}_{i,1} \dots \bar{h}_{i,M_i}]^\top$; all $\bar{h}_{i,j}$, $j \in \{1, \dots, M_i\}$, must satisfy (2.22) to guarantee satisfaction of ϕ_i . The following example illustrates the construction of $\bar{h}_i$.

Example 1 *Consider the STL formula*

$$\begin{aligned} \phi_i = & \mathcal{G}_{[5,10]} (\|x_o - A\|_p < 0.05) \wedge \\ & \mathcal{F}_{[5,15]} (\|x_o - B\|_p < 0.05) \wedge \\ & \mathcal{G}_{[10,20]} \left(\phi_o - \pi < \frac{\pi}{6} \right) \wedge \\ & (\|x_o - C\|_p < 1) \mathcal{U}_{[20,25]} (\|x_o - C\|_p < 0.05) \end{aligned}$$

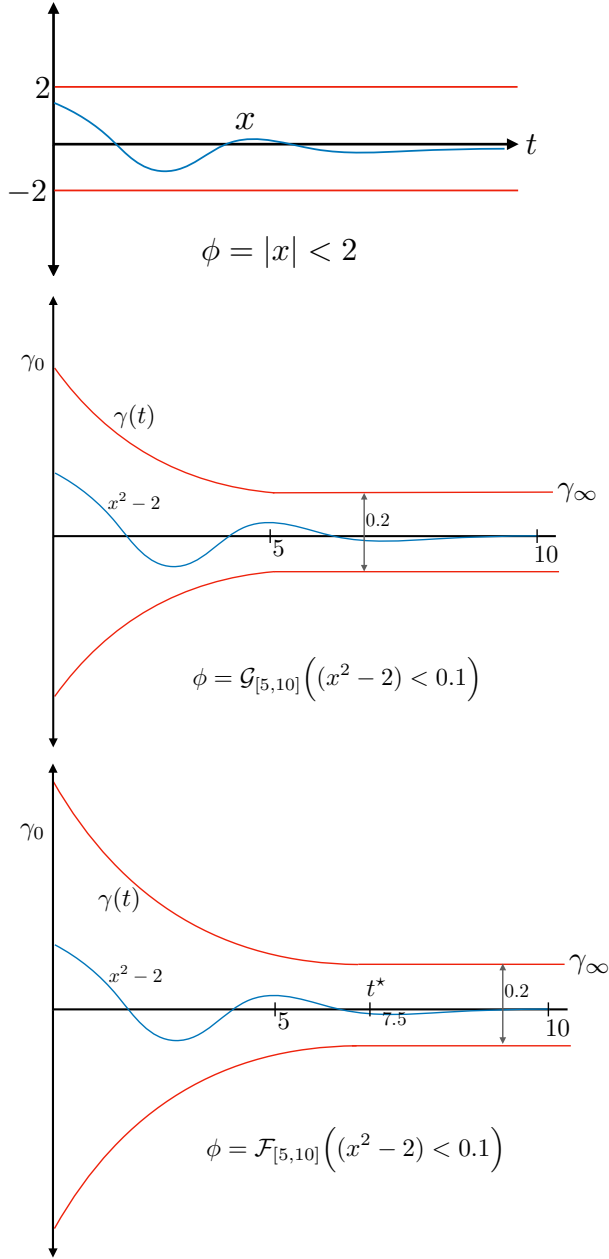


Figure 2.3: Design of performance functions for the respective STL formulas

where A, B and C are some desired positions for x_o . The formula ϕ_i can be broken down into four subtasks, namely,

1. $\mathcal{G}_{[5,10]}(\|x_o - A\|_p < 0.05)$ which requires the object x_o between 5s and 10s to match configuration A within a margin of 0.05;
2. $\mathcal{F}_{[5,15]}(\|x_o - B\|_p < 0.05)$ which requires the object x_o between 5s and 15s to eventually match configuration B within a margin of 0.05;
3. $\mathcal{G}_{[10,20]}(\phi_o - \pi < \frac{\pi}{6})$ requires the object to assume a roll angle of π between 10s and 20s within a margin of $\pi/6$; and finally
4. $(\|x_o - C\|_p < 1)\mathcal{U}_{[20,25]}(\|x_o - C\|_p < 0.05)$ requires the object to assume configuration C with an accuracy of 1 until the object gets close enough to C within the margin 0.05 in the interval $[20, 25]$ s.

The function

$$\bar{h}_i(x_o, t) = \begin{bmatrix} \bar{h}_{i,1}(x_o, t) \\ \bar{h}_{i,2}(x_o, t) \\ \bar{h}_{i,3}(x_o, t) \\ \bar{h}_{i,4}(x_o, t) \\ \bar{h}_{i,5}(x_o, t) \end{bmatrix},$$

corresponding to ϕ_i , is of the form,

$$\bar{h}_i(x_o, t) = \begin{bmatrix} \beta_{i,1}(t) h_{i,1}(x_o) \\ \beta_{i,2}(t) h_{i,2}(x_o) \\ \beta_{i,3}(t) h_{i,3}(x_o) \\ \beta_{i,4}(t) h_{i,4}(x_o) \\ \beta_{i,5}(t) h_{i,5}(x_o) \end{bmatrix} = \begin{bmatrix} \beta_{i,1}(t) \|x_o - A\|_p \\ \beta_{i,2}(t) \|x_o - B\|_p \\ \beta_{i,3}(t) (\phi_o - \pi) \\ \beta_{i,4}(t) \|x_o - C\|_p \\ \beta_{i,5}(t) \|x_o - C\|_p \end{bmatrix},$$

where $M_i = 5$ and $\beta_{i,j}(t)$ are switching functions of the form (2.21), $j \in \{1, \dots, 5\}$.

Disjunctions:

Finally, we consider disjunctions of the form $\phi = \phi_1 \vee \phi_2 \vee \dots \vee \phi_R$ where each ϕ_i is a conjunction of subformulas of the form (2.20), $i \in \{1, \dots, R\}$. As discussed before, each ϕ_i corresponds to a predicate function $h_i(x_o, t) \in \mathbb{R}^{M_i}$; Since satisfying ϕ_i for any i , satisfies ϕ , we use a cost metric to decide on initially choosing such a ϕ_i . More specifically, we define the cost function $J_i(h_i(x_o, t_0))$, which captures the robust satisfaction of the STL subtask ϕ_i , with t_0 being the first time instant appearing in the subformulas $\phi_{i,j}$, $i \in \{1, \dots, R\}$, $j \in \{1, \dots, M_i\}$. One such example of J_i is $J_i(h(x_o, t_0)) = \frac{1}{M_i} \sum_{j \in \{1, \dots, M_i\}} \left| \frac{h_{i,j}(x_o, t_0)}{\gamma_{i,j}(t_0)} \right|$, which chooses a ϕ_i that is, on average, furthest from the funnel boundaries at t_0 . We select then the

subtask ϕ_ℓ that satisfies $\ell = \arg \min_i J_i(h_i(x_o(t_0), t_0))$. We do this automatically by defining the variable

$$k_\ell(t) = \begin{cases} 1 & \text{if } \ell = \arg \min_i J_i(h_i(x_o(t_0), t_0)) \\ 0 & \text{otherwise.} \end{cases}$$

and finally selecting the predicate function for ϕ as $\tilde{h} = [\tilde{h}_1^\top, \dots, \tilde{h}_R^\top]^\top$, with

$$\tilde{h}(x_o, t) = \bar{K}(t) [\bar{h}_1(x_o, t) \quad \dots \quad \bar{h}_R(x_o, t)]^\top$$

where $\bar{K}(t) = \text{blkdiag}\{[k_i I_{M_i}]_{i \in \{1, \dots, R\}}\}$. An example of this strategy is shown below.

Example 2 Consider the STL formula,

$$\phi = \left[\mathcal{G}_{[5,10]}(\|x_o - A\|_p < 1) \wedge F_{[10,15]}(\|x_o - B\|_p < 0.05) \right] \vee \left[\mathcal{F}_{[5,15]}(\|x_o - C\|_p < 0.05) \right]$$

where A, B and C are desired attainable positions for x_o . The predicate function corresponding to ϕ is

$$\tilde{h}(x_o, t) = \left[\begin{array}{cc|c} k_1(t) & 0 & 0 \\ 0 & k_1(t) & 0 \\ \hline 0 & 0 & k_2(t) \end{array} \right] \left[\begin{array}{c} \beta_1(t) \|x_o - A\|_p \\ \beta_2(t) \|x_o - B\|_p \\ \hline \beta_3(t) \|x_o - C\|_p \end{array} \right].$$

Remark 1 In the aforementioned procedure on the derivation of k_ℓ for the disjunction sub-formulae, the minimization involves the time instant t_0 (see $J_i(h(x_o(t_0), t_0))$). For instance, the algorithm might choose the index $\ell = \arg \min_i J_i(h(x_o(t_0), t_0))$ at $t = t_0$, choosing to satisfy ϕ_ℓ , but there might exist t' and ℓ' such that $\ell' = \arg \min_i J_i(h(x_o(t_0), t)) \neq \ell$, for all $t \geq t'$. We can extend the proposed algorithm to handle such cases by considering a time-varying index $\ell(t) = \arg \min_i J_i(h(x_o(t_0), t))$. To retain the differentiability of $\tilde{h}$, we can replace the $\min(\cdot)$ operator with a differentiable approximation (see, e.g., [31]).

We note that this stage of planning occurs offline, based on the STL formula that is available to the agents.

Controller Design

In this section, we present the decentralised control design that guarantees compliance with the funnel of the previous section and consequently, satisfaction of the STL task. Before proceeding, we impose the following required assumptions.

Assumption 2 The object's pose does not result in a singular $J_{o_r}(x_o(t))$, i.e. $\theta_o(t) \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$, for all $t \geq 0$.

Assumption 3 The function $\tilde{h}(x_o, t)$ satisfies the following properties:

- $\tilde{h}(x_o, t)$ is continuously differentiable in $\mathbb{R}^3 \times \mathbb{T}^3 \times \mathbb{R}_{\geq 0}$ and $\tilde{h}$, $\frac{\partial \tilde{h}}{\partial x_o}$, and $\frac{\partial \tilde{h}}{\partial t}$ are uniformly bounded in t for all $x_o \in \mathbb{R}^3 \times \mathbb{T}^3$.
- For every $c_1 > 0$, there exists a $c_2 > 0$ such that $\{x_o \in \mathbb{R}^3 \times \mathbb{T}^3 : \|\tilde{h}(x_o, t)\| \leq c_1\} \subset \{x_o \in \mathbb{R}^3 \times \mathbb{T}^3 : \|x_o\| \leq c_2\}$.

Assumption 2 is required [48] for the controllability of the object's pose (see (2.6)). Assumption 3 provides simple differentiability and boundedness conditions for $\tilde{h}$ and that boundedness of $\tilde{h}(x_o, t)$ implies the boundedness of x_o .

Next, let ℓ be the index of the chosen subformulae to be satisfied in the disjunction operator, i.e., $k_\ell = 1$ and $k_i = 0$, for all $i \in \{1, \dots, R\} \setminus \{\ell\}$. Then, the task to be satisfied is given by the function $\tilde{h}_\ell(x_o, t) = \bar{h}_\ell(x_o, t) = [\bar{h}_{\ell,1}, \dots, \bar{h}_{\ell,M_\ell}]^\top$. We provide now the control design.

Step I-a. Choose the performance functions

$$\gamma_h(t) = \text{diag}\{\gamma_{h_{\ell,1}}, \gamma_{h_{\ell,2}}, \dots, \gamma_{h_{\ell,M_\ell}}\}$$

and,

$$\gamma_{h_{\ell,j}}(t) = (\gamma_{h_{\ell,j}}^0 - \gamma_{h_{\ell,j}}^\infty) \exp(-l_{\ell,j}t) + \gamma_{h_{\ell,j}}^\infty,$$

with $\gamma_{h_{\ell,j}}^0 > \bar{h}_{\ell,j}(x_o(0), 0)$; $l_{\ell,j} \in \mathbb{R}_{>0}$ and $\gamma_{h_{\ell,j}}^\infty$ are chosen such that, when $-\gamma_{h_{\ell,j}}(t) < \bar{h}_{\ell,j}(x_o, t) < \gamma_{h_{\ell,j}}(t)$ holds, the task is satisfied, for $j \in \{1, 2, \dots, M_\ell\}$. In the special case that $\phi_{\ell,j}$ has the form $\|x_o - A\| < z$, for some $x_o \in \mathbb{R}^3 \times \mathbb{T}^3$, $z > 0$, we set $\gamma_{h_{\ell,j}}^\infty = z$.

Step I-b. Define the normalised errors $\xi_h \in \mathbb{R}^L$ by

$$\xi_h = \begin{bmatrix} \xi_{h_{\ell,1}} \\ \vdots \\ \xi_{h_{\ell,M_\ell}} \end{bmatrix} = \gamma_h^{-1}(t) \bar{h}_\ell(x_o, t) \quad (2.24)$$

and design the reference velocity v_r as

$$v_r(\xi_h, t) = -g_s J_{o_r}(x_o) \frac{\partial \bar{h}_\ell}{\partial x_o} \gamma_h^{-1}(t) r_h(\xi_h) \varepsilon_h(\xi_h) \quad (2.25)$$

where g_s is a positive constant and the signals $\varepsilon_h : (-1, 1)^{M_\ell} \rightarrow \mathbb{R}^{M_\ell}$ and $r_h : (-1, 1)^{M_\ell} \rightarrow \mathbb{R}^{M_\ell \times M_\ell}$ are

$$\varepsilon_h(\xi_h) = \begin{bmatrix} \varepsilon_{h_{\ell,1}}(\xi_{h_{\ell,1}}) \\ \vdots \\ \varepsilon_{h_{\ell,M_\ell}}(\xi_{h_{\ell,M_\ell}}) \end{bmatrix}, \quad r_h(\xi_h) = \text{diag}\{[r_{h_{\ell,j}}(\xi_{h_{\ell,j}})]_{j \in \{1, \dots, M_\ell\}}\},$$

with

$$\varepsilon_{h_{\ell},j} = \ln \left(\frac{1 + \xi_{h_{\ell},j}}{1 - \xi_{h_{\ell},j}} \right), \quad r_{h_{\ell},j} = \frac{2}{1 - \xi_{h_{\ell},j}^2}, \quad j \in \{1, \dots, M_{\ell}\}.$$

Step II follows the lines presented in [20].

Step II-a. Define the velocity error $e_v \in \mathbb{R}^6$ as,

$$e_v = \begin{bmatrix} e_{v_1} \\ \vdots \\ e_{v_6} \end{bmatrix} = v_o - v_r(\xi_h, t), \quad (2.26)$$

and velocity performance functions

$$\gamma_v(t) = \text{diag}\{\gamma_{v_1}, \gamma_{v_2}, \dots, \gamma_{v_6}\}$$

where

$$\gamma_{v_n}(t) = (\gamma_{v_n}^0 - \gamma_{v_n}^{\infty}) \exp(-l_{v_n} t) + \gamma_{v_n}^{\infty},$$

with parameters $\gamma_{v_n}^0 > |e_{v_n}(t_0)|$, $\gamma_{v_n}^{\infty} \in (0, \gamma_{v_n}^0)$ and $l_{v_n} > 0$, $\forall n \in \{1, \dots, 6\}$.

Step II-b. Define the normalised velocity errors $\xi_v \in \mathbb{R}^6$:

$$\xi_v = \begin{bmatrix} \xi_{v_1} \\ \vdots \\ \xi_{v_6} \end{bmatrix} = \gamma_v^{-1}(t) e_v, \quad (2.27)$$

where $\gamma_v = \text{diag}\{[\gamma_{v_m}]_{m \in \{1, \dots, 6\}}\}$, and design the decentralized control law as

$$u(\xi_h, \xi_v, t) = \begin{bmatrix} u_1(\xi_h, \xi_v, t) \\ \vdots \\ u_N(\xi_h, \xi_v, t) \end{bmatrix} = -C_g G^*(q) \gamma_v^{-1}(t) r_v(\xi_v) \varepsilon_v(\xi_v) \quad (2.28)$$

where

$$G^*(q) = [J_{o_1}^{-1}(q_1), \dots, J_{o_N}^{-1}(q_N)] \in \mathbb{R}^{6N \times 6},$$

$$C_g = g_v \text{diag}\{[c_i I_6]_{i \in \mathcal{N}}\} \in \mathbb{R}^{6N \times 6N},$$

and the signals $\varepsilon_v : (-1, 1)^6 \rightarrow \mathbb{R}^6$ and $r_v : (-1, 1)^6 \rightarrow \mathbb{R}^{6 \times 6}$ are

$$\varepsilon_v(\xi_v) = \begin{bmatrix} \varepsilon_{v_1}(\xi_{v_1}) \\ \vdots \\ \varepsilon_{v_6}(\xi_{v_6}) \end{bmatrix}, \quad r_v(\xi_v) = \text{diag}\{[r_{v_n}(\xi_{v_n})]_{n \in \{1, \dots, 6\}}\},$$

with

$$\varepsilon_{v_n} = \ln \left(\frac{1 + \xi_{v_n}}{1 - \xi_{v_n}} \right), \quad r_{v_n} = \frac{2}{1 - \xi_{v_n}^2}, \quad n \in \{1, \dots, 6\}.$$

Note that the STL planning, i.e., the design of performance functions in Step I-a and Step II-a is performed offline, whereas the control input (2.28) is applied online by the agents.

Remark 2 The control protocol guarantees the funnel containment by enforcing the normalized errors $\xi_{h_{\ell,j}}, \xi_{v_i}$ to remain strictly within $(-1, 1)$, which is equivalent to guaranteeing the boundedness of the transformed signals $\varepsilon_{h_{\ell,j}}, \varepsilon_{v_i}$, respectively. These signals, which are used in the control design, operate similarly to reciprocal barrier functions [50]. In particular, they admit higher negative or positive values as $\bar{h}_{\ell,j}(x_o, t)$ and e_{v_i} approach the funnel boundaries, enforcing in that way the funnel containment while implicitly counteracting the unknown dynamic terms.

The correctness of the proposed control scheme is proven in the following theorem.

Theorem 1 Consider N agents rigidly grasping an object with coupled dynamics (2.11) subject to an STL formula ϕ of the form (2.16). Assume that $\lambda_{\min}\left(\frac{\partial \bar{h}_{\ell}}{\partial x_o} \frac{\partial \bar{h}_{\ell}}{\partial x_o}^{\top}\right) \geq \kappa > 0$, for all $t \geq 0$. Then the decentralized control (2.28) guarantees $|\bar{h}_{\ell,j}(x_o(t), t)| < \gamma_{\ell,j}(t)$, for all $t \geq 0$, and the boundedness of all closed loop signals.

Proof 1 Consider first the dynamics of the normalised errors

$$\begin{aligned}\dot{\xi}_h &= \gamma_h^{-1}(t)(\dot{\bar{h}}(x_o, t) - \dot{\gamma}_h(t)\xi_h), \\ \dot{\xi}_v &= \gamma_v^{-1}(t)(\dot{e}_v - \dot{\gamma}_v(t)\xi_v).\end{aligned}$$

which, after using (2.11), (2.26),

$$\dot{\xi}_h = \gamma_h^{-1}(t) \left(\frac{\partial \bar{h}}{\partial x_o} J_{o_r}^{-1}(x_o) v_o + \frac{\partial \bar{h}}{\partial t} - \dot{\gamma}_h(t)\xi_h \right), \quad (2.29a)$$

$$\dot{\xi}_v = \gamma_v^{-1}(t) \left(\tilde{M}^{-1}(q) \left[G(q)u - \tilde{C}(q, \dot{q})v_o - \tilde{g}(q, \dot{q}) - \tilde{w}(q, t) \right] - \dot{v}_r(\xi_h, t) - \dot{\gamma}_v(t)\xi_v \right). \quad (2.29b)$$

and substituting for (2.25) and (2.28), becomes,

$$\begin{aligned}\dot{\xi}_h &= f_h(\xi_h, t) \\ &= -g_s \frac{\partial \bar{h}_{\ell}}{\partial x_o} \frac{\partial \bar{h}_{\ell}}{\partial x_o}^{\top} \gamma_h^{-1}(t) \gamma_h^{-1}(t) r_h(\xi_h) \varepsilon_h(\xi_h) + \\ &\quad \gamma_h^{-1}(t) \left(\frac{\partial \bar{h}_{\ell}}{\partial x_o} J_{o_r}^{-1}(x_o) \gamma_v(t) \xi_v + \frac{\partial \bar{h}_{\ell}}{\partial t} - \dot{\gamma}_h(t)\xi_h \right),\end{aligned} \quad (2.30a)$$

$$\begin{aligned}\dot{\xi}_v &= f_v(\xi_h, \xi_v, t) \\ &= \gamma_v^{-1}(t) \left(\tilde{M}^{-1}(q) \left[-g_v \gamma_v^{-1}(t) r_v(\xi_v) \varepsilon_v(\xi_v) - \right. \right. \\ &\quad \left. \left. \tilde{C}(q, \dot{q})v_o - \tilde{h}(q, \dot{q}) - \tilde{w}(q, t) \right] - \dot{v}_r(\xi_h, t) - \dot{\gamma}_v(t)\xi_v \right),\end{aligned} \quad (2.30b)$$

which, by using (2.24)-(2.28), can be written compactly as

$$\dot{\xi} = f(\xi, t) = \begin{bmatrix} f_h(\xi, t) \\ f_v(\xi, t) \end{bmatrix} \quad (2.31)$$

where $\xi = [\xi_h^\top, \xi_v^\top] \in \mathbb{R}^{M_\ell+6}$. Define next the open and nonempty set $\Omega_\xi = \Omega_{\xi_h} \times \Omega_{\xi_v} \subset \mathbb{R}^{M_\ell+6}$ with $\Omega_{\xi_h} = (-1, 1)^{M_\ell}$ and $\Omega_{\xi_v} = (-1, 1)^6$. We now proceed in two steps: We first prove that there exists a maximal solution $\xi : [t_0, \tau_{\max}) \rightarrow \Omega_\xi$ to (2.31) and then show that $\tau_{\max} = \infty$ which then completes the proof. By choosing $\gamma_h(t)$ and $\gamma_v(t)$ as discussed in Steps I-b, II-a, respectively, and owing to Assumptions 1 and 2, we ensure that $\xi(t_0) \in \Omega_\xi$. In view of (2.11), (2.24)-(2.28), and Assumptions 2, 3, one can conclude that $f(\xi, t)$ is continuous in t and locally Lipschitz in ξ over Ω_ξ . Therefore, [51, Theorem 54] dictates the existence of a maximal solution $\xi : [t_0, \tau_{\max}) \rightarrow \Omega_\xi$. Thus,

$$\xi_{h_{\ell,j}}(t) = \frac{\bar{h}_{\ell,j}(x_o, t)}{\gamma_{h_{\ell,j}}(t)} \in (-1, 1), \quad (2.32a)$$

$$\xi_{v_n}(t) = \frac{e_{v_n}(t)}{\gamma_{v_n}(t)} \in (-1, 1), \quad (2.32b)$$

$\forall j \in \{1, \dots, M_\ell\}$, $n \in \{1, \dots, 6\}$, $t \in [t_0, \tau_{\max})$ from which we conclude $\bar{h}_{\ell,j}(x_o, t)$ and $e_{v_n}(t)$ are bounded by $\gamma_{h_{\ell,j}}(t)$ and $\gamma_{v_n}(t)$ respectively, for $j \in \{1, \dots, M_\ell\}$, $n \in \{1, \dots, 6\}$, $t \in [t_0, \tau_{\max})$.

Consider now the positive definite and radially unbounded function

$$V_h(\varepsilon_h) = \frac{1}{2} \varepsilon_h^\top \varepsilon_h$$

Its time derivative reads

$$\dot{V}_h = \varepsilon_h^\top (\xi_h) r_h(\xi_h) \dot{\xi}_h,$$

and, after using (2.30a),

$$\begin{aligned} \dot{V}_h &\leq -g_s \left\| \varepsilon_h^\top (\xi_h) r_h(\xi_h) \gamma_h^{-1}(t) \frac{\partial \bar{h}_\ell}{\partial x_o} \right\|^2 + \\ &\varepsilon_h^\top (\xi_h) r_h(\xi_h) \gamma_h^{-1}(t) \left(\frac{\partial \bar{h}_\ell}{\partial x_o} J_{o_r}^{-1}(x_o) \gamma_v(t) \xi_v + \frac{\partial \bar{h}_\ell}{\partial t} - \dot{\gamma}_h(t) \xi_h \right), \end{aligned}$$

Note next that $\gamma_v(t)$ and $\dot{\gamma}(t)$ are bounded for all $t \geq t_0$, and $\|\xi_h\| \leq \sqrt{M_\ell}$, $\|\xi_v\| \leq \sqrt{6}$, for all $t \in [t_0, \tau_{\max})$ from (2.32). In view of Assumption 3, the boundedness of $\bar{h}_{\ell,j}(x_o, t)$ from (2.32) implies the boundedness of x_o and the boundedness of $\frac{\partial h_\ell(x_o, t)}{\partial x_o}$, $\frac{\partial h_\ell(x_o, t)}{\partial t}$. Moreover, Assumption 2 and the continuity of $J_{o_r}^{-1}$ implies its boundedness. Note that (2.32) imply that the aforementioned bounds are independent of $\tau_{\max}$. Hence, $\dot{V}_h$ reduces to

$$\dot{V}_h \leq -g_s \lambda_{\min} \left(\frac{\partial \bar{h}_\ell}{\partial x_o} \frac{\partial \bar{h}_\ell}{\partial x_o}^\top \right) \left\| \varepsilon_h^\top (\xi_h) r_h(\xi_h) \gamma_h^{-1}(t) \right\|^2 + \left\| \varepsilon_h^\top (\xi_h) r_h(\xi_h) \gamma_h^{-1}(t) \right\| \bar{B}_h,$$

where $\bar{B}_h$ is a constant independent of $\tau_{\max}$, satisfying $\bar{B}_h \geq \left\| \frac{\partial \bar{h}_\ell}{\partial x_o} J_{o_r}^{-1}(x_o) \gamma_v(t) \xi_v + \frac{\partial \bar{h}_\ell}{\partial t} - \dot{\gamma}_h(t) \xi_h \right\|$. Therefore, one concludes that

$\dot{V}_h < 0 \Leftrightarrow \|\varepsilon_h^\top(\xi_h) r_h(\xi_h) \gamma_h^{-1}(t)\| > \frac{\bar{B}_h}{g_s \kappa}$, which, by noting $r_{h_m} > 2$, is equivalent to,

$$\dot{V}_h < 0 \Leftrightarrow \|\varepsilon_h(\xi_h)\| > \frac{\bar{B}_h \max\{\gamma_{h_{\ell,j}}^0\}_{j \in (1, \dots, M_\ell)}}{g_s \kappa}.$$

Therefore, it holds that $\|\varepsilon_h(\xi_h)\| \leq \bar{\varepsilon}_h$, where

$$\bar{\varepsilon}_h = \max \left\{ \|\varepsilon_h(\xi_h(0))\|, \frac{\bar{B}_h \max\{\gamma_{h_{\ell,j}}^0\}_{j \in (1, \dots, M_\ell)}}{g_s \kappa} \right\}$$

$\forall t \in [t_0, \tau_{\max})$, and by invoking the inverse logarithmic function,

$$-1 < \frac{\exp(-\bar{\varepsilon}_h) - 1}{\exp(-\bar{\varepsilon}_h) - 1} = \underline{\xi}_h \leq \xi_{h_{\ell,j}}(t) \leq \bar{\xi}_h = \frac{\exp(\bar{\varepsilon}_h) - 1}{\exp(\bar{\varepsilon}_h) - 1} < 1. \quad (2.33)$$

Hence, $v_r(\xi_h, t)$ in (2.25) and consequently $v_o = \gamma_v(t) \xi_v + v_r(\xi_h, t)$ remain bounded for all $t \in [0, \tau_{\max})$. Proceeding in a similar manner with the function $V_v = \frac{1}{2} \varepsilon_v^\top \varepsilon_v$, we conclude that $\|\varepsilon_v(\xi_v)\| \leq \bar{\varepsilon}_v$, where

$$\bar{\varepsilon}_v = \max \left\{ \|\varepsilon_v(\xi_v(0))\|, \frac{\bar{B}_v \max\{\gamma_{h_n}^0\}_{n \in (1, \dots, 6)}}{g_v \lambda_{\min}(\tilde{M})} \right\}$$

$\forall t \in [t_0, \tau_{\max})$, where $\lambda_{\min}(\cdot)$ denotes the minimum eigenvalue; by invoking the inverse logarithmic function, we obtain

$$-1 < \frac{\exp(-\bar{\varepsilon}_v) - 1}{\exp(-\bar{\varepsilon}_v) - 1} = \underline{\xi}_v \leq \xi_{h_n}(t) \leq \bar{\xi}_v = \frac{\exp(\bar{\varepsilon}_v) - 1}{\exp(\bar{\varepsilon}_v) - 1} < 1, \quad (2.34)$$

for all $n \in \{1, \dots, 6\}$, $t \in [t_0, \tau_{\max})$, and hence the boundedness of the proposed control law (2.28). More details regarding this step are provided in [20, Sec. 4.1]. What remains to be shown is that $\tau_{\max} = \infty$. Towards this end, note from (2.33), (2.34) that $\xi(t) \in \Omega'_\xi$, where Ω'_ξ is a compact subset of Ω_ξ . Therefore, by invoking [51, Prop. C.3.6], we conclude the forward completeness of the solution and $\tau = \infty$.

Remark 3 The satisfaction of $\lambda_{\min} \left(\frac{\partial \bar{h}_\ell}{\partial x_o} \frac{\partial \bar{h}_\ell}{\partial x_o}^\top \right) \geq \kappa > 0$ is related to the number and structure of the STL sub-tasks encoded in $\bar{h}_\ell$. More specifically, it requires the matrix $\frac{\partial \bar{h}_\ell}{\partial x_o} \frac{\partial \bar{h}_\ell}{\partial x_o}^\top \in \mathbb{R}^{M_\ell \times M_\ell}$ to have full rank M_ℓ , which imposes the requirement on the rank of $\frac{\partial \bar{h}_\ell}{\partial x_o} \in \mathbb{R}^{M_\ell \times 6}$ to be M_ℓ . The latter leads to the following necessary conditions. Firstly, the number M_ℓ of sub-tasks must not be larger than six, i.e., $M_\ell \leq 6$. Intuitively, the number of tasks the robotic system can execute simultaneously is restricted by the number of degrees of freedom in the task space, which is

six. The overall number of sub-tasks can be increased by defining $\bar{h}_\ell$ to encode more tasks that do not overlap in time; we omit the formal description, however, for ease of exposition. Secondly, $\frac{\partial \bar{h}_\ell}{\partial x_o}$ must have linearly independent rows, which translates to the requirement of linearly independent sub-tasks encoded in $\bar{h}_\ell$. One can identify tasks that do not satisfy this assumption and remove potential conflicting dependencies a priori. It should be stressed, however, that condition $\lambda_{\min}\left(\frac{\partial \bar{h}_\ell}{\partial x_o} \frac{\partial \bar{h}_\ell}{\partial x_o}^\top\right) \geq \kappa > 0$ is only a sufficient condition for the theoretical analysis of the proposed control algorithm; practically, the latter may guarantee the satisfaction of the STL tasks even if the condition is not satisfied.

Experimental Validation

Now we are prepared to provide some experimental validation to the proposed PPC control under STL tasks. Consider $N = 2$ six degree-of-freedom HEBI-Robotics arms¹ rigidly grasping a wooden box of dimensions $29cm \times 30.5cm \times 17cm$, and weight $2.05kg$, as depicted in Figure 2.1. The robots get the object's position (x_o) and velocity (v_o) from the coupled object-agent kinematics as mentioned in the above Section. The arms lack an on-board computer and thus communicate the states (x_o, v_o) to a central computer that calculates the reference velocity and the control command. The central computer is an eight-core 1.6 GHz CPU with 16 GB of RAM, and the communication frequency is 150Hz. The arms share the load equally i.e. $c_1 = c_2 = 0.5$. The experiments were performed by cancelling an estimate of gravity forces on the arms and object.

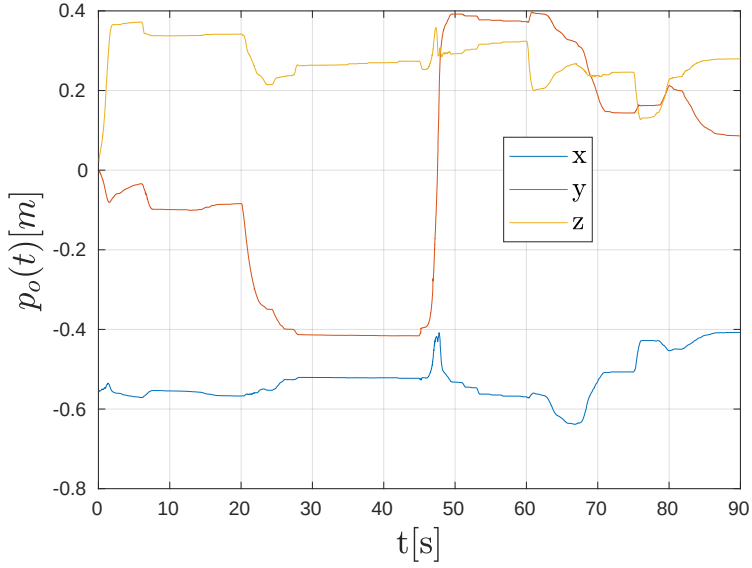
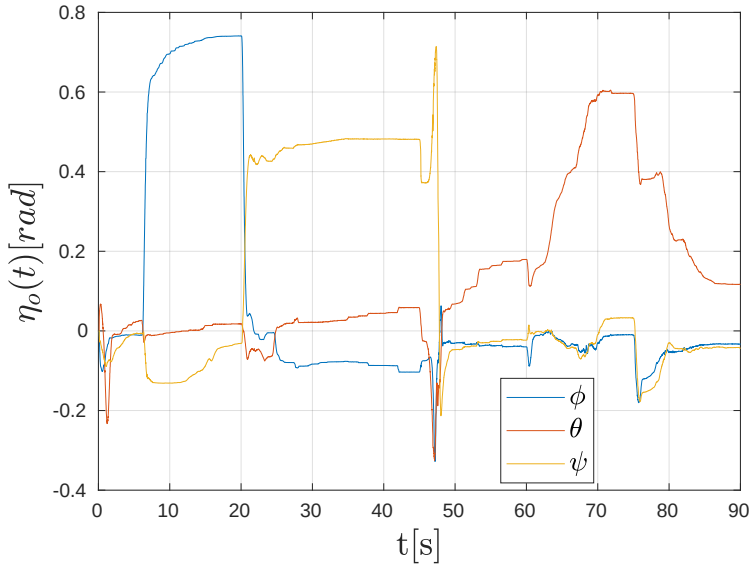
The object's task is encoded in the following formula,

$$\phi := \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5$$

where,

$$\begin{aligned} \phi_1 &= (\|x_o - A\| < 2.2) \mathcal{U}_{[0,20]}(\|x_o - B\| < 4) \\ \phi_2 &= G_{[20,45]}(\|x_o - C\| < 2.5) \\ \phi_3 &= F_{[45,60]}(\|x_o - D\| < 2.2) \\ \phi_4 &= F_{[60,75]}(\|x_o - E\| < 3) \\ \phi_5 &= F_{[75,90]}(\|x_o - F\| < 3), \end{aligned}$$

¹The video of the experiments can be found here: <https://youtu.be/CfoWh5PEYPo>

Figure 2.4: Evolution of the object's position $p_o(t)$.Figure 2.5: Evolution of the Euler angles of the object $\eta_o(t)$.

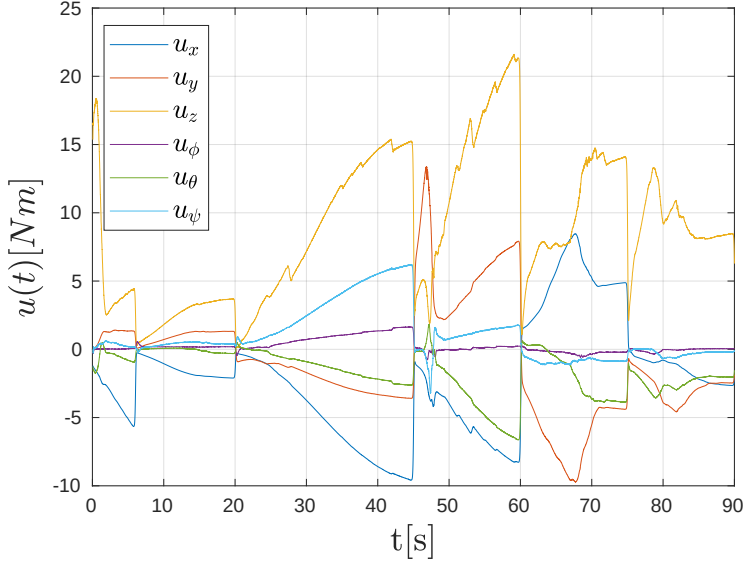


Figure 2.6: Control input to the robots $u(t)$.

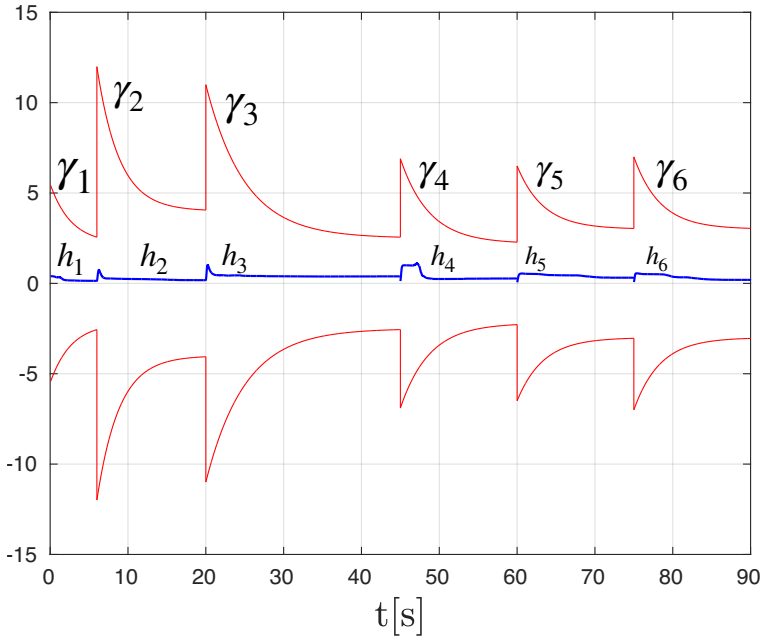


Figure 2.7: Evolution of $\bar{h}_i(x_o, t)$ in blue and performance functions $\gamma_i(t)$ in red for $i = 1, \dots, 6$.

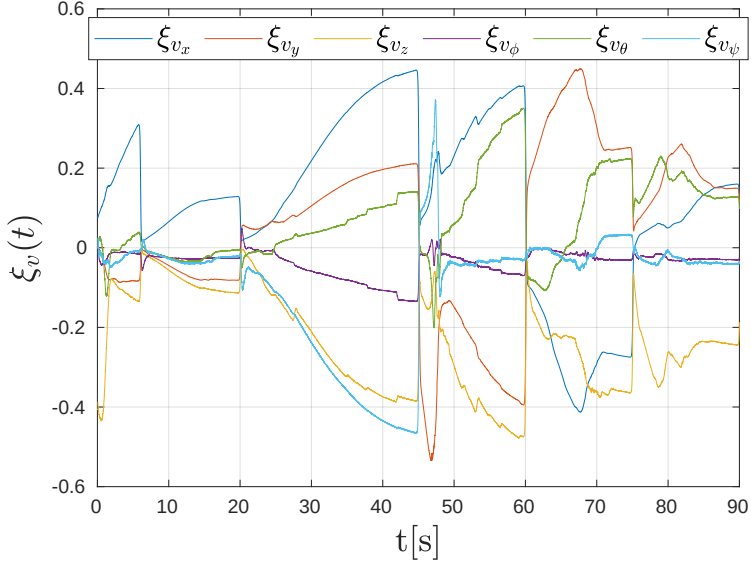


Figure 2.8: Evolution of normalised velocity errors $\xi_v(t)$.

with,

$$\begin{aligned}
 A &= [-0.7, 0, 0.4, 0, 0, 0]^\top \\
 B &= [-0.7, 0, 0.4, \pi/4, 0, 0]^\top \\
 C &= [-0.7, -0.5, 0.35, 0, 0, \pi/4]^\top \\
 D &= [-0.7, 0.5, 0.4, 0, 0, 0]^\top \\
 E &= [-0.35, 0, 0.35, 0, \pi/8, 0]^\top \\
 F &= [-0.5, 0, 0.35, 0, 0, 0]^\top,
 \end{aligned}$$

corresponding to desired setpoints for the object's pose $x_o = [p_o^\top, \eta_o^\top]^\top$. More specifically, ϕ requires the object to assume a sequence of poses in the respective time intervals, as dictated by ϕ_i , $i \in \{1, \dots, 5\}$. The STL tasks are encoded as

follows,

$$\bar{h}(\mathbf{x}_o, t) = \begin{bmatrix} \bar{h}_1(x_o, t) \\ \bar{h}_2(x_o, t) \\ \bar{h}_3(x_o, t) \\ \bar{h}_4(x_o, t) \\ \bar{h}_5(x_o, t) \\ \bar{h}_6(x_o, t) \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{1+\exp(-20(t+0.1))} - \frac{1}{1+\exp(-20(t-6.1))} \right) \|x_o - A\| \\ \left(\frac{1}{1+\exp(-20(t-6.1))} - \frac{1}{1+\exp(-20(t-20.1))} \right) \|x_o - B\| \\ \left(\frac{1}{1+\exp(-20(t-20.1))} - \frac{1}{1+\exp(-20(t-45.1))} \right) \|x_o - C\| \\ \left(\frac{1}{1+\exp(-20(t-45.1))} - \frac{1}{1+\exp(-20(t-60.1))} \right) \|x_o - D\| \\ \left(\frac{1}{1+\exp(-20(t-60.1))} - \frac{1}{1+\exp(-20(t-75.1))} \right) \|x_o - E\| \\ \left(\frac{1}{1+\exp(-20(t-75.1))} - \frac{1}{1+\exp(-20(t-90.1))} \right) \|x_o - F\| \end{bmatrix}.$$

The object is initially fixed at $[-0.7, 0, 0, 0, 0, 0]^\top$. The parameters of the performance functions are chosen as

$$\begin{aligned} \gamma_h^0 &= [5.5 \quad 69.32 \quad 466.5 \quad 8.8 \times 10^5 \quad 2.3 \times 10^8 \quad 2.3 \times 10^{10}], \\ \gamma_h^\infty &= [2.2 \quad 4 \quad 2.5 \quad 2.2 \quad 3 \quad 3], \\ l &= [0.35 \quad 0.35 \quad 0.2 \quad 0.25 \quad 0.3 \quad 0.3], \end{aligned}$$

and the control gains as

$$\begin{aligned} g_s &= \text{diag}\{10, 10, 20, 10, 10, 10\} \\ g_v &= \text{diag}\{5, 5, 10, 5, 5, 5\}. \end{aligned}$$

The experimental results plots are presented here. Figure 2.4, 2.5 show the position and orientation, respectively, of the object executing the STL formula ϕ . Figure 2.6 depicts the control input of one of the robots, which is identical to the second robot's. Figure 2.7 illustrates the evolution of the predicate functions $h_i(\mathbf{x}_o, t)$, along with the respective performance functions that correspond to the 6 sub-formulas in ϕ . One can easily conclude that ϕ is satisfied. Note that, for better visualization, the activation functions and the nonactive $h_i(\mathbf{x}_o, t)$ are not shown in Figure 2.7. Figure 2.8 depicts the evolution of $\xi_v(t)$ which respects $\xi_v \in (-1, 1)$ and is thus inside the prescribed velocity funnels. Compared to existing results [31], the experiments here can handle a richer STL fragment and define tasks to exploit all the 6 degrees-of-freedom of the system. Additionally, the dynamics are assumed to be unknown and we avoid any approximations when considering *disjunctions*.

2.4 Leader-Follower Cooperative Manipulation under STL Constraints

In the Section 2.3 above, we explored cooperative manipulation in a homogenous group of agents where every agent knew the desired satisfactory tasks. Unlike the literature discussed above, here, we work with mobile manipulators, restrict the knowledge of tasks to only a subset of agents and are able to enforce transient constraints without the knowledge of the dynamical terms of the system.

Problem Formulation

The system dynamics are slightly more nuanced in this problem. Assume there are $N = N_f + N_l$ number of agents, where N_f denoted the number of followers that are agnostic to the STL task, and N_l denotes the number of leaders that are aware of the STL task. The system dynamics for the followers are similar to those derived in (2.5), but are now indexed with f for follower agents,

$$\bar{M}_f(q)\dot{v}_f + \bar{C}(q, \dot{q})v_f + \bar{g}_f(q) = u_f - \lambda_f. \quad (2.35)$$

For the leader agents, indexed with l , the dynamics are,

$$\bar{M}_l(q)\dot{v}_l + \bar{C}(q, \dot{q})v_l + \bar{g}_l(q) = u_l - \lambda_l, \quad (2.36)$$

where, where,

- $v = [v_1^\top, \dots, v_{N_{\{f,l\}}}^\top] \in \mathbb{R}^{6N_{\{f,l\}}}$,
- $\bar{M}_{\{f,l\}} = \text{diag}\{\bar{M}_i\} \in \mathbb{R}^{6N_{\{f,l\}} \times 6N_{\{f,l\}}}$,
- $\bar{C}_{\{f,l\}} = \text{diag}\{\bar{C}_i\} \in \mathbb{R}^{6N_{\{f,l\}} \times 6N_{\{f,l\}}}$,
- $\bar{g}_{\{f,l\}} = [\bar{g}_1^\top, \dots, \bar{g}_{N_{\{f,l\}}}^\top]^\top \in \mathbb{R}^{6N_{\{f,l\}}}$,
- $u_{\{f,l\}} = [u_1^\top, \dots, u_{N_{\{f,l\}}}^\top]^\top \in \mathbb{R}^{6N_{\{f,l\}}}$, and,
- $\lambda_{\{f,l\}} = [\lambda_1^\top, \dots, \lambda_{N_{\{f,l\}}}^\top]^\top \in \mathbb{R}^{6N_{\{f,l\}}}$.

The object dynamics remain unchanged from (2.6), but are restated here for clarity,

$$M_o(q)\dot{v}_o + C(q, \dot{q})v_o + g_o(q) = \lambda_o. \quad (2.37)$$

The coupled follower-object dynamics obtained are similar to (2.11), but apply only to the follower agents grasping the object.

$$\tilde{M}_f\dot{v}_o + \tilde{C}_f v_o + \tilde{g}_f = G_f u_f, \quad (2.38)$$

where,

$$\tilde{M}_f = M_o + G_f \bar{M}_f G_f^\top, \quad (2.39)$$

$$\tilde{C}_f = C_o + G_f \bar{M}_f \dot{G}_f^\top + G_f C_f G_f^\top, \quad (2.40)$$

$$\tilde{g}_f = g_o + G_f \bar{g}_f. \quad (2.41)$$

$$(2.42)$$

Finally, the coupled leader-follower-object dynamics are obtained by balancing the forces exerted on the object by the leader and the follower agents i.e., $\lambda_o = G_f u_f + G_l u_l$, resulting in,

$$\tilde{M}(q)\dot{v}_o + \tilde{C}(q, \dot{q})v_o + \tilde{g}(q, \dot{q}) = G_f^\top(q)u_f + G_l^\top(q)u_l \quad (2.43)$$

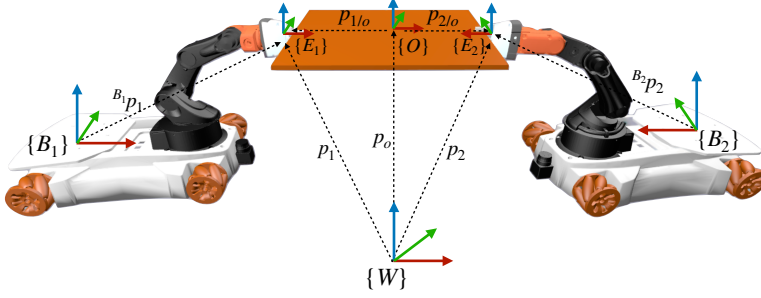


Figure 2.9: Two mobile manipulators grasping an object.

where, $\tilde{M} = \tilde{M}_f + G_l \bar{M}_l G_l^\top$, $\tilde{C} = \tilde{C}_f + G_l \bar{M}_l \dot{G}_l^\top + G_l \bar{C}_l G_l^\top$ and $\tilde{g} = \tilde{g}_f + G_l^\top \bar{g}_l$.

The setup includes having both follower and leader agents as mobile manipulators, see Figure 2.9, consisting of both a mobile base and a manipulator arm. This arrangement provides us with redundant degrees of freedom, allowing for the prescription of additional tasks in the redundant directions. The problem then is to design u_f and u_l , such that the object satisfies a user-defined STL formula. The STL formula we consider is of the same form as (2.16). More formally,

Problem 2 Consider a system comprising $N = N_f + N_l$ agents and a spatio-temporal tasks encoded via an STL formula ϕ . Design a decentralised stiffness controller, $u_f \in \mathbb{R}^{6N_f}$, enabling the follower agents to lift the rigidly grasped object. Concurrently, design a controller $u_l \in \mathbb{R}^{6N_l}$ for the leader agents, ensuring that it can effectively manipulate the object to satisfy ϕ .

Solution Approach

Our solution approach unfolds as follows: We first design a decentralised stiffness controller for each follower agent to compensate for the object's gravitational forces. This controller incorporates adaptive gravity compensation for the object since its mass is unknown to the agents. Additionally, a decentralized controller is not associated with real-time problems, such as channel overload or latency, which can arise over a communication network. Consequently, it enables the follower-object agents system to behave like a mass-spring-damper system, which a leader agent can then manipulate. To this end, we develop a PPC for the leader agent, which is tasked with enforcing spatio-temporal constraints on the object.

- (A) Design an adaptive update law for the followers to compensate for the object's gravity vector g_o .
- (B) Design u_f for the follower agents such that the dynamics (2.38) take the following form,

$$\tilde{M}_f \ddot{v}_o + (\tilde{C}_f + K_v) \dot{v}_o - J_{o_r}^{-1} K_x \Delta x_e = 0 \quad (2.44)$$

where K_x and K_v are the desired stiffness and damping matrices, respectively, and Δx_e denotes a position error to be specified later. This allows the follower-object system to behave like a mass-spring-damper system, allowing us to exert forces on a virtual spring with the desired stiffness and damping properties.

- (C) Develop the implementation of u_f to map to joint torques τ_f , considering the redundant degrees of freedom inherent in the mobile manipulator.
- (D) Design a suitable controller u_l for the leader to impose the desired spatio-temporal tasks on the object.

Controller Design

We follow the above procedure in the same order to solve Problem 2.

(A) *Adaptive Gravity Compensation:*

The gravity vector g_o of the object can be parameterised as [33],

$$g_o = Y_o \gamma_o \quad (2.45)$$

where Y_o is a known regressor matrix and γ_o is the unknown parameter vector. Define the adaptive update law as

$$\dot{\hat{\gamma}}_o = -\Gamma Y_o v_o \quad (2.46)$$

where Γ is a diagonal matrix composed of positive parameters. The control objective, then, is to estimate the required gravity compensation such that,

$$\lim_{t \rightarrow \infty} (\hat{g}_o - g_o) = 0 \quad (2.47)$$

where $\hat{g}_o = Y_o \hat{\gamma}_o$ denotes the estimate of the gravity vector, and $\hat{\gamma}_o$ being the estimate of the unknown physical parameter vector.

(B) *Follower-Only Controller Design:*

We are now ready to design the controller u_f for the follower agents with the goal to shape the dynamics (2.38) into (2.44) in order to stabilise the object's position x_o around a time-varying desired pose x_{o_d} , with the error represented by $\Delta x_e = x_{o_d} - x_o$. We select x_{o_d} as an exponential moving average of x_o , governed by the stable dynamics

$$\tau \dot{x}_{o_d} + x_{o_d} = x_o$$

where τ is a positive time constant. This formulation of the error will allow the follower-object system to function akin to a mass-spring-damper system around the

equilibrium point x_{o_d} . By establishing such a moving average, we aim to ensure that the object stabilises around this point in the absence of external forces applied by the leader agent. Such a low pass filter provides robustness and a degree of compliance to external disturbances. We now design the stiffness-based adaptive controller for the followers as follows:

$$u_f = g_f + G_f^+(Y_o \hat{\gamma}_o + J_{o_r}^{-1} K_x \Delta x_e - K_v v_o) \quad (2.48)$$

where $G_f^+ = [J_{o_1}^{-1}(q_1), \dots, J_{o_N}^{-1}(q_{N_f})] \in \mathbb{R}^{6N_f \times 6}$.

Note that we employ a stiffness-based controller rather than an impedance-based controller. The rationale for this choice stems from the fact that impedance control requires shaping the desired inertia matrix, a task generally avoided due to the complexity of designing and calculating the inertia matrix. Furthermore, impedance control requires measuring the object's acceleration, which tends to be a highly noisy signal or requires measuring external wrenches [37]. Now, we are ready to prove the stability of the follower-object system. This proof is necessary because, when the leader is not manipulating the object, we require the follower-object system to still stabilise around the pose where the leader last released the object.

Theorem 2 *Consider N_f followers rigidly grasping an object with the coupled dynamics (2.38). Then, under the adaptive control law (2.48) and the estimation update law (2.46), the object pose x_o is asymptotically stabilised around the pose x_{o_d} .*

Proof 2 *We first construct the following positive-definite Lyapunov function candidate*

$$V = \frac{1}{2} v_o^\top \tilde{M}_f v_o + \frac{1}{2} \tilde{\gamma}_o^\top \Gamma^{-1} \tilde{\gamma}_o + \frac{1}{2} \Delta x_e^\top K_x \Delta x_e \quad (2.49)$$

where $\tilde{\gamma}_o = \gamma_o - \hat{\gamma}_o$ is the estimation error vector. Taking the time derivative of the above candidate function,

$$\begin{aligned} \dot{V} &= v_o^\top \tilde{M}_f \dot{v}_o + \frac{1}{2} v_o^\top \dot{\tilde{M}}_f v_o + \tilde{\gamma}_o^\top \Gamma^{-1} \dot{\tilde{\gamma}}_o + \Delta x_e^\top K_x \Delta \dot{x}_e \\ &= v_o^\top (G_f u_f - \tilde{C}_f v_o - Y_o \gamma_o - G_f g_f) + \frac{1}{2} v_o^\top \dot{\tilde{M}}_f v_o + \tilde{\gamma}_o^\top \Gamma^{-1} \dot{\tilde{\gamma}}_o + \Delta x_e^\top K_x \Delta \dot{x}_e \\ &= v_o^\top (G_f u_f - Y_o \gamma_o - G_f g_f) + \frac{1}{2} v_o^\top (\dot{\tilde{M}}_f - 2\tilde{C}_f) v_o + \tilde{\gamma}_o^\top \Gamma^{-1} \dot{\tilde{\gamma}}_o + \Delta x_e^\top K_x \Delta \dot{x}_e \end{aligned}$$

Now note that, $\dot{\tilde{\gamma}}_o = \dot{\gamma}_o$, $\Delta \dot{x}_e = \dot{x}_{o_d} - \dot{x}_o = -\frac{\Delta x_e}{\tau} - J_{o_r}^{-1} v_o$, and, $\dot{\tilde{M}}_f - 2\tilde{C}_f$ is skew-symmetric [34]. Hence,

$$\dot{V} = v_o^\top (G_f u_f - Y_o \gamma_o - G_f g_f - J_{o_r}^{-1} K_x \Delta x_e) + \tilde{\gamma}_o^\top \Gamma^{-1} \dot{\tilde{\gamma}}_o - \Delta x_e^\top \frac{K_x}{\tau} \Delta x_e$$

Furthermore, using (2.48) and (2.46) in the above equation, we obtain the bound as follows,

$$\begin{aligned}\dot{V} &= v_o^\top (Y_o \hat{\gamma}_o - Y_o \gamma_o) + \hat{\gamma}_o^\top Y_o v_o - v_o^\top K_v v_o - \Delta x_e^\top \frac{K_x}{\tau} \Delta x_e \\ &= -v_o^\top K_v v_o - \Delta x_e^\top \frac{K_x}{\tau} \Delta x_e \leq 0.\end{aligned}$$

The above function is negative semi-definite and vanishes if and only if $v_o = 0$ and $\Delta x_e = 0$. By applying the Lasalle invariance principle [52], the theorem is proved.

The controller (2.48) is decentralised due to the grasp matrix term G_f^+ , and follower each agent i implements its corresponding

$$u_{f_i} = g_{f_i} + J_{o_i}^{-1} (Y_o \hat{\gamma}_o + J_{o_r}^{-1} K_x \Delta x_e - K_v v_o).$$

Also, observe that we can choose small gains for K_x and K_v as we do not enforce strict trajectory tracking requirements with respect to x_{o_d} . Other approaches, such as solely performing gravity compensation on the object, could result in the object moving indefinitely once the leader releases it. Alternatively, defining a static setpoint might force the object to return to this fixed point after manipulation, which undermines the task being performed. A more natural choice is to allow the object to stabilize around the point where the leader last released it; this rationale motivates the design of the desired position as an exponential moving average. By doing this, we also enforce a certain degree of compliance in the system. Additionally, in the simulations, we observed that the adaptive estimate of the object's mass, $\hat{\gamma}_o = \hat{m}_o$, actually converges to $1/N_f$ -th of the original estimate. This makes intuitive sense as we expect all the N_f followers to share the load equally; proving this relationship is part of the future work. Next, we will explore how to implement such a controller on mobile manipulators and also how to exploit the redundant degrees of freedom natural to such robots.

(C) Implementation of Follower Controller:

Here, we demonstrate how to distribute the input wrench into torques for the mobile base and the manipulator. Define ${}^{B_i}p_i \in \mathbb{R}^6$ (see Figure 2.9) as the pose vector of the end-effector $\{E_i\}$ in the frame of the mobile base $\{B_i\}$. Partition the joint-space states into those pertaining to the mobile base and those belonging to the manipulator. Denote $n_i = n_i^b + n_i^m$, where n_i^b represents the degrees of freedom of the mobile base, and n_i^m represents the degrees of freedom of the manipulator. The torques $\tau_i = [(\tau_i^b)^\top, (\tau_i^m)^\top]^\top$ and the Jacobian $J_i = [J_i^b, J_i^m]$ are partitioned into components associated with the base and the manipulator, respectively. To utilize the redundant degrees of freedom provided by the mobile manipulator, we employ the null-space projection of the Jacobian [53]. Through the null space, we facilitate the motion of the base without affecting the motion of the end-effector. The intuition is to enable the base to follow the object by detecting changes only in the ${}^{B_i}p_i$

vector. By implementing such a strategy, the entire follower-object structure would detect changes in the position of the end-effector and compensate individually by generating motions in the null space. The implementation we propose is,

$$\tau_i = J_i^\top u_i + (I - J_i^+ J_i) \begin{bmatrix} \tau_{b_i} \\ \mathbf{0}_{n_i^m \times 1} \end{bmatrix}$$

where J_i^+ is the pseudo-inverse and $\tau_{b_i} \in \mathbb{R}^{n_i^b}$ is the mobile base controller given by,

$$\tau_{b_i} = -J_i^b K_{b_x} ({}^{B_i}p_i^d - {}^{B_i}p_i).$$

Here, K_{b_x} is a positive-definite gain matrix and ${}^{B_i}p_i^d$ represents the desired pose of the end-effector. In this work, we have chosen this vector to correspond to the pose at the initial grasp. This approach ensures that the base tracks the movement of the object in response to any external forces applied to the end-effector, allowing for the coordinated movement of the entire follower-object assembly through this localized implementation of the base controller.

(D) Leader Controller Design:

The leader is primarily responsible for manipulating the object such that it satisfies the given spatio-temporal task encoded by ϕ . An efficient way to achieve this is by employing PPC like we did in Section 2.3. PPC has the following advantages which we exploit in the design of our algorithm:

- First, it allows us to impose prescribed, user-defined transient and steady-state responses allowing us to incorporate constraints both in state and time on the evolution of the system. This is a very useful property since it allows imposition of a multitude of user-defined performance requirements on system evolution;
- Second, the control design does not make use of dynamical terms such as $\tilde{M}(q)$, $\tilde{C}(q, \dot{q})$, and $\tilde{g}(q, \dot{q})$ but only requires the positive definiteness of $\tilde{M}$. This is beneficial since the controller may be susceptible to model uncertainties and external disturbances. Moreover, accurately determining such matrices can be a challenge for real-world manipulation systems. Most relevant works either consider known dynamics or uncertainties bounded with known bounds or parametric uncertainties;
- Compared to methods such as tube-based Model Predictive Control(MPC) [54] which requires the need of solving online optimization problems, PPC comes with minimal computational burden. PPC along with its robustness properties and ease of implementation is a suitable design method for such systems;
- Finally, it provides us means to design decentralised control laws by distributing performance requirements among N agents.

For the exposition below, we focus on the scenario involving a single leader, i.e., $N_l = 1$. However, the findings are extendable to scenarios in which multiple leaders cooperate with each other.

Using (2.48), the leader-follower-object dynamics (2.43) takes the form,

$$\tilde{M}\dot{v}_o + (\tilde{C} + K_v)v_o + (g_o - Y_o\hat{\gamma}_o) + G_l^\top g_l - J_{o_r}^{-1}K_x\Delta x_e = G_l^\top u_l. \quad (2.50)$$

The goal now is to design the leader controller u_l so that a desired spatio-temporal task on the object is enforced by achieving the PPC specifications.

Define the position error as $e_x(t) = x_o - x_d$ where x_d is the desired trajectory that arises from the STL task (see Section 2.3) to be tracked by the object and $x_o = [p_o^\top, \eta^\top]^\top \equiv [px, py, pz, \phi, \theta, \psi]^\top$ is the object position. We now provide the control design.

Step I

The the position error $e_x \in \mathbb{R}^6$ is,

$$e_x = \begin{bmatrix} e_{px} \\ \vdots \\ e_{\psi} \end{bmatrix} = x_o - x_d \quad (2.51)$$

and the position performance functions $\gamma_x(t) = \text{diag}\{\gamma_{px}, \gamma_{py}, \gamma_{pz}, \gamma_{\phi}, \gamma_{\theta}, \gamma_{\psi}\}$ and $\gamma_x(t) = (\gamma_x^0 - \gamma_x^\infty) \exp(-lt) + \gamma_x^\infty$ with $\gamma_x^0 > |e_x(0)|$. Choose $l \in \mathbb{R}_+$ and γ_x^∞ such that the desired spatio-temporal task is satisfied if $-\gamma_x(t) < e_x(t) < \gamma_x(t)$ holds. Define the normalised errors $\xi_x \in \mathbb{R}^6$ by,

$$\xi_x = \begin{bmatrix} \xi_{px} \\ \vdots \\ \xi_{\psi} \end{bmatrix} = \gamma_x^{-1}(t)e_x(t) \quad (2.52)$$

and design the reference velocity as

$$v_r(t) = -c_1 J_{o_r} \gamma_x^{-1}(t) r_x(\xi_x) \varepsilon_x(\xi_x) \quad (2.53)$$

where c_1 is a positive constant and the signals $\varepsilon_x : (-1, 1)^6 \rightarrow \mathbb{R}^6$ and $r_x : (-1, 1)^6 \rightarrow \mathbb{R}^{6 \times 6}$ are $\varepsilon_x(\xi_x) = [\varepsilon_{px}, \dots, \varepsilon_{\psi}]$ and $r_x(\xi_x) = \text{diag}([r_i]_{i \in \{px, \dots, \psi\}})$, with,

$$\varepsilon_{x_i} = \ln \left(\frac{1 + \xi_{x_i}}{1 - \xi_{x_i}} \right), \quad r_{x_i} = \frac{2}{1 - \xi_{x_i}^2}, \quad i \in \{px, py, pz, \phi, \theta, \psi\}.$$

Step II

Define the velocity error $e_v \in \mathbb{R}^6$ as,

$$e_v = \begin{bmatrix} e_{vx} \\ \vdots \\ e_{\omega_z} \end{bmatrix} = v_o - v_r \quad (2.54)$$

where $v_o = [\dot{p}_o^\top, \omega_o^\top]^\top \equiv [vx, vy, vz, \omega_x, \omega_y, \omega_z]^\top$. Let the velocity performance functions $\gamma_v(t) = \text{diag}(\gamma_{vx}, \gamma_{vy}, \gamma_{vz}, \gamma_{\omega_x}, \gamma_{\omega_y}, \gamma_{\omega_z})$ and $\gamma_v(t) = (\gamma_v^0 - \gamma_v^\infty) \exp(-lt) + \gamma_v^\infty$ with $\gamma_v^0 > |e_v(0)|$. Choose $l \in \mathbb{R}_+$ and γ_v^∞ such that the desired velocity constraints are satisfied if $-\gamma_v(t) < e_v(t) < \gamma_v(t)$ holds.

Step III

Define the normalised errors $\xi_v \in \mathbb{R}^6$ by,

$$\xi_v = \begin{bmatrix} \xi_{vx} \\ \vdots \\ \xi_{\omega_z} \end{bmatrix} = \gamma_v^{-1}(t) e_v(t) \quad (2.55)$$

and design the control law in (2.50) as

$$u_l(t) = -c_2 J_{o_l}^{-1} \gamma_v^{-1}(t) r_v(\xi_x) \varepsilon_v(\xi_v) \quad (2.56)$$

where J_{o_l} is the object to leader Jacobian, c_2 is a positive constant and the signals $\varepsilon_v : (-1, 1)^6 \rightarrow \mathbb{R}^6$ and $r_v : (-1, 1)^6 \rightarrow \mathbb{R}^{6 \times 6}$ are $\varepsilon_v(\xi_v) = [\varepsilon_{vx}, \dots, \varepsilon_{\omega_z}]$ and $r_v(\xi_v) = \text{diag}([r_i]_{i \in \{vx, \dots, \omega_z\}})$, with,

$$\varepsilon_{v_i} = \ln \left(\frac{1 + \xi_{v_i}}{1 - \xi_{v_i}} \right), r_{v_i} = \frac{2}{1 - \xi_{v_i}^2}, i \in \{vx, vy, vz, \omega_x, \omega_y, \omega_z\}.$$

Before we demonstrate that the proposed control scheme ensures the containment of trajectories within the designated performance functions, we will state the following assumption.

Assumption 4 *The object's pose does not result in a singular $J_{o_r}(x_o(t))$, i.e. $\theta_o(t) \in [-\theta, \theta]$, for all $t \geq 0$, with $\theta \in (0, \frac{\pi}{2})$.*

Theorem 3 *Consider a leader-follower system rigidly grasping an object with coupled dynamics (2.50) and subject to spatio-temporal tasks requiring x_o to track a desired trajectory x_d under timing constraints. Then the controller (2.56) guarantees $-\gamma_x(t) < e_x(t) < \gamma_x(t)$ for all $t \geq 0$, and the boundedness of all closed loop signals.*

Proof 3 *The proof follows along the lines of Proof of Theorem 1.*

Above, we developed a controller for a leader agent aimed at fulfilling the spatio-temporal tasks associated with the object. This controller is not required when a leader agent is substituted by a human operating the object. The strength of the framework developed herein lies in the decoupling of the follower-object system from the leader system. Consequently, the leader can be any entity that possesses knowledge of the tasks imposed on the object being carried.

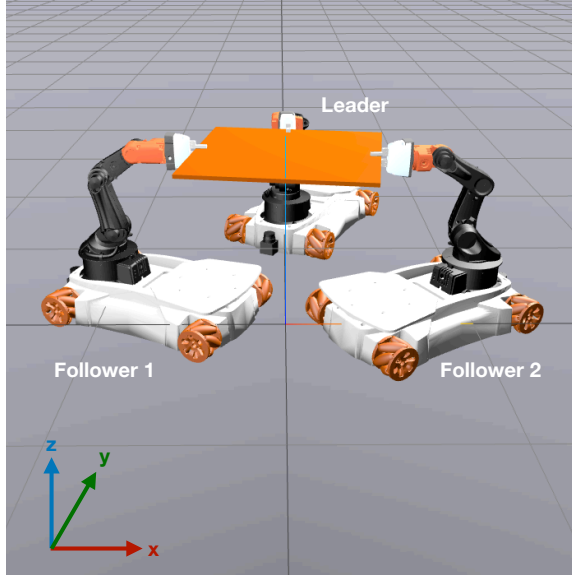


Figure 2.10: Setup of 3 YouBots grasping an object.

Simulation Validation

We conduct computer simulations using the realistic environment created by the Drake toolbox [2]. The mobile manipulators under consideration are the KUKA YouBots, which feature an omnidirectional mobile base and a 5-degree-of-freedom manipulator arm equipped with a finger gripper. Our simulation setup includes two follower agents and one leader agent, along with a slab (of mass $m_o = 0.05[\text{kg}]$) representing the object, as shown in Figure 2.10. Let $x_o(t) = [p_o^\top, \eta_o^\top]^\top$ denote the pose of the object's centre of mass, and the initial conditions are $x_o(0) = [0, 0, 0.4, 0, 0, 0]^\top$. The STL formula we impose on the object here is,

$$\begin{aligned}
 \varphi = & \mathcal{F}_{[0,55]} \left(\|p_o - [1, 1, 0.4]^\top\| \leq 3 \right) \wedge \\
 & \mathcal{F}_{[55,105]} \left(\|p_o - [0, 1.414, 0.4]^\top\| \leq 3 \right) \wedge \\
 & \mathcal{F}_{[105,155]} \left(\|p_o - [-1, 1, 0.4]^\top\| \leq 3 \right) \wedge \\
 & \mathcal{F}_{[155,205]} \left(\|p_o - [-1.414, 0, 0.4]^\top\| \leq 3 \right) \wedge \\
 & \mathcal{F}_{[205,255]} \left(\|p_o - [0, 0, 0.4]^\top\| \leq 3 \right) \wedge \\
 & \mathcal{G}_{[255,350]} \left(\|p_o - [0, 0, 0.4]^\top\| \leq 3 \right).
 \end{aligned}$$

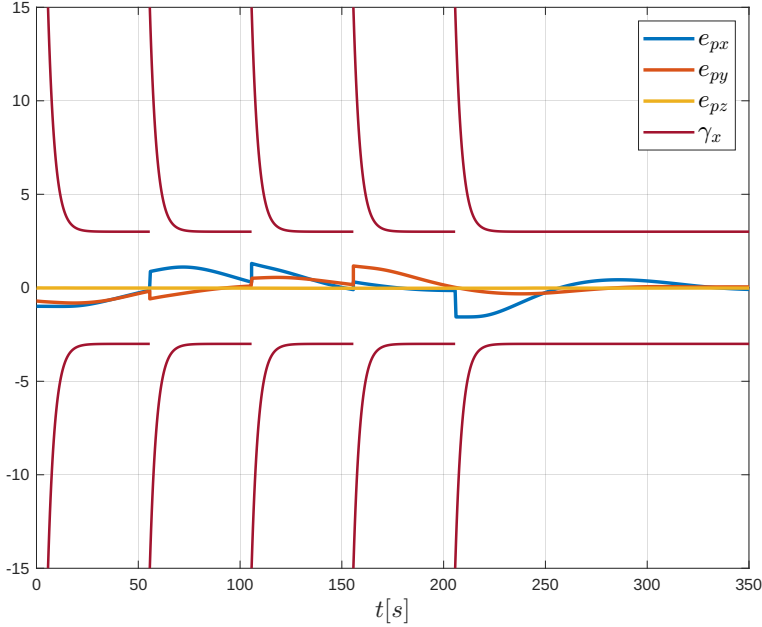


Figure 2.11: Evolution of (2.51) along with the corresponding funnels.

Based on our Section 2.3, the STL formula above can be transformed into the following task explanation:

- The desired position p_o of the object should be $[1, 1, 0.4][m]$ at $t_1 = 55[s]$, (see Figure 2.15a),
- Next, the object's desired position should be $[0, 1.414, 0.4][m]$ at $t_2 = 105[s]$, (see Figure 2.15b),
- Next, the object's desired position should be $[-1, 1, 0.4][m]$ at $t_3 = 155[s]$, (see Figure 2.15c),
- Next, the object's desired position should be $[-1.414, 0, 0.4][m]$ at $t_4 = 205[s]$, (see Figure 2.15d),
- And finally, the object should return to the initial conditions at $t_5 = 255[s]$ (see Figure 2.15e) and stay there until $t_6 = 350[s]$.

Only the leader is aware of the aforementioned tasks. The error was defined as $e(t) = p_o - p_{o_d}$ where p_{o_d} is the desired position mentioned above. The position and velocity performance functions were chosen as,

$$\gamma(t) = (15 - 3) \exp(-0.3(t - t^*)) + 3$$

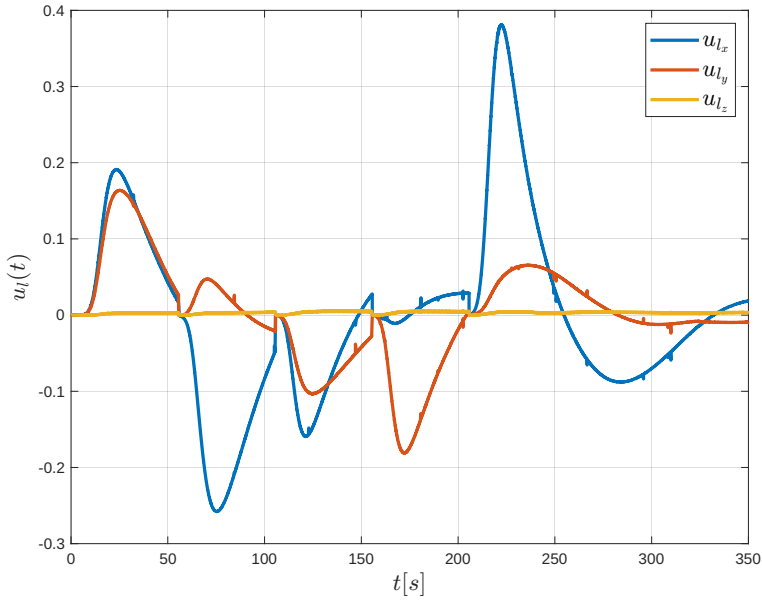
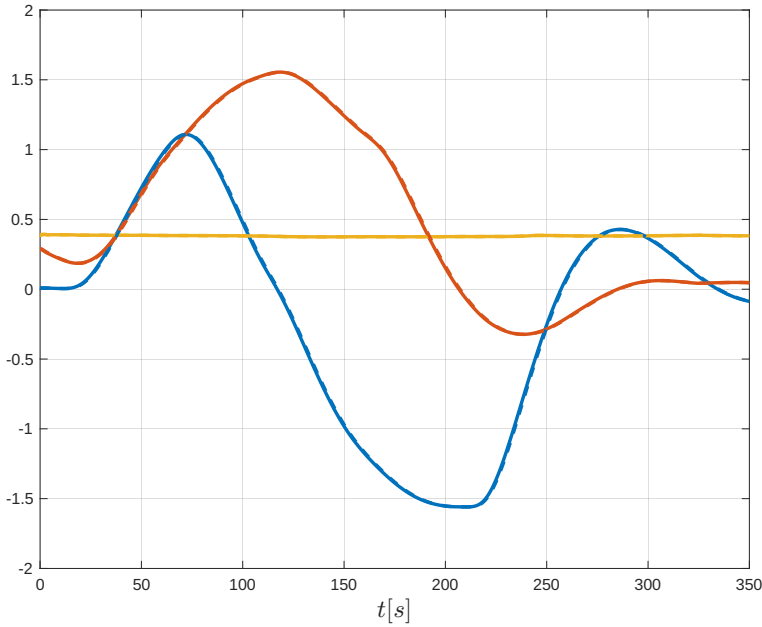


Figure 2.12: Leader control input from (2.50).

Figure 2.13: Evolution of x_o and x_{od} .

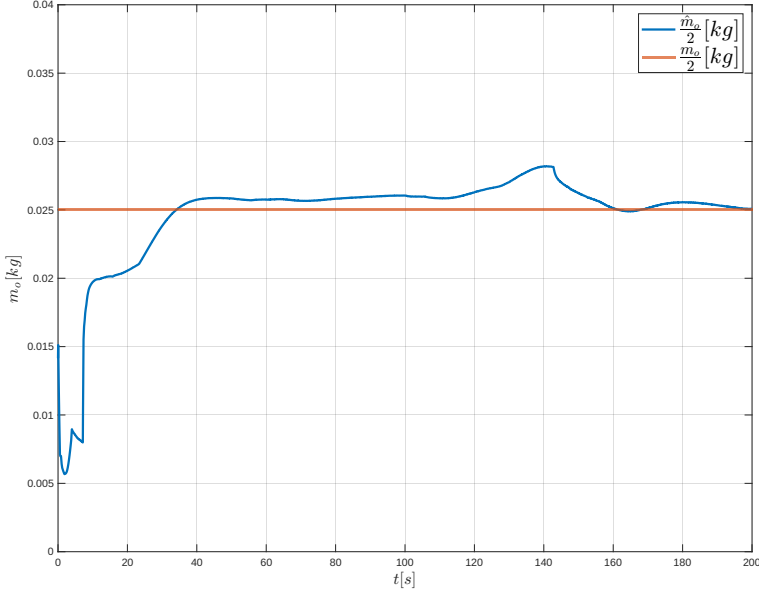


Figure 2.14: Estimated effort by follower 1 using (2.46).

where $t^* \in \{0, 55, 105, 155, 205\}[\text{s}]$. The results are presented in Figure 2.11-2.14, and snapshots of the simulation environment are depicted in Figure 2.15. In Figure 2.11, we observe the five funnels that enforce the desired transient constraints on the object's position error. It is noted that the errors are contained within the funnels. Figure 2.12 illustrates the leader control input (2.56), while Figure 2.13 displays the object's position as a solid line and the exponential moving average as a dotted line. As can be seen, x_{o_d} closely follows the original position x_o . In Figure 2.14, the follower agents implement the adaptive law (2.14) to estimate the mass of the object. As can be seen, the mass of the object converges to $0.025[\text{kg}]$ which is half of the original mass. This is due to the fact that there are two follower agents that share the load of the object equally and, hence, arrive at this estimate.

2.5 Conclusion

In Section 2.3, we addressed the cooperative manipulation problem under STL tasks for systems with uncertain dynamics. The main contribution is the design of a control mechanism that guarantees the satisfaction of an STL-encoded task for a cooperatively manipulated object. First, we translated the problem of STL satisfaction into one of ensuring state evolution within a time-varying funnel. We then employed the Prescribed Performance Control (PPC) methodology to guarantee compliance with this funnel and, consequently, satisfaction of the task. The con-

trol mechanism exhibits the following attributes. First, it is decentralised, meaning each agent computes its own control signal without requiring communication with the rest of the team. Second, it does not rely on any knowledge of the dynamics of the robotic agents or the object, nor does it require estimation or approximation models. Third, it operates without force/torque sensors at the grasping points between the agents and the object. Finally, we avoid approximations in the robustness metrics of STL and consider a richer STL fragment compared to previous works.

In Section 2.4, we addressed an even more challenging problem: how to solve Problem 1 when not all agents are aware of the STL task. We presented a decentralised control algorithm for cooperative manipulation within a leader-follower framework. We demonstrated the decoupling of the primary objective—fulfilling spatio-temporal tasks—from the secondary objective of carrying the object. The primary task is executed by the leader agent using a PPC controller, while the follower agents focus solely on compensating for the object’s gravitational force. We further validated the proposed framework through simulations involving KUKA YouBot robots.

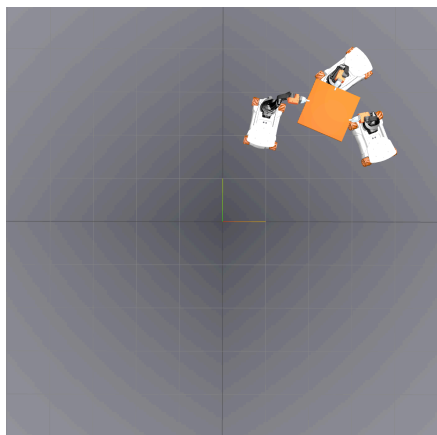
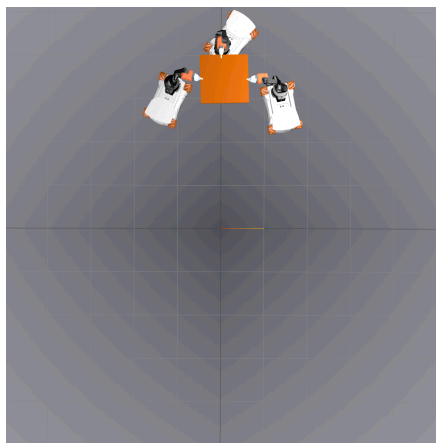
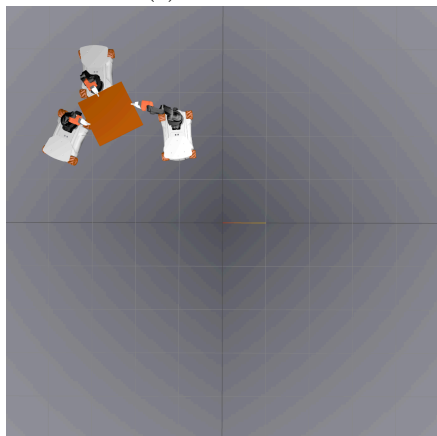
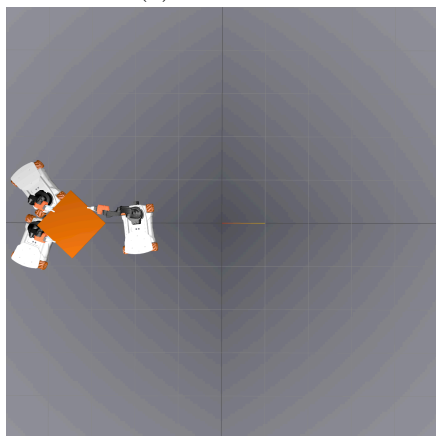
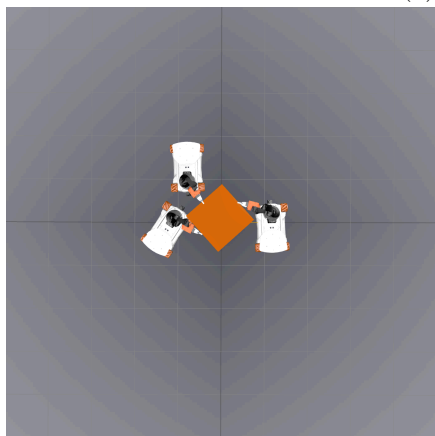
(a) At $t = 55s$ (b) At $t = 105s$ (c) At $t = 155s$ (d) At $t = 205s$ (e) At $t = 255s$

Figure 2.15: Snapshots of the simulation environment at various timestamps.

Chapter 3

Geometric Path-Planning for Multi-Manipulator Systems

In Chapter 2, we asked the question: given a restricted fragment of STL, what can we achieve in terms of control design? The limitations in STL expressiveness were directly tied to the guarantees we could provide through our control framework. But what happens if we relax that constraint? What if we no longer require strong theoretical guarantees, and instead are satisfied with probabilistic guarantees, provided we can reason with more expressive STL formulas? That is the central question of this chapter.

Here, we explore the trade-off between expressiveness and guarantees, and shift our focus towards geometric path planning problems. Given an arbitrary STL formula, can we find a signal, or a trajectory, that satisfies it? Since STL allows reasoning over continuous-time, continuous-space signals, we are not restricted to interpreting these signals solely as robot states. In fact, this chapter adopts an even more general perspective: can we find any signal that satisfies a given STL formula? We approach this problem in two parts:

First, we develop a cooperative sampling-based motion planning algorithm for two autonomous agents under coupled tasks expressed as STL constraints. The algorithm builds incrementally two spatio-temporal trees, one for each agent, by sampling points in an extended space, which consists of a compact subset of the time domain and the physical space of the agents. The trees are built by checking if newly sampled points form edges in time and space that satisfy certain parts of the coupled task. Therefore, the constructed trees represent time-varying trajectories in the agents' state space that satisfy the task. The algorithm is distributed in the sense that the agents build their trees individually by communicating with each other. The proposed algorithm inherits the properties of probabilistic completeness and computational efficiency of the original sampling-based procedures.

Second, we present MAPS²: a distributed algorithm that allows multi-robot systems to deliver coupled tasks expressed as STL constraints. Classical control

theoretical tools addressing STL constraints either adopt a limited fragment of the STL formula or require approximations of min/max operators. Meanwhile, works maximising robustness through optimisation-based methods often suffer from local minima, thus relaxing any completeness arguments due to the NP-hard nature of the problem. Endowed with probabilistic guarantees, MAPS² provides an autonomous algorithm that iteratively improves the robots' trajectories. The algorithm selectively imposes spatial constraints by taking advantage of the temporal properties of the STL. The algorithm is distributed in the sense that each robot calculates its trajectory by communicating only with its immediate neighbours as defined via a communication graph. We illustrate the efficiency of MAPS² by conducting extensive simulation and experimental studies, verifying the generation of STL satisfying trajectories.

This chapter is based on [55] and [56].

3.1 Introduction

Autonomous robots can address complex problems when guided by a clear set of constraints. These constraints may arise from the robot's physical limitations, such as joint limits, or be defined by human-specified tasks, such as pick-and-place operations. A powerful way to impose such constraints is through logic-based tools, which enable formal reasoning about the robot's intended behaviour. These tools allow us to describe behaviour across multiple levels of abstraction, from interactions among internal components to high-level task execution [57]. Their strong expressivity makes it possible to encode complex mission requirements into logical formulas, offering significant versatility. Unlike simple point-to-point navigation, such logic, such as STL, can capture a wide range of sophisticated planning objectives [58]. Recent research has increasingly focused on using logic-based tools to specify behavioural requirements for robots, followed by the development of algorithms that generate trajectories satisfying those requirements. Such is the focus of this Chapter.

Another important property of autonomous robots is their ability to coordinate and work in teams. The use of multiple robots is often necessary in situations where a single robot is either insufficient, the task is high-energy demanding, or unable to physically perform certain tasks. For instance, a single aerial vehicle tasked with surveying a 10 km² farmland would require multiple battery recharges, regular data transfers and may take an excessive amount of time. In contrast, using multiple aerial vehicles can alleviate some of these issues. However, multi-robot systems present their own set of challenges, such as communication overload, the need for a central authority for commands, and high computational demands. The challenge, therefore, is to derive solutions for multi-robot problems utilising logic-based tools, ensuring the achievement of specified high-level behaviour.

There exist numerous works that consider planning under temporal logic specifications. Such specifications include both qualitative properties, such as linear tem-

poral logic (LTL) [59], and quantitative spatio-temporal properties, such as metric temporal logic (MTL) [60–63], metric interval temporal logic (MITL) [20, 64–66] or signal temporal logic (STL) [22, 23, 31, 67–71]. MTL and MITL usually specify tasks over finite-state spaces, requiring the abstraction of the underlying continuous-time and -state systems to discrete ones. However, such abstractions bring several drawbacks, such as loss of information and risk of state explosion. Using STL avoids the aforementioned drawbacks and allows one to express specifications in continuous time and space. Additionally, as presented before, STL is a good choice as it is accompanied by a robustness metric which allows us to determine the extent of satisfaction compared to only absolute satisfaction [58].

In the domain of single-agent motion planning, different algorithms have been proposed to generate safe paths for robots. Sampling-based algorithms, such as CBF-RRT [72], have achieved success in providing a solution to the motion planning problem in dynamic environments. However, they do not consider high-level complex mission specifications. Works that impose high-level specifications in the form of LTL, such as [19, 73–75], resort to a hybrid hierarchical control regime resulting in abstraction and explosion of state-space. While a mixed integer program can encode this problem for linear systems and linear predicates [76], the resulting algorithm has exponential complexity, making it impractical for high-dimensional systems, complex specifications, and long duration tasks. To address this issue, [77] proposes a more efficient encoding for STL to reduce the exponential complexity in binary variables. Additionally, [78] introduces a new metric, discrete average space robustness, and composes a Model Predictive Control (MPC) cost function for a subset of STL formulas.

Several works consider the multi-agent motion-planning problem under STL tasks [23, 79–83]. The work [23, 79] develops distributed feedback control algorithms based on funnel control and barrier functions to guide robots toward satisfying the underlying STL tasks. The limitations of continuous spaces, however, such as the inability to explore the entire time and state spaces, leads to the consideration of only specific STL fragments, losing the full expressivity of STL. Works that consider the entire STL fragment adopt centralized solutions based on mixed-integer linear programming (MILP) [80–83]. However, MILP programs can be computationally demanding and yield long running times. Furthermore, centralized algorithms do not scale with the number of agents and are sensitive to faults, since a single computer unit plans the actions of all agents. In multi-agent temporal logic control, works such as [20, 84] employ workspace discretisation and abstraction techniques, which we avoid in this article due to it being computationally demanding. Some approaches to STL synthesis involve using mixed-integer linear programming (MILP) to encode constraints, as previously explored in [22, 85, 86]. However, MILPs are computationally intractable when dealing with complex specifications or long-term plans because of the large number of binary variables required in the encoding process. The work in [81] encodes a new specification called multi-agent STL (MA-STL) using mixed integer linear programs (MILP). However, the predicates here depend only on the states of a single agent, can only represent poly-

tope regions, and finally, temporal operations can only be applied to a single agent at a time. In contrast, this work explores coupled constraints between robots and predicates are allowed to be of nonlinear nature.

As a result, researchers have turned to transient control-based approaches such as gradient-based, neural network-based, and control barrier-based methods to provide algorithms to tackle the multi-robot STL satisfaction problem [77]. Such approaches, at the cost of imposing dynamical constraints on the optimisation problem, often resort to using smooth approximations of temporal operators at the expense of completeness arguments or end-up considering only a smaller fragment of the syntax [31, 87–89]. STL’s robust semantics are used to construct cost functions to convert a synthesis problem to an optimisation problem that benefits from gradient-based solutions. However, such approaches result in non-smooth and non-convex problems and solutions are prone to local minima [90].

The general motion-planning problem, even for a two-dimensional configuration space with rectangular obstacles is PSPACE-hard [91], thus, a common approach is sampling-based planners. In such planners, the free space is modeled using a graph with nodes and edges. The motion planning problem then reduces to a graph-search problem where a collision free path is found thereon. By constructing this graph, a multidimensional space search is reduced to finding a path in the graph. Thus, we develop distributed sampling-based planners in this chapter, thereby inheriting their efficiency and probabilistic completeness properties for both cooperating-agent and multi-robot systems. We avoid approximations and consider the full expressivity of the STL syntax. The proposed solution adopts a purely geometric approach to the STL planning problem in both cooperating-agent and multi-robot contexts. Our current focus is on the planning problem itself, specifically the generation of trajectories that fulfil STL constraints, rather than on handling dynamical constraints or designing control techniques for trajectory execution.

3.2 Preliminaries

The preliminaries in this section include an introduction to certain STL concepts, an introduction to RRT — a sampling-based path planner, a graph topology introduction and a brief introduction to distributed optimisation.

STL Parse Tree

In this Chapter, the whole STL syntax is considered and is of the form (1.1) and is written below again for clarity,

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

Each STL formula is valid over a time horizon defined as follows,

Definition 2 ([92]) *The time horizon $\text{th}(\varphi)$ of an STL formula φ is recursively defined as,*

$$\text{th}(\varphi) = \begin{cases} 0, & \text{if } \varphi = \mu \\ \text{th}(\varphi_1), & \text{if } \varphi = \neg\varphi_1 \\ \max\{\text{th}(\varphi_1), \text{th}(\varphi_2)\}, & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ b + \max\{\text{th}(\varphi_1), \text{th}(\varphi_2)\}, & \text{if } \varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2. \end{cases} \quad (3.1)$$

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

Assumption 5 *The STL formula is in positive normal form i.e., it does not contain the negation operator.*

The above assumption does not cause any loss of expression of the STL syntax (1.1). As shown in [86], any STL formula can be written in positive normal form by moving the negation operator to the predicate.

An STL formula defined recursively over the form (1.1) can be represented as a tree data structure, called the STL parse tree. An STL parse tree can be constructed as follows:

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

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

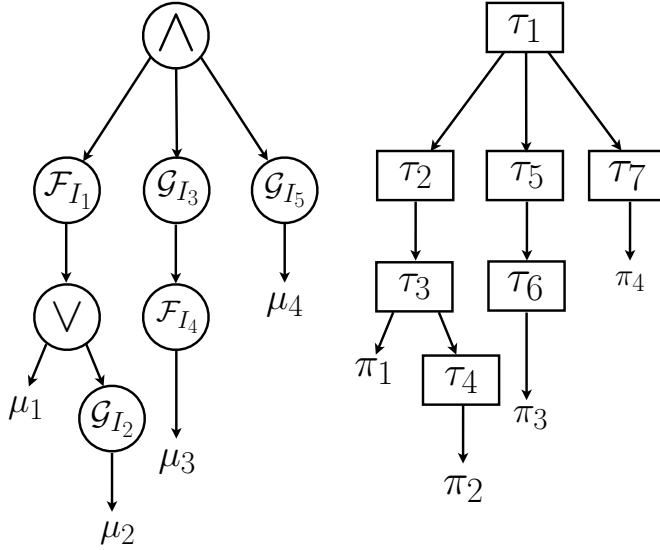


Figure 3.1: STL parse tree and satisfaction variable tree for the formula in (3.2).

maps uniquely to a predicate variable π_i , where i is an enumeration of the nodes in the satisfaction variable tree. An example of construction of such trees is shown below.

Example 3 *The STL parse tree and the satisfaction variable tree for the STL formula*

$$\varphi = \mathcal{F}_{I_1}(\mu^{h_1} \vee \mathcal{G}_{I_2}(\mu^{h_2})) \wedge \mathcal{G}_{I_3} \mathcal{F}_{I_4}(\mu^{h_3}) \wedge \mathcal{G}_{I_5}(\mu^{h_4}). \quad (3.2)$$

are shown in Figure 3.1. From the trees, one obtains the implications $\tau_2 = +1 \implies (\mathbf{x}, t) \models \mathcal{F}_{I_1}(\mu^{h_1} \vee \mathcal{G}_{I_2}(\mu^{h_2}))$, and $\tau_7 = +1 \implies (\mathbf{x}, t) \models \mathcal{G}_{I_5}(\mu^{h_4})$.

Rapidly Exploring Random Tree (RRT)

A Rapidly-exploring Random Tree (RRT), proposed in [93], is path planning algorithm designed to efficiently search for paths in non-convex, high-dimensional spaces. Beginning from the initial configuration (x_0), the tree progressively expands towards points randomly selected (x_{samp}) from the configuration space $\mathcal{C}$. For each sample, the algorithm identifies the nearest existing vertex (x_{nearest}) within the tree and attempts to establish a connection. If this connection is valid i.e., traversing collision free regions and respecting all system constraints, then the sample is added as a new vertex in the tree. Due to uniform random sampling, vertices with

larger associated Voronoi regions are more likely to be expanded. Consequently, the tree exhibits an inherent tendency to extend towards unexplored regions of the configuration space.

To control the rate of expansion, a maximum extension distance is typically enforced (*step*). When a sampled point lies beyond this threshold from the nearest node, the algorithm instead advances the tree by the maximum permitted distance in the direction of the sample. This mechanism ensures controlled, incremental growth while maintaining the exploratory character of the tree, with random samples dictating the expansion direction and the growth limit governing step size.

Algorithm 1: RRT

Input: $x_0, K, step$

Output: $\mathcal{T}$

```

1  $\mathcal{V} \leftarrow \{x_0\};$ 
2  $\mathcal{E} \leftarrow \emptyset;$ 
3  $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E});$ 
4 for  $k = 1$  to  $K$  do
5    $x_{\text{samp}} \leftarrow \text{Sample}(\mathcal{C});$ 
6    $x_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}, x_{\text{samp}});$ 
7    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{samp}}, x_{\text{nearest}}, step);$ 
8    $\mathcal{V} \leftarrow \mathcal{V} \cup x_{\text{new}};$ 
9    $\mathcal{E} \leftarrow \mathcal{E} \cup \{x_{\text{nearest}}, x_{\text{new}}\};$ 
10 return  $\mathcal{T};$ 

```

The pseudocode for RRT is presented in Algorithm 1.

- **Sample**($\mathcal{C}$): Generates a random sample from the configuration space $\mathcal{C}$.
- **Nearest**($\mathcal{T}, x_{\text{samp}}$): Iterates over all vertices v in the set $\mathcal{V}$ of tree $\mathcal{T}$, computes the distance between x_{samp} and each v using a specified distance function, and returns the nearest vertex x_{nearest} .
- **Steer**($x_{\text{samp}}, x_{\text{nearest}}, step$): Determines a new vertex x_{new} in the direction of x_{samp} from x_{nearest} , such that the distance between x_{nearest} and x_{new} does not exceed the value of *step*.

Figure 3.2 illustrates two example runs of the RRT algorithm in the presence of obstacles.

Graph Topology

A graph — specifically, a finite, undirected, simple graph as defined in [94] — is composed of a finite set of elements known as the vertex set, denoted by $\mathcal{V}$. The elements of $\mathcal{V}$ are referred to as vertices or nodes. The relationships between these

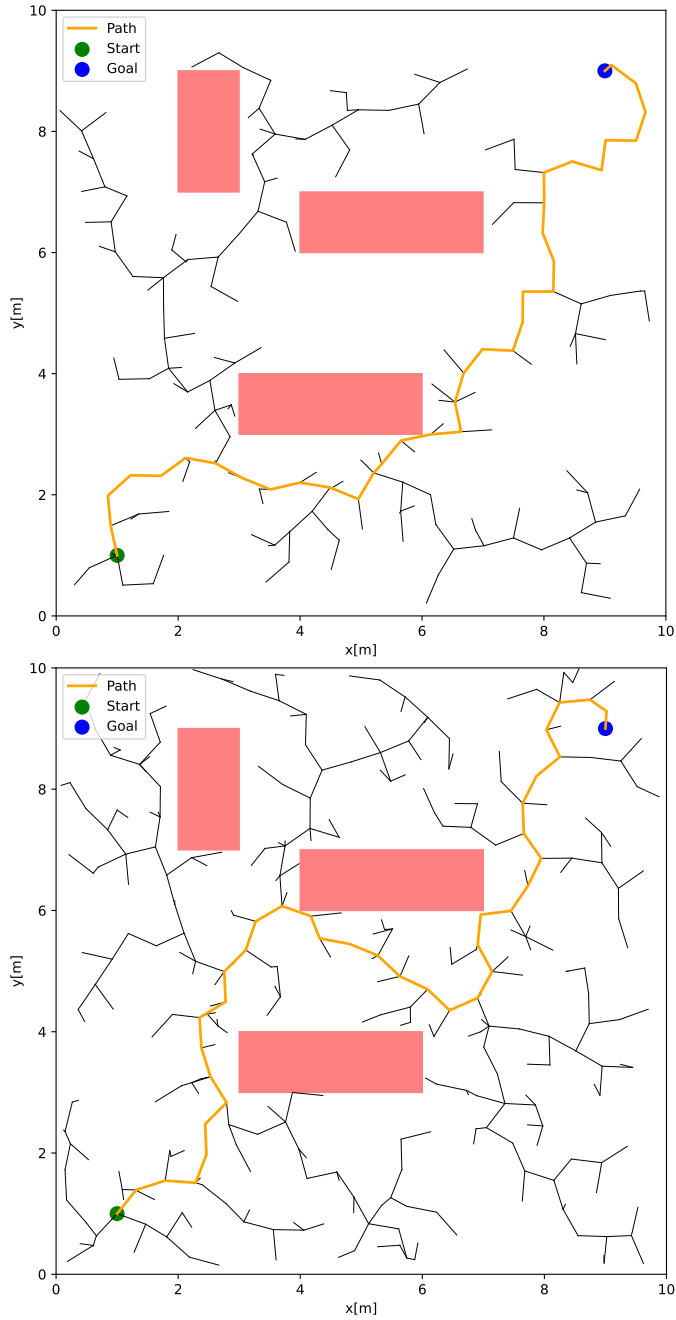


Figure 3.2: Examples of RRT algorithm.

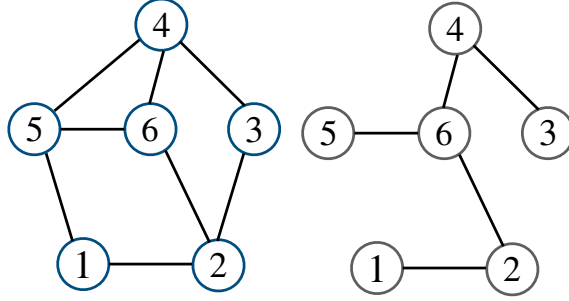


Figure 3.3: Two graphs representing different communication topologies.

vertices are captured by the edge set, $\mathcal{E}$, which is a subset of the Cartesian product $\mathcal{V} \times \mathcal{V}$. Each element of $\mathcal{E}$ represents a connection between two vertices. Formally, a graph is expressed as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. A graph is said to be connected if a path exists between every pair of vertices. In this context, a path refers to a sequence of edges that links one vertex to another.

Example 4 Consider a graph with vertex set $\mathcal{V} = \{1, 2, 3, 4, 5, 6\}$ and two distinct edge sets: $\mathcal{E}_1 = \{(1, 2), (1, 5), (2, 3), (2, 6), (3, 4), (4, 5), (4, 6), (5, 6)\}$ (left) and $\mathcal{E}_2 = \{(1, 2), (2, 6), (3, 4), (4, 6), (5, 6)\}$ (right), as shown in Figure 3.3. In both cases, the graphs are connected, as there exists a path between every pair of vertices.

In this chapter, we will clearly distinguish between the use of graphs and trees to avoid confusion. The term graph refers specifically to the communication network between robots, where nodes represent individual robots and edges represent communication links between them. In contrast, the term tree refers to data structures used in motion planning algorithms, such as RRT, where the tree is grown incrementally in the configuration space. In this context, vertices represent possible robot configurations, and edges represent feasible motion between these configurations. Throughout the chapter, graph terminology will be reserved exclusively for describing robot communication networks, while tree terminology will be used when discussing path planning structures.

Distributed Optimisation

A classical optimisation problem considers finding a decision variable $x \in \mathcal{X}$ that minimises a given objective function. A generic problem is of the form,

$$\min f(x), \quad (3.3)$$

$$\text{s.t. } x \in \mathcal{X}. \quad (3.4)$$

where $f(x)$ is the cost (objective) to be minimised. In classical problems, the function $f(x)$ and $\mathcal{X}$ are assumed to be available in one place, i.e., on a centralised

computer. In contrast, in more modern problem, the data is inherently distributed among agents or locations. This means no single agent has access to the entire objective or constraints. Instead, each agent might only know portion of the cost function or a subset of constraints. Such problems are referred to as distributed optimisation problems [95–97]. The goal is to find a solution that minimises the global cost function while satisfying the constraints, even though the data is distributed across multiple agents.

There are two primary classes of distributed optimisation problems: cost-coupled and decision-coupled.

In cost-coupled optimisation problems, all agents minimise a common global cost function, with each agent subject to independent local constraints. These problems are formulated as:

$$\min f(x_1, \dots, x_m), \quad (3.5)$$

$$\text{s.t. } x_i \in \mathcal{X}_i, \forall i = 1, \dots, m. \quad (3.6)$$

The algorithms that solve the problem of the above form are the Jacobi algorithm and the Gauss-Seidel algorithm.

In decision-coupled optimisation problems, each agent has its own local cost function, which depends on a global decision variable shared across all agents. The objective in these problems is to minimise the sum of individual cost functions, with the shared decision variable required to satisfy the constraints of every agent. Formally, this is expressed as:

$$\min \sum_i f_i(x), \quad (3.7)$$

$$\text{s.t. } x \in \mathcal{X}_i, \forall i = 1, \dots, m. \quad (3.8)$$

The algorithms that solve the problem of the above form are the proximal minimisation algorithm and the Alternating Direction Method of Multipliers (ADMM) algorithm.

3.3 Cooperative Sampling-Based Motion Planning under STL Constraints

This section develops a cooperative sampling-based planner for coupled tasks. Such tasks can arise, for example, in the planning of a dual-arm mobile manipulator, such as the one shown in Figure 3.4, which was part of the EU Horizon 2020 project *CANOPIES*.

Problem Formulation

Consider the motion-planning problem of two autonomous agents subject to a coupled task expressed as an STL formula φ of the form (1.1). We assume that the



Figure 3.4: TIAGo Mobile Manipulator developed by PAL Robotics.

agents evolve in $\mathbb{R}^n$, with $n \in \mathbb{N}$, characterized by the state trajectories $\mathbf{x}^i : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^n$, for $i \in \{1, 2\}$.

More specifically, the problem we consider is the design of time-varying trajectories $y_i : [0, \infty) \rightarrow \mathbb{R}^n$, starting at the initial configurations $y_i(0) = \mathbf{x}^i(0)$, $i \in \{1, 2\}$, that satisfy a user-defined coupled STL task φ , i.e., $(y, 0) \models \varphi$. with $y := [y_1^\top, y_2^\top]^\top$.

We currently do not focus on dynamics of the agent or the explicit control design that guarantees tracking of the trajectory $y_i(t)$; we assume that the agents are endowed with a control algorithm that allows such tracking. Instead, we focus on the planning problem, i.e., the derivation of continuous time-varying trajectories that satisfy φ . Unlike previous works in the related literature (e.g., [31]), we consider the entire fragment of STL and develop a distributed and efficient sampling-based algorithm.

Solution Approach

Inspired by standard sampling-based motion-planning techniques, this work develops a cooperative time-augmented sampling-based algorithm. In particular, the algorithm builds incrementally two *spatio-temporal* trees, one for each agent, by sampling points in the *coupled* state and time domain of the agents. The algorithm checks if the new edges to be added to the trees satisfy certain parts of the given task. In that way, the resulting spatio-temporal trees represent time-varying trajectories in the agents' state space that satisfy the given spatio-temporal task.

Finally, the algorithm can be executed in a distributed manner since the agents build their own trees by communicating with each other. The proposed algorithm is relevant to the approach of [98], which introduces a time-augmented sampling-based algorithm for single-agent systems under STL tasks. Similarly, [71] proposes a sampling-based algorithm for single-agent systems, encoding a temporal-logic task in a cost function to be minimized, without sampling in the time domain.

The solution consists of a variation of the standard Rapidly-exploring Random Tree (RRT) algorithm (Section 3.2) and it derives a time-varying trajectory $y(t)$ that satisfies φ . The key element is the incorporation of time in the sampling process, leading to a *time-augmented* sampling method. Compared to the original sampling-based algorithms, we replace the obstacle collision-checking procedure with a procedure that checks whether a specific fragment of the constructed tree violates the given STL formula. The aforementioned properties allow the direct derivation of a time-varying trajectory $y(t)$ that satisfies the given STL task encoded in φ . This problem poses additional challenges since it involves determining satisfaction of φ , which is defined over a continuous signal, using only point-wise data. We propose the following algorithm, called STLcoRRT, to address these challenges.

The STLcoRRT algorithm comprises two main components, namely the sampling-based procedure to build a tree, and the STL verification procedure that decides the satisfaction of a STL formula. Each sample corresponds to a point $z = (t, \mathbf{x}) \in \mathcal{Z} \subset \mathbb{R}_+ \times \mathbb{R}^n$, which the STL verification procedure decides if it is feasible with respect to the STL formula φ ; if yes, it approves it and adds it as a vertex to the tree. Every added edge is verified against a formula and a path traversing any edge ensures the satisfaction of φ by construction. The termination of the STLcoRRT algorithm guarantees the generation of a trajectory that satisfies the formula.

In what follows, we introduce first the sampling-based procedure and then the overall STLcoRRT algorithm below.

Main Results

Sampling-based Procedure: The traditional RRT algorithm samples a point $\mathbf{x}_{\text{samp}}$ from $\mathbb{R}^n$, finds its nearest neighbour $\mathbf{x}_{\text{nearest}}$ from the existing tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, and draws an edge of predefined length $step$ to create a new vertex $\mathbf{x}_{\text{new}}$. The inputs to the RRT algorithm are a starting vertex $\mathbf{x}(0)$, representing the initial condition of a path, a goal region, signifying the termination of the algorithm (once $\mathbf{x}_{\text{new}}$ is sampled in the goal region), and a hyper-parameter $step$ representing the edge length. An edge from $\mathbf{x}_{\text{nearest}}$ to $\mathbf{x}_{\text{new}}$ is added to $\mathcal{E}$ in the direction of $\mathbf{x}_{\text{samp}}$. The Euclidean distance between $\mathbf{x}_{\text{new}}$ and $\mathbf{x}_{\text{nearest}}$ is user defined and denoted by $step$. The STLcoRRT algorithm, as opposed to the traditional RRT algorithm, generates a spatio-temporal tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges. Each vertex in the set $\mathcal{V} = \{z_1, z_2, \dots\}$ is given by $z_i = (t_i, \mathbf{x}_i) \in \mathcal{Z} \subset \mathbb{R}_+ \times \mathbb{R}^n$. By sampling in time, the termination of STLcoRRT generates a trajectory in space

and time while RRT only returns a path in space. The continuous sample space $\mathcal{Z}$ is predefined based on the range of variables $\mathbf{x}_i$ and the time horizon of the STL formula as defined in Definition 2. We assume the states $\mathbf{x}_i$ are sampled from a closed set. Another distinction is that in STLcoRRT, a vertex $z_{\text{new}} = (t_{\text{new}}, \mathbf{x}_{\text{new}})$ is sampled if and only if $t_{\text{new}} > t_{\text{nearest}}$ and subsequently any z_{new} mentioned henceforth is considered to be ahead in time with respect to z_{nearest} . Along with this restriction, every eligible z_{new} goes through a STL verification process deciding if the vertex could be a satisfactory point in the final trajectory satisfying $y(t) \models \varphi$.

Overall Algorithm (STLcoRRT): To solve the coupled motion planning problem for two cooperating agents, agent i and agent j , we partition the algorithm into three modules. From the point of view of agent i , the first module is about sampling in a time region where agent j has already explored, i.e. Algorithm 4: **PastAgent**. The second module is sampling in a time region where agent j is yet to explore, i.e. Algorithm 3: **FutureAgent**. And the final module is verification of the STL formula based on the sampled points, i.e. Algorithm 5: **EdgeApproval**. For the discussions below, we use index i to refer to an agent that is currently running the pseudo-code and index j for the other agent. We now describe the main algorithm and modules in detail.

- **STLcoRRT:** The main body of STLcoRRT starts with the initialisation of the tree $\mathcal{T}_i$ (not to be confused with the STL parse tree and the satisfaction variable tree) with the initial conditions $z_0^i = (0, \mathbf{x}^i(0))$ (line 1-5). In the pseudo-code, we denote the time component of a sample $z^i = (t^i, \mathbf{x}^i)$ as $z^i.t^i$ i.e., $t^i := z^i.t^i$. Agent i then requests the updated tree of agent j (line 8). The sampling takes place between lines 10-12 where the functions **Sample**, **Nearest** and **Steer** are standard (see Section 3.2) and borrowed from the original RRT algorithm [93];
 - **Sample** samples a random point $z_{\text{samp}}^i = (t_{\text{samp}}^i, \mathbf{x}_{\text{samp}}^i)$ in $\mathcal{Z}$,
 - **Nearest** finds the nearest node $z_{\text{nearest}}^i = (t_{\text{nearest}}^i, \mathbf{x}_{\text{nearest}}^i)$ with respect to the Euclidean distance in $\mathcal{T}_i$ to the sampled point z_{samp}^i , and,
 - **Steer** finds a node $z_{\text{new}}^i = (t_{\text{new}}^i, \mathbf{x}_{\text{new}}^i)$ which is *step* units apart from z_{nearest}^i in the direction of z_{samp}^i .

The condition in line 9 imposes every sample to be sampled ahead in time compared to its nearest neighbour. This guarantees every path in the tree to only move forward in time. Each agent samples points sequentially and builds its tree after communicating with the other agent. We distinguish two scenarios in this sampling procedure: a new sample $z_{\text{new}}^i = (t_{\text{new}}^i, \mathbf{x}_{\text{new}}^i)$ of one agent is at a point in time t_{new}^i when (1) the other agent j hasn't built a tree yet i.e. $t_{\text{new}}^i > \max\{t_k^j\}$, $k \in \{1, \dots, |\mathcal{V}_j|\}$ or (2) the other agent j has existing edges passing through this time instant t_{new}^i i.e. $t_{\text{new}}^i < \max\{t_k^j\}$, $k \in \{1, \dots, |\mathcal{V}_j|\}$. In the former case, each agent uses the function **FutureAgent**

Algorithm 2: STLcoRRT: Agent i perspective

Input: $\varphi, z_0^i, step, L, \mathcal{Z}_{goal}^i$, and $\mathcal{Z}$
Output: $\mathcal{T}_i$ and $path_i$

```

1  $\mathcal{V}_i \leftarrow \{z_0^i\};$ 
2  $\mathcal{E}_i \leftarrow \emptyset;$ 
3  $\mathcal{T}_i \leftarrow (\mathcal{V}_i, \mathcal{E}_i);$ 
4  $t_0 \leftarrow 0;$ 
5  $path_i \leftarrow \emptyset;$ 
6  $\tau \leftarrow \{-1\};$ 
7 while  $z^i \notin \mathcal{Z}_{goal}^i$  do
8    $\mathcal{T}_j \leftarrow \text{Agent } j;$ 
9   while  $z_{new}^i.t < z_{nearest}^i.t$  do
10      $z_{smp}^i \leftarrow \text{Sample}(\mathcal{Z});$ 
11      $z_{nearest}^i \leftarrow \text{Nearest}(\mathcal{T}_i, z_{smp}^i);$ 
12      $z_{new}^i \leftarrow \text{Steer}(z_{nearest}^i, z_{smp}^i, step);$ 
13   if  $z_{new}^i.t > \max(\mathcal{V}_j.t)$  then
14      $\mathcal{T}_i, \mathcal{T}_j \leftarrow \text{FutureAgent}(\varphi, step, L, z_{new}^i, z_{nearest}^i, \tau, \mathcal{T}_i, \mathcal{T}_j);$ 
15   else
16      $\mathcal{T}_i \leftarrow \text{PastAgent}(\varphi, L, z_{new}^i, z_{nearest}^i, \tau, \mathcal{T}_i, \mathcal{T}_j);$ 
17  $path_i \leftarrow path_i \cup \text{PlotPath}(\mathcal{T}_i)$ 

```

and in the latter case each agent uses the function **PastAgent** to decide the satisfiability of φ . This process is repeated until $z^i \in \mathcal{Z}_{goal}^i$ and terminates by finding the trajectory $path_i$. The goal region is $\mathcal{Z}_{goal}^i = \{z^i \in \mathcal{Z} \mid t^i \geq \text{th}(\varphi)\}$. Both the agents simultaneously run STLcoRRT and communicate with the other agent after every edge addition. We further need the following definition

Definition 3 (Vertex Incidence) *A vertex z_{new}^j is said to be incident on an edge $\{z_{nearest}^i, z_{new}^i\}$ if and only if $t_{nearest}^i < t_{new}^j < t_{new}^i$.*

Before we proceed with the discussion of the rest of the modules, we focus on how to evaluate an STL formula φ using sampled points. Recall that STL is a predicate-based logic and predicates are defined over predicate functions which are functions of states of both agents. To evaluate if a sampled point z_{new}^i could be a potential point in the final satisfiable path for formula φ , we require points of both agent states $z_{new}^i = (t_{new}, \mathbf{x}_{new}^i)$ and $z_{new}^j = (t_{new}, \mathbf{x}_{new}^j)$ at the same time instant t_{new} . The pair $\{z_{new}^i, z_{new}^j\}$ is used to evaluate the predicate and therein the formula φ . Suppose the edge resultant from adding z_{new}^i to $\mathcal{T}_i$ is $\{z_{nearest}^i, z_{new}^i\}$. As in obstacle avoidance problems, where an edge connecting two non-collision vertices can still intersect an obstacle, we follow

the same principle and evaluate points on the edge to ensure such points do not violate φ . We pick $L \in \mathbb{N}$ points on the edge $\{z_{\text{nearest}}^i, z_{\text{new}}^i\}$ as

$$z^i = z_{\text{nearest}}^i + l(z_{\text{new}}^i - z_{\text{nearest}}^i),$$

where $l = 0, \frac{1}{L-1}, \frac{2}{L-1}, \dots, \frac{L-2}{L-1}, 1$. Each z^i corresponds to z^j via $z^i.t = z^j.t$ over which we evaluate φ . The hyper-parameter L specifies the discretisation of the edge.

• **FutureAgent:**

Function 3: FutureAgent	
Input: $\varphi, \text{step}, L, z_{\text{new}}^i, z_{\text{nearest}}^i, \tau, \mathcal{T}_i$, and $\mathcal{T}_j$	
Output: $\mathcal{T}_i$ and $\mathcal{T}_j$	
1	$\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z_{\text{new}}^i;$
2	$\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z_{\text{nearest}}^i, z_{\text{new}}^i\};$
3	while $z_{\text{new}}^i.t < z_{\text{nearest}}^i.t$ or $z_{\text{new}}^j.t \notin (z_{\text{nearest}}^i.t, z_{\text{new}}^i.t)$ do
4	$z_{\text{samp}}^j \leftarrow \text{Sample}(\mathcal{Z});$
5	$z_{\text{nearest}}^j \leftarrow \text{Nearest}(\mathcal{T}_j, z_{\text{samp}}^j);$
6	$z_{\text{new}}^j \leftarrow \text{Steer}(z_{\text{nearest}}^j, z_{\text{samp}}^j, \text{step});$
7	$z_{\text{inter}}^i \leftarrow \text{Interpolate}(\{z_{\text{nearest}}^i, z_{\text{new}}^i\}, z_{\text{new}}^j.t);$
8	for $l \leftarrow 0$ to L by $1/(L-1)$ do
9	$z^i = z_{\text{nearest}}^i + l(z_{\text{inter}}^i - z_{\text{nearest}}^i);$
10	$z^j \leftarrow \text{Interpolate}(\{z_{\text{nearest}}^j, z_{\text{new}}^j\}, z^i.t);$
11	$\pi, \tau \leftarrow \text{EdgeApproval}(\varphi, z^i, z^j, \tau);$
12	if $\neg \pi$ then
13	$\mathcal{V}_i \leftarrow \mathcal{V}_i \setminus z_{\text{new}}^i;$
14	$\mathcal{E}_i \leftarrow \mathcal{E}_i \setminus \{z_{\text{nearest}}^i, z_{\text{new}}^i\};$
15	return $\mathcal{T}_i$ and $\mathcal{T}_j$
16	$\mathcal{V}_j \leftarrow \mathcal{V}_j \cup z_{\text{new}}^j;$
17	$\mathcal{E}_j \leftarrow \mathcal{E}_j \cup \{z_{\text{nearest}}^j, z_{\text{new}}^j\};$
18	$\mathcal{V}_i \leftarrow \mathcal{V}_i \setminus z_{\text{new}}^i;$
19	$\mathcal{E}_i \leftarrow \mathcal{E}_i \setminus \{z_{\text{nearest}}^i, z_{\text{new}}^i\};$
20	$\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z_{\text{inter}}^i;$
21	$\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z_{\text{nearest}}^i, z_{\text{inter}}^i\};$

This function is called when agent i samples a point $z_{\text{new}}^i = (t_{\text{new}}^i, \mathbf{x}_{\text{new}}^i)$ where $t_{\text{new}}^i > t_k^j$ for all $k \in (1, 2, \dots, |\mathcal{V}_j|)$ as seen in Figure 3.5. It starts by temporarily adding the sampled vertex z_{new}^i to the tree of agent i (lines 1-2). Next, it communicates this $\{z_{\text{nearest}}^i, z_{\text{new}}^i\}$ to agent j and agent j samples indefinitely until a sampled point lies incident to the temporarily added edge of agent i , lines 3-6 and seen in Figure 3.5 (center). Then, the algorithm

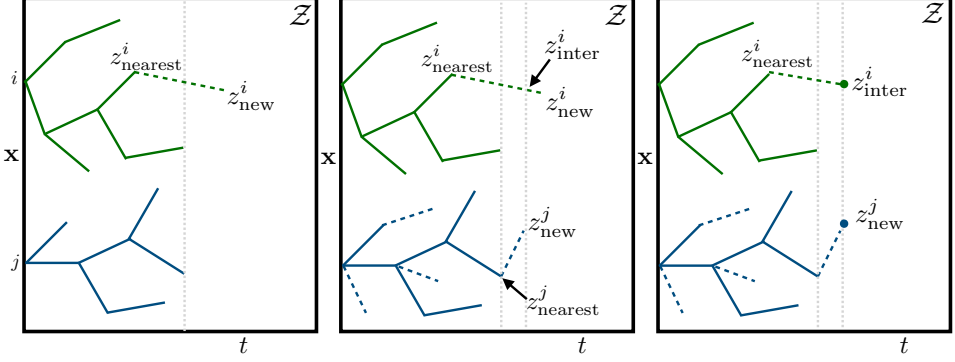


Figure 3.5: **FutureAgent** illustration (i) Left: Agent i adds an edge where Agent j has no incident edges (lines 1-2), (ii) Center: Agent j samples until it finds a vertex incident to Agent i 's new edge (lines 3-6), (iii) Right: Agent i interpolates to create a vertex pair $\{z_{\text{inter}}^i, z_{\text{new}}^j\}$ (line 7).

linearly interpolates in line 7 to obtain the vertex $z_{\text{inter}}^i = (t_{\text{new}}^j, \mathbf{x}_{\text{inter}}^i)$ as seen in Figure 3.5 (right). This is obtained by solving for $\mathbf{x}_{\text{inter}}^i$ element-wise as the solution of,

$$\mathbf{x}_{\text{inter}}^i = \left(\frac{\mathbf{x}_{\text{nearest}}^i - \mathbf{x}_{\text{new}}^i}{t_{\text{nearest}}^i - t_{\text{new}}^i} \right) (t_{\text{new}}^j - t_{\text{new}}^i) + \mathbf{x}_{\text{new}}^i.$$

We define a function **Interpolate** that solves the aforementioned computation. We can now sample points over the edge $\{z_{\text{nearest}}^i, z_{\text{inter}}^i\}$ and all the corresponding interpolated points from $\mathcal{T}_j$ to evaluate the STL formula through the **EdgeApproval** function, depicted in Function 5 and discussed below. By $L \in \mathbb{N}$ we chose the number of equally spaced points on the edge $\{z_{\text{nearest}}^i, z_{\text{inter}}^i\}$ in the evaluation of φ . The new vertex z_{new}^i of agent i is deleted if it does not satisfy the STL formula (lines 13-14).

- **PastAgent:**

This function, depicted in Function 4, is called when an agent i samples a point $z_{\text{new}}^i = (t_{\text{new}}^i, \mathbf{x}_{\text{new}}^i)$ and when there is at least one edge in $\mathcal{T}_j$ that is being incident to z_{new}^i ; an illustration is shown in Figure 3.6; **PastAgent** begins by evaluating z_{inter}^j from the incident edges using linear interpolation as seen in lines 1-9. In line 7, as in the case of **FutureAgent**, we chose L samples over the edge $\{z_{\text{nearest}}^i, z_{\text{new}}^i\}$ and find the corresponding interpolated z_{inter}^j 's from $\mathcal{T}_j$ to evaluate **EdgeApproval**. If all the pairs result in $\pi = \top$, then the node z_{new}^i and the edge $\{z_{\text{nearest}}^i, z_{\text{new}}^i\}$ are added to the tree $\mathcal{T}_i$, see lines 5-11.

- **EdgeApproval:** Function 5 is recursively defined and takes as input an STL formula of the form (1.1), a pair of points $\{z^i, z^j\}$ from the trees of both

Function 4: PastAgent

Input: $\varphi, L, z_{\text{new}}^i, z_{\text{nearest}}^i, \tau, \mathcal{T}_i$, and $\mathcal{T}_j$
Output: $\mathcal{T}_i$

```

1 foreach  $\{z_m^j, z_{m+1}^j\} \in \mathcal{E}_j$  do
2    $t_m^j \leftarrow z_m^j.t$ ;
3    $t_{m+1}^j \leftarrow z_{m+1}^j.t$ ;
4   if  $t_m^j < t_{\text{new}}^i < t_{m+1}^j$  then
5     for  $l \leftarrow 0$  to  $L$  by  $1/(L-1)$  do
6        $z^i = z_{\text{nearest}}^i + l(z_{\text{new}}^i - z_{\text{nearest}}^i)$ ;
7        $z^j \leftarrow \text{Interpolate}(\{z_m^j, z_{m+1}^j\}, z^i.t)$ ;
8        $\pi, \tau \leftarrow \text{EdgeApproval}(\varphi, z^i, z^j, \tau)$ ;
9       if  $!\pi$  then return  $\mathcal{T}_i$ ;
10  $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z_{\text{new}}^i$ ;
11  $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z_{\text{nearest}}^i, z_{\text{new}}^i\}$ ;
```

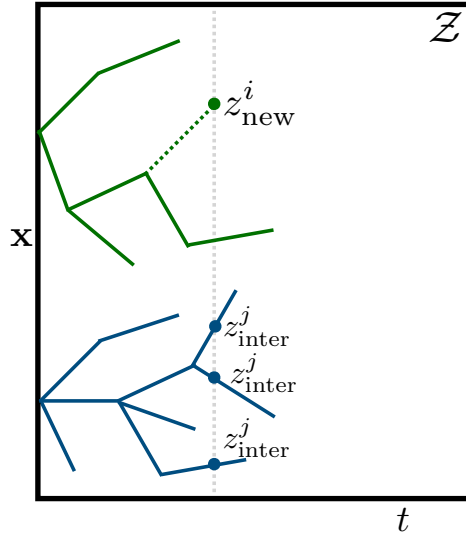


Figure 3.6: **PastAgent** illustration: Agent i samples z_{new}^i and interpolates in line 7 to find all incident Agent j samples.

agents, and returns a truth value π to determine if the vertices satisfy the STL formula. Along with π , it returns a variable $\tau(\varphi)$ taking values in the set $\{-1, +1\}$ indicating satisfaction of the STL sub-formula φ . The satisfaction of an STL formula cannot be determined by an individual point $z_{\text{new}}^i = (t_{\text{new}}^i, x_{\text{new}}^i)$. However, a point z_{new}^i along with z_{new}^j can momentarily determine the satisfaction at that particular instant in time, since the formula consists of both temporal and spatial variables. Next, we present a set of rules for adding a pair of sampled points $z_{\text{new}}^i, z_{\text{new}}^j$ to the tree based on the STL formula φ in (1.1).

- $\varphi = \top$: Any sampled point z_{new}^i is added to the tree.
- $\varphi = \mu$: A sampled point z_{new}^i is added to the tree, if and only if $\mu = \top$ with,

$$\mu = \begin{cases} \top & h(\mathbf{x}_{\text{new}}^i, \mathbf{x}_{\text{new}}^j, t_{\text{new}}) \geq 0 \\ \perp & h(\mathbf{x}_{\text{new}}^i, \mathbf{x}_{\text{new}}^j, t_{\text{new}}) < 0 \end{cases},$$

where the predicate function $h(\mathbf{x}_{\text{new}}^i, \mathbf{x}_{\text{new}}^j, t_{\text{new}})$ may encapsulate coupled constraints on states of agents i, j .

- $\neg\varphi$: A sampled point z_{new}^i is added to the tree if and only if it is not added to the tree for φ .
- $\varphi = \varphi_1 \vee \varphi_2$: A sampled point z_{new}^i is added to the tree if it is added to the tree for φ_1 or φ_2 in the sense presented above. But the point-wise evaluation of an STL formula does not track history of satisfaction of parts of the formula. For example, consider the STL formula

$$\varphi = \varphi_1 \vee \varphi_2 = (\mathcal{G}_{[1,2]}\mu_1 \wedge \mathcal{F}_{[4,5]}\mu_2) \vee (\mathcal{G}_{[1,2]}\mu_3 \wedge \mathcal{F}_{[4,5]}\mu_4)$$

and notice that **EdgeApproval** will make no distinction between the said formula and

$$(\mathcal{G}_{[1,2]}\mu_1 \wedge \mathcal{F}_{[4,5]}\mu_4) \vee (\mathcal{G}_{[1,2]}\mu_3 \wedge \mathcal{F}_{[4,5]}\mu_2).$$

To avoid such a scenario, the algorithm is designed to work as follows: If, in the construction of the tree, a sampled point satisfies any sub-formula of φ_1 , then we deactivate φ_2 and if a sampled point satisfies any sub-formula of φ_2 , then we deactivate φ_1 . In the pseudo-code, if a child node of φ_1 , denoted by $\varphi_1 >$, satisfies $\tau(\varphi_1 >) = +1$, then $\varphi_1 \vee \varphi_2 = \varphi_1 \vee \perp$. And if a child node of φ_2 , denoted by $\varphi_2 >$, satisfies $\tau(\varphi_2 >) = +1$, then $\varphi_1 \vee \varphi_2 = \perp \vee \varphi_2$. In the example presented above, if

$$z((0, 2), \mathbf{x}) \models \mathcal{G}_{[1,2]}\mu_1,$$

then

$$\varphi = (\mathcal{G}_{[1,2]}\mu_1 \wedge \mathcal{F}_{[4,5]}\mu_2) \vee \perp$$

and if

$$z((0, 2), \mathbf{x}) \models \mathcal{G}_{[1,2]}\mu_3,$$

then

$$\varphi = \perp \vee (\mathcal{G}_{[1,2]}\mu_3 \wedge \mathcal{F}_{[4,5]}\mu_4).$$

It may appear that by formulating disjunction as presented above we deliberately make a choice of satisfying either φ_1 or φ_2 . The algorithm is designed such that φ_1 or φ_2 is satisfied depending on whether a sub-formula from φ_1 or φ_2 is satisfied prior. The choice of sub-formula from φ_1 or φ_2 is completely random and depends on the evolution of the tree. Thus the choice of φ_1 or φ_2 is still random.

- $\varphi = \varphi_1 \wedge \varphi_2$: A sampled point z_{new}^i is added to the tree if it is added to the tree for φ_1 and φ_2 .
- $\varphi = \mathcal{F}_I\varphi$: In the case of an eventually operator, it is possible that a single sample z_{new}^i could satisfy $\mathcal{F}_I\varphi$. And it is also possible that such a point might not be a part of the final trajectory. Thus, to ensure this satisfying point is included in our final trajectory we first extract and store the path connecting z_o^i and z_{new}^i . The trees of both agents are then reset and only the nodes z_{new}^i and z_{new}^j are added to $\mathcal{V}_i$ and $\mathcal{V}_j$ respectively. By doing this, any final trajectory will pass through z_{new}^i as it is the only sample connecting the prior tree and the new tree, lines 21-26. For $I = [a, b]$, if $z_{\text{new}}^i.t < b$ the sampled point z_{new}^i is added to the tree. Any sampled point where $z_{\text{new}}^i.t > b$ is not added to the tree, this stops the tree from growing beyond $z_{\text{new}}^i.t > b$ until we find a satisfying point. Once a point is sampled that satisfies φ , we remove the hold on tree expansion i.e. the tree is allowed to expand beyond $z_{\text{new}}^i.t > b$. Once $\tau(\varphi) = +1$, then this implies that $\tau(\mathcal{F}_I\varphi) = +1$ for the sample z_{new}^i . Note that if $\varphi = \mu$, then $\tau(\varphi) = +1 \Leftrightarrow \mu = \top$.
- $\varphi = \mathcal{G}_I\varphi$: If $z_{\text{new}}^i.t \in [a, b]$, the sampled point z_{new}^i is added to the tree if $\tau(\varphi) = +1$. If $t_{\text{new}} \notin [a, b]$ then z_{new}^i is added to the tree. This means the algorithm does not proceed if it does not satisfy $\mathcal{G}_I\varphi$. Again note $\tau(\varphi) = +1 \Leftrightarrow \mu = \top$ if $\varphi = \mu$.
- $\varphi = \varphi_1 \mathcal{U}_I \varphi_2$: The until operator is treated as $\mathcal{G}_{[z_{\text{new}}.t, z_{\text{new}}.t]}\varphi_1$ until $\tau(\mathcal{F}_{[z_{\text{new}}.t, z_{\text{new}}.t]}\varphi_1) = +1$. The interval is a singleton set since the satisfaction instance of φ_2 cannot be assessed apriori.

The function **EdgeApproval** is recursively defined over (1.1) and hence covers the entire STL formula. Below we see an example of how **EdgeApproval** treats nested temporal operators.

Example 5 Let $\varphi = \mathcal{G}_{[0,20]}\mathcal{F}_{[1,3]}\mu$. Formula φ requires μ to hold at least every two seconds in the interval $[1, 23]$ s. Recalling **EdgeApproval**(φ), once $\mu = \top$ in the interval $[1, 3]$ s, line 20 in Function 5 returns $\tau(\mathcal{F}_{[1,3]}\mu) = +1$ and

Function 5: EdgeApproval**Input:** $\varphi, z_{new}^i, z_{new}^j, \tau$ **Output:** π, τ

```

1  switch  $\varphi$  do
2    case  $\top$  do
3      | return  $\top, \tau$ 
4    case  $\mu$  do
5      | if  $\mu$  then return  $\top, \tau$ ;
6      | else return  $\perp, \tau$ ;
7    case  $\neg\varphi$  do
8      | return  $\neg\text{EA}(\varphi, \tau)$ 
9    case  $\varphi_1 \vee \varphi_2$  do
10     | if  $z_{new}^i.t \in t_0 \oplus I$  and  $\text{EA}(\varphi_1, \tau) = (\cdot, +1)$  then
11     |   return  $\text{EA}(\varphi_1, \tau) \vee \text{EA}(\perp, \tau)$ ;
12     | else if  $z_{new}^j.t \in t_0 \oplus I$  and  $\text{EA}(\varphi_2, \tau) = (\cdot, +1)$  then
13     |   return  $\text{EA}(\perp, \tau) \vee \text{EA}(\varphi_2, \tau)$ ;
14     | else
15     |   return  $\text{EA}(\varphi_1, \tau) \vee \text{EA}(\varphi_2, \tau)$ ;
16   case  $\varphi_1 \wedge \varphi_2$  do
17     | return  $\text{EA}(\varphi_1, \tau) \wedge \text{EA}(\varphi_2, \tau)$ 
18   case  $\mathcal{F}_I\varphi$  do
19     | if  $z_{new}^i.t \in t_0 \oplus I$  and  $\text{EA}(\varphi, \tau) = (\top, \cdot)$  then
20     |    $\tau \leftarrow +1$ ;
21     |    $\text{path}_i \leftarrow \text{PlotPath}(\mathcal{T}_i)$ ;
22     |    $\text{path}_j \leftarrow \text{PlotPath}(\mathcal{T}_j)$ ;
23     |    $\mathcal{V}_i \leftarrow \emptyset, \mathcal{E}_i \leftarrow \emptyset$ ;
24     |    $\mathcal{V}_j \leftarrow \emptyset, \mathcal{E}_j \leftarrow \emptyset$ ;
25     |    $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z_{new}^i$ ;
26     |    $\mathcal{V}_j \leftarrow \mathcal{V}_j \cup z_{new}^j$ ;
27     |   return  $\text{EA}(\varphi, \tau)$ ;
28     | else if  $z_{new}^i.t \in t_0 \oplus I$  and  $\text{EA}(\varphi, \tau) = (\perp, \cdot)$  then
29     |   return  $\neg \text{EA}(\varphi, \tau)$ ;
30     | else
31     |   return  $\text{EA}(\top, \tau)$ ;
32   case  $\mathcal{G}_I\varphi$  do
33     | if  $z_{new}^i.t \in t_0 \oplus I$  then
34     |   return  $\text{EA}(\varphi, \tau)$ 
35     | else
36     |   if  $z_{new}^i.t > I.2$  then  $\tau \leftarrow +1$ ;
37     |   return  $\text{EA}(\top, \tau)$ 
38   case  $\varphi_1 \mathcal{U}_I \varphi_2$  do
39     | while  $\text{EA}(\mathcal{F}_I\varphi_2, \tau) \neq (\cdot, +1)$  do
40     |   return  $\text{EA}(\mathcal{G}_{[z_{new}^i.t, z_{new}^i.t]}\varphi_1, \tau)$ 
41   | return  $\text{EA}(\mathcal{F}_{[z_{new}^i.t, z_{new}^i.t]}\varphi_2, \tau)$ 

```

/* EdgeApproval is abbreviated as EA for readability purposes

*/

the initial conditions are reset to $z_o \leftarrow z_{new}$. This ensures $t_o \leftarrow z_{new}.t$ and the new interval is $[1 + z_{new}.t, 3 + z_{new}.t]$. The next case called is $\mathcal{G}_I\varphi$ which returns $\mathcal{G}_I(\top)$ in line 34 and **EdgeApproval** returns $(\top, +1)$. This process repeats until the entire interval $[0, 23]s$ is covered.

Once the algorithm terminates i.e. when the tree reaches the designated goal region, a path $(y_1(t), y_2(t))$ is generated by **PlotPath** as done in the standard RRT [93]. The paths $(y_1(t), y_2(t))$ satisfy the STL formula by construction since every node in the tree satisfies the STL formula via the **EdgeApproval** function. The generated trajectories can be verified using the robust semantics presented in Section 1.2 by evaluating for $\rho(\varphi, \mathbf{x}, t)$. STLcoRRT, unlike monitoring algorithms, generates trajectories that satisfy the STL formula by construction. The inter-dependency of coupled tasks between agents poses a significant challenge in solving the problem in a distributed way while covering the entire STL formula rather than a fragment of it. We believe that the coupled-agents approach treated here is a first step towards solving the more general multi-agent motion planning under STL task problem.

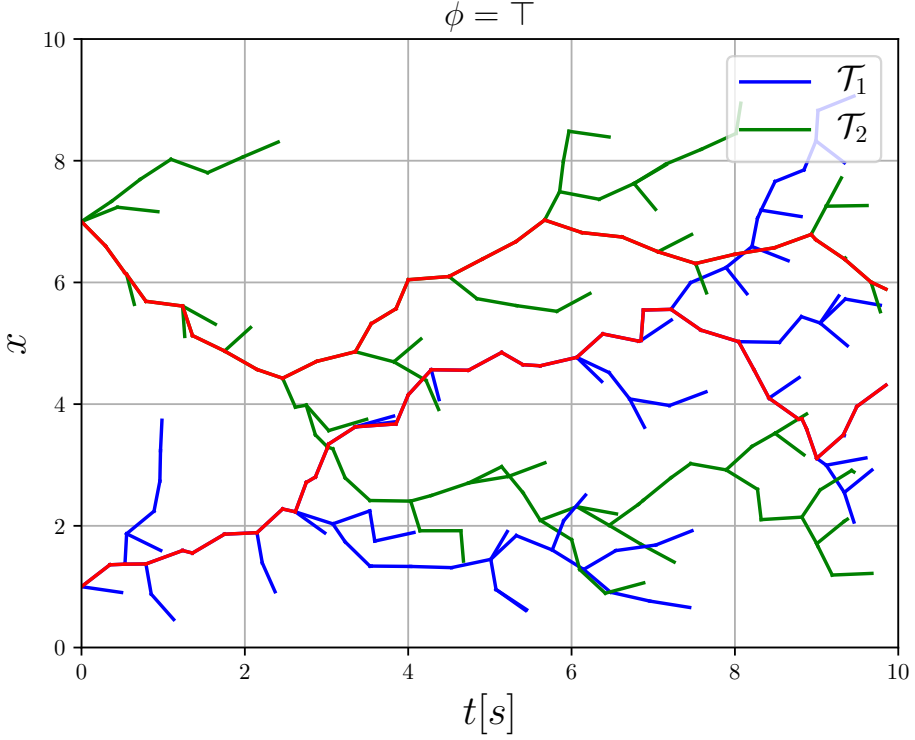
This concludes the presentation of the STLcoRRT algorithm. As with sampling based methods, we can take various steps to speed up the algorithm such as:

- Changing the *step* to a larger value for high dimensions;
- A maximum number of nodes can be specified in Algorithm 2 by changing Line 7 to a **for** loop;
- An iteration limit can be imposed on the second condition in Line 3 of Function 3. Since in some cases it could be too narrow to sample in $(z^{\text{nearest}}.t, z_{new}^i.t)$.

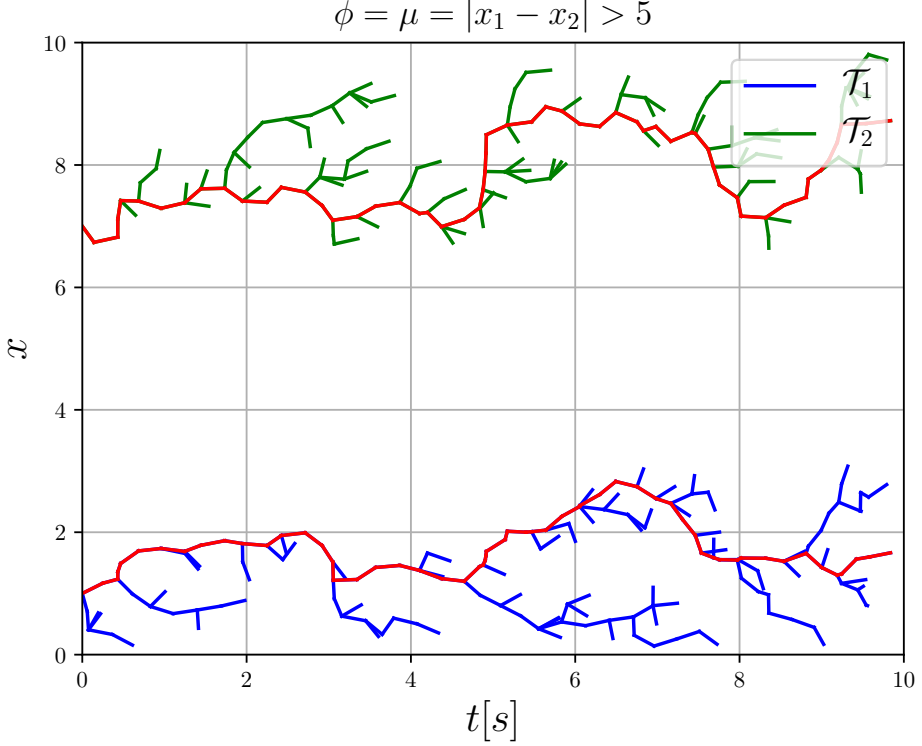
Simulations

In this Section, simulations are presented to illustrate the performance of STLcoRRT.

- Figure 3.7: The formula $\phi = \top$ does not impose any constraints on the agents. The algorithm is able to sample points in the entire space and the trees of both agents are able to grow freely.
- Figure 3.8: The formula $\phi = |x_1 - x_2| > 5$ is a coupled constraint and requires both agents to be at least 5 units apart. Agent 1 (in blue) and Agent 2 (in green) build their trees ensuring that every added vertex is atleast 5 units away from all incident vertices of the other agent such that any final trajectory will satisfy the formula.

Figure 3.7: Trees for the formula $\phi = \top$.

- Figure 3.9: The formula $\phi = \mathcal{G}_{[4,6]}|x_1 - x_2| < 2$ requires in the interval $[4, 6]$ units, every added vertex is less than two units apart with respect to the other agent. The trees are built satisfying this constraint in the interval $[4, 6]$ as seen in the figure.
- Figure 3.10: The formula $\phi = \mathcal{F}_{[4,6]}|x_1 - x_2| > 8$ requires the agents to be 8 units apart at any one instant in the interval $[4, 6]$. All sampled vertices $(z_{\text{new}}^i, z_{\text{new}}^j)$ are added to the tree until $|x_1 - x_2| > 8$ becomes true, indicated by red crosses in the figure. Once the condition $|x_1 - x_2| > 8$ holds, the corresponding vertices are set as new starting vertices for the STLcoRRT algorithm and are thus included in the final path.
- Figure 3.11: The formula $\phi = \mathcal{G}_{[2,8]}(x_1 < 1) \wedge \mathcal{G}_{[4,6]}|x_1 - x_2| < 2$ requires $x_1 < 1$ in the interval $[2, 8]$ and $|x_1 - x_2| < 2$ in the interval $[4, 6]$. This is evident from the figure as Agent 1 samples only in the set $\{x_1 < 1\}$ in the interval $[2, 8]$ and Agent 2 comes closer satisfying $|x_1 - x_2| < 2$ in the interval $[4, 6]$.


 Figure 3.8: Trees for the formula $\phi = \mu = |x_1 - x_2| > 5$.

- Figure 3.12: The formula $\phi = \mathcal{F}_{[4,6]}(|x_1 - x_2| > 8) \vee \mathcal{G}_{[6,8]}|x_1 - x_2| < 2$ requires the agents to be 8 units apart at any one instant in the interval $[4, 6]$ or $|x_1 - x_2| < 2$ in the interval $[6, 8]$. In Figure 3.12a we see that the algorithm chose to satisfy the subformula $\mathcal{F}_{[4,6]}|x_1 - x_2| > 8$ while in other run $\mathcal{G}_{[6,8]}|x_1 - x_2| < 2$ was satisfied as seen in Figure 3.12b.
- Figure 3.13: The nested formula, $\varphi = \mathcal{G}_{[0,10]}\mathcal{F}_{[1,3]}|x_1 - x_2| > 8$, requires as task to ‘satisfy $|x_1 - x_2| > 8$ every 2 seconds in the interval $[1, 13]$ ’. As seen over two example runs in Figure 3.13a and Figure 3.13b, each red cross represents the satisfaction of $\mathcal{F}_{[\star^1, \star^2]}$ where $[\star^1, \star^2]$ are set to $[1, 3]$ at $t = 0$ and re-evaluated after every satisfaction of the eventually.

3.4 Multi-Robot Autonomous Motion Planning under STL Constraints

In Section 3.3, we addressed the STL motion planning problem for two coupled agents. There, we extended the conventional Rapidly-exploring Random Trees

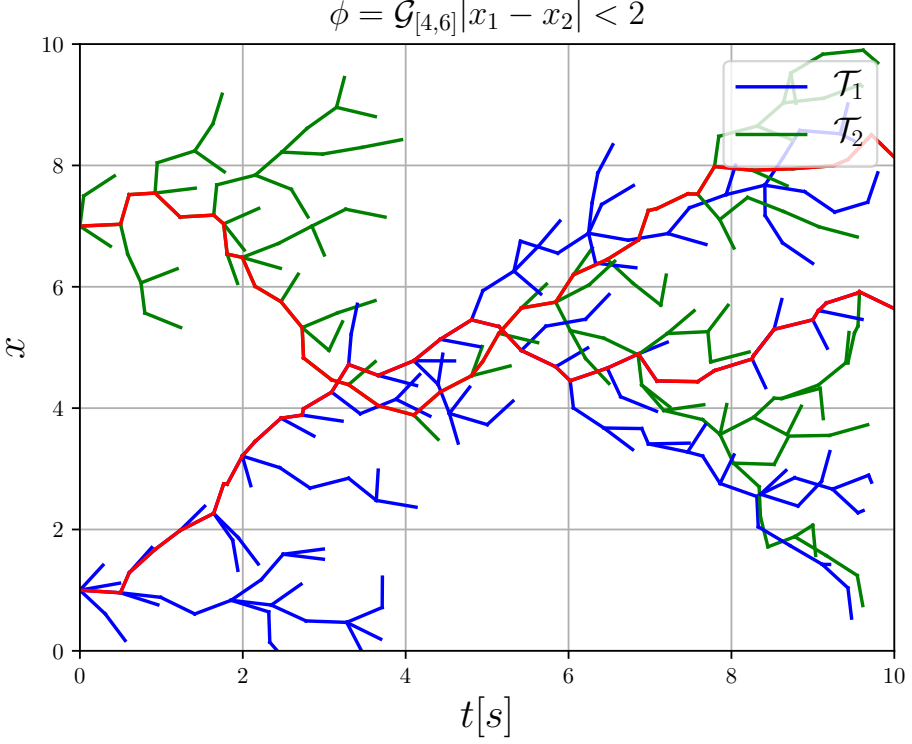


Figure 3.9: Trees for the formula $\phi = \mathcal{G}_{[4,6]}|x_1 - x_2| < 2$.

(RRT) algorithm to sample in both the time and space domains. Our approach incrementally built spatio-temporal trees through which we enforced space and time constraints as specified by the STL formula. The algorithm employed a sequential planning method, wherein each agent communicated and waited for the other agent to build its tree. In contrast, this section addresses the STL motion planning problem for multiple robots. Here, our algorithm adopts a distributed optimisation-based approach, where spatial and temporal aspects are decoupled to satisfy the STL formula. Instead of constructing an incremental tree, as done in the previous work, we introduce a novel metric called the *validity domain* and initialise the process with an initial trajectory. Additionally, we present experimental validation results and introduce a novel STL verification architecture.

Problem Formulation

Let the state trajectory of the i th robot be $\mathbf{x}_i : \mathbb{R}_{\geq 0} \rightarrow \mathcal{W}_i$, where $\mathcal{W}_i \subset \mathbb{R}^{n_i}$ denotes its state space and let $\mathcal{W} = \mathcal{W}_1 \times \mathcal{W}_2 \times \dots \times \mathcal{W}_{\mathbf{N}}$ where $\mathcal{W} \subset \mathbb{R}^{n_1} \times \dots \times \mathbb{R}^{n_{\mathbf{N}}}$ is the coupled space of all robot states. The problem tackled in this paper is stated

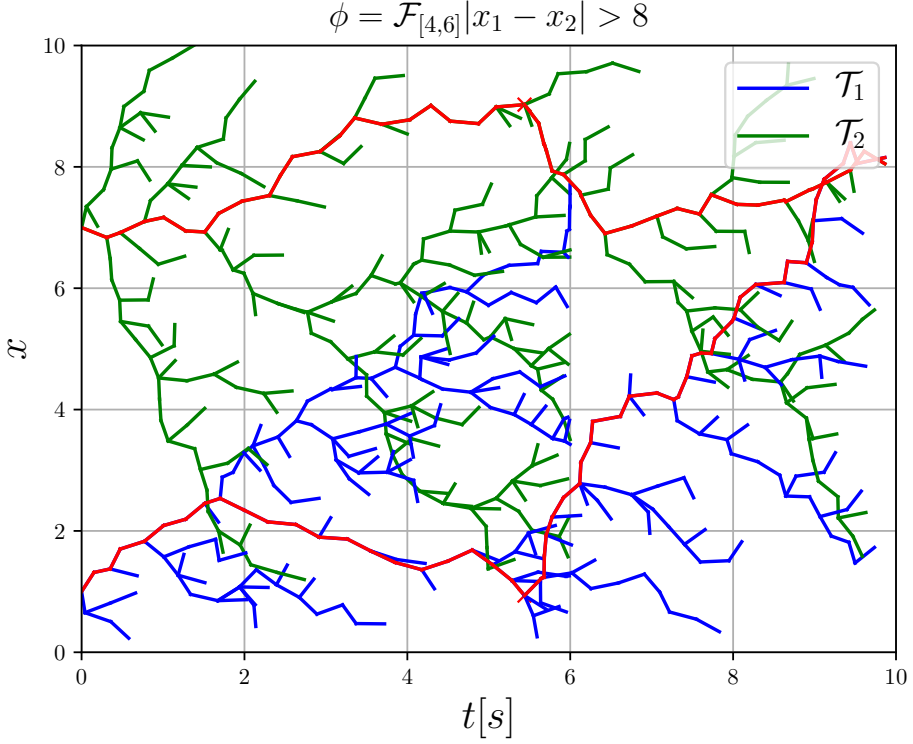


Figure 3.10: Trees for the formula $\phi = \mathcal{F}_{[4,6]}|x_1 - x_2| > 8$.

as follows;

Problem 3 *Given an STL formula φ that specifies tasks on a multi-robot system with N robots, design a distributed algorithm to find continuous time-varying trajectories $\mathbf{y}_i : [0, \text{th}(\varphi)] \rightarrow \mathcal{W}_i$, starting at an initial configuration $\mathbf{y}_i(0) = \mathbf{x}_i(0)$, such that $(\mathbf{y}, t) \models \varphi, \forall t \in [0, \text{th}(\varphi)]$ with $\mathbf{y} := [\mathbf{y}_1^\top, \mathbf{y}_2^\top, \dots, \mathbf{y}_N^\top]^\top$.*

We do not currently address the closed-loop stability of the underlying multi-robot system. Instead, we focus on the trajectory generation aspect and rely on existing low-level control approaches to track the generated trajectories. For more information, see Remark 6. The above problem is addressed assuming that at least one such solution exists. This will help us provide probabilistic completeness guarantees later on. Formally, we state the following assumption:

Assumption 6 *There exists at least one $\mathbf{y}$ such that $(\mathbf{y}, t) \models \varphi$.*

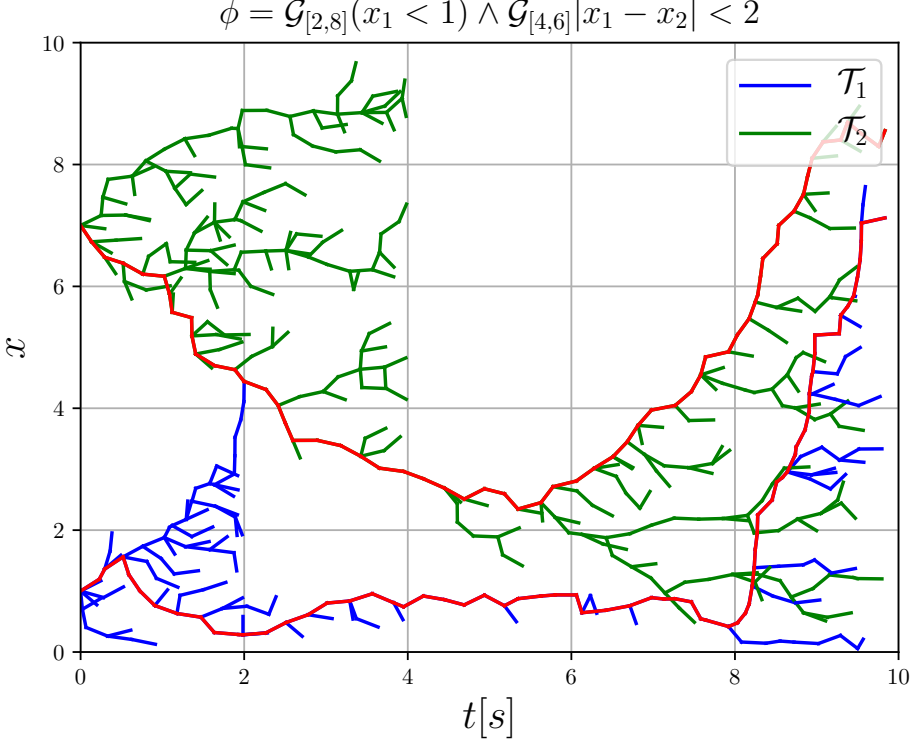


Figure 3.11: Trees for the formula $\phi = \mathcal{G}_{[2,8]}(x_1 < 1) \wedge \mathcal{G}_{[4,6]}|x_1 - x_2| < 2$.

Solution Approach

In this Section, we propose **MAPS**² - **M**ulti-**R**obot **A**utonomous Motion **P**lanning under **S**ignal Temporal Logic **S**pecifications - to address the multi-robot motion planning problem subject to coupled STL constraints. The algorithm encodes these constraints into an optimisation function and selectively activates them based on the temporal requirements of the STL formula. While doing so, each robot only communicates with its neighbours and iteratively searches for STL satisfying trajectories. The algorithm ensures distributed trajectory generation to satisfy STL formulas that consist of coupled constraints for multiple robots. The main contributions of this section are summarised in the following attributes:

- The algorithm's effectiveness lies in its ability to distribute STL planning for multiple robots and in providing a mechanism to decouple the STL formula among robots, thereby facilitating the distribution of tasks.
- As opposed to previous work, it covers the entire STL formula and is not limited to a smaller fragment. It reduces conservatism by eliminating the

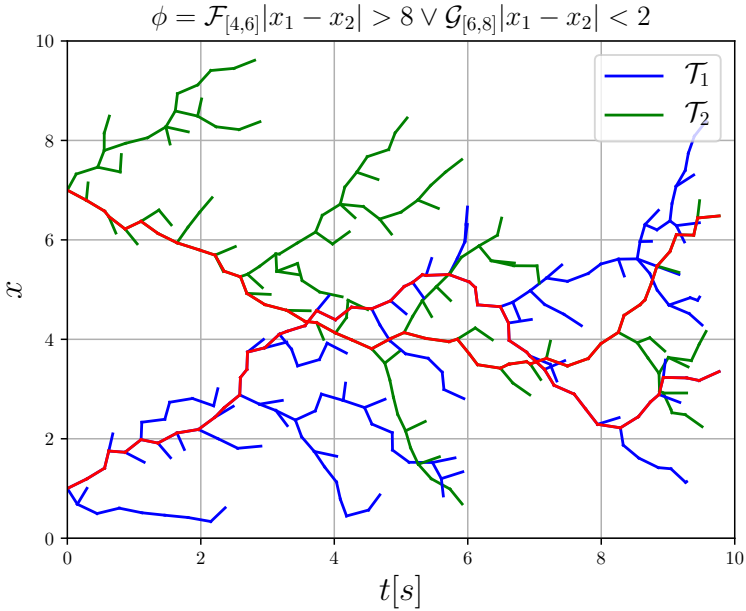
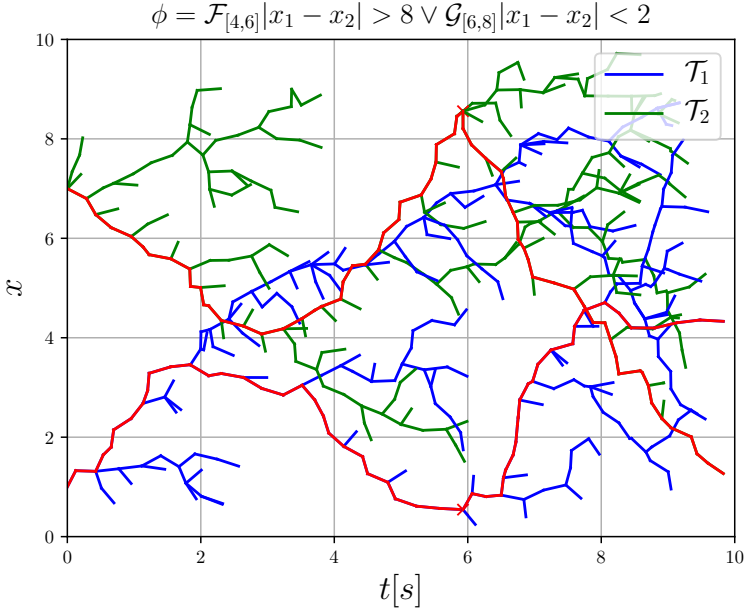
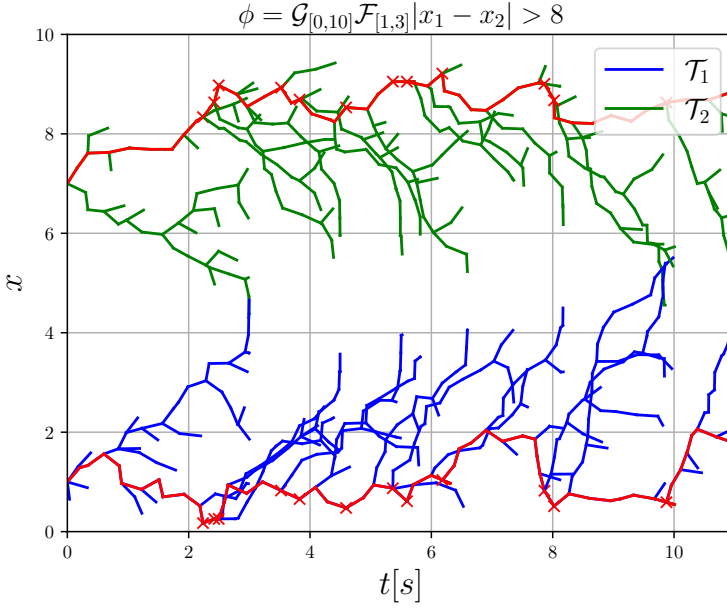
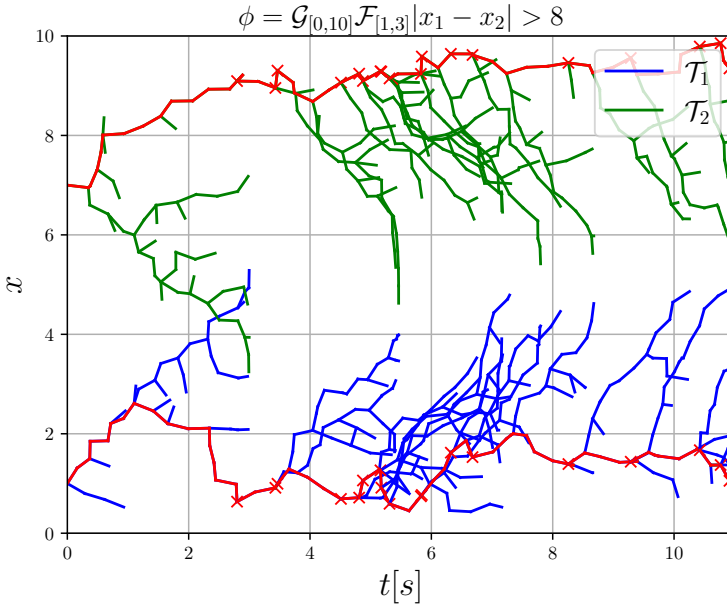


Figure 3.12: Two examples runs of $\phi = \mathcal{F}_{[4,6]}(|x_1 - x_2| > 8) \vee \mathcal{G}_{[6,8]}|x_1 - x_2| < 2$.



(a) Example run 1



(b) Example run 2

Figure 3.13: Two examples runs of $\phi = \mathcal{G}_{[0,10]}\mathcal{F}_{[1,3]}(|x_1 - x_2| > 8)$.

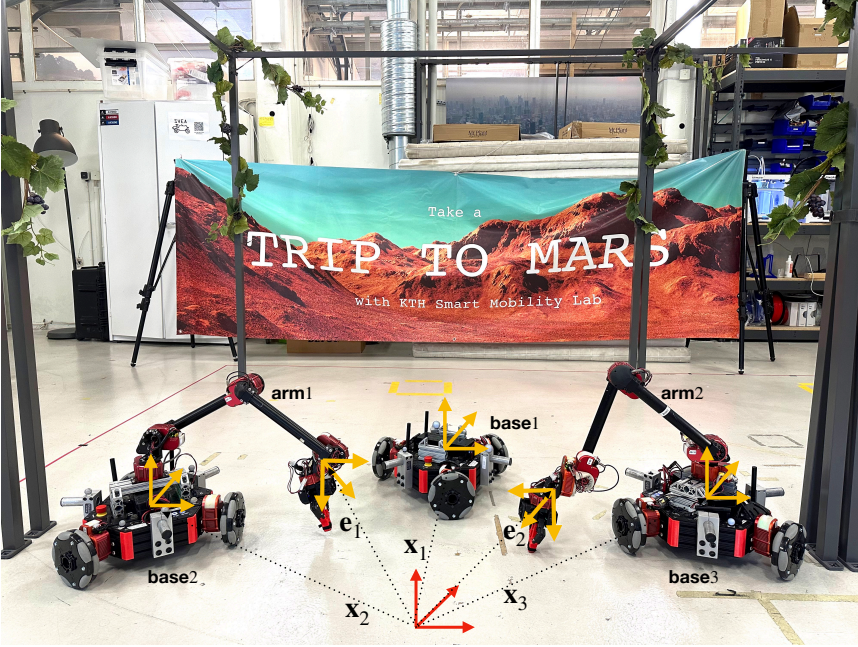


Figure 3.14: Experimental setup with three mobile bases and two 6-dof manipulators

need for approximations of max/min operators and samples in continuous time to avoid abstractions.

- It incorporates a wide range of coupled constraints (both linear and nonlinear) into the distributed optimisation framework, enabling the handling of diverse tasks such as pick-and-place operations and time-varying activities like trajectory tracking.
- We present extensive simulation and hardware experiments (on the setup shown in Figure 3.14) that demonstrate the execution of complex tasks using MAPS².

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

STL Decomposition

In this section, we present how to retrieve spatial and temporal constraints from a given STL formula φ of the form 1.1.

Spatial Constraints:

The spatial constraints in an STL formula are encoded in the predicate functions $h(\mathbf{x}(t))$ defined in (1.2). Such predicate functions can enforce constraints on an individual robot or coupled constraints on multiple robots. A robot i with states $\mathbf{x}_i \in \mathcal{W}_i$ is subject to independent state constraints $\mathbf{x}_i \in \mathcal{X}_i \subseteq \mathcal{W}_i$ and coupled state constraints $\mathbf{x}_i \in \mathcal{X}_{ij} \subseteq \mathcal{W}$ for any $j \in \{1, \dots, \mathbf{N}\} \setminus i$. The corresponding predicate functions are defined by the inequalities

$$\begin{aligned} \alpha h_i(\mathbf{x}_i) &\leq 0, \quad \alpha \in \{1, 2, \dots, r_i\} \\ \beta h_{ij}(\mathbf{x}_i, \mathbf{x}_j) &\leq 0, j \in \{1, \dots, \mathbf{N}\} \setminus i, \beta \in \{1, 2, \dots, s_i\} \end{aligned} \quad (3.9)$$

where αh_i and βh_{ij} are continuous functions; $\alpha h_i : \mathcal{W}_i \rightarrow \mathbb{R}$ specifies constraints on robot i and r_i is the number of such constraints, and, $\beta h_{ij} : \mathcal{W}_i \times \mathcal{W}_j \rightarrow \mathbb{R}$ specifies coupled constraints between robots i and j with s_i being the number of such constraints. The state constraint sets are then defined as,

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

We further consider that $\beta h_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \beta h_{ji}(\mathbf{x}_j, \mathbf{x}_i)$, which states that agent i and j receive the same STL formula. The inequalities αh_i and βh_{ij} are predicate function inequalities of the form (1.2). A predicate function inequality $h(\mathbf{x}_i(t)) \leq 0$ corresponds to state constraint $\alpha h_i(\mathbf{x}_i) \leq 0$ and $h(\mathbf{x}_i(t), \mathbf{x}_j(t)) \leq 0$ corresponds to the state constraint $\beta h_{ij}(\mathbf{x}_i, \mathbf{x}_j) \leq 0$. We also make the following assumption:

Assumption 7 *If robot i and j share a coupled constraint, then they can continuously communicate with each other.*

The coupled state constraints βh_{ij} then define an undirected communication graph over the multi-robot system. We further denote by $\mathcal{N}_i$ the set of all such robots that share a constraint with agent i . For robot i , cast the constraints (3.9) into the cost function F^i as

$$F^i := \sum_{\alpha=1}^{r_i} \frac{1}{2} \max \left(0, \alpha h_i \right)^2 + \sum_{\beta=1}^{s_i} \frac{1}{2} \max \left(0, \beta h_{ij} \right)^2. \quad (3.10)$$

Observe that $F^i : \mathcal{W} \rightarrow \mathbb{R}_{\geq 0}$ and $F^i = 0$ if and only if all the constraints in (3.9) are satisfied. Then, enforcing conditions (3.9) is equivalent to finding $\mathbf{x}_i$ for a given $\mathbf{x}_j$ ($j \in \mathcal{N}_i$) such that $F^i = 0$. This problem can be posed as,

$$\min_{\mathbf{x}_i \in \mathcal{W}_i} F^i \quad (3.11)$$

whose solution $\mathbf{x}_i^*$ satisfies $F^i(\mathbf{x}^*) = 0$. In the cost function (3.10), to reduce computational costs, we only minimise $h(\mathbf{x}(t))$ when $h(\mathbf{x}(t)) > 0$ while leaving

$h(\mathbf{x}(t)) \leq 0$ unchanged. This leads us to minimise: $F^i = \max(0, h(\mathbf{x}(t)))$, which results in $F^i = h(\mathbf{x}(t))$ when $h(\mathbf{x}(t)) > 0$ and $F^i = 0$ when $h(\mathbf{x}(t)) \leq 0$. Additionally, squaring the function penalises larger errors more than smaller ones. Other cost functions that enforce $h(\mathbf{x}(t)) \leq 0$ within the validity domain can also be considered. For example, the cost function

$$F^i := \sum_{\alpha=1}^{r_i} \alpha h_i^2 + \sum_{\beta=1}^{s_i} \beta h_{ij}^2$$

could also be used. However, it was not our first choice, as we aimed to minimise $h(\mathbf{x}(t))$ only when $h(\mathbf{x}(t)) > 0$, whereas this cost function would attempt to minimise $h(\mathbf{x}(t))$ regardless of its sign. Additionally, our formulation is general and works for any type of STL formula as any type of objectives can be encoded in the STL formula, from which, we can extract the predicate functions and minimise the function F^i .

The solution for finding the global minimum of a nonconvex function is a subject of extensive research. We argue that employing gradient descent with random initialisations is adequate for addressing this problem, particularly since the initialisations are sampled from a compact set, $\mathcal{W}_i$. Furthermore, using the knowledge that the minimum of the function, $F^i(\mathbf{x}) = 0$, acts as a stopping criterion and facilitates the attainment of the desired solution. We direct readers to the seminal work in [99], which presents a distributed gradient descent algorithm for multi-agent systems. Additionally, under certain assumptions, [100] demonstrates that gradient descent with a constant step size avoids entrapment at saddle points. Gradient descent is also shown to efficiently manage most reach-avoid constraints without the need for re-initialisation, given that such constraints are expressible using norms. For our application of gradient descent, we utilise the function presented in Function 6.

Function 6 implements the gradient descent algorithm as described in Algorithm 9.3 of [101], utilising initial conditions $\mathbf{x}_i$, step size δ , maximum number of iterations L' , and activation variables λ_{ij} as inputs. The activation variables are presented later in Function 8. It returns the optimised states $\mathbf{x}_i^*$ as output. In line 8, the function `GradientComputation()` computes the gradient, either analytically or numerically. The stopping criterion is met either when a feasible state is determined, indicated by $F^i = 0$, or when the iteration count exceeds L' (line 11), which may occur due to multiple conflicting predicates active within F^i . This situation arises because the algorithm accounts for the possibility that the *eventually* operator may not be satisfied at every sampled point within its validity domain. This occurs, for example, if $\varphi = \mathcal{F}_{[0,5]}\mathcal{G}_{[0,5]}\mu_1 \wedge \mathcal{G}_{[5,10]}\mu_2$, and there is a conflict between μ_1 and μ_2 (i.e., $\nexists \mathbf{x}_i^* \in \mathcal{W}_i$ such that $\mu_1(\mathbf{x}_i^*) \leq 0 \wedge \mu_2(\mathbf{x}_i^*) \leq 0$). In such cases, it becomes necessary for μ_1 to be true exclusively within the interval $[0, 5][s]$ and for μ_2 to be true exclusively within the interval $[5, 10][s]$.

The robots solve their respective optimisation problem cooperatively in a distributed manner via inter-neighbour communication. This makes the problem dis-

Function 6: DistributedOptimisation

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

Output: $\mathbf{x}_i^*$

```

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

tributed, as every interaction between robots is part of the communication graph. Given the nature of the optimisation problem, there is a trade-off between robustness and optimisation performance since $\mathbf{x}^*$ converges to the boundaries imposed by the STL formula constraints, making it vulnerable to potential perturbations. However, introducing a slack variable into the equation can enhance robustness, albeit at the cost of sacrificing completeness arguments. The example below shows how to construct the optimisation functions F^i .

Example 6 Consider a system with 3 agents and the corresponding states $\{x_1, x_2, x_3\}$, and let the STL formula be:

$$\varphi = (\|x_1 - x_2\| > 5) \wedge (\|x_2 - x_3\| < 2);$$

then, the functions F^i , for $i \in \{1, 2, 3\}$, are,

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

In this section, our focus lies on the planning aspect rather than specifically on the tools to solve (3.11). While we employ gradient descent to seek a solution in this article, it's worth noting that other optimisation methods could also be utilised by capitalising on the information that $F^i(\mathbf{x}_i^*) = 0$. Now that spatial constraints are encoded into the optimisation problem, we are ready to encode temporal constraints

in the following section, thus completing our STL decomposition into spatial and temporal constraints.

Temporal Constraints:

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

Definition 4 *The validity domain $\text{vd}(\bar{\varphi})$ of each path $\bar{\varphi}$ of an STL formula φ , is recursively defined as*

$$\text{vd}(\bar{\varphi}) = \begin{cases} 0, & \text{if } \bar{\varphi} = \mu^h \\ \text{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \neg \bar{\varphi}_1 \\ [a, b], & \text{if } \bar{\varphi} = \mathcal{G}_{[a,b]} \mu^h \\ a \oplus \text{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \mathcal{G}_{[a,b]} \bar{\varphi}_1, \bar{\varphi}_1 \neq \mu^h \\ t^* + T^* \oplus \text{vd}(\bar{\varphi}_1), & \text{if } \bar{\varphi} = \mathcal{F}_{[a,b]} \bar{\varphi}_1 \end{cases} \quad (3.12)$$

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

The above definition of t^* is necessary due to the redundancy of the eventually operator; we must ascertain the specific instances where the eventually condition is met to ensure finding a feasible trajectory. Additionally, we need to maintain the history of T^* for nested temporal operators which require recursive satisfaction. The validity domain is determined for each path of an STL formula in a hierarchical manner, beginning at the root of the tree, and each path has a distinct validity domain. The number of leaf nodes in an STL formula is equal to the total number of validity domains. In Definition 4, we do not include the operators $\wedge$ and $\vee$ because they do not impose temporal constraints on the predicates and thus inherit the validity domains of their parent node. If there is no parent node, operators $\wedge$ and $\vee$ inherit the validity domains of their child node.

Remark 4 *The validity domain is specially defined in the following cases. If a path contains only predicates, the validity domain of μ^h is equal to the time horizon of φ (i.e., $\text{vd}(\mu^h) = \text{th}(\varphi)$). Furthermore, if a path contains nested formulas with the same operators, such as $\bar{\varphi} = \mathcal{G}_{[1,10]} \mathcal{G}_{[0,2]} \mu^h$, then the validity domain of $\bar{\varphi}$ is equal to the time horizon of the path (i.e., $\text{vd}(\bar{\varphi}) = \text{th}(\bar{\varphi})$). For example, $\text{vd}(\mathcal{G}_{[1,10]} \mathcal{G}_{[0,2]} \mu^h) = \text{th}(\bar{\varphi}) = [1, 12]$.*

Example 7 *Consider the following examples of the validity domain:*

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

Regarding the STL formula in equation (3.2), the validity domains are defined for the following paths: $\mathcal{F}_{I_1}\mu^{h^1}$, $\mathcal{F}_{I_1}\mathcal{G}_{I_2}\mu^{h^2}$, $\mathcal{G}_{I_3}\mathcal{F}_{I_4}\mu^{h^3}$, and $\mathcal{G}_{I_5}\mu^{h^4}$.

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

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

Proof 4 The proof follows from the construction of the optimisation function (3.10) and the validity domain. Notice that if the optimisation problem (3.11) converges to the desired minima at $F^i(\mathbf{x}) = 0$, then $\mu^{\alpha h_i} = \top$ and $\mu^{\beta h_{ij}} = \top$ for all $\alpha \in \{1, \dots, r_i\}$ and $\beta \in \{1, \dots, s_i\}$. Next, by definition, the validity domain is defined for the STL formula and if F^i is minimised during the validity domain, then $\mathbf{y}(t) \models \varphi$.

In the next Section, we present how to integrate the validity domain with the optimisation problem in (3.11), completing thus the spatial and temporal integration.

Main Results

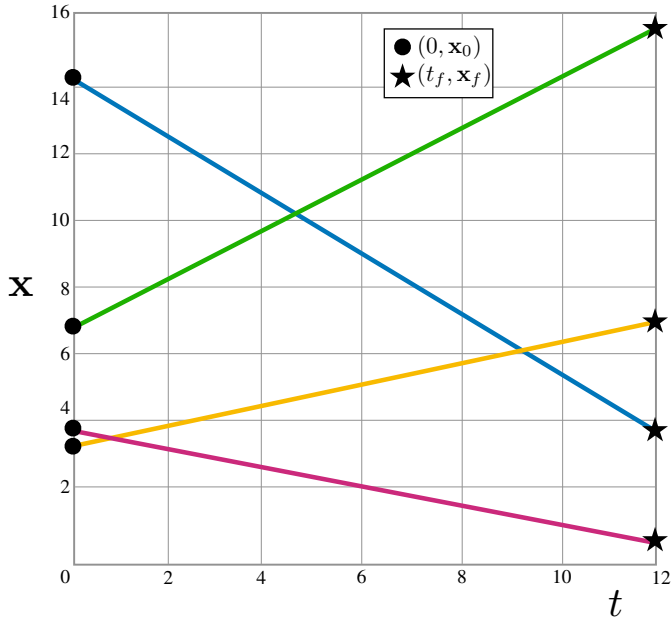
In this section, we present the algorithm for generating continuous trajectories that meet the requirements of a given STL formula φ . The algorithm is executed by the robots offline in a distributed manner, in the sense that they only communicate with their neighbouring robots. The algorithm builds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

In what follows, we give a high-level description of the algorithm. The general idea is to start with an initial trajectory that spans the time horizon of the formulas $\text{th}(\varphi)$, then repeatedly sample random points along the trajectory and use gradient-based techniques to find solutions that satisfy the specification at these points. More specifically, the algorithm begins by connecting the initial and final points $z_0^i = \{0, \mathbf{x}_0^i\}$ and $z_f^i = \{t_f^i, \mathbf{x}_f^i\}$ with a single edge $\mathcal{E}_i = \{(z_0^i, z_f^i)\}$. The initial conditions $z_0^i = \{t_0^i, \mathbf{x}_0^i\}$ depend on the robot's initial position and time. The final conditions are chosen to be $z_f^i = \{\text{th}(\varphi) + \epsilon, \mathbf{x}_f^i\}$ where $\epsilon > 0$ and $\mathbf{x}_f^i \in \mathcal{W}_i$. The final states $\mathbf{x}_f^i$ can be randomly chosen since the states in the interval $[0, \text{th}(\varphi)]$ will be determined by the algorithm based on the constraints imposed by φ . The algorithm then randomly selects a time instant $t^0 \in [0, \text{th}(\varphi)]$ and uses linear interpolation to determine the states of each robot at that time, denoted by $\mathbf{x}^0$. The robots then solve the distributed optimisation problem (3.11) to find new positions $\mathbf{x}^*$ that meet the specification at time t^0 . The algorithm then repeats this process at a user-specified time density, updating the trajectories as necessary. The result is a trajectory that asymptotically improves the task satisfaction of the STL formula.

Example 8 Before we get into the technical details, let us consider an example of 4 agents, represented by the colours blue, green, yellow and magenta, to illustrate the procedure. Suppose, at a specific instance in time, say t^0 , the STL formula requires agent 1 (blue) and agent 2 (green) to be more than 6 units apart and agent 3 (yellow) and agent 4 (magenta) to be closer than 6 units i.e., for $\epsilon > 0$,

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

We begin the process by connecting the initial and final points z_0^i and z_f^i with an initial trajectory for all agents, as shown in Figure 3.15a. Each agent's vertex set is $\mathcal{V}_i$ and consists of the start and end points denoted by z_0^i and z_f^i respectively, while its edge set $\mathcal{E}_i$ contains only one edge connecting the start and end points. From the initial trajectory, the algorithm randomly selects a point at time instance t^0 from the entire time domain and uses linear interpolation to determine the state of each agent at that time. The agents solve (3.11) using the initial position $\mathbf{x}^0$ to find new position $\mathbf{x}^*$, as seen in Figure 3.15b. As shown in Figure 3.16a, the distributed optimisation problem (3.11) is solved, resulting in a solution $\mathbf{x}^*$, in which agent 1



(a) Initial trajectory

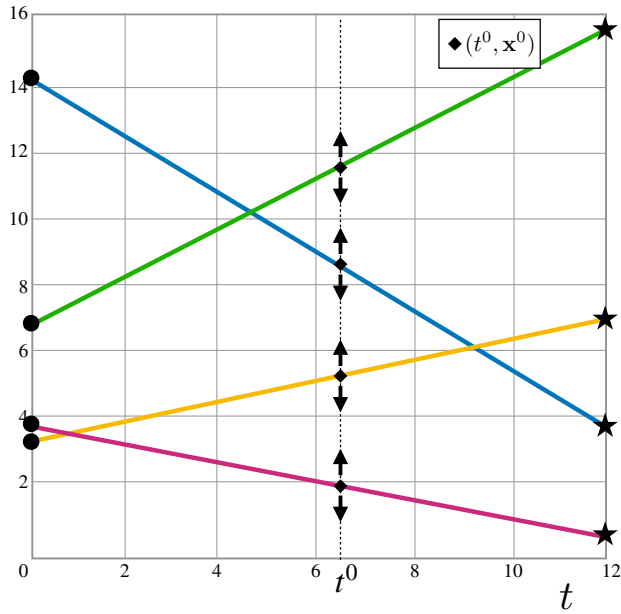
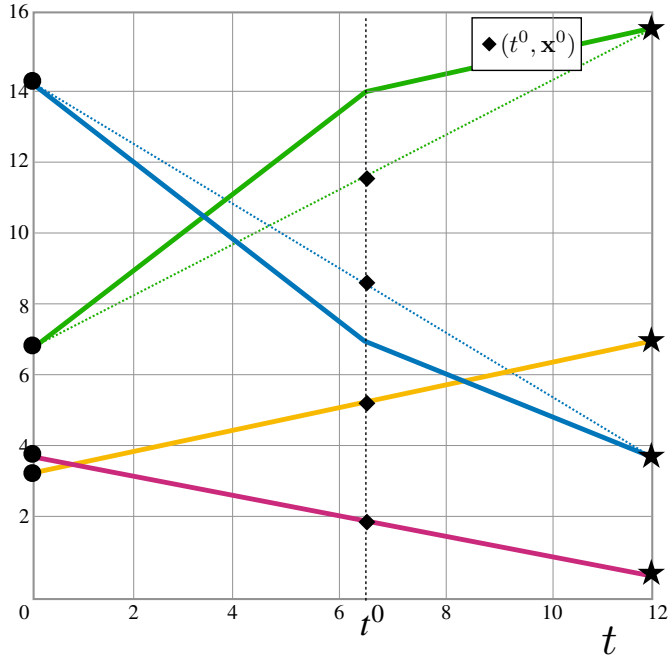
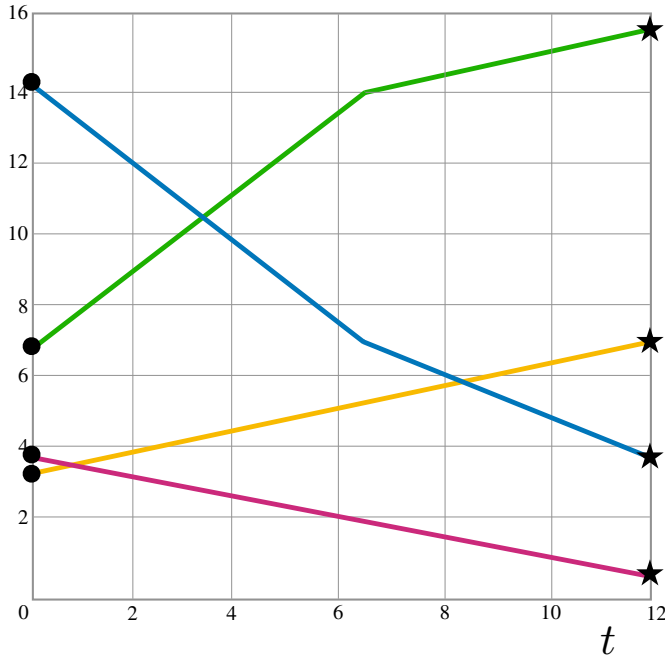
(b) A random time instance t^0

Figure 3.15: Illustration of the proposed algorithm.



(a) Optimal solution $\mathbf{x}^*$



(b) Shaped trajectory

Figure 3.16: Illustration of the proposed algorithm.

and agent 2 are positioned so that they are more than 6 units apart and agent 3 and agent 4 remain undisturbed. The latter is the result of using functions of the form $1/2 \max(0, h_{ij})^2$, and since agent 3 and agent 4 already satisfy the requirements, i.e., $h_{34} < 0$, the function is valued 0. The newly determined positions of agents 1 and 2 are added to the tree, allowing the trajectory to be shaped to meet the requirements. The updated trajectory can be seen in Figure 3.16b. This process of randomly selecting a point in time, determining the state of the agents and updating their positions is repeated for a user-defined number of times L , to ensure that the trajectory satisfies the STL formula φ throughout the time horizon.

The architecture of the algorithm MAPS² (short for ‘**m**ulti-robot **a**n anytime motion **p**lanning under signal temporal logic specifications’), is depicted in Figure 3.17.

The algorithm, outlined in Algorithm 7, begins with an initial trajectory connecting z_0^i and z_0^f , along with a random seed and design constants as input (see lines 1-3). The random seed ensures that all robots select the same time instance. The algorithm proceeds by repeatedly sampling a time instance within the interval, interpolating states at the said time instance, applying gradient descent to minimise the function (3.10), and either adding or discarding the resulting optimal solution. This process is repeated until the total number of vertices, L , is reached, see lines 5-14. This is also illustrated in Figure 3.15a-3.16b.

These steps are implemented as follows: In line 7, the `SearchSort()` function separates the vertices $\mathcal{V}_i$ into two sets based on their time values: one set with time values lower than t^0 (the vertex with the highest time in this set is indexed with ‘index’), and another with values greater than t^0 (the vertex with the lowest time in this set is indexed with ‘index + 1’). The corresponding vertices are $z_{\text{index}}^i = \{t_{\text{index}}^i, \mathbf{x}_{\text{index}}^i\}$ and $z_{\text{index}+1}^i = \{t_{\text{index}+1}^i, \mathbf{x}_{\text{index}+1}^i\}$. Then, the algorithm uses linear interpolation in line 8 via the function `Interpolate()` to obtain the vertex $z_{\text{inter}}^i = \{t^0, \mathbf{x}_{\text{inter}}^i\}$. This is obtained by solving for $\mathbf{x}_{\text{inter}}^i$ element-wise as the solution of

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

The vertex z_{inter}^i is the initial condition to solve the optimisation problem (3.11); and once a solution z_{opt}^i is obtained, it is added to the vertex set $\mathcal{V}_i$ in line 10. The edge set $\mathcal{E}_i$ is reorganised to include z_{opt}^i in lines 11-13. Moreover, as a safeguard, if a solution remains undiscovered following L iterations, line 16 initiates a reset procedure. This involves setting the satisfaction variable for all *eventually* operators back to -1 and restarting the search. Since we assume that at least one viable solution always exists (refer to Assumption 6), the absence of a solution occurs solely when an *eventually* operator is satisfied at an impractical instance of time. Such an impractical instance of time affects the solution of the algorithm since there are redundancies in picking the satisfaction instance ($\mathcal{F}_{[a,b]}\mu \models \varphi$ if $\mu = \top$ at any single instance in $[a, b]$). By resetting these operators, the algorithm aims to locate a solution under feasible instances.

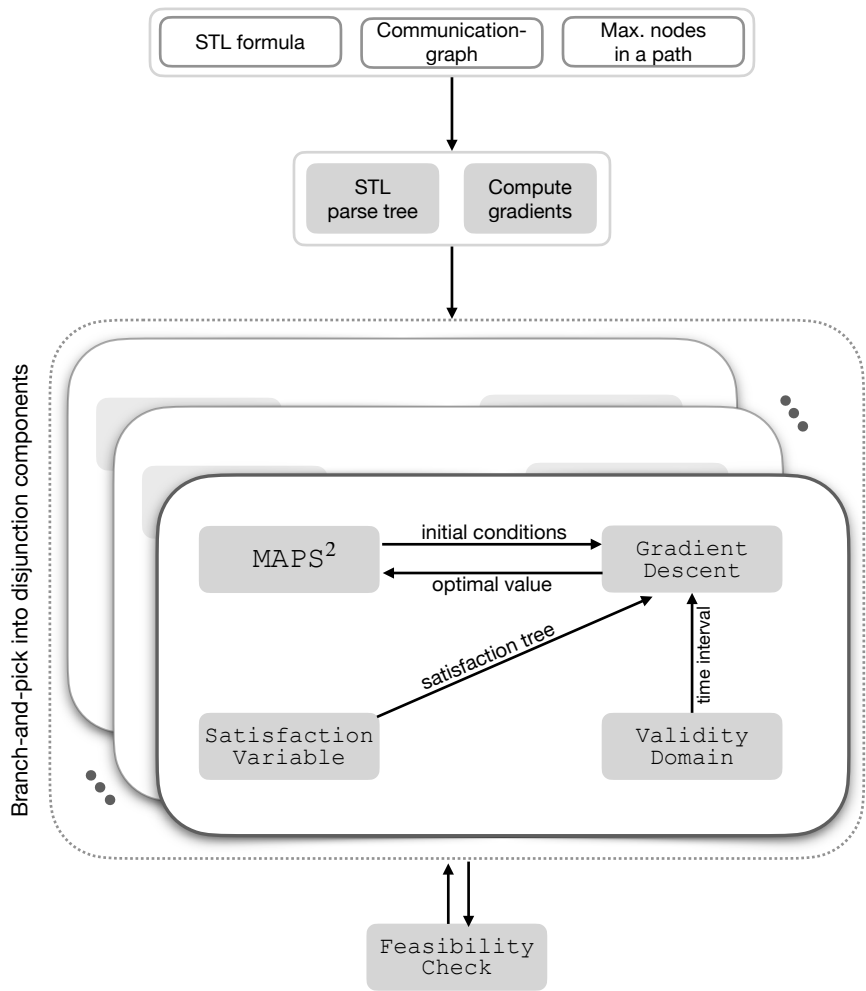


Figure 3.17: Architecture of the provided algorithm

Algorithm 7: MAPS²

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

Output: $\mathcal{T}_i$

```

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

- **GradientDescent:** The function is presented in Function 8 and computes the optimal value, z_{opt}^i , by solving the problem presented in equation (3.11). This allows the robots to compute vertices that locally do not violate the STL formula. Once z_{opt}^i is determined through Function 6, the satisfaction variables are updated in Function 9.

Based on the validity domain, the Function 8 determines which predicate functions are active in (3.10) at every sampled time instance t^0 . The Function `ValidityDomain()` in line 3 calculates the validity domains of each path $\bar{\varphi}$ based on Definition 4. Among the set of predicate functions $\{h_{ij} | \forall j \in \mathcal{N}_i\}$ associated with a robot i , a binary variable $\lambda_{ij} \in \{0, 1\}$ is assigned to determine whether a predicate function is active or not. It is set to 1 if the predicate is active and 0 otherwise. For example,

- If $\varphi_1 = \mathcal{G}_{[5,10]}(\|x_1 - x_2\| \leq 2)$, then $\lambda_{12} = 1$ whenever $t^0 \in [5, 10]$ and 0 otherwise.
- If $\varphi_2 = \mathcal{F}_{[10,15]}(\|x_3\| \leq 5)$, then $\lambda_3 = 1$ whenever $t^0 \in [10, 15]$ and 0 otherwise. Once $x_3(t) \models \varphi_2$, $\lambda_3 = 0 \forall t$.

We distinguish three cases: if the sampled point belongs to the validity domain

Function 8: GradientDescent

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

Output: z_{opt}^i, τ

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

of a single *eventually* operator and/or a single *always* operator, $\lambda_{ij} = 1$. If the sampled point belongs to the validity domain of multiple *eventually* operators, we activate only one of them at random, that is, $\lambda_{ij} = 1$ only for one of them. This avoids enforcing conflicting predicates as it can happen that multiple *eventually* operators may not be satisfied at the same time instance (For example $\varphi = \mathcal{F}_{[0,1]}(x > 0) \wedge \mathcal{F}_{[0,1]}(x < 0)$); see lines 6-13.

In lines 16-24, the algorithm updates the satisfaction variable of all paths in the STL formula that impose restrictions on robot i 's states. The algorithm goes bottom-up, starting from the **leaf** node to the **root** node. First, it determines if z_{opt}^i is the desired minimum (i.e., $F^i(\mathbf{x}_{opt}^i) \leq 0$) in line 18, and in lines 19-23, the algorithm updates the satisfaction variable of all nodes in the path $\bar{\varphi}$ through the function **SatisfactionVariable()**. If z_{opt}^i is not the desired minimum, then all the satisfaction variables of the path $\bar{\varphi}$ are reset to -1 in line 25. This could result from conflicting predicates at the same time instance.

• **SatisfactionVariable:**

Function 9: SatisfactionVariable	
Input: node, $z_{opt}^i = \{t^0, \mathbf{x}^i\}$	
Output: τ, t^*	
1	case ($node = \mathcal{F}_I$) do
2	$\tau(\mathcal{F}_I) = +1$;
3	$t^* = t^0$;
4	return τ, t^* ;
5	case ($node = \mathcal{G}_I$) do
6	if $\text{robust}(\mathcal{G}_I) \geq 0$ then
7	$\tau(\mathcal{G}_I) = +1$;
8	return τ, t^*
9	case ($node = \wedge$) do
10	$\tau(\wedge) = +1$;
11	return τ, t^*

This function, presented in Function 9, updates the *satisfaction variable tree*, τ . The aforementioned procedure decides if the satisfaction variable corresponding to each node listed is $+1$ (satisfied) or -1 (not yet satisfied). The discussion of handling disjunction operators is deferred to later as they are handled differently. Considering the premise that the predicate is true, as indicated in line 18 of Function 8, we evaluate the satisfaction variable as follows:

- $\mathcal{F}_I$: The satisfaction variable of the eventually operator is updated along with $t^* = t^0$. This updated t^* is used to determine the new validity

domains in line 3 of Function 8; see Example 3 for an illustration of this procedure.

- $\mathcal{G}_I$: Unlike the *eventually* operator, determining $\tau(\mathcal{G}_I)$ necessitates the computation of robustness over the entire validity domain of the operator. The function **robust**() uses the robust semantics of the STL presented in [6]. Particularly, it samples a user-defined number of points in the interval $\text{vd}_{ij}^G()$ and computes $\inf_{t \in \text{vd}_{ij}^G} h_{ij}(\mathbf{x}^i(t))$. If the robustness is non-negative, indicating satisfaction of the task, the value of $\tau(\mathcal{G}_I)$ is updated to +1.
- $\wedge$: This set node returns the satisfaction variable as +1 since it does not impose spatial or temporal restrictions.
- *Branch-and-Pick for Disjunctions*: In our approach, we address disjunctions as follows: Given an STL formula of the form $\varphi = \bigvee_{i \in 1, \dots, K} \phi_i$, which can also be represented as $\varphi = \vee(\phi_1, \phi_2, \dots, \phi_K)$, we divide it into K individual STL formulas. The agents then run Algorithm 7 separately for each $\varphi = \phi_i$, where $i \in 1, \dots, K$. For instance, consider the STL formula represented as (3.2) of Example 3,

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

We branch it into two STL formulas:

$$\phi_1 = \mathcal{F}_{I_1} \mu_1 \wedge \mathcal{G}_{I_3} \mathcal{F}_{I_4}(\mu_3) \wedge \mathcal{G}_{I_5}(\mu_4)$$

and

$$\phi_2 = \mathcal{F}_{I_1} \mathcal{G}_{I_2}(\mu_2) \wedge \mathcal{G}_{I_3} \mathcal{F}_{I_4}(\mu_3) \wedge \mathcal{G}_{I_5}(\mu_4),$$

as illustrated in Figure 3.18. The search terminates when any branch of the disjunction satisfies the condition $\tau(\text{root}) \neq +1$, as specified on line 5 of Algorithm 1. We acknowledge that this naive method of handling disjunctions can result in exponential growth with the addition of more operators. An alternative approach, akin to the branch-and-bound method from optimisation [102], involves evaluating the robustness of each ϕ_i for $i \in 1, \dots, K$ and executing MAPS² only for the formulas that show a faster increase in satisfaction. However, this strategy might necessitate a higher level of communication among robots which goes beyond their existing communication network and possibly require a central authority to coordinate task fulfilment. For example, the STL formula

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

comprises disjunction between $\phi_1 = \mathcal{G}_{[0,5]}(x_1 < 5)$ and $\phi_2 = \mathcal{F}_{[0,5]}(|x_2 - x_3| > 2)$. Observe that ϕ_1 requires no inter-robot communication, while ϕ_2

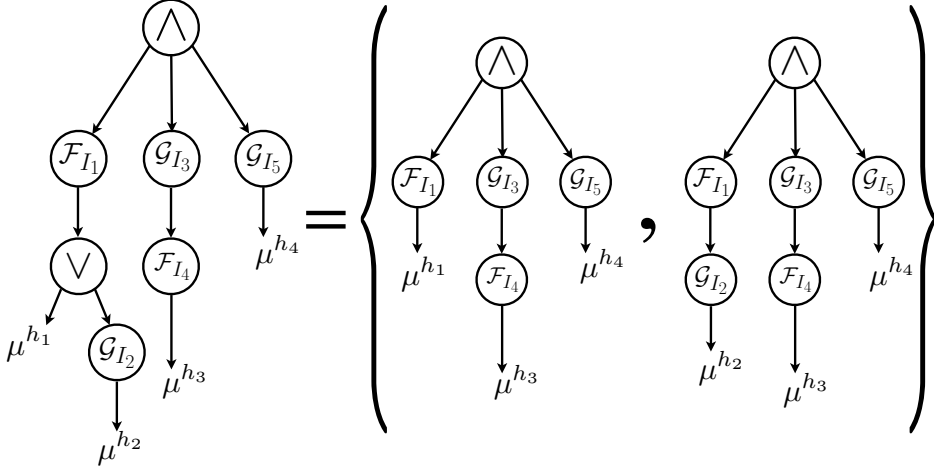


Figure 3.18: Disjunction representation for disjunctive components using STL parse tree.

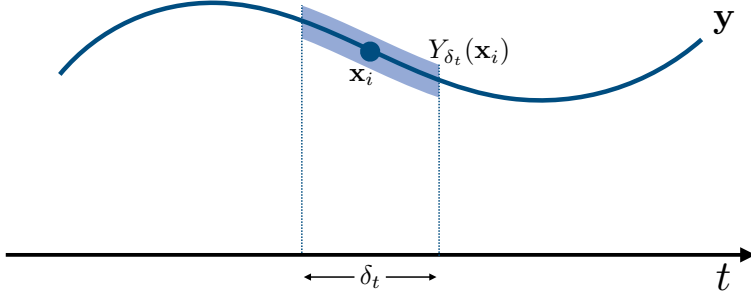
necessitates communication between robots 2 and 3. In the implementation of a method akin to branch-and-bound, we would branch into two formulas, ϕ_1 and ϕ_2 , and repeatedly switch between them if we observe the robustness of one formula decaying faster compared to the other. This switching must be performed by a central authority that observes the decay in robustness. If the switching is decided among the robots, then robot 1 of ϕ_1 needs to communicate the robustness decay with the network of robots 2 and 3. This requires robot 1 to establish communication with the network of robots 2 and 3 in order to decide which branch to grow, thereby necessitating communication links where none existed before. Without such a communication link, both ϕ_1 and ϕ_2 would need to be satisfied using the naive approach presented in our work. This motivates our choice to use the naive approach.

Analysis

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

Let the set $\mathcal{S} \subseteq \mathcal{W}$ be a compact set where a trajectory $\mathbf{y} : [0, \text{th}(\varphi)] \rightarrow \mathcal{S}$ satisfies the STL formula. Along the lines of [103], let a trajectory $\mathbf{y}$ be located on the boundary of the set $\mathcal{S}$, the satisfiable set, dividing $\mathcal{W}$ into a feasible set $\mathcal{S}$ and an infeasible set $\mathcal{W} \setminus \mathcal{S}$.

Starting with an initial linear trajectory in the augmented time-space domain, each uniformly sampled time point t^0 corresponds to a position $\mathbf{x}_{\text{inter}}$ either in $\mathcal{S}$ or $\mathcal{W} \setminus \mathcal{S}$. If $\mathbf{x}_{\text{inter}} \in \mathcal{S}$, we leave it unchanged as it meets the requirements. But


 Figure 3.19: Illustration of $Y_{\delta_t}(\mathbf{x}_i)$

if $\mathbf{x}_{\text{inter}} \notin \mathcal{S}$, we use gradient descent to reach a point on $\mathbf{y}$, since it lies on the boundary of the constraints' set.

Next, divide the trajectory $\mathbf{y} : [0, \text{th}(\varphi)] \rightarrow \mathcal{S}$ into $L + 1$ points $\mathbf{x}_i$, where $0 \leq i \leq L$ and $\mathbf{y}(\text{th}(\varphi)) = \mathbf{x}_f = \mathbf{x}_L$ by dividing the time duration into equal intervals of δ_t . Without loss of generality, assume that the points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are separated by δ_t in time. With $L\delta_t = \text{th}(\varphi)$, the probability of sampling a point in an interval of length δ_t can be calculated as $p = \frac{\delta_t}{\text{th}(\varphi)}$. If $\delta_t \ll \text{th}(\varphi)$, then $p < 1/2$. Denote the sequential covering class¹ of trajectory $\mathbf{y}$ as $Y_{\delta_t}(\mathbf{x}_i)$. The length of $Y_{\delta_t}(\mathbf{x}_i)$ is δ_t in the time domain and is centered at $\mathbf{x}_i$. See Figure 3.19 for reference. A trial is counted as successful if we sample a point t^0 within the interval $\delta_t/2$ on either side of $\mathbf{x}_i$, that is, within $Y_{\delta_t}(\mathbf{x}_i)$. If there are L successful trials, the entire trajectory $\mathbf{y}$ is covered, and the motion planning problem is solved. Consider k total samples, where $k \gg L$, and treat this as k Bernoulli trials with success probability p since each sample is independent with only two outcomes. We are now ready to state the following lemma.

Lemma 2 *Let a constant L and probability p such that $p < \frac{1}{2}$. Further, let k represent the number of samples taken by the MAPS² algorithm. Then, the probability that MAPS² fails to sample a segment after k samples is at most $\frac{(k-L)p}{(kp-L)^2}$.*

Proof 5 *The probability of not having L successful trials after k samples can be expressed as:*

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

and according to [104], if $p < \frac{1}{2}$, we can upper bound this probability as:

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

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

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

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

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

Proof 6 *The algorithm initiates by setting all satisfaction variables, τ , to -1 , as inputs to Algorithm 7. These variables are updated in Function 9 designed for evaluating whether τ meets the satisfaction criteria. The function adjusts τ in accordance with the definition of STL operators, ensuring that updates accurately reflect the satisfaction status. Furthermore, the update to $\tau(\text{leaf})$ within Function 8 (referenced at line 20) occurs only when the condition $F^i \leq 0$ is met. This condition indicates that all active predicates are satisfied by definition. Thus, no satisfaction variable is incorrectly updated.*

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

Theorem 4 *Algorithm 7 is probabilistically complete.*

Proof 7 *The proof follows from Lemmas 1, 2, and 3. From Lemma 1 and Lemma 3, we know that every sample added to the trajectory satisfies the STL formula. Thus, what needs to be shown is that the algorithm samples infinitely many times and covers the entire time horizon. From Lemma 2, we know that the probability of covering the entire time horizon is $1 - \mathbf{P}[X_k \leq L]$. Suppose the Algorithm 7 reaches $J = L'$ samples without finding a feasible solution, then it discards J samples as seen in line 16 of Algorithm 7. Given Assumption 6, we have $J < \infty$, and since J is the number of discarded samples, we also have $J \leq k$ where k is the total number of samples sampled so far (including the discarded ones). Thus, the probability of the trajectory satisfying the STL formula is $1 - \frac{((k-J)-L)p}{((k-J)p-L)^2}$, which approaches one as $k \rightarrow \infty$. Thus, the algorithm is probabilistically complete.*

Remark 5 *Our algorithm can be endowed in a post-processing stage with a module that smoothens the trajectory to avoid large accelerations. However, care needs to be taken since the smoothened paths may no longer satisfy the STL formula. One could also use more sophisticated approaches like B-splines to impose velocity and acceleration limits as shown in [105].*

Remark 6 *At present, our approach does not incorporate kinematic or dynamic constraints. Incorporation of such constraints could be attempted by either deploying the kinodynamic version of the RRT algorithm [106], or by using an existing low-level controller to track the generated open-loop trajectories. Some examples of such controllers include the Model Predictive Controller [107] and the input constrained Prescribed Performance Controller [108, 109]. This incorporation is by no means straightforward but requires fusion with another type of methodological machinery that goes beyond the scope of the current work. Moreover, such controllers have been developed for a large variety of dynamical systems and hence the proposed algorithm is practical and applicable to a large class of robots.*

Simulations

In this section, we present simulations of various scenarios encountered in a multi-robot system. Restrictions are imposed using an STL formula and MAPS² is utilised to create trajectories that comply with the STL formula. In the following we consider 4 agents, with $\delta = 0.1$, $\eta = 0.01$ and $L = L' = 100$. The simulations were run on an 8 core Intel[®] Core[™] i7 1.9GHz CPU with 16GB RAM. ²

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

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

where $\{i, j\} \in \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$. As depicted in Figure 3.20, all four agents maintain a distance of at least 1 unit from each other during the interval [20, 80][s]. The maximum computation time by any agent is 0.1143[s].

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

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

As seen in Figure 3.21, agents 1 and 3 and agents 2 and 4 approach each other within a distance of 1 unit during the specified interval. It's worth noting that the algorithm randomly selects the specific time t^* within the continuous interval [40, 60][s] at which the satisfaction occurs. The maximum computation time by any agent is 0.0637[s].

²The project code can be found here <https://github.com/sewlia/Maps2>

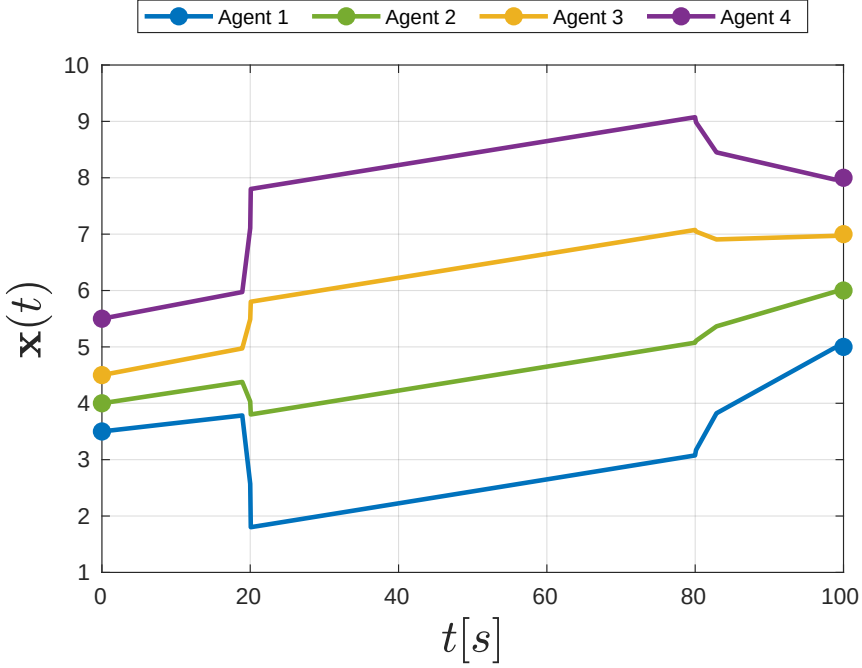


Figure 3.20: Collision Avoidance

- *Stability*: The next task is that of stability, which is represented by the STL formula $\mathcal{F}_{[a_1, b_1]} \mathcal{G}_{[a_2, b_2]} \mu$. This formula requires that μ must always hold within the interval $[t^* + a_2, t^* + b_2]$, where $t^* \in [a_1, b_1]$. This represents stability, as it requires μ to always hold within the interval $[t^* + a_2, t^* + b_2]$, despite any transients that may occur in the interval $[a_1, t^*]$. Figure 3.22 presents a simulation of the following STL formula:

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

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

- *Recurring Tasks*: The next scenario is that of recurring tasks. This can be useful when an autonomous vehicle needs to repeatedly survey an area at regular intervals, a bipedal robot needs to plan periodic foot movements, or a ground robot needs to visit a charging station at specified intervals. The STL formula to express such requirements is given by $\mathcal{G}_{[a_1, b_1]} \mathcal{F}_{[a_2, b_2]} \mu$, which reads as ‘beginning at $a_1[s]$, μ must be satisfied at some point in the interval

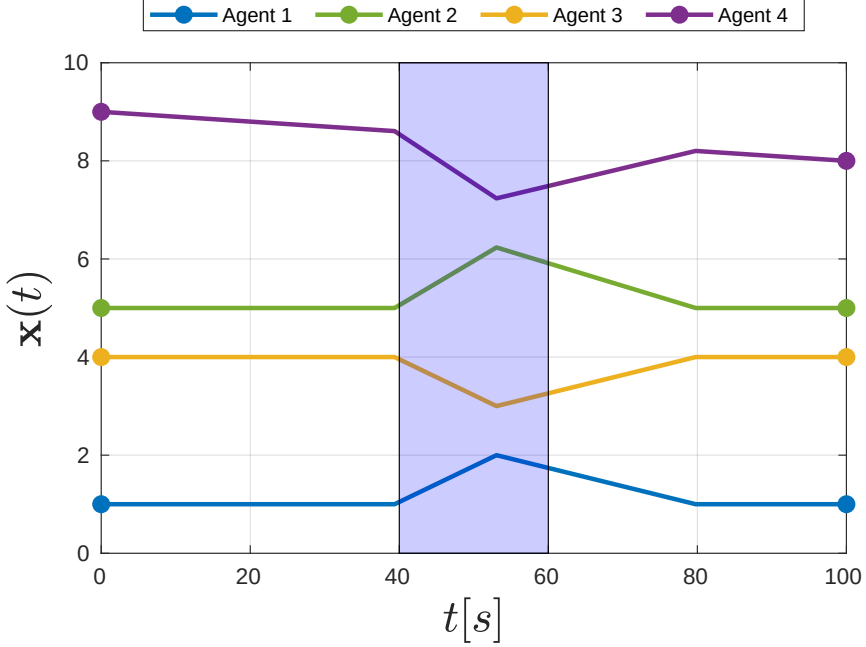


Figure 3.21: Rendezvous

$[a_1 + a_2, a_1 + b_2][s]$ and this should be repeated every $[b_2 - a_2][s]$.’ A simulation of the following task is shown in Figure 3.23:

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

Every 20[s], the condition $|x_1 - x_3| \leq 1$ is met. It’s worth noting that the specific time t^* at which satisfaction occurs is randomly chosen by the algorithm. The maximum computation time by any agent is 0.2017[s]. In reference to Remark 5, an example of post-processing the trajectories is shown in Figure 3.24 for the STL formula,

$$\varphi = \mathcal{G}_{[0,100]} \mathcal{F}_{[0,20]}(\|x_1 - x_2\| \leq 1). \quad (3.13)$$

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

- *Overall Case Study:* In this case study, we demonstrate the application of the aforementioned scenarios by setting up the following tasks:

- Agent 1 always stays above 8 units.

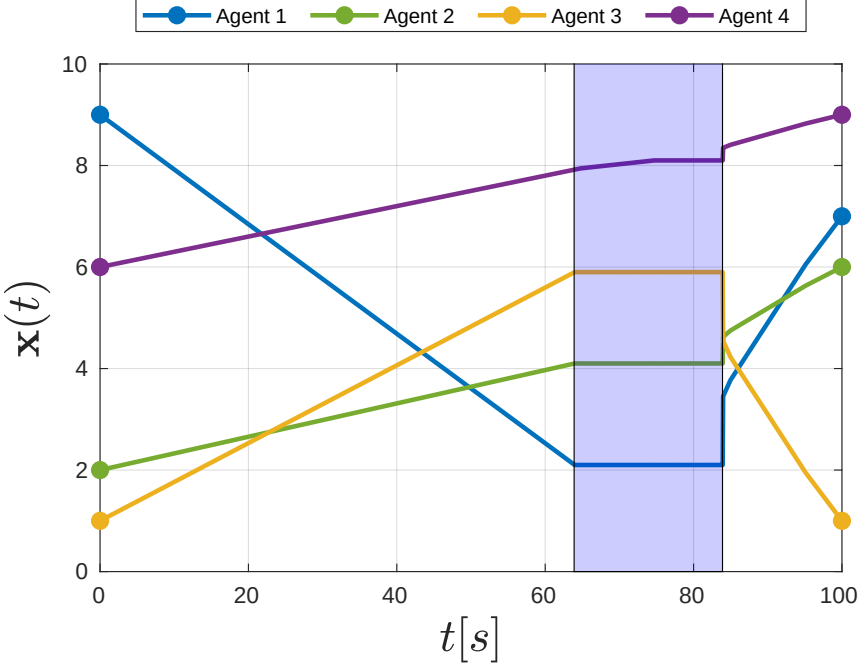


Figure 3.22: Stability

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

The STL formula for the above tasks is as follows:

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

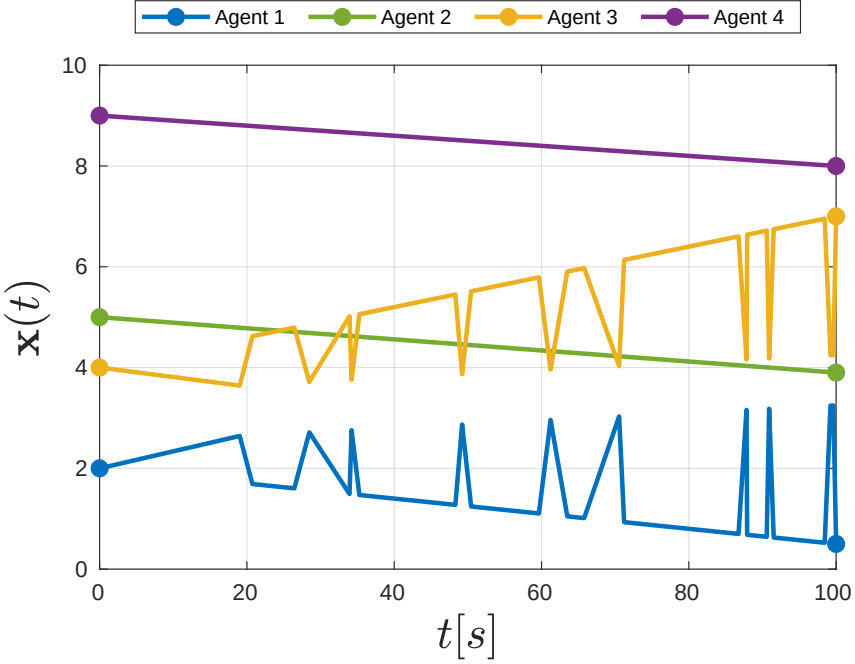


Figure 3.23: Recurring Tasks

The parameter L was increased to 1000, and η was decreased to 0.001. In Figure 3.25, we show the resulting trajectories of each agent generated by MAPS² satisfying the above STL formula. The maximum computation time by any agent is 4.611[s].

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

$$\varphi = \mathcal{G}_{[10,90]} \left[\|x_i - x_j\| \geq 0.01 \wedge \|x_i - (50, 50)\| \leq 5 \right] \quad (3.14)$$

$\forall i, j \in \{1, 2, \dots, 100\}$ and $i \neq j$. Note that the above STL formula has 5150 predicates. In the interval $[10, 90]$ [s], the STL formula dictates every agent to be at least 0.01[m] apart from every other agent and to be at least 5[m] close to the centre point (50, 50)[m]. The simulation results are shown in Figure 3.26-3.28 where the Figures 3.26a-3.28a are the trajectories before the start of the algorithm while Figures 3.26b-3.28b shows the trajectories at the end of $j = 1000$ iterations, as mentioned in Algorithm 7. The simulation took 17.84[s] to complete without parallelisation. The faster computation can be attributed to the nature of the design of the cost function in (3.11), which allows for

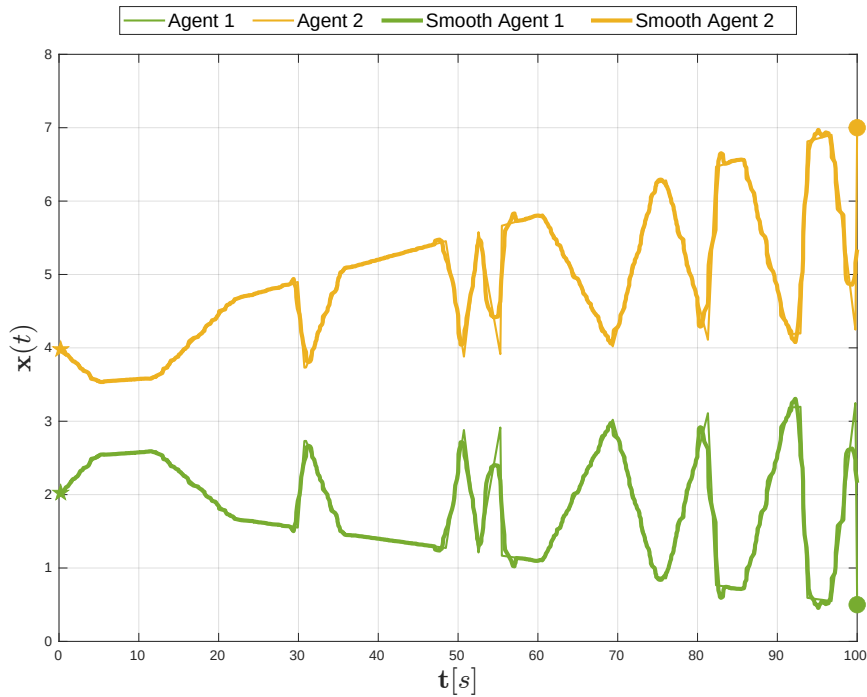


Figure 3.24: Non-smooth and smooth paths for the formula (3.13)

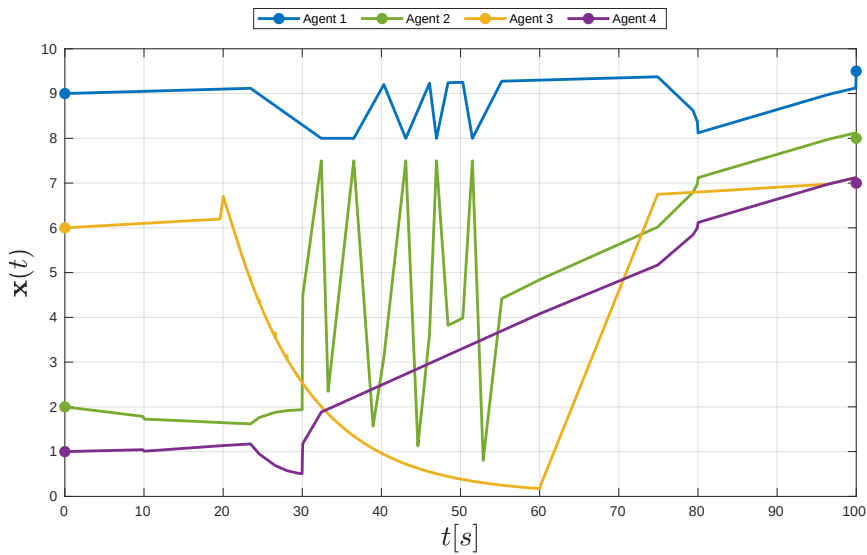


Figure 3.25: Overall Case Study

points that do not violate the formula not to be changed. The robustness of the STL formula is shown in Figure 3.29, a negative robustness signifies task satisfaction. Here, the robustness converges to 0, because the robustness for an *always* operator reflects the worst-case scenario. It is important to note that computing the result for Figure 3.29 required 12 hours and 10 minutes of computation time since it had to be performed centrally.

Experiments

This Section presents an experimental demonstration of the proposed algorithm. The multi-robot setup involves three robots, as shown in Figure 3.14, and consists of 3 mobile bases and two 6-DOF manipulator arms. The locations of the three bases are denoted as $\mathbf{x}_1 \in \mathbb{R}^2$, $\mathbf{x}_2 \in \mathbb{R}^2$, and $\mathbf{x}_3 \in \mathbb{R}^2$, respectively. Base 2 and base 3 are equipped with manipulator arms, whose end-effector positions are represented as $\mathbf{e}_1 \in \mathbb{R}^3$ and $\mathbf{e}_2 \in \mathbb{R}^3$, respectively.

The STL formula defining the tasks is the following,

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

The above task involves collision avoidance constraints that are always active given by the subformula

$$\bar{\varphi}_1 = (\|\mathbf{x}_1 - \mathbf{x}_2\| \geq 0.6) \wedge (\|\mathbf{x}_2 - \mathbf{x}_3\| \geq 0.6) \wedge (\|\mathbf{x}_3 - \mathbf{x}_1\| \geq 0.6).$$

Next, in the duration $[10, 125][s]$, base 1 surveils the arena and follows a circular time varying trajectory given by the subformula

$$\bar{\varphi}_2 = (\mathcal{G}_{[10,125]} \|\mathbf{x}_1 - c_1(t)\| \leq 0.05)$$

where $c_1(t)$ is the circular trajectory. In the duration $[30, 70][s]$, end-effector 1 tracks a virtual point $0.35[m]$ over base 1 to simulate a pick-and-place task, given by the subformula

$$\bar{\varphi}_3 = \mathcal{G}_{[30,70]} \|\mathbf{e}_1 - [\mathbf{x}_1^\top, 0.35]^\top\| \leq 0.01 \wedge \mathcal{G}_{[30,70]} \|\mathbf{x}_2 - c_2(t)\| \leq 0.05$$

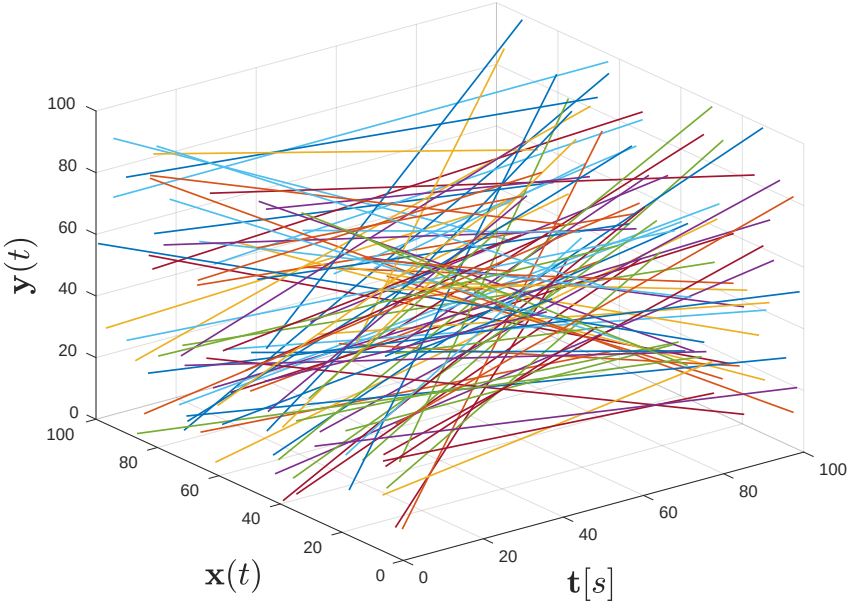
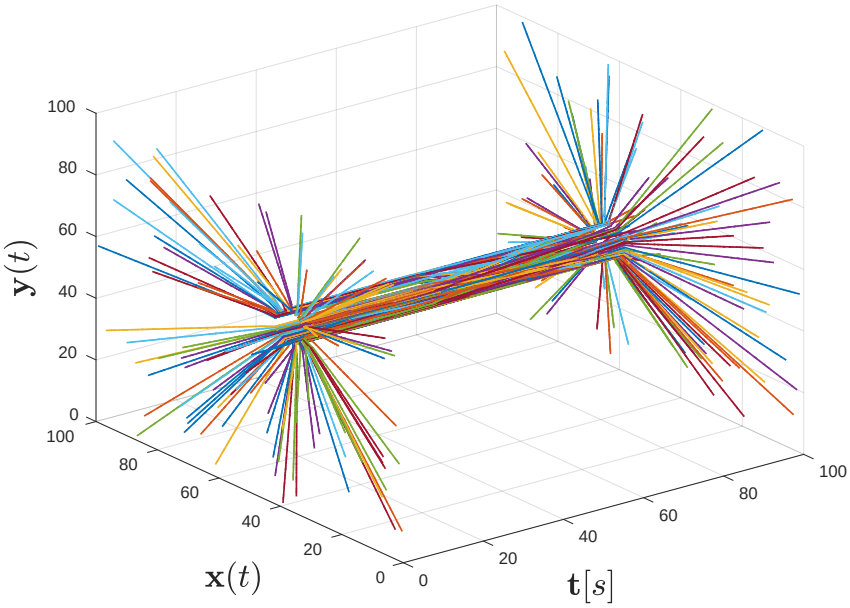
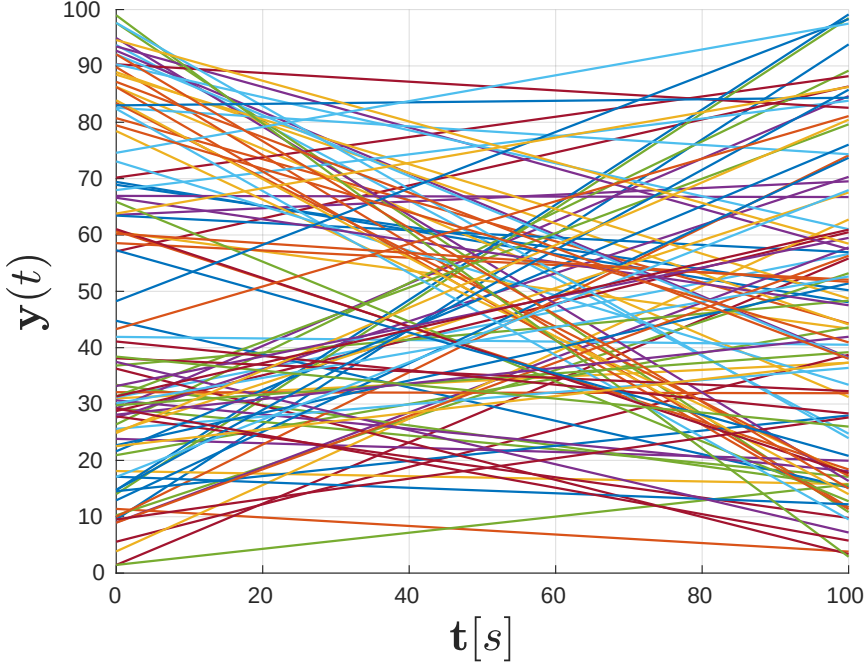
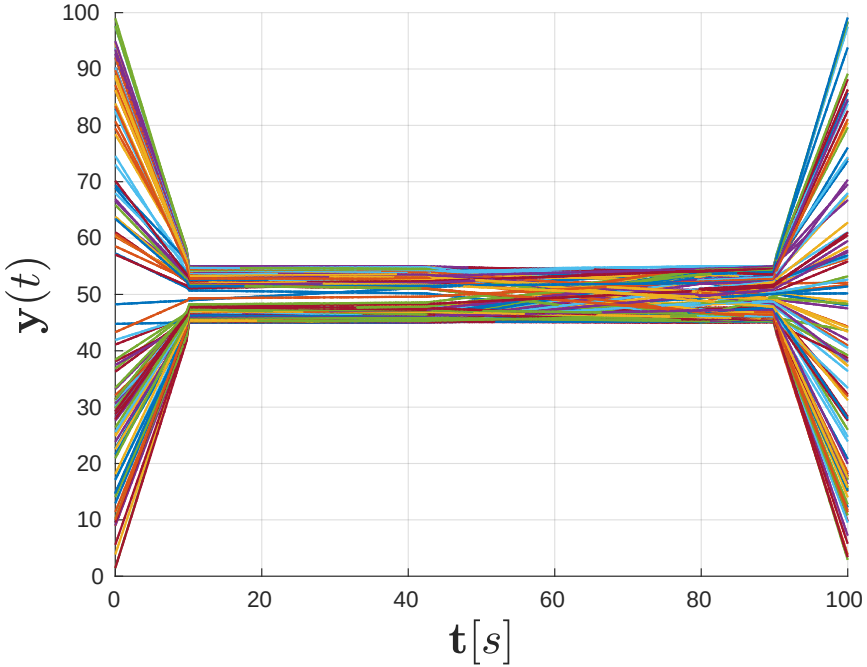
(a) 3D view at $j = 0$ (b) 3D view at $j = 1000$

Figure 3.26: 3D Simulation of trajectory generation for 100 agents for the STL formula (3.14).

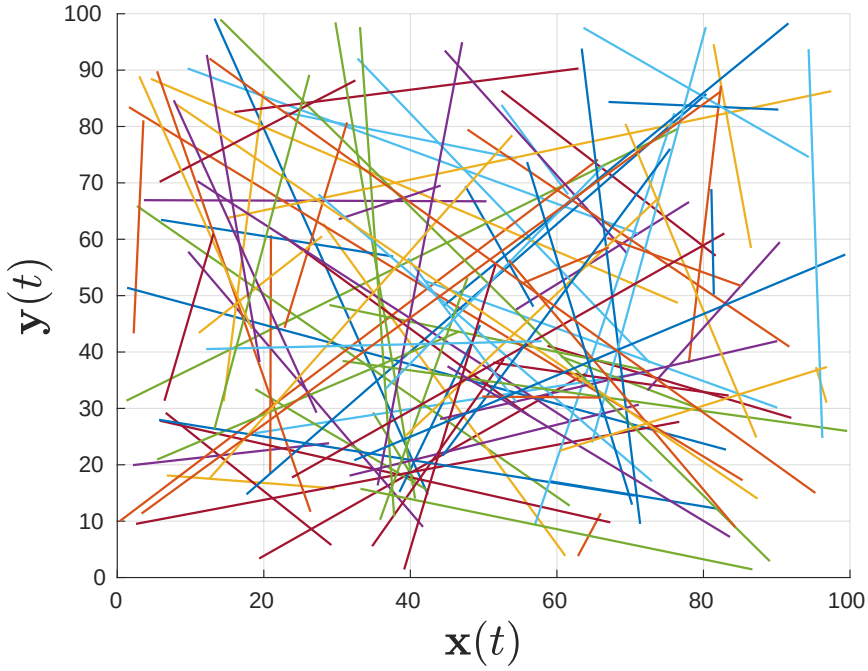
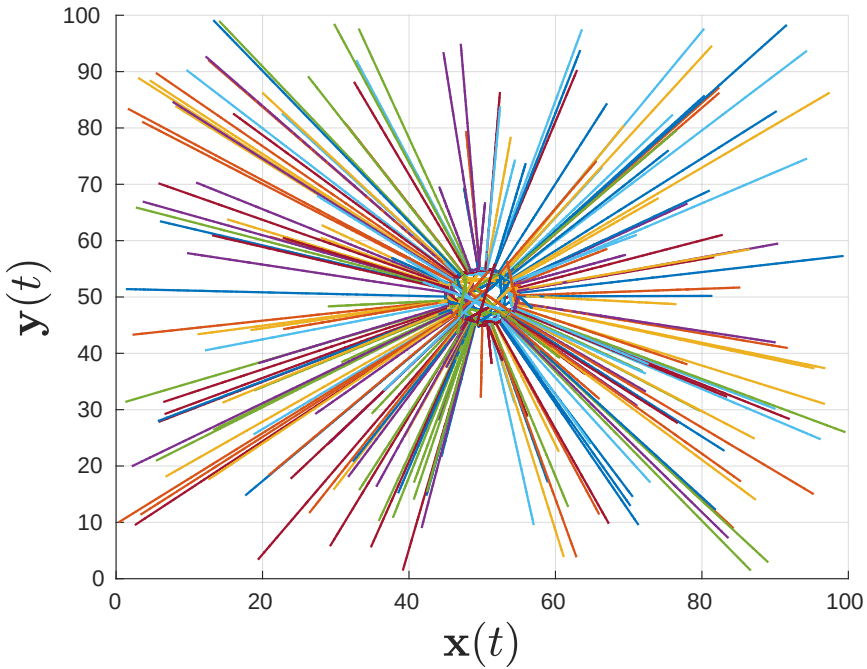


(a) $t - y$ axis view at $j = 0$



(b) $t - y$ axis view at $j = 1000$

Figure 3.27: $t - y$ axis simulation of trajectory generation for 100 agents for the STL formula (3.14).

(a) $x - y$ axis view at $j = 0$ (b) $x - y$ axis view at $j = 1000$ Figure 3.28: $x - y$ axis simulation of trajectory generation for 100 agents for the STL formula (3.14).

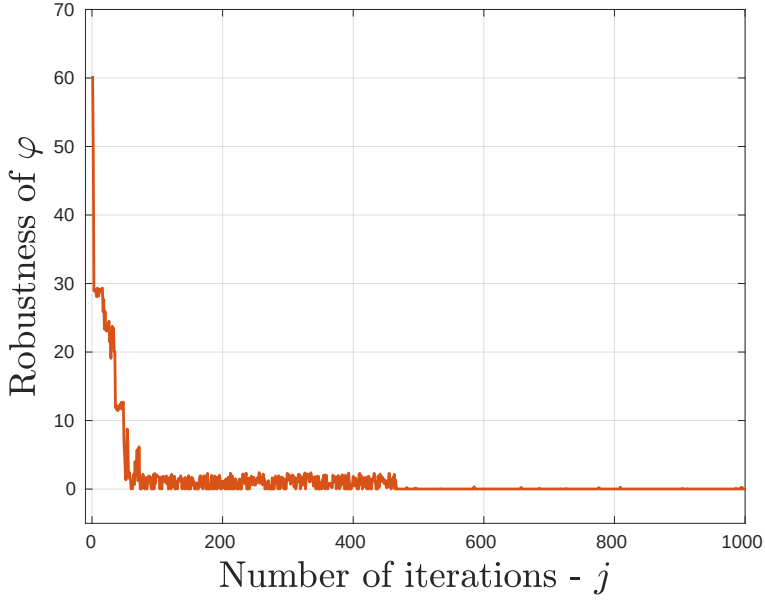


Figure 3.29: Robustness of the STL formula in (3.14)

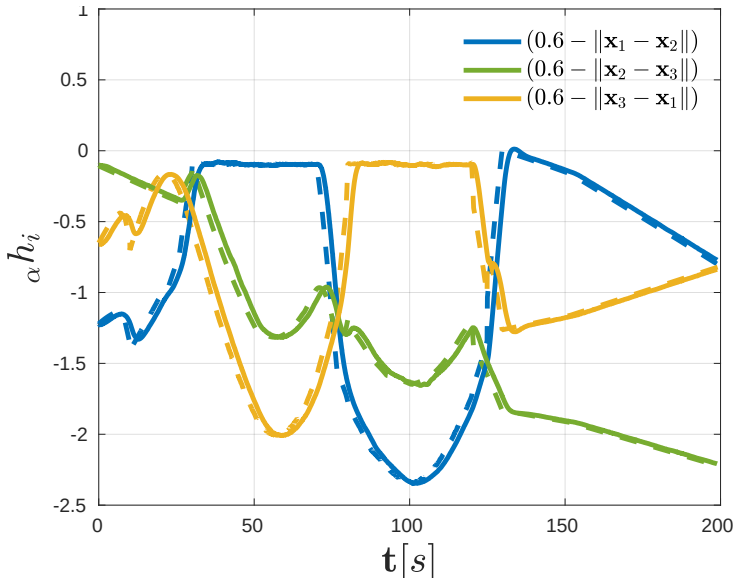


Figure 3.30: Collision constraints

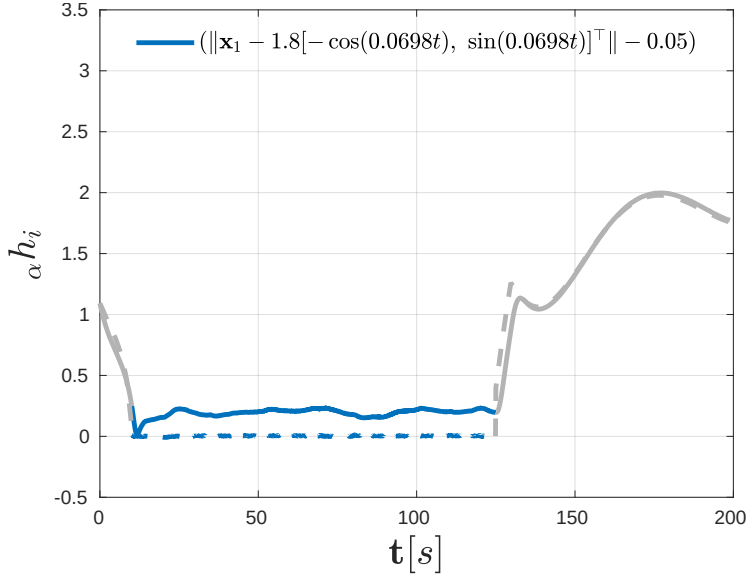


Figure 3.31: Base 1 trajectory tracking

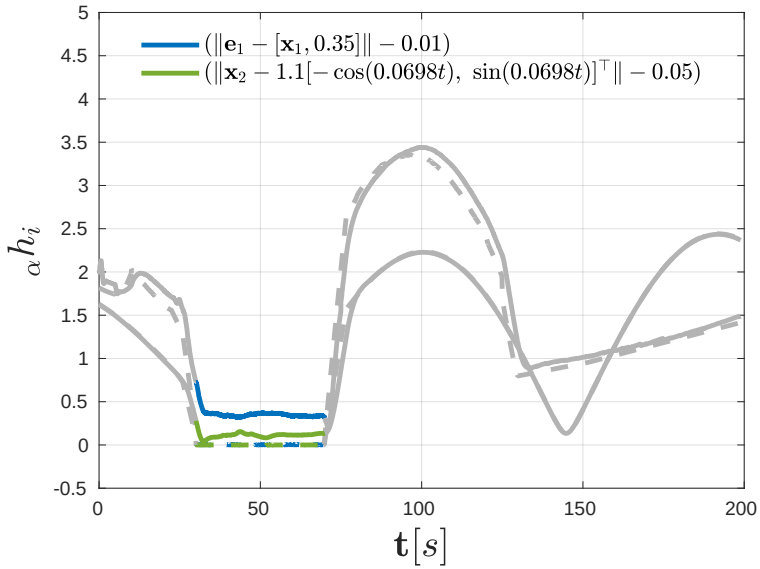


Figure 3.32: Virtual point tracker by base 2 and end effector 1

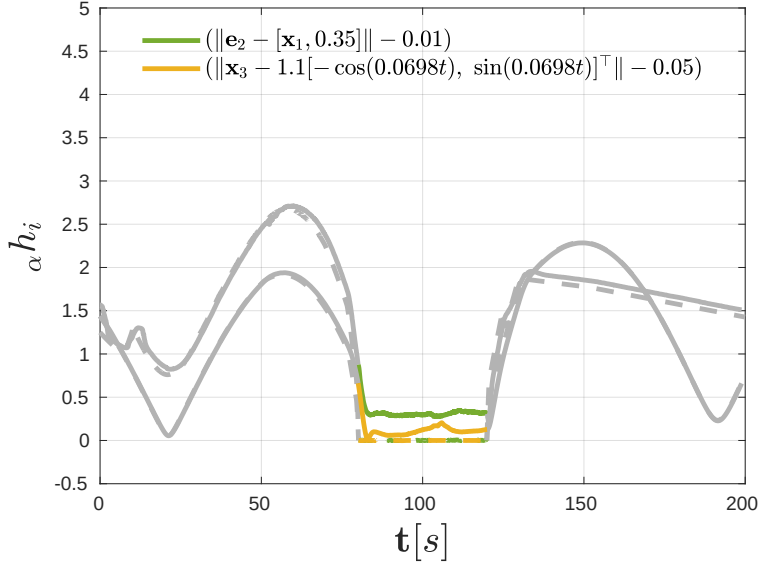


Figure 3.33: Virtual point tracker by base 3 and end effector 2

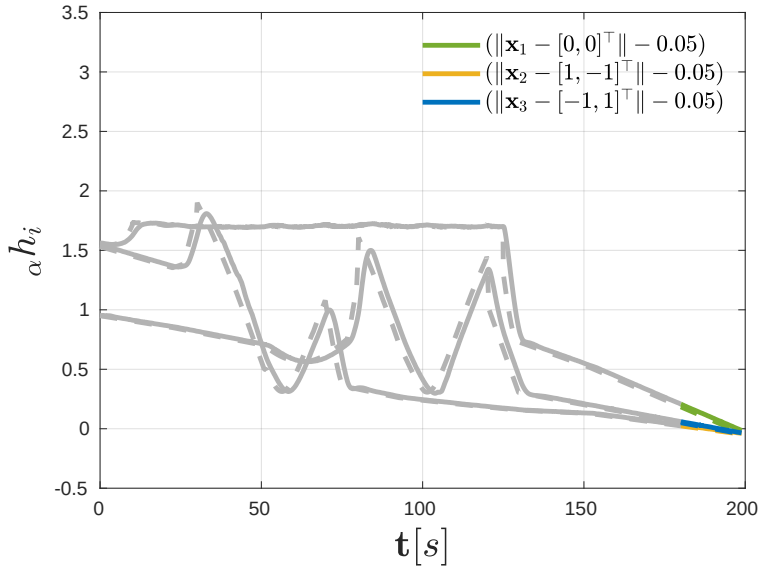


Figure 3.34: Desired final position of basess

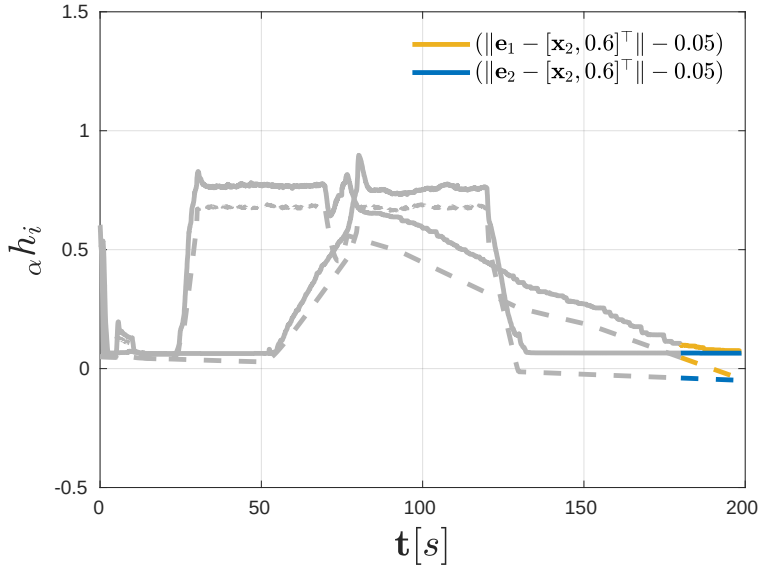
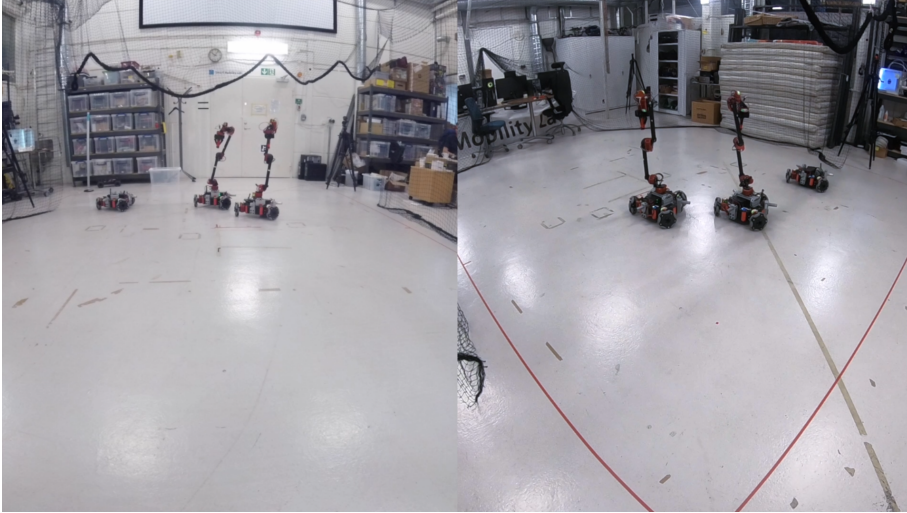


Figure 3.35: Desired final position of end effectors

Figure 3.36: At $t = 12[s]$, when $\bar{\varphi}_1$ and $\bar{\varphi}_2$ are active

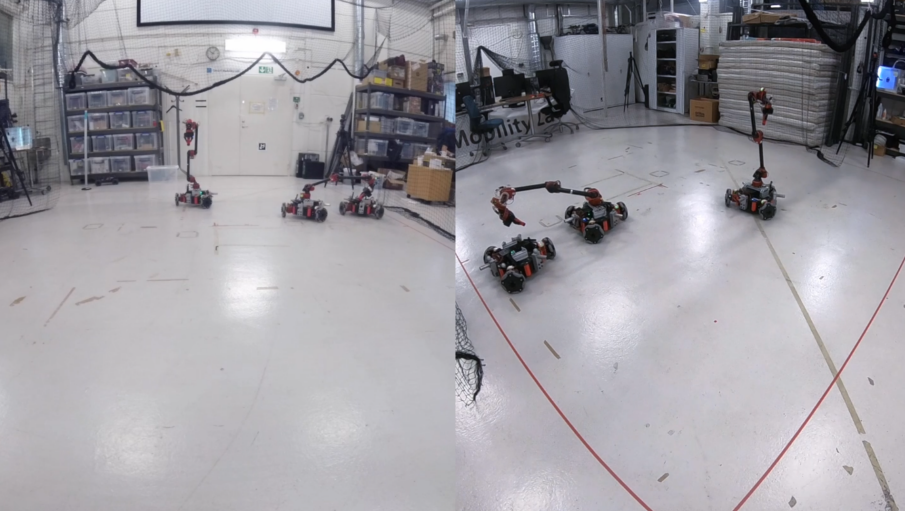


Figure 3.37: At $t = 50[s]$, when $\bar{\varphi}_1$, $\bar{\varphi}_2$ and $\bar{\varphi}_3$ are active

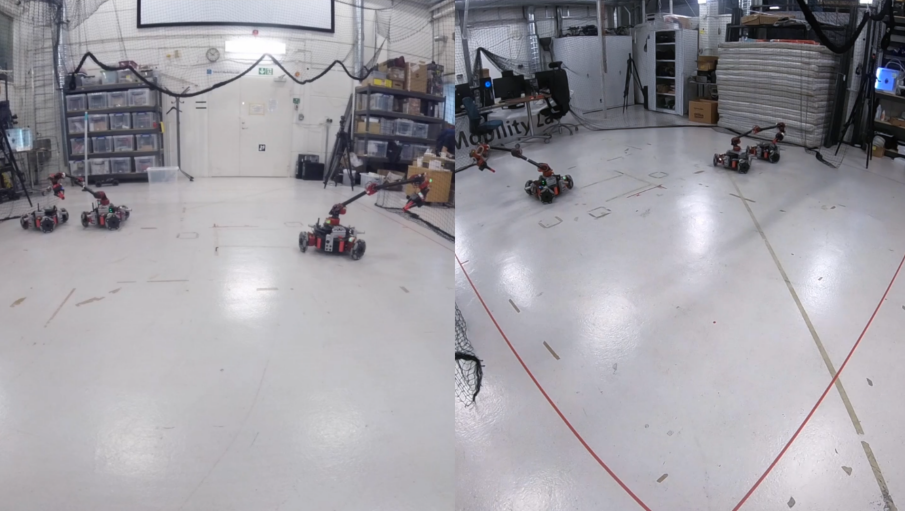


Figure 3.38: At $t = 100[s]$, when $\bar{\varphi}_1$, $\bar{\varphi}_2$ and $\bar{\varphi}_4$ are active

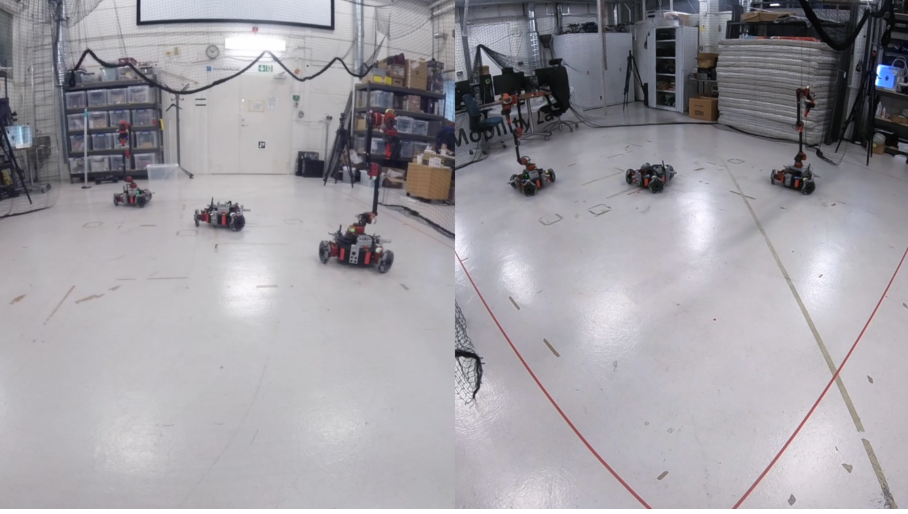


Figure 3.39: At $t = 190[s]$, when $\bar{\varphi}_1$ and $\bar{\varphi}_5$ are active

where $c_2(t)$ is the circular trajectory. Similarly, in the duration $[80, 120][s]$, end-effector 2 takes over the task to track a virtual point $0.35[m]$ over base 1, given by the subformula

$$\bar{\varphi}_4 = \mathcal{G}_{[80,120]} \| \mathbf{e}_2 - [\mathbf{x}_1^\top, 0.35]^\top \| \leq 0.01 \wedge \mathcal{G}_{[80,120]} \| \mathbf{x}_3 - c_2(t) \| \leq 0.05.$$

Finally, eventually in the duration $[180, 200][s]$, the robots assume a final position given by the subformula

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

The results are shown in Figure 3.30-3.35, where the x-axis represents time in seconds, and the y-axis represents the predicate functions defined by (3.9). The dashed line in the plots represents the predicate functions of the trajectories obtained by solving the optimisation problem (3.11), while the solid line represents the predicate functions of the actual trajectories by the robots. In the context of (3.9), negative values indicate task satisfaction. However, due to the lack of an accurate model of the robots and the fact that the optimisation solution converges to the boundary of the constraints, the tracking is imperfect, and we observe slight violations of the formula by the robots in certain cases. Nonetheless, the trajectories generated by the algorithm do not violate the STL formula. The coloured lines represent the functions that lie within the validity domain of the formula. Figure

3.30 shows that the collision constraint imposed on all 3 bases is not violated, and they maintain a separation of at least 60 cm. In Figure 3.31, base 1 tracks a circular trajectory in the interval $[10, 125]$ seconds. In Figures 3.32 and 3.33, the end effectors mounted on top of bases 2 and 3 track a virtual point over the moving base 1 sequentially. In the last 20 seconds, the bases and end effectors move to their desired final positions, as seen in Figures 3.34 and 3.35. The maximum computation time by any robot is 3.611[s]. Figure 3.36-3.39 shows front-view and side-view at different time instances during the experimental run³.

3.5 Conclusion

In Section 3.3, we presented an efficient sampling-based algorithm for cooperative motion planning between two agents under STL specifications. We augment the traditional RRT method to generate spatio-temporal trees through which we enforce space and time constraints specified by the STL. The proposed algorithm is distributed as agents independently build their trees by communicating with each other.

In 3.4, we proposed MAPS², a distributed planner that solves the multi-robot motion-planning problem subject to tasks encoded as STL constraints. By using the notion of validity domain and formulating the optimisation problem as shown in (3.11), MAPS² transforms the spatio-temporal problem into a spatial planning task, for which efficient optimisation algorithms already exist. Task satisfaction is probabilistically guaranteed in a distributed manner by presenting an optimisation problem that necessitates communication only between robots that share coupled constraints. Extensive simulations involving benchmark formulas and experiments involving varied tasks highlight the algorithms functionality.

³The video of the experiments can be found here: <https://youtu.be/YkuiPu0erMg>

Chapter 4

Trajectory Optimisation for Multi-Manipulator Systems

In Chapter 2, we designed control algorithms for the cooperative manipulation problem, and in Chapter 3, we developed geometric planning algorithms for multi-robot systems. The natural next step is to combine the two approaches and apply them to more challenging tasks such as obstacle avoidance and passing through narrow passages. This is the focus of this Chapter.

4.1 Introduction

Applications of cooperative manipulation, such as warehouse automation, smart construction, and cooperative transport, often involve navigating through constrained environments. By constrained, we refer to the presence of static obstacles and narrow passages. In this chapter, we address the following question: how do we navigate effectively in such environments?

Obstacle avoidance in manipulation is a well-studied problem, typically divided into two categories: (i) obstacle avoidance for mobile manipulators, and (ii) obstacle avoidance for cooperative manipulators transporting a shared object. The path planning problem in the presence of obstacles is generally nonlinear, non-convex, and PSPACE-complete; finding optimal paths is NP-hard.

The seminal work of [110] introduced artificial potential fields—specifically FIRM (Force Inducing an Artificial Repulsion from the Surface)—for low-level obstacle avoidance. More recent approaches, such as [111], operate at a higher level by constructing Graphs of Convex Sets (GCS) to represent the collision-free workspace, planning shortest paths through these regions. In [112], the authors propose a whole-body trajectory optimisation method for mobile manipulators by constructing convex free-space regions and tracking references using MPC. Similarly, [113] solves the problem in two stages: first, using wavefront expansion to identify free regions; then applying potential fields to avoid reactive obstacles. We also recently

proposed a wavefront-based obstacle avoidance method for mobile manipulators in [114].

In cooperative manipulation, most existing work assumes 2D environments or avoids reconfiguration altogether [115]. Some methods, such as [116], assume full-system models but do not address mobile manipulators. Others, like [117], use potential-field-like costs in trajectory optimisation and employ MPC for stabilisation, but do not consider full trajectory tracking. The work in [118] is closest to our approach but excludes mobile cooperative manipulation due to the added complexity introduced by system redundancy.

To the best of our knowledge, no prior work addresses cooperative manipulation through narrow passages. These scenarios are particularly challenging due to the high risk of local minima, collisions, and failure to meet tracking requirements. Potential field-based methods are ill-suited for such cases, as the arms often need to pass very close to obstacles—leading to large gradient magnitudes that destabilise the controller. Our approach addresses this by first decoupling the planning of mobile bases from the planning of manipulator arms, enabling the computation of collision-free base trajectories. This significantly reduces the number of degrees of freedom involved in planning for the full mobile cooperative manipulation setup. Next, we softly track these base trajectories while computing joint angles that avoid collisions. This allows for the generation of locally collision-free arm motions as the bases move along the planned paths. Finally, we track the desired joint configurations using a controller. We develop an inverse kinematics algorithm that softly tracks both the base trajectories and the object trajectory, while still maintaining collision-free motion. We present simulation results to demonstrate the effectiveness of our proposed framework.

4.2 Problem Formulation

System Dynamics

The system consists of N mobile robotic arms rigidly grasping an object. Let $\{W\}$ be the inertial frame. The mobile base of the i th arm is associated with a frame $\{B_i\}$, and its end-effector is associated with a frame $\{E_i\}$, as shown in Figure 4.1. The position of the base in the $\{W\}$ frame is denoted by $b_i \in \mathbb{R}^2$, and the position of the end-effector by $p_i \in \mathbb{R}^3$. The dynamics of each mobile manipulator are governed as in (2.1),

$$M_i(q)\ddot{q}_i + C_i(q, \dot{q})\dot{q}_i + g_i(q) = \tau_i \quad (4.1)$$

where

$$q_i = \begin{bmatrix} b_i \\ \theta_i \end{bmatrix} \in \mathbb{R}^{n_i},$$

with $b_i \in \mathbb{R}^2$ denotes the position of the base of the i th arm, and $\theta_i \in \mathbb{R}^{n_i-2}$ representing its joint angles. The relationship between q_i and p_i is governed by the

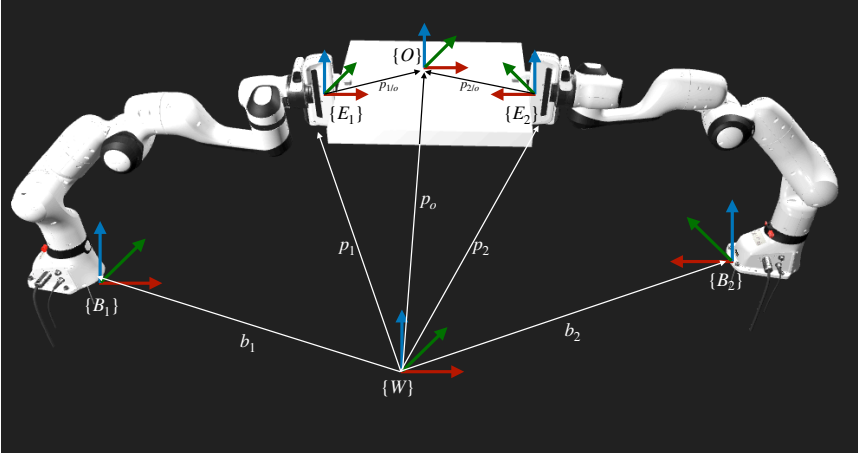


Figure 4.1: Visualisation of frames of two panda arms rigidly grasping an object (we consider panda arms developed by Franka Emika [3]).

forward kinematics (1.3),

$${}^W X_i^{E_i} = f(q_i) \quad (4.2)$$

where

$${}^W X_i^{E_i} = \begin{bmatrix} {}^W R_i^{E_i} & p_i \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}$$

represents the homogeneous transformation matrix from the $\{E_i\}$ frame to the $\{W\}$ frame, and $f(q_i)$ is a function that maps the joint configuration q_i to the end-effector position p_i . The agent dynamics (4.1) are defined in joint space and can be expressed in task space as (2.4), which is coupled to the object dynamics (2.6) through the grasping constraint. This results in the coupled object-agent dynamics, similarly to (2.11),

$$\tilde{M}\dot{v}_o + \tilde{C}v_o + \tilde{g} + \tilde{w} = Gu \quad (4.3)$$

where

$$\tilde{M} = M_o + G\bar{M}G^\top \quad (4.4)$$

$$\tilde{C} = C_o + G\bar{M}\dot{G}^\top + GCG^\top \quad (4.5)$$

$$\tilde{g} = g_o + G\bar{g} \quad (4.6)$$

$$\tilde{w} = w_o + G\bar{w}. \quad (4.7)$$

An introduction to the derivation of the coupled dynamics above is given in Section 2.2.

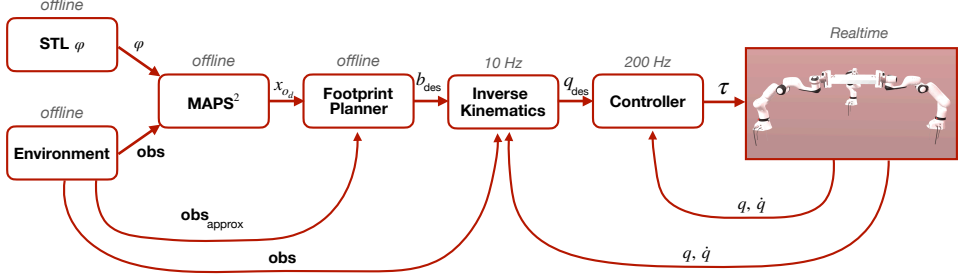


Figure 4.2: Proposed solution architecture.

Environment

Let $\mathcal{W} \subset \mathbb{R}^3$ denote the workspace within which the robots operate, and let $\mathcal{O} \subset \mathcal{W}$ represent the region occupied by static obstacles.

Assumption 8 *The obstacle-free workspace $\mathcal{W} \setminus \mathcal{O}$ is non-empty, and the obstacle region $\mathcal{O}$ remains static over time.*

This is a standard assumption that allows planning in the obstacle free region.

Problem 4 *Consider a system of N agents rigidly grasping an object, and let an STL formula φ (of the form (1.1)) specify the desired spatio-temporal behaviour of the object state x_o . The objective is to design control torques τ_i for each agent $i = 1, \dots, N$ in accordance with the dynamics in (4.1), such that the resulting x_o satisfies φ , i.e., $x_o \models \varphi$.*

Note that the above problem requires determining the joint torques for individual agents based on a specification defined over the object, in contrast to Chapter 2, where the control effort is typically allocated to individual agents first, followed by the computation of joint torques.

4.3 Trajectory Planning and Control for Cooperative Manipulators in Constrained Environments

Solution Approach

The proposed solution approach, illustrated in Figure 4.2, is as follows:

- Compute a desired collision-free trajectory for the object x_{o_d} using MAPS² (see Section 3.4).
- Determine desired collision-free base positions using the *Footprint Planner* for each robot i , $b_{i_{des}}$, based on x_{o_d} .

- Compute the desired joint angles q_{ides} via *Inverse Kinematics*, given b_{ides} and x_{od} .
- Design a control law to track the desired joint angles q_{ides} for each agent $i = 1, \dots, N$.

Main Results

- *MAPS²*: Based on the STL formula, the MAPS² algorithm computes a desired collision-free trajectory for the object $x_{od}(t)$. We post-process the trajectory using a two-step smoothing method. First, we interpolate the original waypoints using cubic splines to obtain a temporally dense and smooth trajectory. Then, we apply Gaussian smoothing to the interpolated positions to further reduce high-frequency noise. Finally, smoothed velocities are computed as time derivatives, and the result is represented as a Cubic Hermite Piecewise Polynomial for downstream use.
- *Footprint Planner*: The problem of directly computing the joint angles given the desired object trajectory x_{od} quickly becomes intractable, as it involves optimising over $N \times n_i$ states, along with non-convexities introduced by obstacles and forward kinematics. Moreover, optimising over a time horizon $[0, t_f]$ with T total time steps results in $T \times N \times n_i$ decision variables. To alleviate this challenge, we decompose the planning problem for mobile manipulators into two parts: planning for the mobile bases and planning for arms. Given a desired x_{od} , our goal is to compute a collision-free set of base trajectories $b_{des} = [b_{1des}^\top \ \cdots \ b_{Ndes}^\top]^\top$. Key considerations include:
 - The base trajectories should not intersect with one another.
 - The base trajectories should avoid collisions with obstacles.
 - The base velocities should satisfy motion constraints.
 - The base trajectories should be feasible with respect to the desired object trajectory x_{od} .
 - The base trajectories should remain mutually feasible, i.e., the agents should not drift too far apart from one another releasing the object.

This problem can be formulated as a trajectory optimisation problem, where the objective is to find b_{des} that satisfies the above constraints. Let the

trajectory x_{o_d} be discretised into T time steps over the horizon $[0, t_f]$.

$$\min_{b_{des}} \sum_{k=1}^T \sum_{i=1}^N \sum_{\substack{j=1 \\ i \neq j}}^N w_i \left(\|b_{ides}[k] - b_{jdes}[k]\|^2 - \alpha_i^2 \right)^2 \quad (4.8a)$$

$$\text{s.t.} \quad \left\| \frac{1}{N} \sum_{i=1}^N b_{ides}[k] - x_{o_d}(t) \right\|^2 \leq \epsilon \quad (4.8b)$$

$$-\eta \leq b_{ides}[k+1] - b_{ides}[k] \leq \eta, \quad \forall i = 1, \dots, N \quad (4.8c)$$

$$b_{des}[k] \in \mathcal{W} \setminus \mathcal{O}_{\text{approx}} \quad (4.8d)$$

$$b_{des}[0] = x_{o_d}(0) \quad (4.8e)$$

$$b_{des}[t_f] = x_{o_d}(t_f) \quad (4.8f)$$

- 4.8a defines the cost function that penalises the deviation of the squared distance between any two bases from the target squared distance α_i^2 . If the arms grasp the object in non-symmetric configurations (see Figure 4.3), then α_i will vary depending on the kinematic reach from each base to its end-effector. The weights w_i are positive scalars that can be tuned based on the desired strictness in maintaining the target inter-base distances. In this formulation, we consider every pairwise inter-base distance and enforce desired distances between them.
- 4.8b ensures that the centroid of the base positions remains close to the desired object trajectory x_{o_d} . The object trajectory here considers only the x and y coordinates, as z is fixed for the bases. This constraint maintains the proximity of the centroid formed by all base positions to x_{o_d} generated by the MAPS² algorithm, which already provides a global feasible trajectory—an advantage we aim to transfer to the base trajectories.
- 4.8c ensures that the base velocities remain within the upper and lower bounds defined by the maximum velocity η . The value of η is typically specified by the manufacturer of the mobile base.
- 4.8d represents the obstacle avoidance constraint in two dimensions. The obstacle region $\mathcal{O}$ is assumed to be convex, and $\mathcal{O}_{\text{approx}}$ refers to its 2D projection. This constraint ensures that the base positions remain collision-free within the workspace.
- 4.8e and 4.8f define the initial and final conditions for the base trajectories.
- *InverseKinematics:* The desired base trajectories b_{des} (denoted as $\bar{b}_{des}$ here) enable the computation of desired joint angles $q_{des} = [q_{1des}^\top \quad \dots \quad b_{Ndes}^\top]^\top$. The considerations are as follows:
 - The arms should avoid collisions with obstacles.

- The arms should ensure tracking of the desired base trajectories $b_{i\text{des}}$ and the desired object trajectory x_{o_d} .
- The arms should maintain configurations that avoid kinematic singularities.
- The end-effectors should satisfy the rigid grasp constraints on the object.
- The solution must respect the joint limit constraints.

These considerations can be formulated as an optimisation problem, where the objective is to find the desired joint angles q_{des} that satisfy the above requirements. Let the trajectory q_{des} be discretised into T time steps over the horizon $[0, t_f]$.

$$\min_{q_{\text{des}}} \|q_{\text{des}}[k] - q_0\|^2 + \sum_{i=1}^N \left\| \left({}^W R_i^{E_i} p_{i/o} + p_i \right)[k] - x_{o_d}(t) \right\|^2 + c_{\text{collision}} \quad (4.9a)$$

$$\text{s.t.} \quad p_{i/o}[k] = p_{i/o}[0] \quad (4.9b)$$

$${}^O R_i^{E_i}[k] = {}^O R_i^{E_i}[0] \quad (4.9c)$$

$$\bar{b}_{\text{des}}[k] - \nu \leq b_{\text{des}}[k] \leq \bar{b}_{\text{des}}[k] + \nu \quad (4.9d)$$

$${}^W X_i^{E_i}[k] = f(q_{i\text{des}}[k]) \quad (4.9e)$$

- 4.9a defines the cost function, which consists of three terms. The first term penalises deviations of the joint angles from configurations that avoid singularities, where q_0 denotes such desirable configurations. The second term enforces tracking of an offset point — coinciding with the object's centre of mass — towards the desired position x_{o_d} . Note that ${}^W R_i^{E_i} p_{i/o} + p_i$ corresponds to p_o . The third term introduces a collision cost, which accounts for full 3D obstacles. We employ the potential function

$$c_{\text{collision}} = o_1 \log(1 + \exp(o_2(o_3 - p_d[k]))),$$

where o_1 , o_2 , and o_3 are tunable constants, and p_d is the distance to the closest point in the obstacle space $\mathcal{O}$. This function is chosen for being smooth and finite everywhere, non-decreasing, and lower-bounded by $o_1 \log(1 + \exp(o_2 o_3))$.

- 4.9b and 4.9c ensure that the rigid grasp constraints are satisfied. The first constraint ensures that the offset point $p_{i/o}$ remains fixed, while the second constraint ensures that the end-effector orientation remains fixed.
- 4.9d ensures that the base positions remain within a certain distance ν from the desired base positions $\bar{b}_{\text{des}}$. This allows for some flexibility in positioning to enable collision avoidance, while still maintaining tracking of the base trajectory.

- 4.9e finally represents the forward kinematics constraint.
- *Control Design:* The *Inverse Kinematics* solution provides the desired joint angles q_{des} based on trajectories computed from *MAPS*² and *Footprint Planner*. The next step is to design a control law to track these desired joint angles. We use a Proportional-Derivative(PD) control law:

$$\tau_i = -k_p(q_{i\text{des}}[k] - q_i(t)) - k_v\dot{q}_i(t), \quad t \in [t_k, t_{k+1}) \quad (4.10)$$

where k_p and k_v are positive gains that can be tuned based on the desired performance. The desired joint angles q_{des} are updated at discrete time instances t_k , and held constant (using a zero-order hold) throughout the interval $[t_k, t_{k+1})$.

4.4 Simulations

The simulations are conducted on three Panda mobile manipulators rigidly grasping an object, as shown in Figure 4.3. The system is simulated using the DRAKE physics engine [2]. The workspace spans an area of $10[\text{m}] \times 10[\text{m}]$ and contains obstacles and narrow passages of various shapes, see Figure 4.4. The goal will be to visit the waypoints designated in green while avoiding the obstacles shown in red.

Several implementation caveats, not captured in the methodology described above, are worth noting:

- Physics-based simulators generally do not support rigid grasps due to the lack of support for closed kinematic chains. To address this, we simulate grasping using linear bushings, which model spring-damper systems allowing limited compliance.
- Obstacles in the environment are approximated in the *Footprint Planner* using 2D superellipsoidal shapes. These approximations, illustrated in Figure 4.5, are necessary because modeling polygonal obstacles typically requires binary variables, which are not supported by the nonlinear solver SNOPT when used with nonlinear constraints. Figures 4.5 and 4.6 show a top view of the workspace, i.e., the xy -plane as seen from the positive z -axis.
- The collision cost function $c_{\text{collision}}$ is activated only when the distance between the object and an obstacle falls below a certain threshold. This avoids unnecessary computation when the object is far from obstacles.
- Collision checking is performed only on AutoDiff models of the arms and object. Therefore, the entire system is converted into a unified AutoDiff model to compute $c_{\text{collision}}$.
- The proximity (collision) regions of the arms are approximated as boxes to ensure compatibility with the DRAKE toolbox. Collision checking with other

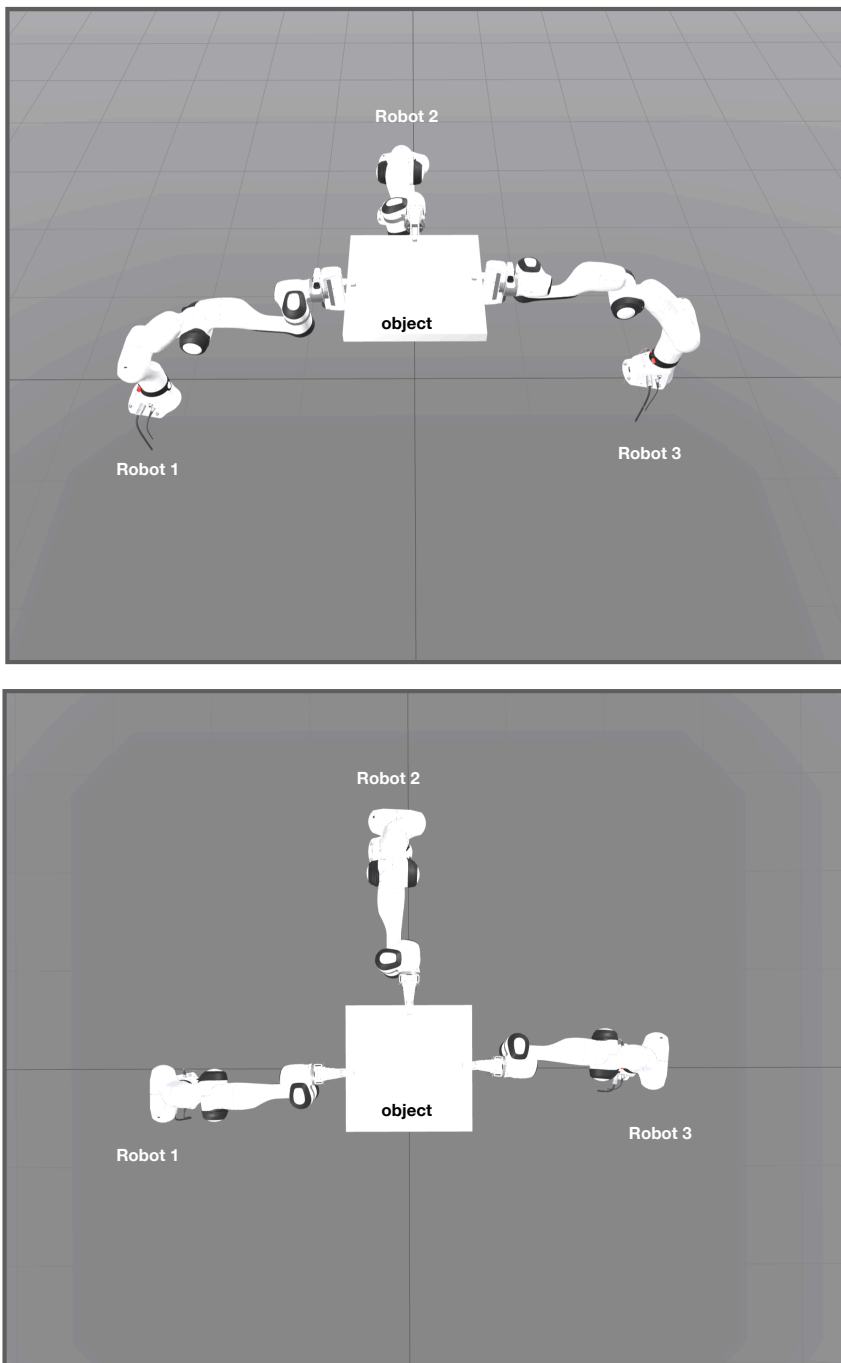


Figure 4.3: Setup of 3 panda arms developed by Franka Emika grabbing an object.

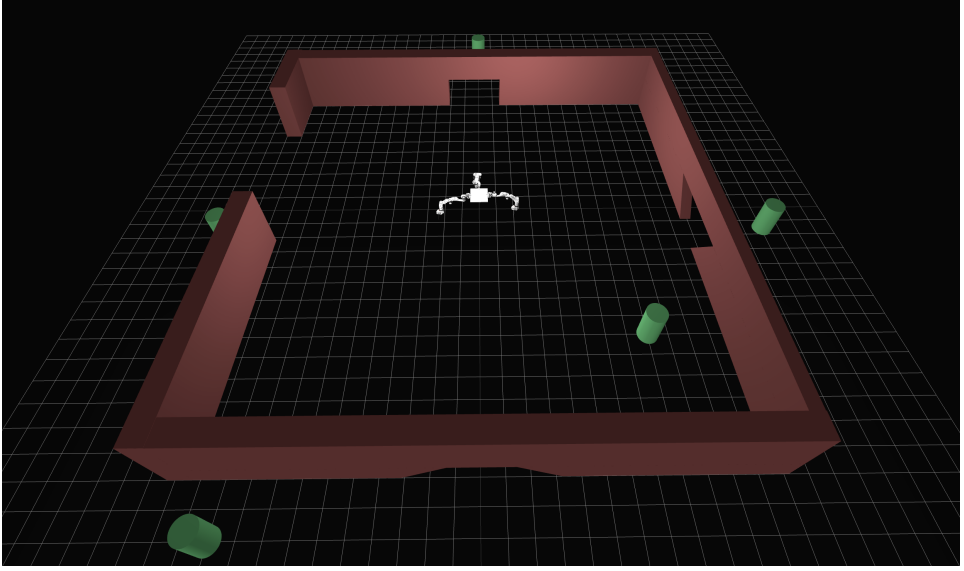


Figure 4.4: The workspace with obstacles shown in red, desired waypoints in green, and robots in white.

geometries such as meshes, spheres, or convex hulls is not yet supported. These boxes are then used to compute distances from obstacles and evaluate the collision cost $c_{\text{collision}}$.

The desired STL task to be executed is:

$$\begin{aligned} \varphi = & \mathcal{G}_{[37,38]}(\|x_o - \begin{bmatrix} 0 \\ 8 \\ 0.6 \end{bmatrix}\| \leq \epsilon) \wedge \mathcal{G}_{[87,88]}(\|x_o - \begin{bmatrix} 8 \\ -1 \\ 0.6 \end{bmatrix}\| \leq \epsilon) \wedge \\ & \mathcal{G}_{[112,113]}(\|x_o - \begin{bmatrix} 4 \\ -4 \\ 0.6 \end{bmatrix}\| \leq \epsilon) \wedge \mathcal{G}_{[150,151]}(\|x_o - \begin{bmatrix} -5 \\ -8 \\ 0.6 \end{bmatrix}\| \leq \epsilon) \wedge \\ & \mathcal{G}_{[200,201]}(\|x_o - \begin{bmatrix} -7 \\ -1 \\ 0.6 \end{bmatrix}\| \leq \epsilon) \wedge \mathcal{F}_{[220,230]}(\|x_o - \begin{bmatrix} 0 \\ 0 \\ 0.6 \end{bmatrix}\| \leq \epsilon) \wedge \\ & \mathcal{G}_{[0,250]}(\|x_o - \text{obs}\| \geq 0.5) \end{aligned}$$

which requires visiting 5 waypoints with a margin of $\epsilon > 0$ and avoiding obstacles with the margin of $0.5[m]$ as the dimensions of the object being held are $0.5[m] \times 0.5[m] \times 0.04[m]$. In Figure 4.4, the first waypoint is the top-most green point, the second is the right-most green point, the third lies inside the walls at the bottom-right corner, the fourth is at the bottom-left outside the walls, and the

final waypoint returns to the left-most green point. The final destination is the origin—the position of the robots shown in Figure 4.4. In this figure, the positive x-axis points to the right, and the positive y-axis points to the top.

The trajectory computed by MAPS² is shown in the top part of Figure 4.9d, where we verify that all waypoints are visited and obstacles are avoided. The solution from *FootprintPlanner* is displayed in the bottom part of the same figure, where we observe that all bases successfully avoid the obstacle approximations. Both these figures are accompanied with the plots of the real workspace of Figure 4.4.

The tracked base position error is shown in Figure 4.9d, where a non-zero tracking error is observed, as *InverseKinematics* is allowed some relaxation. Once the desired base trajectories b_{des} are obtained, we use *InverseKinematics* to compute the desired joint angles q_{des} . The collision cost $c_{\text{collision}}$ is shown in Figure 4.7a, along with its active region. Finally, the object's position tracking is shown in Figures 4.7b and 4.8a.

Figure 4.9 shows snapshots of the simulation run, where the arms are successfully passing through various shapes and avoiding obstacles.

4.5 Conclusion

This chapter presented a framework for motion planning in cooperative manipulation under constrained environments and STL tasks. A reference trajectory is first obtained from MAPS². This trajectory is passed to the *Footprint Planner*, which generates base trajectories that avoid obstacles. These base trajectories are then processed by the *Inverse Kinematics* algorithm to compute the joint angles to be tracked. We demonstrated that these joint angles can be successfully tracked using a controller. Simulation results with three Panda manipulators show that our framework enables safe obstacle avoidance and traversal through narrow passages while accurately tracking the object trajectory.

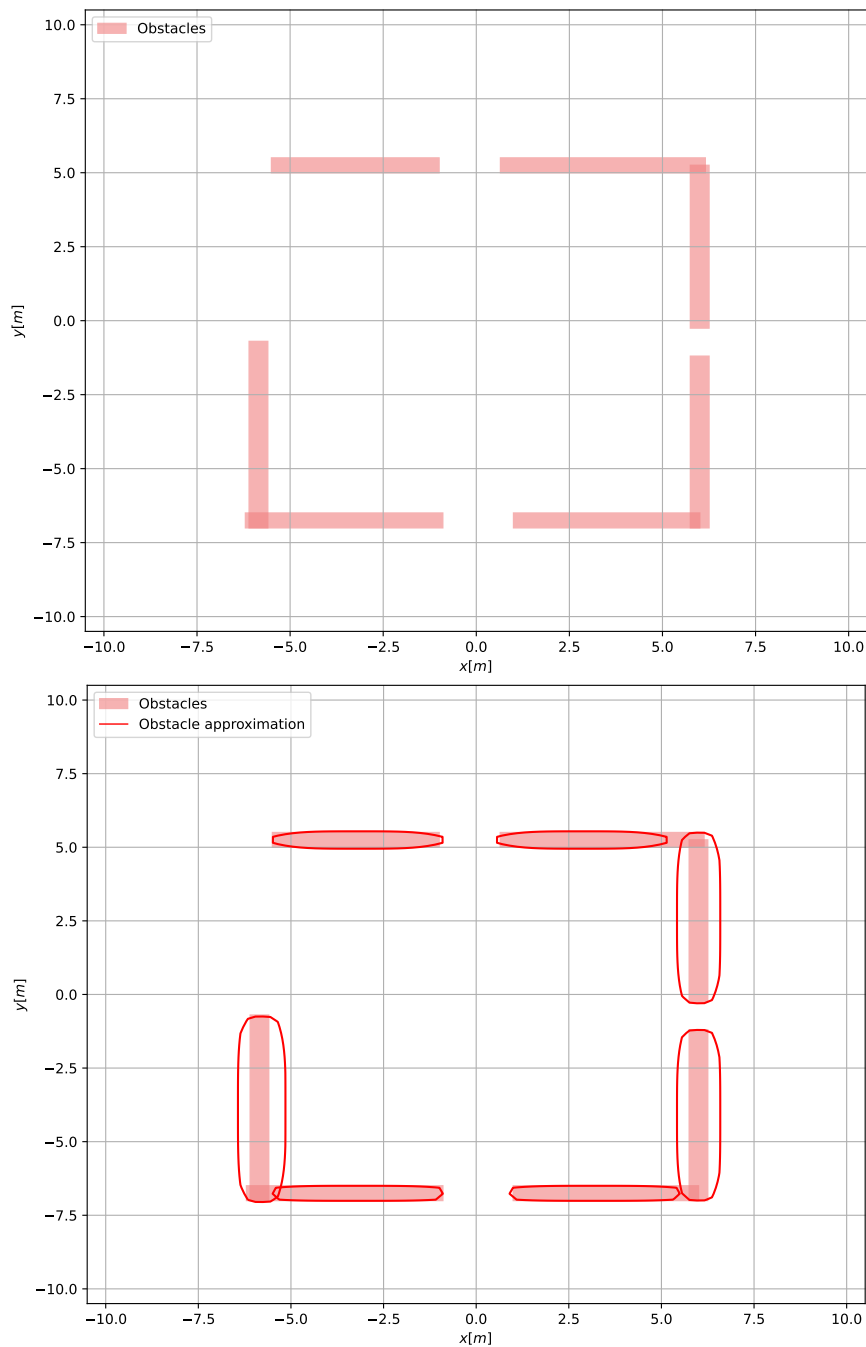


Figure 4.5: Obstacles and its approximation in the workspace $\mathcal{W}$.

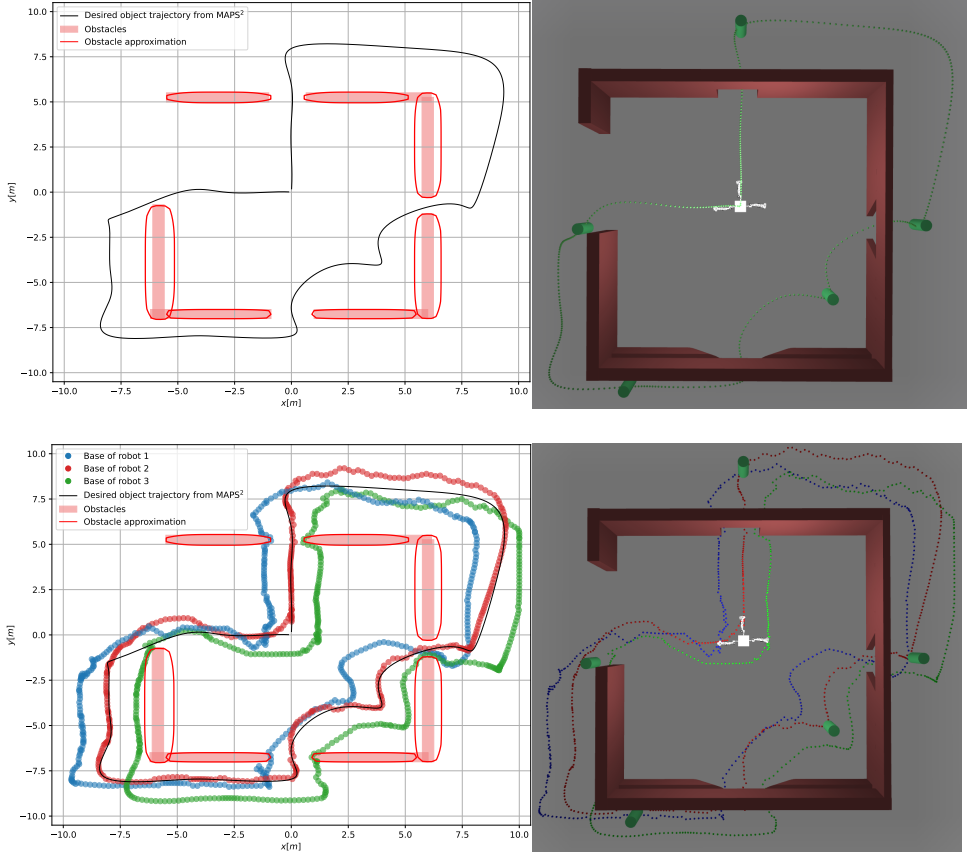
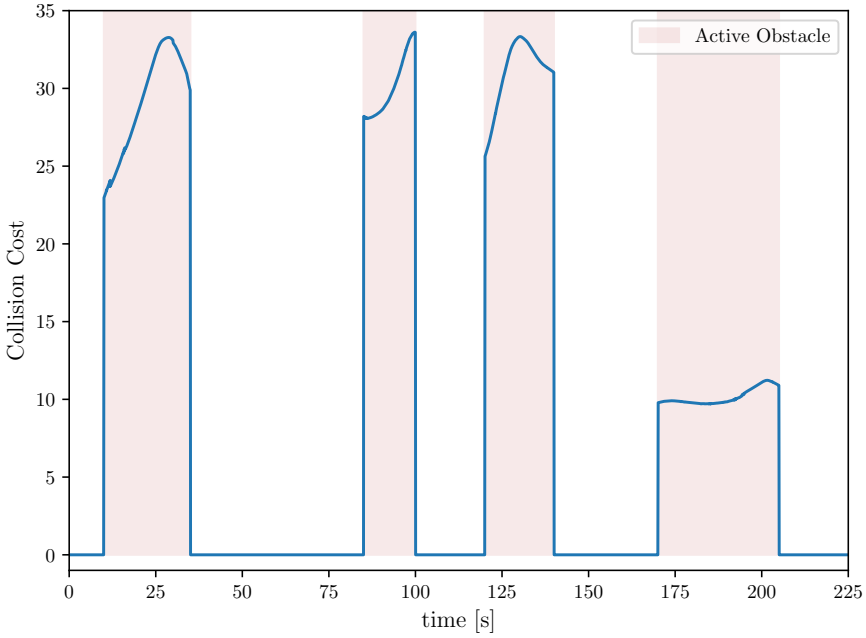
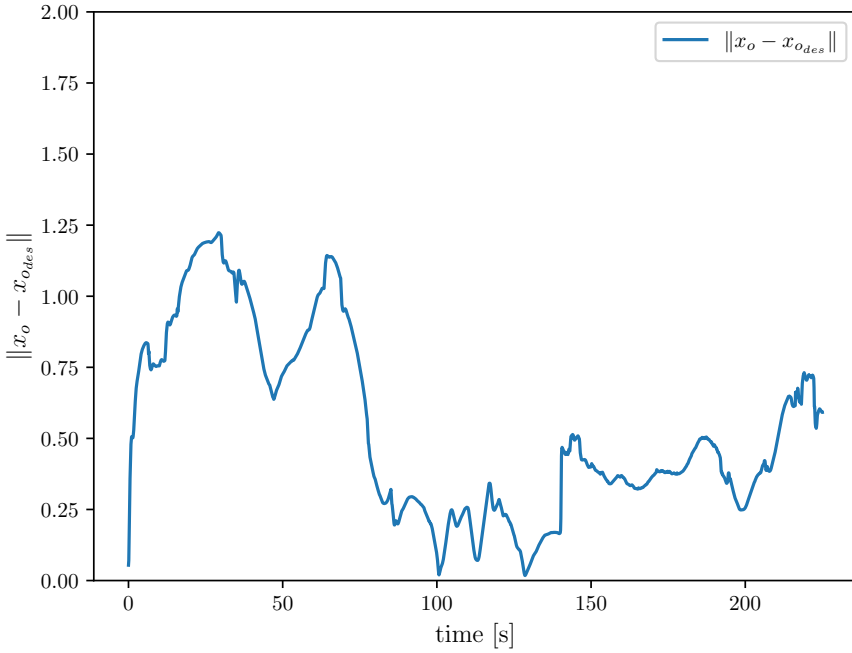


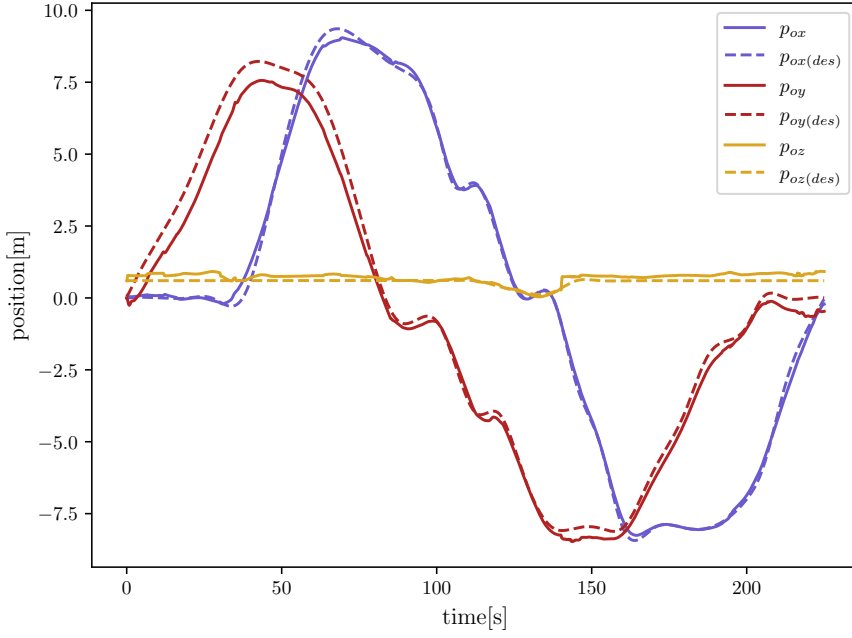
Figure 4.6: Desired trajectory from MAPS² along with the solution to (4.8)



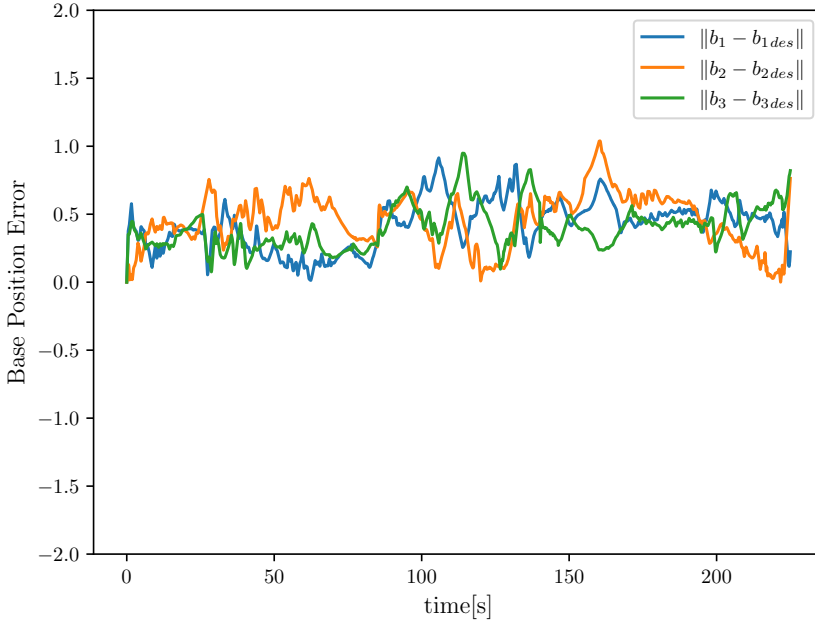
(a) Collision cost over time.



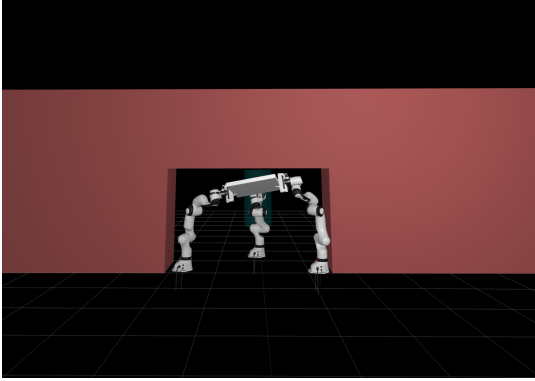
(b) Object position error during execution.



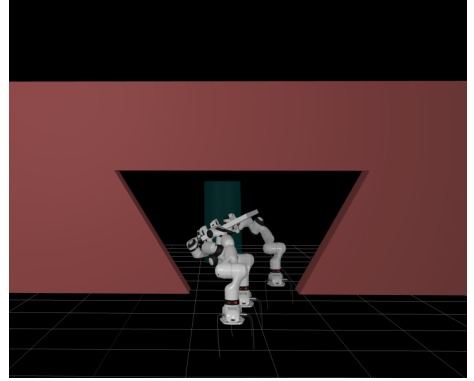
(a) Object trajectory tracking.



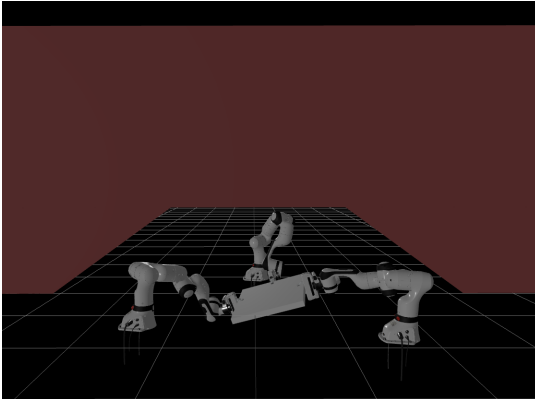
(b) Base footprint tracking.



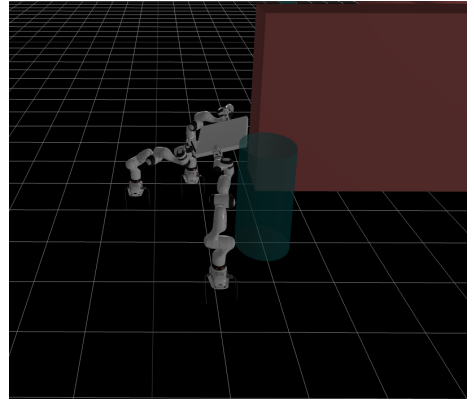
(a) Passing through a square shape.



(b) Passing through a triangle shape.



(c) Passing through trapezoid shape.



(d) Avoiding obstacle.

Figure 4.9: Snapshots of a simulation run.

Chapter 5

Conclusion and Future Directions

Right, we are at the end now. Let's conclude.

In Chapter 2, we designed PPC-based control algorithms for two scenarios: (i) all agents know the task and share the load equally, and (ii) a subset of agents know the task while the rest assist in carrying the load. In doing so, we provided task guidelines using a smaller fragment of STL formula (still expressive enough) and designed funnels that enforce the task. We presented both simulation and experimental results to demonstrate the proposed methodology.

In Chapter 3, we asked: why limit ourselves to a smaller fragment of the STL formula? What can we achieve if we don't constrain ourselves and instead use the full STL formula? At the cost of only providing probabilistic guarantees, we designed two algorithms: (i) STLcoRRT—a sampling-based cooperative algorithm inspired by RRT that generates trajectories for cooperating agents satisfying the STL formula; and (ii) MAPS²—a distributed optimisation algorithm that extends the problem to multiple agents and generates trajectories for any number of agents coupled by STL constraints. A common theme in both algorithms is sampling not just in space, as is conventional in planning algorithms, but also in time. This is necessary as STL imposes both spatial and temporal constraints. We presented both simulation and experimental results demonstrating the tracking of the generated trajectories.

Finally, in Chapter 4, we combined the planning and control frameworks to solve more challenging tasks such as transporting an object using cooperative mobile manipulators in a constrained environment. We split the problem into three parts: (i) generating offline base trajectories that are obstacle-free, (ii) generating joint angles that are obstacle-free and fulfil the task, and (iii) tracking the desired joint angles using a controller. We provided simulation results in a complex environment to evaluate the proposed framework.

5.1 Future Directions

Robotics—especially manipulation—is a vast field of research. Future directions could span any stage: sensing, estimation, identification, planning, control, physical structure, environment, or task description. However, I will focus on the directions I can reason about more clearly.

Task description: STL imposes constraints on a signal. In this thesis, we interpreted this signal as the trajectory of the centre of mass of the object being manipulated. In Chapter 3, however, we did not impose any dynamical constraints, and the signal could represent any continuous-time output of a physical or computational system. Each STL formula is specified by a human—that is, a designer formulates the task they want the robot to perform and translates it into an STL formula. However, as tasks become more complex, the translated formula may no longer accurately reflect the intended behaviour. There is no built-in mechanism to verify whether an STL formula truly captures the desired task. Moreover, an STL formula can admit many solutions, including the empty set in the case of infeasibility, or solutions that are syntactically correct but semantically undesired.

This limitation can be alleviated in two ways: (i) the designer can use MAPS² to generate trajectories and manually verify whether they are acceptable, or search for alternative solutions if needed; (ii) develop methods to translate directly from natural language to STL. For instance, [119] uses a transformer model to map English to STL, and [120] explores large language models for translating natural language into temporal logic.

Thus, the main challenges lie in maintaining semantic correctness, handling ambiguity, and verifying logical soundness, especially for safety-critical applications.

Grasping: Throughout this thesis, we assumed that the object is rigidly grasped by the robots. This assumption allowed us to shift focus away from grasping and towards challenges in planning and control. Addressing grasping in depth requires thorough reasoning about contact modelling, perception, and grasp pose synthesis—topics beyond the scope of this work.

However, grasping remains a fundamental challenge in manipulation. A few directions worth exploring are: (i) Many objects do not conform to the shape of the gripper, requiring the use of rolling contacts [121]; (ii) Considering nonprehensile manipulation, where objects are moved without being grasped. There has been limited research on specifying such tasks using temporal logic frameworks. (iii) Finally, cooperative manipulation of non-rigid objects, which offers more flexibility in managing internal forces but also requires reasoning about object deformation.

Generalisation and dynamical constraints: In Chapter 2, Remark 3 allowed us to enforce only six tasks at any given time. This becomes challenging when multiple obstacle avoidance constraints are present, as observed in Chapter 4. One possible direction is to use more general predicate functions that can encompass multiple tasks, or to rethink the control design to avoid relying on a full-rank Jacobian.

Additionally, in Chapter 3, we intentionally avoided dynamical constraints to maintain a more general framework. While this approach is valid—and applicable to a wide range of signals constrained by STL—it would be worthwhile to study how incorporating dynamics, a critical aspect of manipulation, affects the resulting algorithms.

Unified framework: As we have seen throughout this thesis, we addressed control in Chapter 2, planning in Chapter 3, and attempted to merge both in Chapter 4. Yet, a missing link remains: the development of a unified framework.

The vision, or rather the dream, is to have robots scattered across a workspace, where STL formulas dynamically trigger the transport of various objects, for example in a warehouse. When a task is activated, the optimal robots closest to the object are selected and deployed to grasp it. This requires determining an optimal grasp pose, generating an optimal approach trajectory, and detecting feasible grasp points. Once the object is grasped, the robots transport it and then repeat the sequence, including object release. This process again requires trajectory planning for each robot and dynamic obstacle avoidance. Developing such a unified framework, encapsulated in a single system, would be a significant contribution to robotics.

This need not be achieved by a single planner or controller. Pre-grasp planning could be handled by an RRT-based planner for speed. Grasp detection could leverage recent advances in diffusion models [122]. Finger-level grasping could use a simple on-off controller. Cooperative manipulation could follow the approach of Chapter 2, while trajectory planning could be based on the ideas from Chapter 3, extended to include dynamic constraints. All of these modules could communicate through a unified optimisation problem, operating continuously and coherently.

This is certainly an ambitious goal. However, if the robots are homogeneous, the environment is known, and the controllers are well-tuned, it would be a highly promising direction to explore. It would represent a modular yet integrated form of intelligence—not in the sense of what we see in large language models, but as a robust system capable of executing complex tasks in a dynamic environment. A true labour of love.

References

- [1] S. Haddadin, S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin, “The franka emika robot: A reference platform for robotics research and education,” *IEEE Robotics & Automation Magazine*, vol. 29, no. 2, pp. 46–64, 2022.
- [2] R. Tedrake and the Drake Development Team, “Drake: Model-based design and verification for robotics,” 2019.
- [3] S. Haddadin, “The franka emika robot: A standard platform in robotics research,” *IEEE Robotics & Automation Magazine*, 2024.
- [4] G. E. M. Anscombe, “Aristotle and the sea battle,” *Mind*, vol. 65, no. 257, pp. 1–15, 1956.
- [5] V. Goranko and A. Rumberg, “Temporal Logic,” in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta and U. Nodelman, eds.), Metaphysics Research Lab, Stanford University, Summer 2024 ed., 2024.
- [6] O. Maler and D. Nickovic, “Monitoring temporal properties of continuous signals,” in *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pp. 152–166, Springer, 2004.
- [7] S. Y. Nof, *Handbook of industrial robotics*. John Wiley & Sons, 1999.
- [8] J. G. C. Devol, “Programmed article transfer,” June 13 1961. US Patent 2,988,237.
- [9] J. Haviland and P. Corke, “Manipulator differential kinematics: Part i: Kinematics, velocity, and applications,” *IEEE Robotics & Automation Magazine*, 2023.
- [10] N. Lord, “The calculus of variations, by bruce van brunt. pp. 290.£ 54. 2004. isbn 0 387 40247 0 (springer).,” *The Mathematical Gazette*, vol. 89, no. 516, pp. 566–568, 2005.
- [11] N. S. Manton, “The principle of least action in dynamics,” 2013.

- [12] P.-L. MOREAU DE MAUPERTUIS, “Accord de différentes loix de la nature qui avoient jusqu’ici paru incompatibles,” *Histoire de l’academie royale des sciences et des belles-lettres de Berlin [pour l’annee], 1744*, pp. 417–426, 1748.
- [13] L. Euler, *Methodus Inveniendi Lineas Curvas Maximi Minimive Proprietate Gaudentes: Chapter II, A Method of Finding Absolute Maxima and Minima for Curved Lines*. 2013. Translation by Ian Bruce.
- [14] J. L. Lagrange, *Mécanique analytique*, vol. 1. Mallet-Bachelier, 1853.
- [15] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.
- [16] M. Sewlia, C. K. Verginis, and D. V. Dimarogonas, “Cooperative object manipulation under signal temporal logic tasks and uncertain dynamics,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11561–11568, 2022.
- [17] M. Sewlia, C. K. Verginis, and D. V. Dimarogonas, “Leader-follower cooperative manipulation under spatio-temporal constraints,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10312–10317, IEEE, 2024.
- [18] T. G. Sugar and V. Kumar, “Control of cooperating mobile manipulators,” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 94–103, 2002.
- [19] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, “Temporal logic motion planning for dynamic robots,” *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.
- [20] C. K. Verginis and D. V. Dimarogonas, “Timed abstractions for distributed cooperative manipulation,” *Autonomous Robots*, vol. 42, pp. 781–799, Apr. 2018.
- [21] A. Linard and J. Tumova, “Active learning of signal temporal logic specifications,” in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pp. 779–785, IEEE, 2020.
- [22] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, “Model predictive control with signal temporal logic specifications,” in *53rd IEEE Conference on Decision and Control*, pp. 81–87, IEEE, 2014.
- [23] L. Lindemann and D. V. Dimarogonas, “Control barrier functions for multi-agent systems under conflicting local signal temporal logic tasks,” *IEEE control systems letters*, vol. 3, no. 3, pp. 757–762, 2019.

- [24] O. Khatib, K. Yokoi, K. Chang, D. Ruspini, R. Holmberg, and A. Casal, "Co-ordination and decentralized cooperation of multiple mobile manipulators," *Journal of Robotic Systems*, vol. 13, no. 11, pp. 755–764, 1996.
- [25] J. T. Wen and K. Kreutz-Delgado, "Motion and force control of multiple robotic manipulators," *Automatica*, vol. 28, no. 4, pp. 729–743, 1992.
- [26] H.-C. Lin, J. Smith, K. K. Babarahmati, N. Dehio, and M. Mistry, "A projected inverse dynamics approach for multi-arm cartesian impedance control," *International Conference on Robotics and Automation (ICRA)*, pp. 1–5, 2018.
- [27] S. Erhart, D. Sieber, and S. Hirche, "An impedance-based control architecture for multi-robot cooperative dual-arm mobile manipulation," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 315–322, 2013.
- [28] A.-N. Ponce-Hinestroza, J.-A. Castro-Castro, H.-I. Guerrero-Reyes, V. Parra-Vega, and E. Olguyn-Dyaz, "Cooperative redundant omnidirectional mobile manipulators: Model-free decentralized integral sliding modes and passive velocity fields," *International Conference on Robotics and Automation (ICRA)*, pp. 2375–2380, 2016.
- [29] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 346–352, 2015.
- [30] S. Saha and A. A. Julius, "Task and motion planning for manipulator arms with metric temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 379–386, 2017.
- [31] 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)*, pp. 2997–3002, Dec. 2017.
- [32] T. Hsia, "Adaptive control of robot manipulators - a review," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 183–189, 1986.
- [33] P. Tomei, "Adaptive pd controller for robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, pp. 565–570, 1991.
- [34] R. Kelly, R. Carelli, M. Amestegui, and R. Ortega, "On adaptive impedance control of robot manipulators," in *Proceedings, 1989 International Conference on Robotics and Automation*, pp. 572–577 vol.1, 1989.

- [35] T. Yang, N. Sun, Y. Fang, X. Xin, and H. Chen, “New adaptive control methods for n -link robot manipulators with online gravity compensation: Design and experiments,” *IEEE Transactions on Industrial Electronics*, vol. 69, no. 1, pp. 539–548, 2022.
- [36] J. Fujishiro, Y. Fukui, and T. Wada, “Finite-time pd control of robot manipulators with adaptive gravity compensation,” in *2016 IEEE Conference on Control Applications (CCA)*, pp. 898–904, 2016.
- [37] A. Dietrich, C. Ott, and A. Albu-Schäffer, “Multi-objective compliance control of redundant manipulators: Hierarchy, control, and stability,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3043–3050, 2013.
- [38] R. Nebbia Colomba, R. Laha, L. F. Figueredo, and S. Haddadin, “Adaptive admittance control for cooperative manipulation using dual quaternion representation and logarithmic mapping,” in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 107–144, 2022.
- [39] A. Albu-Schaffer, C. Ott, U. Frese, and G. Hirzinger, “Cartesian impedance control of redundant robots: recent results with the dlr-light-weight-arms,” in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3, pp. 3704–3709 vol.3, 2003.
- [40] C. Jiao, L. Yu, X. Su, Y. Wen, and X. Dai, “Adaptive hybrid impedance control for dual-arm cooperative manipulation with object uncertainties,” *Automatica*, vol. 140, p. 110232, 2022.
- [41] M. Van Damme, P. Beyl, B. Vanderborght, V. Grosu, R. Van Ham, I. Vanderniepen, A. Matthys, and D. Lefeber, “Estimating robot end-effector force from noisy actuator torque measurements,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 1108–1113, 2011.
- [42] W.-H. Chen, D. Ballance, P. Gawthrop, and J. O’Reilly, “A nonlinear disturbance observer for robotic manipulators,” *IEEE Transactions on Industrial Electronics*, vol. 47, no. 4, pp. 932–938, 2000.
- [43] W. Dixon, I. Walker, D. Dawson, and J. Hartranft, “Fault detection for robot manipulators with parametric uncertainty: a prediction-error-based approach,” *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, pp. 689–699, 2000.
- [44] A. De Luca and R. Mattone, “Actuator failure detection and isolation using generalized momenta,” in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 1, pp. 634–639 vol.1, 2003.

- [45] A. Tsiamis, C. K. Verginis, C. P. Bechlioulis, and K. J. Kyriakopoulos, “Cooperative manipulation exploiting only implicit communication,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 864–869, 2015.
- [46] M. Logothetis, C. P. Bechlioulis, and K. J. Kyriakopoulos, “Decentralized impedance control of mobile robotic manipulators for collaborative object handling with a human operator,” in *2021 29th Mediterranean Conference on Control and Automation (MED)*, pp. 741–746, 2021.
- [47] M. Zribi, S. Ahmad, and S. Luo, “Adaptive control of redundant multiple robots in cooperative motion,” *Journal of Intelligent and Robotic Systems*, vol. 17, pp. 169–194, 1996.
- [48] B. Siciliano, O. Khatib, and T. Kröger, *Springer handbook of robotics*, vol. 200. Springer, 2008.
- [49] C. P. Bechlioulis and G. A. Rovithakis, “A low-complexity global approximation-free control scheme with prescribed performance for unknown pure feedback systems,” *Automatica*, vol. 50, pp. 1217–1226, Apr. 2014.
- [50] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, “Control barrier function based quadratic programs for safety critical systems,” *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.
- [51] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer-Verlag New York, 1998. OCLC: 1165444109.
- [52] J. P. LaSalle, “Stability theory and invariance principles,” in *Dynamical systems*, pp. 211–222, Elsevier, 1976.
- [53] O. Khatib, “Motion/force redundancy of manipulators,” in *Proceedings of Japan-USA Symposium on Flexible Automation*, vol. 1, pp. 337–342, 1990.
- [54] D. Q. Mayne and E. C. Kerrigan, “Tube-based robust nonlinear model predictive control,” *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 36–41, 2007. 7th IFAC Symposium on Nonlinear Control Systems.
- [55] M. Sewlia, C. K. Verginis, and D. V. Dimarogonas, “Cooperative sampling-based motion planning under signal temporal logic specifications,” in *2023 American Control Conference (ACC)*, pp. 2697–2702, IEEE, 2023.
- [56] M. Sewlia, C. K. Verginis, and D. V. Dimarogonas, “MAPS²: Multi-robot autonomous motion planning under signal temporal logic specifications,” *arXiv preprint arXiv:2309.05632*, 2023.
- [57] L. Lamport, “What good is temporal logic?,” *Information Processing 83, R. E. A. Mason, ed., Elsevier Publishers*, vol. 83, pp. 657–668, May 1983.

- [58] A. Donzé, “On signal temporal logic,” *International Conference on Runtime Verification*, pp. 382–383, 2013.
- [59] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, “Temporal logic motion planning for mobile robots,” *IEEE International Conference on Robotics and Automation*, pp. 2020–2025, 2005.
- [60] J. Ouaknine and J. Worrell, “On the decidability of metric temporal logic,” *IEEE Symposium on Logic in Computer Science (LICS’05)*, pp. 188–197, 2005.
- [61] P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine, and J. Worrell, “Timed temporal logics,” *Models, Algorithms, Logics and Tools*, pp. 211–230, 2017.
- [62] C. K. Verginis, C. Vrohidis, C. P. Bechlioulis, K. J. Kyriakopoulos, and D. V. Dimarogonas, “Reconfigurable motion planning and control in obstacle cluttered environments under timed temporal tasks,” *International Conference on Robotics and Automation*, 2019.
- [63] S. Karaman and E. Frazzoli, “Vehicle routing problem with metric temporal logic specifications,” *IEEE conference on decision and control*, pp. 3953–3958, 2008.
- [64] F. Fotiadis, C. K. Verginis, K. G. Vamvoudakis, and U. Topcu, “Assured learning-based optimal control subject to timed temporal logic constraints,” *IEEE conference on decision and control*, 2021.
- [65] A. Nikou, D. Boskos, J. Tumova, and D. V. Dimarogonas, “On the timed temporal logic planning of coupled multi-agent systems,” *Automatica*, vol. 97, pp. 339–345, 2018.
- [66] C. N. Mavridis, C. Vrohidis, J. S. Baras, and K. J. Kyriakopoulos, “Robot navigation under mitl constraints using time-dependent vector field based control,” *IEEE Conference on Decision and Control (CDC)*, pp. 232–237, 2019.
- [67] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, “Fly-by-logic: Control of multi-drone fleets with temporal logic objectives,” *ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 186–197, 2018.
- [68] N. Mehdipour, C.-I. Vasile, and C. Belta, “Arithmetic-geometric mean robustness for control from signal temporal logic specifications,” *American Control Conference (ACC)*, pp. 1690–1695, 2019.
- [69] L. Lindemann and D. V. Dimarogonas, “Efficient automata-based planning and control under spatio-temporal logic specifications,” *American Control Conference (ACC)*, pp. 4707–4714, 2020.

- [70] G. Yang, C. Belta, and R. Tron, “Continuous-time signal temporal logic planning with control barrier functions,” *American Control Conference (ACC)*, pp. 4612–4618, 2020.
- [71] J. Karlsson, F. S. Barbosa, and J. Tumova, “Sampling-based motion planning with temporal logic missions and spatial preferences,” vol. 53, no. 2, pp. 15537–15543, 2020.
- [72] G. Yang, B. Vang, Z. Serlin, C. Belta, and R. Tron, “Sampling-based motion planning via control barrier functions,” in *Proceedings of the 2019 3rd International Conference on Automation, Control and Robots*, pp. 22–29, 2019.
- [73] A. M. Ayala, S. B. Andersson, and C. Belta, “Temporal logic motion planning in unknown environments,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5279–5284, IEEE, 2013.
- [74] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, “Sampling-based motion planning with temporal goals,” in *2010 IEEE International Conference on Robotics and Automation*, pp. 2689–2696, 2010.
- [75] C. I. Vasile and C. Belta, “Sampling-based temporal logic path planning,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4817–4822, 2013.
- [76] E. M. Wolff, U. Topcu, and R. M. Murray, “Optimization-based trajectory generation with linear temporal logic specifications,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5319–5325, 2014.
- [77] V. Kurtz and H. Lin, “Mixed-integer programming for signal temporal logic with fewer binary variables,” *IEEE Control Systems Letters*, vol. 6, pp. 2635–2640, 2022.
- [78] L. Lindemann and D. V. Dimarogonas, “Robust motion planning employing signal temporal logic,” in *2017 American Control Conference (ACC)*, pp. 2950–2955, IEEE, 2017.
- [79] L. Lindemann and D. V. Dimarogonas, “Feedback control strategies for multi-agent systems under a fragment of signal temporal logic tasks,” *Automatica*, vol. 106, pp. 284–293, 2019.
- [80] D. Gundana and H. Kress-Gazit, “Event-based signal temporal logic synthesis for single and multi-robot tasks,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3687–3694, 2021.
- [81] D. Sun, J. Chen, S. Mitra, and C. Fan, “Multi-agent motion planning from signal temporal logic specifications,” *arXiv preprint arXiv:2201.05247*, 2022.

- [82] A. T. Buyukkocak, D. Aksaray, and Y. Yazıcıoğlu, “Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1375–1382, 2021.
- [83] I. Haghighi, S. Sadraddini, and C. Belta, “Robotic swarm control from spatio-temporal specifications,” *IEEE Conference on Decision and Control (CDC)*, pp. 5708–5713, 2016.
- [84] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, “Temporal-logic-based reactive mission and motion planning,” *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.
- [85] C. Belta and S. Sadraddini, “Formal methods for control synthesis: An optimization perspective,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 115–140, 2019.
- [86] S. Sadraddini and C. Belta, “Robust temporal logic model predictive control,” in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 772–779, 2015.
- [87] M. Charitidou and D. V. Dimarogonas, “Barrier function-based model predictive control under signal temporal logic specifications,” in *2021 European Control Conference (ECC)*, pp. 734–739, 2021.
- [88] F. Chen and D. V. Dimarogonas, “Funnel-based cooperative control of leader-follower multi-agent systems under signal temporal logic specifications,” in *2022 European Control Conference (ECC)*, pp. 906–911, 2022.
- [89] L. Lindemann and D. V. Dimarogonas, “Decentralized robust control of coupled multi-agent systems under local signal temporal logic tasks,” in *2018 Annual American Control Conference (ACC)*, pp. 1567–1573, 2018.
- [90] Y. Gilpin, V. Kurtz, and H. Lin, “A smooth robustness measure of signal temporal logic for symbolic control,” *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 241–246, 2020.
- [91] J. Hopcroft, J. Schwartz, and M. Sharir, “On the complexity of motion planning for multiple independent objects; pspace- hardness of the ”warehouseman’s problem”,” *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.
- [92] P. Yu and D. V. Dimarogonas, “Hierarchical control for uncertain discrete-time nonlinear systems under signal temporal logic specifications,” in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 1450–1455, 2021.
- [93] S. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” *Research Report 9811*, 1998.

- [94] M. Mesbahi and M. Egerstedt, “Graph theoretic methods in multiagent networks,” 2010.
- [95] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.
- [96] A. Falsone, K. Margellos, S. Garatti, and M. Prandini, “Dual decomposition for multi-agent distributed optimization with coupling constraints,” *Automatica*, vol. 84, pp. 149–158, 2017.
- [97] K. Margellos, A. Falsone, S. Garatti, and M. Prandini, “Distributed constrained optimization and consensus in uncertain networks via proximal minimization,” *IEEE Transactions on Automatic Control*, vol. 63, no. 5, pp. 1372–1387, 2017.
- [98] C.-I. Vasile, V. Raman, and S. Karaman, “Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications,” pp. 3840–3847, 2017.
- [99] A. Nedic and A. Ozdaglar, “Distributed subgradient methods for multi-agent optimization,” *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.
- [100] A. Daneshmand, G. Scutari, and V. Kungurtsev, “Second-order guarantees of distributed gradient algorithms,” *SIAM Journal on Optimization*, vol. 30, no. 4, pp. 3029–3068, 2020.
- [101] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [102] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, “Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning,” *Discrete Optimization*, vol. 19, pp. 79–102, 2016.
- [103] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, “Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. i–vii, 2019.
- [104] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1. Wiley, January 1968.
- [105] D. Lapandić, C. K. Verginis, D. V. Dimarogonas, and B. Wahlberg, “Kinodynamic motion planning via funnel control for underactuated unmanned surface vehicles,” *IEEE Transactions on Control Systems Technology*, vol. 32, no. 6, pp. 2114–2125, 2024.

- [106] D. J. Webb and J. P. van den Berg, “Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints,” *ArXiv*, vol. abs/1205.5088, 2012.
- [107] P. Poignet and M. Gautier, “Nonlinear model predictive control of a robot manipulator,” in *6th International Workshop on Advanced Motion Control. Proceedings (Cat. No.00TH8494)*, pp. 401–406, 2000.
- [108] F. Fotiadis and G. A. Rovithakis, “Input-constrained prescribed performance control for high-order mimo uncertain nonlinear systems via reference modification,” *IEEE Transactions on Automatic Control*, vol. 69, no. 5, pp. 3301–3308, 2024.
- [109] P. S. Trakas and C. P. Bechlioulis, “Robust adaptive prescribed performance control for unknown nonlinear systems with input amplitude and rate constraints,” *IEEE Control Systems Letters*, vol. 7, pp. 1801–1806, 2023.
- [110] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.
- [111] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, “Motion planning around obstacles with convex optimization,” *Science robotics*, vol. 8, no. 84, p. eadf7843, 2023.
- [112] M. Spahn, B. Brito, and J. Alonso-Mora, “Coupled mobile manipulation via trajectory optimization with free space decomposition,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12759–12765, IEEE, 2021.
- [113] O. Brock and L. Kavraki, “Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces,” in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, pp. 1469–1474 vol.2, 2001.
- [114] R. C. Y. Wong, M. Sewlia, A. Wiltz, and D. V. Dimarogonas, “Generating and optimizing topologically distinct guesses for mobile manipulator path planning,” 2024.
- [115] K. Patra, A. Sinha, and A. Guha, “Motion planning of nonholonomic cooperative mobile manipulators,” 2025.
- [116] Y. He, M. Wu, and S. Liu, “A cooperative optimization strategy for distributed multi-robot manipulation with obstacle avoidance and internal performance maximization,” *Mechatronics*, vol. 76, p. 102560, 2021.

- [117] N. Lissandrini, C. K. Verginis, P. Roque, A. Cenedese, and D. V. Dimarogonas, “Decentralized nonlinear mpc for robust cooperative manipulation by heterogeneous aerial-ground robots,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1531–1536, 2020.
- [118] Y. He, M. Wu, and S. Liu, “An optimisation-based distributed cooperative control for multi-robot manipulation with obstacle avoidance,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9859–9864, 2020. 21st IFAC World Congress.
- [119] J. He, E. Bartocci, D. Ničković, H. Isakovic, and R. Grosu, “Deepstl – from english requirements to signal temporal logic,” 2022.
- [120] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, “Nl2tl: Transforming natural languages to temporal logics using large language models,” 2024.
- [121] W. S. Cortez, C. K. Verginis, and D. V. Dimarogonas, “A distributed, event-triggered, adaptive controller for cooperative manipulation with rolling contacts,” *IEEE Transactions on robotics*, vol. 39, no. 4, pp. 3120–3133, 2023.
- [122] R. Wolf, Y. Shi, S. Liu, and R. Rayyes, “Diffusion models for robotic manipulation: A survey,” 2025.