Hierarchical Control for Multi-Robot Systems using Contracts

Shashank Dilip Kumar

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Hierarchical Control for Multi-Robot Systems using Contracts

Master of Science Thesis

For the degree of Master of Science in Embedded Systems at Delft University of Technology

Shashank Dilip Kumar

October 24, 2024

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) &
Mechanical Engineering (ME) · Delft University of Technology

Abstract

Modern control systems require control strategies that can handle multiple levels of abstraction while providing formal guarantees of system behavior. Traditional hierarchical control approaches often lack formal interfaces between different layers of abstraction, making it challenging to ensure system-wide properties and extend implementations across different platforms.

This thesis presents a contract-based hierarchical control framework that provides formal guarantees across different layers of abstraction. Building upon recent developments in vertical contracts with heterogeneous refinements, we develop a three-layer control architecture for robot surveillance tasks, decomposing the problem into motion planning, trajectory generation, and tracking control. Each layer operates at its own time scale and uses appropriate model abstractions, with contracts ensuring system-wide guarantees while vertical system relationships enable proper interfacing between layers.

We validate our framework through both simulation studies and physical implementation on Elisa3 robots. The simulation results demonstrate the framework's ability to satisfy high-level temporal logic specifications while maintaining guarantees at each layer. The physical implementation reveals practical challenges in transitioning from simulation to hardware, leading to adaptations in the control architecture that maintain theoretical guarantees while accommodating hardware constraints.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank my supervisor Dr. Manuel Mazo Jr. for his support and guidance during the writing of this thesis. I deeply valued being part of our weekly group meetings, where the discussions with Manuel and the rest of the team not only broadened my research perspectives but also gave me a glimpse into life in academic research.

I would also like to thank my friends for their support and for engaging with me in the most random conversations.

Finally, I would like to thank my mother and father, who are also my closest friends, for their unwavering support throughout my life.

Delft University of Technology
October 24, 2024

Shashank Dilip Kumar

Chapter 1

Introduction

One of the fundamental challenges in control engineering has been the effective control of complex dynamical systems. Such systems, characterized by nonlinear or high-order dynamics, must satisfy various specifications while operating under multiple constraints. This complexity is particularly evident in multi-agent systems, where multiple robots or autonomous agents must coordinate their actions while maintaining individual performance objectives.

The complexity of modern control systems can be effectively managed through hierarchical decomposition strategies. This approach, which systematically breaks down complex systems into simpler, interconnected subsystems, enables structured analysis and control design across multiple time scales. The theoretical foundations of hierarchical systems theory, established in the late 1960s and 1970s to solve optimization problems, continue to provide valuable insights today. The fundamental principle, as described in [3], involves decomposing complex, high-order systems into several simpler, low-order subsystems while ensuring overall system objectives are met.

In a typical hierarchical structure, the "supervisory" unit in the top layer coordinates the "local" subsystem units in the bottom layer to achieve high-level objectives. A key advantage of this approach is its ability to handle different levels of system abstraction: while the lowest layer maintains the full complex model of the system, higher layers can operate with simplified models that mask unnecessary lower-level details. This layered approach to system abstraction simplifies both decision-making and control design across the hierarchy.

The hierarchical design approach finds applications across various domains today. For instance, in automobile manufacturing, the overall production process is decomposed into specialized tasks such as painting, assembly, welding, and inspection, each performed by dedicated robots. This task decomposition transforms the complex manufacturing process

into smaller, simpler sub-tasks that can be executed independently. Similarly, in swarm robotics applications for environment monitoring, hierarchical control enables efficient task distribution among robots surveying specific regions, leading to more effective resource utilization.

Traditional hierarchical control approaches often provide guarantees for individual layer objectives, which collectively ensure desired system behavior. However, these guarantees are typically system or problem-specific and lack generalizability across different implementations. Contract theory [4] offers a solution to this limitation by establishing formal sets of assumptions and guarantees for each system component. These contracts provide a template where components need only satisfy their specified guarantees, enabling modular design and component interchangeability.

However, existing contract compositions primarily work horizontally—for components operating within the same modeling domain. This becomes problematic when dealing with hierarchical systems where different layers operate with heterogeneous models. The work in [5] addresses this challenge by introducing vertical contracts with heterogeneous refinements, enabling interfaces between different modeling domains. Building on this concept, Mazo et al. [1] developed a hierarchical control framework that accommodates different model abstractions operating at different time scales across its layers. Their framework provides formal guarantees through contracts at each layer, which, when composed vertically, ensure desired overall system behaviour. This compositional approach enables independent controller design at each layer, as fulfilling individual layer contracts is sufficient for satisfaction of system-wide guarantees.

Contributions and outline

This thesis extends the work of [1], which was initially demonstrated through simulation experiments for single robot systems. Our framework implements a natural hierarchical decomposition of a robot control problem: from high-level motion planning, through trajectory generation, to low-level tracking control. Our primary contribution focuses on implementing this hierarchical control framework on physical robotic systems and exploring its extension to multi-agent systems.

The remainder of this thesis is organized as follows. Section 2 reviews related work across multiple domains: methods for system abstraction,motion planning using temporal logic specifications, and contract theory and its applications, in control theory and robotics. Section 3 introduces key concepts and mathematical preliminaries essential for understanding our framework. Section 4 develops the theoretical foundation of vertical system relationships, which is crucial for interfacing different layers of our hierarchy. Building upon these relationships, Section 5 presents our structured hierarchical control framework, detailing the design and integration of different control layers.

Section 6 describes the experimental setup, including both simulation and hardware platforms, while Section 7 validates our framework through comprehensive simulation studies.

Section 8 extends these results to physical implementation on Elisa3 robots, addressing the practical challenges encountered in transitioning from simulation to hardware. Finally, Section 9 concludes the thesis with a summary of contributions and discusses potential directions for future research.

Chapter 2

Literature Review

2-1 Abstractions

System abstraction is a fundamental concept in control theory that enables us to compare the behaviours of different systems, typically a complex system with a simpler one. The key idea is to capture essential properties of a complex system while hiding unnecessary details, allowing us to reason about system behaviour at an appropriate level of detail. For instance, when dealing with robot motion control, we might abstract a robot's complex nonlinear dynamics into a simpler kinematic model for high-level planning purposes.

Formal methods like simulation relations provide a rigorous framework for establishing these abstractions [6]. A simulation relation between two systems ensures that any behaviour of the concrete (more detailed) system can be mapped to a corresponding behaviour in the abstract (simpler) system. This formal relationship guarantees that properties verified for the abstract system hold for the concrete system as well.

The power of abstractions becomes particularly evident in hierarchical system design. By creating multiple levels of abstraction, we can decompose complex control problems into more manageable subproblems. Each layer operates with a system model of appropriate complexity: higher layers use simpler abstractions for tasks like planning and decision-making, while lower layers work with more detailed models for precise control execution. This hierarchical decomposition, supported by formal abstractions, enables us to handle complex control problems while maintaining rigorous guarantees across different levels of the hierarchy.

To illustrate the concept of abstraction, consider a car moving in a space partitioned into three regions A, B, and C (see Figure 2-1). At the concrete level, the car's behaviour is described by detailed differential equations capturing its dynamics, including position, velocity, steering angle, and physical constraints. However, for high-level planning, we can

abstract this complex model into a simpler transition system where the car's location is represented only by which partition (A, B, or C) it currently occupies. In this abstracted model, the car's movement between partitions is represented as discrete transitions ($A \rightarrow B$, $B \rightarrow C$), while the detailed dynamics of how the car actually moves within and between partitions are hidden. This abstraction preserves the essential property of reachability between regions while significantly simplifying the system representation for high-level planning tasks.

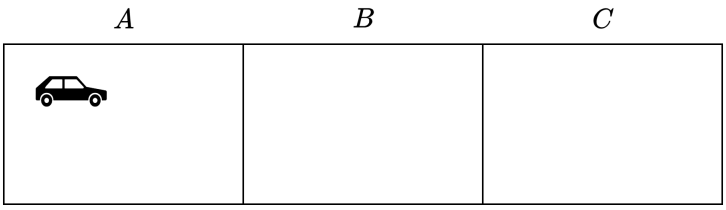

Figure 2-1: Example for system abstraction

2-2 Motion planning for robots

Motion planning is a fundamental problem in robotics where the objective is to generate a sequence of actions that takes a robot from an initial state to a goal state while satisfying various constraints. Traditional motion planning approaches typically focus on point-to-point navigation [7, 8] or trajectory optimization between waypoints [9, 10]. However, many modern robotic applications, particularly in surveillance and monitoring, require more complex behavioural specifications that go beyond simple waypoint navigation - such as periodic area coverage, ordered task execution, or maintaining safety constraints throughout the mission [11, 12, 13, 14].

These complex behavioural requirements have motivated the use of formal methods in robot motion planning, particularly Linear Temporal Logic (LTL) [15, 16]. Temporal logic is a formalism that enables reasoning about sequences of events over time. LTL, first introduced in [17] for specifying and verifying properties of computer programs, has emerged as a powerful tool for specifying the behaviour of dynamical systems. Over the past two decades, temporal logic has gained significant attention in both robotics and control systems research as a powerful tool for specifying complex behavioural requirements that traditional control specifications cannot easily capture. This shift towards temporal logic specifications helps bridge the gap between software systems and control systems, as temporal logic properties can be efficiently verified using simple software models like finite state machines (FSM) [16].

An LTL formula "ϕ" represents an infinite sequence of events, where each event has an unique successor event in time. Properties expressed in natural language can be translated

into LTL formulas using temporal operators, boolean operators and atomic propositions. Consider a simple progress property, which states that every request must eventually be followed by a response [15]. This property can be expressed using the LTL formula:

$$\phi = \Box(request \rightarrow \Diamond response)$$

where $\Box$ and $\Diamond$ are temporal properties "always" and "eventually", and the symbols "request" and "response" are the atomic propositions that take boolean values. This formula specifies that whenever the proposition "request" becomes true, the proposition "response" must become true at some point in the future. Such LTL formulas can be efficiently verified against finite state machines.

Similarly, we can express robot motion planning tasks using LTL formulas. Consider a robot operating in an environment partitioned into finite regions labeled from A to I (see Figure 2-2). For a surveillance task requiring the robot to periodically visit regions A and F while avoiding region E, we can encode these requirements as the LTL formula:

$$\phi = \Box\Diamond A \wedge \Box\Diamond F \wedge \Box\neg E$$

This formula specifies that the robot must infinitely often visit regions A and F ($\Box\Diamond$ indicating "always eventually") while always avoiding region E. Such LTL specifications generate high-level motion plans that can then be refined into concrete robot trajectories using lower-level planning algorithms.

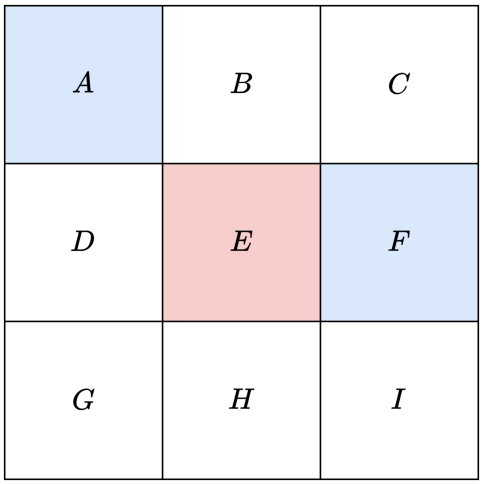

Figure 2-2: An example partitioned space

This formula combines both liveness properties ($\Box\Diamond A$ and $\Box\Diamond F$), which ensure something "good" eventually happens (in this case, regions A and F are visited infinitely often), and safety properties ($\Box\neg E$), which ensure something "bad" never happens (the robot never enters region E). These fundamental property types form building blocks for constructing more complex LTL specifications.

The abstraction methods discussed in the previous section play a crucial role in implementing LTL specifications on dynamical systems. While LTL synthesis generates high-level motion plans using finite state machines (FSMs), robotic systems are typically described by continuous-state dynamical models, often nonlinear and subject to various constraints. This disparity in system representations necessitates a formal bridge between the discrete planning layer and the continuous execution layer.

Simulation relations provide this critical link by establishing formal behavioural relationships between the concrete robot dynamics and the abstract FSM. These relations guarantee that any behaviour possible in the concrete system can be mapped to a corresponding behaviour in the abstract system, ensuring that solutions found at the abstract level (FSM) can be faithfully implemented by the concrete system (robot). This approach not only provides theoretical guarantees of correctness but also enables the systematic refinement of high-level plans into executable trajectories that respect the robot's dynamic constraints[1, 18, 19].

This abstraction naturally leads to a two-layer hierarchical control structure: an upper layer working with the simpler FSM for discrete planning based on LTL specifications, and a lower layer handling the concrete robot dynamics for trajectory execution [1, 20, 21]. This can be viewed as a form of hierarchical control where complex control tasks are decomposed across layers of varying abstraction, with each layer handling tasks appropriate to its level of system complexity. The upper layer focuses on satisfying logical specifications using simplified system representations, while the lower layer manages the detailed dynamic behaviour of the physical system.

We now review existing approaches to LTL-based motion planning for single-robot systems, before discussing extensions to multi-robot scenarios.

2-2-1 Planning for single-agent systems

Control synthesis for LTL specifications has been addressed through various methodologies. The traditional approach relies on automata theory, where the LTL formula is converted to a Büchi/Rabin automaton and the synthesis problem is solved as a two-player game [15, 16]. While this method guarantees completeness, it suffers from state explosion in the Büchi automaton construction, where the state space grows exponentially with both the length of the LTL formula and the number of atomic propositions involved. This issue becomes particularly challenging when specifications include multiple temporal operators or when dealing with intricate behavioural requirements involving multiple system properties. Alternative approaches include graph-based search methods that find satisfying paths in product automata [11, 22], and optimization-based techniques that incorporate LTL constraints into optimization frameworks [23].

There also exists a fragment of LTL called GR(1) (Generalized Reactivity of rank 1) that restricts specifications to environment assumptions and system guarantees, enabling polynomial-time synthesis compared to the doubly exponential complexity of general LTL

synthesis [24, 21, 25, 26]. While traditional LTL synthesis methods compute complete strategies offline, GR(1)'s reduced computational complexity enables online synthesis of reactive controllers that can adapt to changing environment conditions. The authors in [25] develop a receding horizon framework to address the complexity of GR(1) synthesis, with their implementation available through the TuLiP Python toolbox [27].

In [20, 18], hybrid controllers are developed from LTL specifications, with [20] utilizing approximate simulation relations for more robust system abstraction. A notable recent work [28] presents a three-layer hierarchical control architecture for robots in partially observable environments. The framework synthesizes motion plans from syntactically co-safe LTL (scLTL) specifications at the top layer, generates trajectories using MPC at the middle layer, and employs control barrier functions (CBF) at the bottom layer to ensure bounded tracking errors.

An extension of LTL is Signal Temporal Logic (STL), which offers enhanced expressiveness for system specifications [29, 30, 31, 32]. STL, a predicate-based logic interpreted over continuous-time signals, enables the specification of quantitative temporal and spatial properties. A key feature of STL is its ability to measure the degree of specification satisfaction (robustness), in contrast to the boolean satisfaction metrics of LTL. However, while STL offers more expressive power in defining system specifications, it typically incurs higher computational complexity compared to LTL synthesis.

2-2-2 Planning for multi-agent systems

Planning for multi-agent systems introduces additional complexity compared to single-agent LTL synthesis. While individual agents can have local specifications governing their behaviour, the overall system often needs to satisfy global specifications that require coordination among agents. The synthesis problem can be approached either top-down, where a global specification is decomposed into local specifications for each agent, or bottom-up, where individual agent behaviours are composed to satisfy global requirements. Similar to single-agent cases, solutions employ automata-based, graph-search, or optimization methods [33]. However, the key challenge lies in handling the exponential growth of the state space with the number of agents and managing the coupling between agent behaviours through their specifications. This has led to specialized techniques focusing on decomposition and distributed synthesis to manage computational complexity.

Early works like [34] and [35] established frameworks for automatically generating motion plans from global LTL specifications for robot teams. These centralized approaches, while guaranteeing correctness, faced scalability challenges with increasing number of agents. This led to the development of distributed and decentralized solutions.

One direction focuses on decomposing global specifications into local tasks. [36] and [37] propose decentralized frameworks where agents coordinate through local LTL specifications while maintaining global objectives. This approach is further developed in [38], which

addresses both motion and task specifications in a distributed manner. [39] extends this by enabling dynamic plan reconfiguration under local specifications.

Alternative approaches include sampling-based methods [40] for handling global temporal specifications, and optimization-based techniques combining LTL with control barrier certificates [41] for safety-critical applications. [42] focuses on optimal path planning for multi-robot systems while [26] addresses reactive mission planning with dynamic obstacle avoidance and deadlock resolution.

2-2-3 Limitations of existing methods

Existing approaches in hierarchical control and temporal logic-based motion planning, while effective for their specific contexts, have several limitations. Most hierarchical frameworks focus on the decomposition of tasks across layers but lack formal guarantees when different layers use heterogeneous models [20]. For instance, transitioning between discrete plans and continuous controllers often relies on informal bridges between abstraction layers. Additionally, while some works provide system guarantees, these are typically specific to particular system types or controller designs, limiting their broader applicability [28]. Moreover, the independent development of controllers at different hierarchical levels remains challenging, as most approaches tightly couple the design of high-level planners with low-level controllers.

2-3 Contract theory

Contract theory was popularized in the early 1990s to address complexity of systems in the field of software engineering [43]. The primary objective at that time was to improve the reliability of software systems. Contract-based system design is capable of relating systems in different layers of abstraction and also enable component-based design in a horizontal layer. Over time, researchers have extended this concept to cyber-physical systems (CPS), making it possible to incorporate contract theory with physical processes using mathematical meta-theory.

Contract theory, particularly in the form of assume-guarantee contracts (AGC), offers several key advantages in modern system design. First, it provides a formal framework that simplifies complex system design: each component's responsibilities are explicitly defined through its contract, making system-wide properties easier to verify. Second, the compositional nature of contracts enables independent development and verification of components, facilitating parallel development by different teams or organizations.

Moreover, contracts serve as a powerful tool for system integration and maintenance. They provide explicit documentation of component interactions and requirements, making it easier to identify and resolve integration issues. This formal specification of interfaces

and behaviours is particularly valuable in large-scale systems where components may be developed by different suppliers or updated over time.

We now look at an example of an assume-guarantee contract from [4]:

$$\mathcal{C}_1 : \begin{cases} \text{variables: } \begin{cases} \text{inputs: } x, y \\ \text{outputs: } z \end{cases} \\ \text{types: } x, y, z \in \mathbb{R} \\ \text{assumption } A_1\text{: } y \neq 0 \\ \text{guarantee } G_1\text{: } z = x/y \end{cases} \tag{2-1}$$

This simple contract describes a division operation: under the assumption that the denominator y is non-zero (A_1), the contract guarantees that the output z will be equal to x divided by y (G_1). This illustrates the fundamental structure of assume-guarantee contracts, where assumptions specify the conditions under which the component must operate, and guarantees specify what the component promises to deliver under these conditions.

2-3-1 Application to dynamical systems

A direction focused on behavioural specifications, where [44, 45] establish contracts with different perspectives on system behaviour: [45] captures pure output behaviour, while [44] specifies guarantees as input-output behaviours. [46] introduces assume-guarantee contracts as a framework for specifying control system requirements for dynamical systems in driving variable form, where system behaviour and its environment are compared through simulation relations. Both [44, 46] demonstrate the application of contracts in control systems through implementation on vehicle follower systems.

Parametric assume-guarantee contracts [47] enable more precise specification of guarantees through the classification of assumptions into sub-classes, each triggering specific guarantees when fulfilled. This framework has been extended to controller synthesis for interconnected systems in [48], where parametric AGCs are formulated as STL specifications. [49] addresses safety synthesis for complex discrete-time interconnected systems, where controllers are synthesized for each subsystem to ensure operation within their safe sets.

One drawback of traditional contract composition and refinement operations is that they are applicable only to homogeneous systems operating in the same modeling domain. For systems with heterogeneous models, [5] introduces the concept of vertical contracts. These contracts establish behavioural mappings between different modeling domains, enabling contract operations across heterogeneous system models.

Building upon this concept of vertical contracts, [1] develops a hierarchical control framework that addresses the limitations of existing approaches. The framework provides system guarantees across heterogeneous models operating in different signal spaces through vertical contracts. Its compositional design principles allow for independent development of

controllers at each layer. Moreover, the framework offers flexibility in implementation: components or controllers can be replaced with alternative designs as long as they satisfy their respective contracts, maintaining overall system guarantees.

2-4 Proposed solution

Building upon the hierarchical control framework presented in [1], we extend and implement a three-layer control architecture for robot surveillance problems. Each layer operates at a different abstraction level and time scale, with formal contracts ensuring proper interaction between layers. While [1] demonstrated the theoretical framework through simulations, our work focuses on extending it to physical robot implementation and exploring multi-agent scenarios.

The top layer handles high-level motion planning using a finite transition system representation of the robot's workspace. Given an LTL specification describing the surveillance requirements, this layer generates a discrete motion plan that satisfies the high-level task.

The middle layer bridges the gap between discrete planning and continuous motion by generating feasible trajectories that realize the high-level plan. Operating in discrete time, this layer employs Model Predictive Control (MPC) [50] to generate feasible trajectories while respecting the high-level system transitions and other system constraints.

The bottom layer ensures precise trajectory tracking using the robot's full nonlinear dynamics. It first interpolates the discrete trajectory points from the middle layer to generate a continuous reference trajectory. A feedback linearization controller [51, 52] is then designed to track this interpolated trajectory, effectively handling the robot's nonlinear dynamics.

Formal contracts are defined at each layer, with each contract guaranteeing its layer's model abstraction and output property requirements. Through vertical composition of these contracts, we obtain formal guarantees for the overall system behaviour. The interaction between layers is formalized through vertical system relationships, where transducer maps embedded with simulation relations establish abstractions between systems operating in different signal spaces. This formal relationship, combined with compositional design principles, enables the transfer of controller properties across different signal spaces - from finite transitions to discrete-time signals, and from discrete-time to continuous-time signals.

We validate our framework through both simulation and hardware experiments. For the single-robot case, we present comprehensive simulation results followed by implementation on physical Elisa3 robots. We also demonstrate the framework's extensibility to multi-robot systems through simulation studies.

Chapter 3

Preliminaries

We define $\mathcal{L}_\infty^n$ to denote the set of bounded signals from time to Euclidean space $x : \mathbb{R}^+ \to \mathbb{R}^n$, $\mathcal{D}_\infty^n$ for bounded discrete time signals $x_d : \mathbb{N} \to \mathbb{R}^n$, and Σ^ω for the set of infinite-length sequences over some alphabet Σ of finite cardinality, and $\Sigma^{\mathcal{X}}$ for the space of continuous time signals with image on Σ, i.e., $x \in \Sigma^{\mathcal{X}}$. We use $2^{\mathcal{X}}$ to denote the power set of $\mathcal{X}$. We simplify notation by writing $x, y \in \mathcal{L}_\infty$ to denote that both signals belong to their corresponding $\mathcal{L}_\infty^n$, with possibly different n for x and y.

The projection map on the i-th entry is denoted by π_i, e.g., $\pi_2 : (x, y) \mapsto y$. To simplify notation, we sometimes denote by $\mathcal{F}(A)$ the set of all values resulting from applying the function $\mathcal{F}$ to every element in A, i.e., $\mathcal{F}(A) = \{y \mid y = \mathcal{F}(x), x \in A\}$, where $\mathcal{F} : X \to Y$ and $A \in 2^X$. For a differential equation $\dot{\xi}_{u,x}(t) = f(\xi_{u,x}(t), u(t))$, $\xi(0) = x$, we denote $\xi_{u,x}(0 : \tau)$ as the solution to the differential equation $\dot{\xi}_{u,x}(t)$ with initial condition x and input u, restricted to the time segment 0 to τ.

3-1 Systems

3-1-1 System definitions

Definition 3-1.1 (System [6, 16])**.** *A system S is a tuple $S = (X, X_0, U, \longrightarrow, Y, H)$ consisting of*

- *a (possibly infinite) set of states X;*
- *a (possibly infinite) set of initial states $X_0 \subseteq X$;*
- *a (possibly infinite) set of inputs U;*

- *a transition relation* $\longrightarrow \subseteq X \times U \times X$;
- *a set of outputs* Y
- *an output map* $H : X \to Y$

The states in X are referred to as internal states, and the outputs are externally visible. A system is called *finite-state* if the set X is finite. The system is called *infinite-state* if the set X is infinite. System evolution is captured by transitions, which is defined by $(x, u, x') \in \longrightarrow$, with $x, x' \in X$ and $u \in U$. This transition will be denoted as $x \xrightarrow{u} x'$. In the transition relation $x \xrightarrow{u} x'$, x' is called the *u-successor* or *successor* state of x. Similarly, x is called a *u-predecessor* or *predecessor* state of x'. We denote the set of u-successor states of a state x as $Post_u(x)$, and its set of u-predecessor states as $Pre_u(x)$. A state x may have no successors, a single successor, or multiple successor states. We also denote $U(x)$ as the set of inputs $u \in U$ for which $Post_u(x)$ is non-empty. A system is called *deterministic* if for any state $x \in X$ and $u \in U$, there exists at most one successor state. An equivalent expression for deterministic systems is for any state $x \in X$ and any input $u \in U$, $x \xrightarrow{u} x'$ and $x \xrightarrow{u} x''$ imply $x' = x''$, assuming that both x' and x'' exist.

For multi-agent systems, the collective behaviour of all individual transition systems can be captured by constructing their product transition system, which represents all possible combinations of states and transitions of the constituent systems.

Definition 3-1.2 (Product Transition System (PTS) [40])**.** *Given N transition systems $S_i = (X_i, X_{i,0}, U_i, \underset{i}{\longrightarrow}, Y_i, H_i)$, $i = 1, 2, \ldots, N$, the product transition system $S_{\mathcal{P}} = S_1 \otimes S_2 \otimes \cdots \otimes S_N$ is a tuple $S_{\mathcal{P}} = (X_{\mathcal{P}}, X_{\mathcal{P},0}, U_{\mathcal{P}}, \underset{\mathcal{P}}{\longrightarrow}, Y_{\mathcal{P}}, H_{\mathcal{P}})$ consisting of*

- *a (possibly infinite) set of states $X_{\mathcal{P}} = X_1 \times X_2 \times \ldots X_N$;*
- *a (possibly infinite) set of initial states $X_{\mathcal{P},0} \subseteq X_{\mathcal{P}}$;*
- *a (possibly infinite) set of inputs $U_{\mathcal{P}}$;*
- *a transition relation $\underset{\mathcal{P}}{\longrightarrow} \subseteq X_{\mathcal{P}} \times U_{\mathcal{P}} \times X_{\mathcal{P}}$;*
- *a set of outputs $Y_{\mathcal{P}}$*
- *an output map $H_{\mathcal{P}} : X_{\mathcal{P}} \to Y_{\mathcal{P}}$*

The transition $x_{\mathcal{P}} \xrightarrow{u_{\mathcal{P}}} x'_{\mathcal{P}}$ holds true if and only if $\bigwedge_{i=1}^{N}(x_i \xrightarrow{u_i} x'_i)$ is true.

3-1-2 System behaviour

We will now define system behaviour. Given any state $x \in X$, a finite internal behaviour generated from x is a finite sequence of transitions

$$x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} \cdots \xrightarrow{u_{n-2}} x_{n-1} \xrightarrow{u_{n-1}} x_n$$

such that $x_0 = x$ and $x_i \xrightarrow{u_i} x_{i+1}$ for all $0 \leq i < n$, $i, n \in \mathbb{N}$. A finite internal behaviour can be extended to infinite internal behaviour, which is an infinite sequence of states instead of the finite case.

Similarly, using an output map, we can extract an external behaviour for every internal behaviour. A finite external behaviour corresponding to the finite internal behaviour is

$$y_0 \longrightarrow y_1 \longrightarrow y_2 \longrightarrow \cdots \longrightarrow y_{n-1} \longrightarrow y_n$$

with $H(x_i) = y_i \in Y$ for all $0 \leq i < n$, $i, n \in \mathbb{N}_0$. An external behaviour is also represented using the notation $\mathcal{Y} = y_0 y_1 y_2 \dots y_n$. We denote the finite external behaviour defined by the finite internal behaviour generated from state $x \in X_0$ as $\mathcal{B}_x(S)$, where S refers to the system. The set of all external behaviours generated by the system S is given by $\mathcal{B}(S) = \bigcup_{x \in X_0} \mathcal{B}_x(S)$. For output deterministic systems, we can uniquely identify a finite internal behaviour of a system given a finite external behaviour.

Similar to internal behaviour, finite external behaviour can be extended to infinite external behaviour. We denote the infinite external behaviour defined by the infinite internal behaviour generated from state $x \in X_0$ as $\mathcal{B}^\omega_x(S)$, where S refers to the system. The set of all external behaviours generated by the system S is given by $\mathcal{B}^\omega(S) = \bigcup_{x \in X_0} \mathcal{B}^\omega_x(S)$.

3-1-3 Similarity relationships

We will now look at means to establish a relationship between two systems.

Definition 3-1.3 (Simulation relation [6])**.** *Given two systems S_a and S_b with $Y_a = Y_b$, a relation $R \subseteq X_a \times X_b$ is a simulation relation from S_a to S_b if the following conditions are satisfied:*

1. *for every $x_{a0} \in X_{a0}$, there exists $x_{b0} \in X_{b0}$ with $(x_{a0}, x_{b0}) \in R$;*
2. *for every $(x_a, x_b) \in R$ we have $H_a(x_a) = H_b(x_b)$;*
3. *for every $(x_a, x_b) \in R$ we have that $x_a \xrightarrow[a]{u_a} x'_a$ in S_a implies the existence of $x_b \xrightarrow[b]{u_b} x'_b$ in S_b satisfying $(x'_a, x'_b) \in R$.*

If there exists a simulation relation from S_a to S_b, we say that S_b simulates S_a, denoted by $S_a \preceq_{\mathcal{S}} S_b$.

Let us elaborate the meaning of the above definition. Firstly, the initial state in system S_a must be simulated by some initial state in S_b. Secondly, if a state in S_b simulates a state in S_a, then their outputs must be equal. Finally, for a transition in S_a, there must exist an equivalent transition in S_b such that the pair of successor states also satisfy the simulation relation given that the pair of predecessor states satisfy the relation. We call S_b to be an *abstraction* of S_a and S_a to be a *refinement* of S_b. We can also define R to be a bisimulation relation if there exists a simulation relation R from S_a to S_b and a simulation relation R^{-1} from S_b to S_a, and we say that S_a is bisimilar to S_b, denoted by $S_a \cong_{\mathcal{S}} S_b$

When we deal with control systems, in addition to the existence of matching transitions, we also require the existence of control inputs that forces these transitions. Including the condition of matching control inputs help us distinguish when the system is acted upon by the controller and when it is acted upon by the environment. An alternating simulation relation captures this relation, providing an extension to the classic simulation relation.

Definition 3-1.4 (Alternating simulation relation [6])**.** *Given two systems S_a and S_b with $Y_a = Y_b$, a relation $R \subseteq X_a \times X_b$ is an alternating simulation relation from S_a to S_b if the following conditions are satisfied:*

1. *for every $x_{a0} \in X_{a0}$, there exists $x_{b0} \in X_{b0}$ with $(x_{a0}, x_{b0}) \in R$;*
2. *for every $(x_a, x_b) \in R$ we have $H_a(x_a) = H_b(x_b)$;*
3. *for every $(x_a, x_b) \in R$ and for every $u_a \in U_a(x_a)$, there exists $u_b \in U_b(x_b)$ such that for every $x'_b \in Post_{u_b}(x_b)$ there exists $x'_a \in Post_{u_a}(x_a)$ satisfying $(x'_a, x'_b) \in R$.*

If there exists an alternating simulation relation from S_a to S_b, we say that S_a is alternating simulated by S_b, denoted by $S_a \preceq_{\mathcal{AS}} S_b$.

In the case of deterministic systems, where $|Post_{u_a}| \leq 1$ and $|Post_{u_b}| \leq 1$, the definition of alternating simulation relation simplifies to a simulation relation. Specifically, the final condition in each definition is the same.

3-2 Linear Temporal Logic

An LTL formula "ϕ" typically represent an infinite sequence of events, where each event has an unique successor event in time. Events are described using natural language and translated into LTL formulas using special symbols. An LTL formula is constructed using a set of observations or atomic propositions, Boolean operators ($\top$ or "true", $\neg$ or "negation", and $\wedge$ or "conjunction"), and temporal operators ($\bigcirc$ or "next", and $\mathcal{U}$ or "until") [15, 16]. We define the syntax of LTL formulas as follows:

Definition 3-2.1 (LTL syntax [16]). *A (propositional) Linear Temporal Logic (LTL) formula ϕ over a given set of atomic propositions $\mathcal{AP}$ is recursively defined as*

$$\phi = \top \mid p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2, \tag{3-1}$$

where $p \in \mathcal{AP}$ is an atomic proposition, and ϕ, ϕ_1 and ϕ_2 are LTL formulas

Additional operators are defined as follows:

$$\begin{aligned} \phi_1 \vee \phi_2 &:= \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \phi_1 \rightarrow \phi_2 &:= \neg\phi_1 \wedge \phi_2 \\ \phi_1 \leftrightarrow \phi_2 &:= (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1), \end{aligned} \tag{3-2}$$

where $\vee, \rightarrow$ and $\leftrightarrow$ stand for disjunction, implication and equivalence respectively. Additionally, the temporal operators $\Diamond$ ("eventually") and $\Box$ ("always") are defined as

$$\begin{aligned} \Diamond &:= \top \mathcal{U} \phi \\ \Box &:= \neg\Diamond\neg\phi \end{aligned} \tag{3-3}$$

As we mentioned earlier, an LTL formula is represented using an infinite sequence of events. We describe these events as words made of atomic propositions from $\mathcal{AP}$. In other terms, the infinite sequence of words are made over the set of infinite propositions $\mathcal{AP}^\omega$. Formally, we define the LTL semantics as follows:

Definition 3-2.2 (LTL semantics [16][1]). *The satisfaction of formula ϕ over a set of propositions $\mathcal{AP}$ at position $k \in \mathbb{N}$ by word $\sigma = \sigma_0\sigma_1\sigma_2... \in (2^{\mathcal{AP}})^\omega$, denoted by $\sigma \models \phi$, is defined recursively as follows:*

- $\sigma \models \top$;
- $\sigma \models p$ if and only if $p \in \sigma_0$;
- $\sigma \models \phi_1 \wedge \phi_2$ if and only if $\sigma \models \phi_1$ and $\sigma \models \phi_2$;
- $\sigma \models \neg\phi$ if and only if $\sigma \not\models \phi$;
- $\sigma \models \bigcirc\phi$ if and only if $\sigma_1\sigma_2\sigma_3 \ldots \models \phi$;
- $\sigma \models \phi_1 \mathcal{U} \phi_2$ if and only if $\exists j \geq 0$ such that $\sigma_j\sigma_{j+1}\sigma_{j+2} \ldots \models \phi_2$ and $\sigma_i\sigma_{i+1}\sigma_{i+2} \ldots \models \phi_1$ for all $0 \leq i < j$.

In LTL synthesis, we want to find an accepting word σ that satisfies the LTL formula ϕ ($\sigma \models \phi$). Any accepting word σ can be written as a combination of finite prefix of symbols $\sigma_{\text{pre}} \in 2^{\mathcal{AP}}$ followed by the infinite repetition of a finite suffix of symbols $\sigma_{\text{suf}} \in 2^{\mathcal{AP}}$. This is called the prefix-suffix structure, where an accepting word can be represented as $\sigma_{\text{pre}}(\sigma_{\text{suf}})^\omega$. In later sections, we will discuss how to find this accepting word using the system S and the LTL formula ϕ.

Büchi Automaton

When synthesizing a sequence of states that satisfy the LTL formula, we often convert the LTL formula into a Büchi automaton.

Definition 3-2.3 (Büchi automaton [16]). *A (nondeterministic) Büchi automaton is a tuple* $B = (S, S_0, \Sigma, \delta, F)$ *where*

- S *is a finite set of states*
- $S_0 \subseteq S$ *is a finite set of initial states*
- Σ *is the input alphabet*
- $\delta : S \times \Sigma \to 2^S$ *is a nondeterministic transition function*
- $F \subseteq S$ *is the set of accepting(final states)*

3-3 Contract Theory

We begin this section by defining a component and an environment. A component "M" can be considered to be an open-system that accepts inputs from other components and/or the external world and generates outputs. A component is characterized by its variables and behaviour. An environment "E" is defined as the external world along with other components that interact with the component of interest. We define the set of all components by the symbol "$\mathcal{M}$" and the set of all environments by the symbol "$\mathcal{E}$" [4].

A contract "$\mathcal{C}$" is a way of specifying these components by exposing only the necessary information about them. A contract is defined as a pair of a subset of contract implementations and a subset of the environments in which it operates, denoted by $\mathcal{C} = (\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$.

Definition 3-3.1 (Contract [4]). *A contract is a pair* $\mathcal{C} = (\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$ *where:*

- $\mathcal{M}_\mathcal{C} \subseteq \mathcal{M}$ *is a set of implementations of* $\mathcal{C}$
- $\mathcal{E}_\mathcal{C} \subseteq \mathcal{E}$ *is a set of environments for* $\mathcal{C}$

A contract is said to be *consistent* if it possesses implementations, i.e., $\mathcal{M}_\mathcal{C} \neq \varnothing$. A contract is said to be *compatible* if there exists environments in which it can operate, i.e., $\mathcal{E}_\mathcal{C} \neq \varnothing$. We denote a component that implements a contract by $M \models^M \mathcal{C}$ (or equivalently $M \in \mathcal{M}_\mathcal{C}$), and an environment for the contract by $E \models^E \mathcal{C}$ (or equivalently $E \in \mathcal{E}_\mathcal{C}$).

In this work, we will use the notion of assume-guarantee reasoning to define the behaviour of a component in a contract, also called assume/guarantee contract (AGC) theory. The component makes assumptions about the environment, and the guarantees are the contract implementations when the said assumptions hold true. To incorporate assume/guarantee reasoning to the definition of a contract, we redefine 3-3.1 as follows:

Definition 3-3.2 (Assume-guarantee contract [4]). *A contract is a pair $\mathcal{C} = (A, G)$ of assertions, called the assumptions and guarantees, such that*

1. *The set $\mathcal{E}_\mathcal{C}$ of the legal environments for $\mathcal{C}$ collects all components E such that $E \subseteq A$*
2. *The set $\mathcal{M}_\mathcal{C}$ of all components implementing $\mathcal{C}$ is defined by $A \times M \subseteq G$*

where $\times$ is the composition operator for components.

Contract composition combines multiple contracts to describe how different components interact with each other in a system. The overall contract resulting from composition describes the behaviour of a larger subsystem or the overall system.

Definition 3-3.3 (Contract composition [53]). *Given contracts $\mathcal{C}_1$ and $\mathcal{C}'$, their composition, written as $\mathcal{C} \otimes \mathcal{C}'$, is the smallest contract in the refinement order such that:*

1. *Composing implementations of each contract results in an implementation that satisfies the composed contract ($M \times M' \models^M \mathcal{C} \otimes \mathcal{C}'$, $\forall M \models^M \mathcal{C}, M' \models^M \mathcal{C}'$)*
2. *Composing an implementation of $\mathcal{C}$ with the environment of the composed contract results in an environment for $\mathcal{C}'$ and vice-versa ($E \times M' \models^E \mathcal{C}$ and $E \times M \models^E \mathcal{C}'$, $\forall M \models^M \mathcal{C}, M' \models^M \mathcal{C}', E \models^E \mathcal{C} \otimes \mathcal{C}'$)*

Equivalently, these can be expressed as

$$\begin{aligned} &M \times M' \models^M \mathcal{C} \otimes \mathcal{C}' \\ &E \times M' \models^E \mathcal{C} \\ &E \times M \models^E \mathcal{C}' \\ &\forall M \models^M \mathcal{C}, M' \models^M \mathcal{C}', E \models^E \mathcal{C} \otimes \mathcal{C}' \end{aligned} \tag{3-4}$$

We say a contract $\mathcal{C} = (A, G)$ is *saturated* if $G = G \cup \neg A$. A contract $\mathcal{C} = (A, G)$ is equivalent to its saturated form $\mathcal{C} = (A, G \cup \neg A)$. A saturated contract $\mathcal{C} = (A, G)$ is *consistent* if and only if $G \neq \varnothing$ and *compatible* if and only if $A \neq \varnothing$. If two contracts $\mathcal{C}_1$ and $\mathcal{C}_2$ are saturated, the composition $\mathcal{C}_1 \otimes \mathcal{C}_2$ is defined as being the pair (A, G) such that $G = G_1 \cap G_2$ and $A = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2)$ [4].

Chapter 4

Vertical System Relationships

4-1 Signals and transducer maps

To define heterogeneous relationships between layers of control operating in different signal spaces, we first need to establish a form of mapping, which we call *transducer* maps, between the different signal spaces of these layers. We start with the mapping between the continuous and discrete-time signals.

Definition 4-1.1 (Periodic sampler [1]). *Given a continuous-time signal $x : \mathbb{R}^+ \to \mathbb{R}^n$, we define a discrete-time signal x_d using the map $\mathcal{S}_T : \mathcal{L}_\infty \to \mathcal{D}_\infty$ such that*

$$\mathcal{S}_T(x) = x_d,\ x_d(k) = x(kT)\ \forall k \in \mathbb{N}, \tag{4-1}$$

where T is the sampling period. We call the map $\mathcal{S}_T$ as the periodic sampler.

Definition 4-1.2 (Linear periodic interpolator [1]). *Given a discrete-time signal $x_d : \mathbb{N} \to \mathbb{R}^n$, we define a continuous-time signal $\hat{x}(t)$ using the map $\mathcal{T}_T : \mathcal{D}_\infty \to \mathcal{L}_\infty$ such that*

$$\begin{gathered} \hat{x}(t) = \mathcal{T}_T(x_d), \\ \hat{x}(t) = x_d(k) + \tfrac{t-kT}{T}(x_d(k+1) - x_d(k)), \\ \forall t \in [kT, (k+1)T),\ k \in \mathbb{N} \end{gathered} \tag{4-2}$$

where T is the sampling period. We call the map $\mathcal{T}_T$ as the linear periodic interpolator.

Definition 4-1.3 ($(\epsilon, \delta)-T$ inverse sampler [1]). *Given a discrete-time signal $x_d : \mathbb{N} \to \mathbb{R}^n$, we define a map $\mathcal{S}^{-1}_{(\epsilon,\delta)-T} : \mathcal{D}_\infty \to 2^{\mathcal{L}_\infty}$ such that*

$$\mathcal{S}^{-1}_{(\epsilon,\delta)-T}(x_d) := \{x \mid \|x - \mathcal{T}_T(x_d)\|_\infty \leq \delta\} \cap \{x \mid \|x(kT) - x_d(k)\| \leq \epsilon\} \tag{4-3}$$

$\mathcal{S}^{-1}_{(\epsilon,\delta)-T}$ represents the set of continuous-time signals bounded by (ϵ, δ) from the linear periodic interpolator.

The bound ϵ constrains how far the continuous-time signal x can deviate from the discrete-samples of signal x_d and the bound δ constrains how far the continuous-time signal x can deviate from the interpolated signal $\mathcal{T}_T(x_d)$. If $\epsilon = \delta$, we can say $\mathcal{S}^{-1}_{(\epsilon,\delta)-T} := \mathcal{S}^{-1}_{\delta-T}$. Establishing these bounds help account for noisy measurements and imperfections during signal sampling.

Definition 4-1.4 (Quantizer [1]). *Given a discrete-time $\mathbb{T} = \mathbb{N}$, a signal $\mathbb{T} \to \mathcal{X}$, a partition $P \subset 2^{\mathcal{X}}$ of $\mathcal{X} = \cup_{[p_i]\in P[p_i]}$, we denote by p_i the representative of each set $[p_i]$, $p(x) = p$ s.t. $x \in [p]$, and $\Sigma = \{p_i\}$, the alphabet of symbols of the quantizer. We define the map $\mathcal{Q}_P : \mathcal{D}_\infty \to \Sigma^\omega$ such that*

$$\mathcal{Q}_P(x)(t) = p(x(t)),\ \forall t \in \mathbb{N} \tag{4-4}$$

We call the map Q_P as the quantizer with partition P.

Definition 4-1.5 (P-inverse quantizer [1]). *We define the map $\mathcal{Q}_P^{-1} : \Sigma^\omega \to 2^{\mathcal{D}_\infty}$ such that*

$$\mathcal{Q}_P^{-1}(x_q) := \{x_d \,|\, x_d(t) \in [x_q(t)]\},\ \forall t \in \mathbb{N} \tag{4-5}$$

We call the map $\mathcal{Q}_P^{-1}$ as the P-inverse quantizer.

We can further transform quantized signals into a sequence discrete events as follows.

Definition 4-1.6 (Eventifier [1]). *Given a sequence x_q^e defined iteratively starting with $k = 0$, $r = 1$, $x_q^e(0) = x_q(0)$ we define a map $\mathcal{E} : \Sigma^\omega \to \Sigma^\omega$, $\mathcal{E}(x_q) := x_q^e$ such that:*

$$x_q^e(r) = x_q(k+1),\ r \leftarrow r+1, \quad \begin{array}{l} k \leftarrow k+1,\ \textit{if}\ x_q(k+1) \neq x_q(k) \\ k \leftarrow k+1,\ \textit{if}\ x_q(k+1) = x_q(k) \end{array} \tag{4-6}$$

We call the map $\mathcal{E}$ as the eventifier.

Definition 4-1.7 (l-inverse eventifier [1]). *We define the map $\mathcal{E}_l^{-1} : \Sigma^\omega \to 2^{\Sigma^\omega}$ such that*

$$\mathcal{E}_l^{-1}(x_q^e) := \{x_q \in \Sigma_l^\omega \,|\, \mathcal{E}(x_q) = x_q^e\} \tag{4-7}$$

Σ_l^ω is the set of signals satisfying $\forall k \in \mathbb{N}$, $\exists r(k) \leq l$ *s.t.* $x_q(k + r(k)) \neq x_q(k)$. *We call the map $\mathcal{E}_l^{-1}$ as the l-inverse eventifier.*

In simpler terms, an l-inverse eventifier is the set of all signals that have an output symbol change separated by at most l time-steps, which when eventified result in the sequence x_q^e. Figure 4-1, taken from [1], provides a visual representation of the relationships between the transducer maps we have defined.

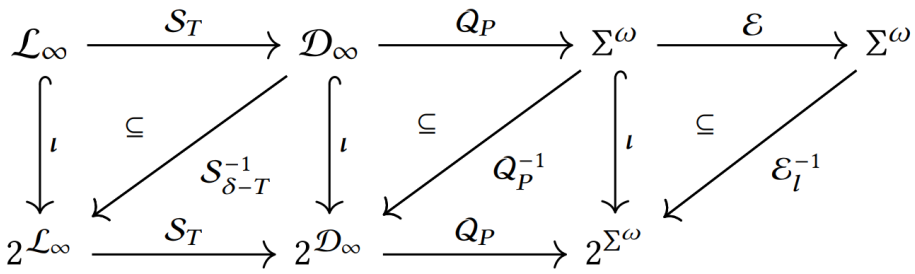

Figure 4-1: Relationship between the transducer maps [1]

4-2 System relationships

In Section 3-1-3, we introduced the concept of simulation relations as a way to establish formal relationships between systems. While traditional simulation relations are effective when comparing systems that share the same signal space, our framework deals with systems that operate across different signal spaces. Therefore, we present a modified definition of simulation relations that incorporates these maps between heterogeneous systems.

Definition 4-2.1 ($\mathcal{F}^{-1}$-simulation relation [1])**.** *Consider two systems S_a and S_b, $\mathcal{F} = \mathcal{F}_u \times \mathcal{F}_y$, $\mathcal{F}_u : U_a \to U_b$, $\mathcal{F}_y : Y_a \to Y_b$ be a transducer map between two spaces with corresponding proper inverse $\mathcal{F}^{-1} = \mathcal{F}_u^{-1} \times \mathcal{F}_y^{-1}, \mathcal{F}_u^{-1} : U_b \to 2^{U_a}$ and $\mathcal{F}_y^{-1} : Y_b \to 2^{Y_a}$. A relation $R_{\mathcal{F}^{-1}} \subseteq X_a \times X_b$ is an $\mathcal{F}^{-1}$-simulation relation from S_a to S_b if the following conditions are satisfied:*

(i) *for every $x_{a,0} \in X_{a,0}$, there exists $x_{b,0} \in X_{b,0}$ with $(x_{a,0}, x_{b,0}) \in R_{\mathcal{F}^{-1}}$;*

(ii) *for every $(x_a, x_b) \in R_{\mathcal{F}^{-1}}$, and for every $u_a \in U(x_a)$, there exists $u_b = \mathcal{F}_u(u_a) \in U(x_b)$ such that for every $x_a \xrightarrow[a]{u_a} x'_a$ in S_a, there exists $x_b \xrightarrow[b]{u_b} x'_b$ in S_b satisfying $(x'_a, x'_b) \in R_{\mathcal{F}^{-1}}$ and $H_a(x_a \xrightarrow[a]{u_a} x'_a) \sim_{\mathcal{F}_y^{-1}} H_b(x_b \xrightarrow[b]{u_b} x'_b)$.*

If there exists an $\mathcal{F}^{-1}$-simulation relation from S_a to S_b, we say that S_b $\mathcal{F}^{-1}$ simulates S_a, denoted by $S_a \preceq_{\mathcal{S},\mathcal{F}^{-1}} S_b$.

The difference between a classical simulation relation (3-1.3) and an $\mathcal{F}^{-1}$ simulation relation is the embedding of a transducer $\mathcal{F}$ in the relation. As we discussed in the previous section, an example of this transducer map $\mathcal{F}$ is a periodic sampler $\mathcal{S}_T$. In that case, system S_a will be modelled in continuous-time and system S_b in discrete-time. When $R_{\mathcal{F}^{-1}}$ is a simulation relation from S_a to S_b and its inverse $R^{-1}_{\mathcal{F}^{-1}}$ is a simulation relation from S_b to S_a, we obtain an $\mathcal{F}^{-1}$-bisimulation relation between S_a and S_b.

We can also define alternating $\mathcal{F}^{-1}$ simulation relation.

Definition 4-2.2 (Alternating $\mathcal{F}^{-1}$-simulation relation [1]). *Consider two systems S_a and S_b, $\mathcal{F} = \mathcal{F}_u \times \mathcal{F}_y$, $\mathcal{F}_u : U_a \to U_b$, $\mathcal{F}_y : Y_a \to Y_b$ be a transducer map between two spaces, with corresponding proper inverse $\mathcal{F}^{-1} = \mathcal{F}_u^{-1} \times \mathcal{F}_y^{-1}, \mathcal{F}_u^{-1} : U_b \to 2^{U_a}$ and $\mathcal{F}_y^{-1} : Y_b \to 2^{Y_a}$. A relation $R_{\mathcal{F}^{-1}} \subset X_a \times X_b$ is an alternating $\mathcal{F}^{-1}$-simulation relation from S_a to S_b if the following conditions are satisfied:*

(i) *for every $x_{a,0} \in X_{a,0}$, there exists $x_{b,0} \in X_{b,0}$ with $(x_{a,0}, x_{b,0}) \in R_{\mathcal{F}^{-1}}$;*

(ii) *for every $(x_a, x_b) \in R_{\mathcal{F}^{-1}}$ and for every $u_a \in U(x_a)$, there exists some $u_b \in U(x_b)$ such that $u_b = \mathcal{F}_u(u_a)$, and for every $x'_b \in Post_{u_b}(x_b)$, there exists $x'_a \in Post_{u_a}(x_a)$ satisfying $(x'_a, x'_b) \in R_{\mathcal{F}^{-1}}$ and $H_a(x_a \xrightarrow[a]{u_a} x'_a) \sim_{\mathcal{F}_y^{-1}} H_b(x_b \xrightarrow[b]{u_b} x'_b)$.*

The above described simulation relations help compare the behaviour of two systems operating in different signal spaces. We also need to define a framework that allow us to use the inputs and outputs of systems defined in one signal space with systems defined in another signal space.

Definition 4-2.3 ($\mathcal{F}$-transduced system [1]). *Given a system $S = (X, X_0, U, \longrightarrow, Y, H)$ and the transducers $\mathcal{F} = \mathcal{F}_u \times \mathcal{F}_y$, $\mathcal{F}^{-1} = \mathcal{F}_u^{-1} \times \mathcal{F}_y^{-1}$ with $\mathcal{F}_u : U \to U_f$, $\mathcal{F}_y : Y \to Y_f$, $\mathcal{F}_u^{-1} : U_f \to 2^U$, and $\mathcal{F}_y^{-1} : Y_f \to 2^Y$, we call the $\mathcal{F}$-transduced system of S:*

$$\mathcal{F}(S, \mathcal{F}^{-1}) = \left(X, X_0, U_f, \underset{f}{\longrightarrow}, Y_f, \mathcal{F}_y \circ H\right)$$

where

$$(x, u_f, x') \in \underset{f}{\longrightarrow} \iff \exists u \in \mathcal{F}_u^{-1}(u_f) \text{ s.t. } (x, u, x') \in \longrightarrow .$$

Definition 4-2.4 ($\mathcal{F}^{-1}$-transduced system [1]). *Given a system $S = (X, X_0, U, \longrightarrow, Y, H)$ and the transducers $\mathcal{F} = \mathcal{F}_u \times \mathcal{F}_y$, $\mathcal{F}^{-1} = \mathcal{F}_u^{-1} \times \mathcal{F}_y^{-1}$ with $\mathcal{F}_u : U \to U_f$, $\mathcal{F}_y : Y \to Y_f$, $\mathcal{F}_u^{-1} : U_f \to 2^U$, and $\mathcal{F}_y^{-1} : Y_f \to 2^Y$, we call the $\mathcal{F}^{-1}$-transduced system of S:*

$$\mathcal{F}^{-1}(S) = \left(X, X_0, 2^{U_f}, \underset{f}{\longrightarrow}, 2^{Y_f}, \mathcal{F}_y^{-1} \circ H\right)$$

where

$$(x, u_f, x') \in \underset{f}{\longrightarrow} \iff \forall u' \in \mathcal{F}_u(u_f), (x, u', x') \in \longrightarrow .$$

To achieve the desired system behavior, we can utilize system composition. Through system composition, it is possible to combine two systems to create an open system that exhibits our target properties. We introduce a composition method that incorporates both feedback and feedforward elements:

Definition 4-2.5 (Open sequential feedback composition [1]). *Given two systems* $S_a = (X_a, X_{a,0}, U_a, \underset{a}{\longrightarrow}, Y_a, H_a)$ *and* $S_b = (X_b, X_{b,0}, U_b, \underset{b}{\longrightarrow}, Y_b, H_b)$*, and a pair* $F = (F_i, F_o)$ *of interconnection maps* $F_i : Y_a \times Y_b \times U_c \to U_a \times U_b$*, with* $F_i = F_{i1} \times F_{i2}$*,* $F_{i1} : Y_a \times Y_b \to U_a$*,* $F_{i2} : Y_a \times U_c \to U_b$*, and* $F_o : Y_a \times Y_b \to Y_c$*, the open sequential feedback composition of* S_a *with* S_b*, denoted* $S_a \|_F S_b$*, is given by the system* $S_c = (X_c, X_{c,0}, U_c, \underset{c}{\longrightarrow}, Y_c, H_c)$ *with:*

- $X_c : X_a \times X_b \times Y_a$
- $X_{c,0} : X_{a,0} \times X_{b,0} \times Y_{a,0}$
- $\underset{c}{\longrightarrow} = \{((x_a, x_b, y_a), u_c, (x'_a, x'_b, y'_a)) \in X_c \times U_c \times X_c \mid (x_a, u_a, x'_a) \in \underset{a}{\longrightarrow}$
  $\wedge\, (x_b, u_b, x'_b) \in \underset{b}{\longrightarrow} \wedge\, y'_a = H_a((x_a, u_a, x'_a))$
  with $u_a = F_{i1}(y_a, H_b(x_b, u_b, x'_b)), u_b = F_{i2}(y_a, u_c)\}$
- $H_c = F_o$

This system composition can be interpreted from a control theory perspective, where S_a represents the plant and S_b serves as the controller, resulting in the controlled system S_c. Figure 4-2 illustrates this feedback composition through a block diagram.

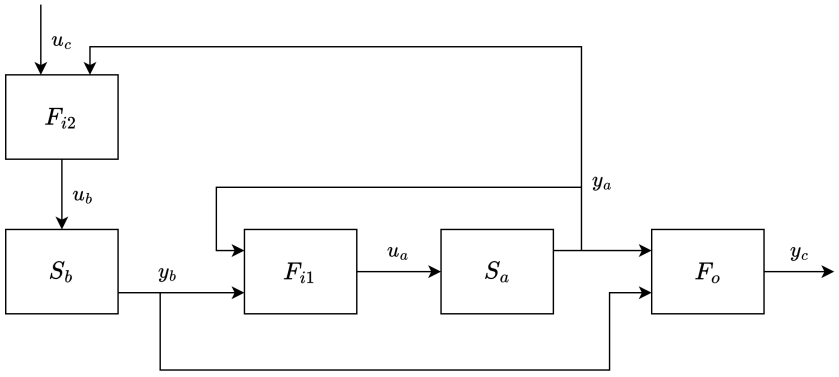

Figure 4-2: Block diagram of open sequential feedback composition

This feedback composition, when combined with our previously defined $\mathcal{F}^{-1}$ simulation relations, provides a powerful framework for translating controller designs across different signal spaces.

Proposition 1 (Controller transference [1]). Let the system $S_b = (X_b, X_{b,0}, \mathcal{F}_u(U_a), \underset{b}{\longrightarrow}, \mathcal{F}_y(Y_a), H_b)$ be an abstraction of system $S_a = (X_a, X_{a,0}, U_a, \underset{a}{\longrightarrow}, Y_a, H_a)$ in the sense that:

$$S_a \preceq_{\mathcal{S},\mathcal{F}^{-1}} S_b \preceq_{\mathcal{AS},\mathcal{F}^{-1}} S_a,$$

with $\mathcal{F} = \mathcal{F}_u \times \mathcal{F}_y$, $\mathcal{F}^{-1} = \mathcal{F}_u^{-1} \times \mathcal{F}_y^{-1}$ proper, $\mathcal{F}_y : Y_a \to Y_b$, $\mathcal{F}_u : U_a \to U_b$, $\mathcal{F}_y^{-1} : Y_b \to 2^{Y_a}$, $\mathcal{F}_u^{-1} : U_b \to 2^{U_a}$. Consider a third system $S_c = (X_c, X_{c,0}, U_c, \underset{c}{\longrightarrow}, Y_c, H_c)$ composable with S_b via an interconnection mapping F. The following holds:

$$\mathcal{F}(S_a) \parallel_F S_c \preceq_S \mathcal{F} \circ \mathcal{F}^{-1}(S_b \parallel_F S_c) \preceq_{\mathcal{AS}} \mathcal{F}(S_a) \parallel_F S_c.$$

For the proof of proposition 1, we ask the reader to refer [1].

Chapter 5

Structured Hierarchical Control System Design

In this work, we implement a novel approach to hierarchical control systems that composes control systems vertically, in contrast to the more common horizontal compositions prevalent in the literature. The framework we implement, developed by Mazo et al. [1], addresses the limitations of existing hierarchical control structures, which typically fall into three categories:

- Hierarchical structures where each layer operates in the same modeling domain and has system guarantees,
- Hierarchical structures where each layer operates in different modeling domains but lacks comprehensive system guarantees, or
- Hierarchical structures where controllers at different layers are tightly coupled, making independent development and modification difficult.

The framework proposed by Mazo et al. [1] allows us to:

- Model each layer using different signal spaces, and operate each layer at different frequencies.
- Obtain guarantees that the behavior of the concrete system at the bottom layer is accurately simulated by the abstracted system at the top layer.
- Enable modular development where controllers can be independently designed and readily substituted.

In the next section, we define the hierarchical control framework.

5-1 Contract template for the hierarchical system

We use contracts to formulate the hierarchical system framework. A contract template C_i for each layer i is defined as follows [1]:

$$\begin{array}{rcll} A_i := & M_{i-1}\wedge & \text{(Model abstraction from layer below)} \\ & P_{i+1} & \text{(Signal property from the layer above)} \\ G_i := & M_i\wedge & \text{(Model abstraction of this layer)} \\ & P_i & \text{(Signal property of this layer)} \end{array} \tag{5-1}$$

Utilizing the system relationships and feedback composition defined in Section 4-2, the model abstraction at each layer M_i is defined as:

$$M_i := S_i \parallel_{F_i} \Pi_i \preceq_{S,\mathcal{F}_i^{-1}} S_{i+1} \preceq_{\mathcal{AS},\mathcal{F}_i^{-1}} S_i \parallel_{F_i} \Pi_i, \tag{5-2}$$

where $\mathcal{F}_i$ is the transducer map that maps the signal space of S_i to the signal space of S_{i+1} and F_i is the interconnection mapping for the controller at the i-th layer. The model abstraction guarantee establishes that the behaviour of the controlled system obtained by composing system S_i with the controller Π_i is $\mathcal{F}_i^{-1}$ simulated by the abstracted system S_{i+1}. The signal properties of each layer P_i is also referred to as the output property of the composed system $S_i \parallel_{F_i} \Pi_i$. The signal property P_i enforces a specific output structure for each layer's controlled system, allowing other layers to make reliable assumptions about the form of signals they will receive. This structure simplifies and promotes independent controller design at each layer.

Each control layer is responsible for fulfilling its contract guarantees based on assumptions about the output signals and model abstractions it expects to receive from adjacent layers. To obtain guarantees for the desired overall system behaviour, it is sufficient to check the satisfaction of an overall contract obtained by composing all the individual contracts at each layer. The overall contract $\mathcal{C}_C = \bigotimes \mathcal{C}_i = (A_C, G_C)$, where

$$\begin{array}{rcll} A_C := & S_1 & \text{(Ground truth model)} \\ G_C := & \bigwedge_{i=1}^{N} M_i\wedge & \text{(Intermediate models)} \\ & \bigwedge_{i=1}^{N} P_i & \text{(All signal properties)} \end{array} \tag{5-3}$$

A key result of this framework is that if S_1 (the lowest-level system) is true, the conjunction of models $\bigwedge_{i=1}^{N} M_i$ establishes a formal chain of abstractions. This ensures that the composed controllers recursively satisfy:

$$\begin{gathered} \left(\mathcal{F}_i(\tilde{S}_i) \parallel_{F_{i+1}} \Pi_{i+1}\right) \models \mathcal{F}_i \circ \mathcal{F}_i^{-1}(P_{i+1}),\ \forall i = 1, 2, \ldots, N \\ \tilde{S}_{i+1} = \left(\mathcal{F}_i(\tilde{S}_i) \parallel_{F_{i+1}} \Pi_{i+1}\right), \quad \tilde{S}_1 = S_1 \parallel_{F_1} \Pi_1. \end{gathered} \tag{5-4}$$

While the contract template discussed thus far applies to a single control plant, it can be extended to multi-agent systems where several control plants need to satisfy a combined

global specification. If we can decompose this global high-level specification into individual plans for each plant, then the satisfaction of individual contracts by each plant ensures the fulfillment of the global specification.

In this thesis, we implement this framework to create a robust foundation for a hierarchical control system with guarantees across different modeling domains and time scales. The following sections will detail the specific implementation of each layer in our hierarchical control structure, demonstrating how this theoretical framework translates into practical control algorithms for complex robotic systems.

5-2 System description

Our aim is to design a hierarchical control system suitable for robotic applications. To achieve this, we develop a control architecture that comprises of three distinct layers, each operating at varying levels of system complexity and frequency:

1. The highest level focuses on abstract motion planning in a quantized state space.
2. The middle layer generates concrete trajectories using information from the top layer.
3. The bottom layer tracks the generated trajectory in continuous-time.

This hierarchical approach allows us to manage the complexity of control tasks by decomposing the problem into more manageable sub-problems at each level. Figure 5-1 provides a visual representation of our hierarchical control system architecture.

The subsequent system and controller descriptions are inspired by the work of Mazo et al. [1]. In the following sections, we will elaborate on each layer of this hierarchical structure, detailing the systems and controllers that comprise each level.

5-2-1 High-level system

The top layer defines the system in terms of high-level objectives governing the overall behavior. At this level, the positional space is divided into a finite set of cells or partitions, representing system states. A motion planner synthesizes a sequence of system states satisfying the given LTL specifications. The system complexity is kept relatively abstract, delegating more complex motion plan implementation details to lower layers.

We describe the high-level system as a FSM. Formally, we denote this FSM system as $S_3 = (X_3, X_{3,0}, U_3, \underset{3}{\longrightarrow}, Y_3, H_3)$, where

- $X_3 = \Sigma$

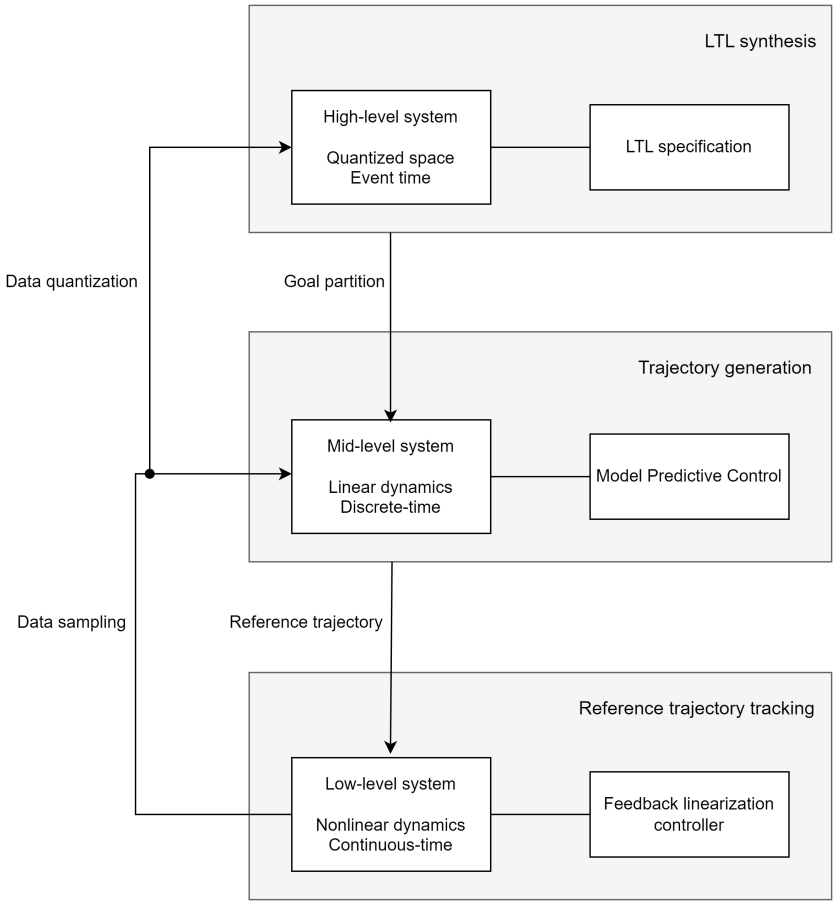

Figure 5-1: Block diagram of the proposed hierarchical control system

- $X_{3,0} = \Sigma$
- $U_3 = \Sigma \times \Sigma$
- $x_3 \xrightarrow[3]{u_3} x_3' : x_3 = u_3(1) \wedge x_3' = u_3(2) \wedge \overline{[x_3]} \cap \overline{[x_3']} \neq \emptyset$
- $Y_3 = U_3$
- $H_3 : (x_3, u_3, x_3') \mapsto u_3$

Here, Σ represents a finite set of cell labels. We index a cell label as c_ij, where $i = 1, 2, \ldots p$ and $j = 1, 2, \ldots, q$, with p and q denoting the number of cells in a row and column respectively. The input space U_3 combines two cell labels, indicating the current cell and the successor cell for movement. The transition condition $\xrightarrow[3]{u_3}$ ensures that the current and successor cells are adjacent. This system operates in event time, progressing to the next step only after completing the current transition, thus allowing for flexible planning. Figure 5-2 illustrates the partitioning of the position space (environment).

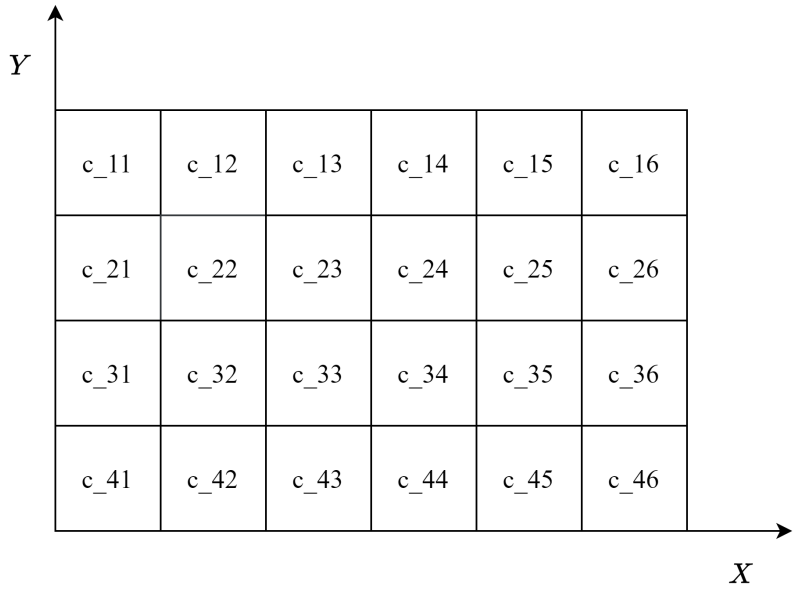

Figure 5-2: Partitioned environment used by the high-level system

5-2-2 Mid-level system

The middle layer of our hierarchical control structure serves as an intermediary between abstract high-level plans and concrete low-level execution. Its primary function is to translate the high-level motion plan into feasible trajectories. This layer processes the output from the top layer, generating a trajectory that facilitates the system's transition from the current cell to the successor cell in a finite number of steps. The resulting planned trajectory is then passed to the bottom layer for precise tracking of the interpolated trajectory.

A key advantage of this approach is that the mid-level system can operate without considering the complex nonlinear dynamics of the bottom layer. Instead, it is defined by the discretization of a single integrator model. This simplification is possible because the bottom layer is tasked with precise tracking of the mid-level trajectory, allowing the mid-level system to behave as a discretized single integrator system.

The dynamics of the mid-level system can be described as follows:

$$\begin{aligned} x[k+1] &= \begin{pmatrix} I & TI \\ 0 & I \end{pmatrix} x[k] + \begin{pmatrix} 0 \\ I \end{pmatrix} u[k] \\ x[k] &= \begin{pmatrix} x_m[k] \\ y_m[k] \\ v_{x_m}[k] \\ v_{y_m}[k] \end{pmatrix} \in \mathbb{R}^4,\ u[k] = \begin{pmatrix} \Delta v_{x_m}[k] \\ \Delta v_{y_m}[k] \end{pmatrix} \in \mathbb{R}^2 \end{aligned} \tag{5-5}$$

In this model, the states of the mid-level dynamics comprise the position and linear velocities along the X and Y coordinates. The inputs to the mid-level dynamics are the changes

in velocities between successive time steps along the X and Y directions.

Formally, we define the mid-level system $S_2 = (X_2, X_{2,0}, U_2, \xrightarrow[2]{}, Y_2, H_2)$ as:

- $X_2 = \begin{pmatrix} x_m[k] \\ y_m[k] \\ v_{x_m}[k] \\ v_{y_m}[k] \end{pmatrix} \in \mathcal{X}_2 \subseteq \mathbb{R}^4$
- $X_{2,0} = X_2$
- $U_2 = \left\{ [u[k]]_{k=0}^{N-1} : u[k] = \begin{pmatrix} \Delta v_{x_m}[k] \\ \Delta v_{y_m}[k] \end{pmatrix} \in \mathcal{U}_2 \subseteq \mathbb{R}^2 \right\}$
- $(x, u, x') \in \xrightarrow[2]{} \iff \exists$ a sequence $[x[k]]_{k=0}^{N}$ with $x[k] \in X_2$ such that $x = x[0], x' = x[N]$ and $\forall k \in \{0, 1, \ldots, N-1\}, x[k]$ obeys the dynamics in equation 5-5
- $Y_2 = \left\{ [x[k]]_{k=0}^{N} : x[k] \in X_2 \right\}$
- $H_2 : (x, u, x') \mapsto [x[k]]_{k=0}^{N}$

The input to the system S_2 is a sequence of inputs $u[k]_{k=0}^{N-1}$, while its output is the sequence of states $x[k]_{k=0}^{N}$ obtained by applying the input U_2. This formulation allows us to use the output sequence from S_2 for the trajectory interpolation required by the bottom layer. It's important to note that both $x[k]$ and $u[k]$ are subject to constraints dictated by environmental factors and physical limitations of the system.

5-2-3 Low-level system

The bottom layer of our hierarchical control structure plays a critical role in ensuring proper tracking of the mid-level trajectory, thereby enabling seamless operation of the middle and top layers. This layer receives a sequence of discrete-time points from the middle layer, which it must track accurately. To achieve smooth and continuous motion, we employ a nonlinear interpolation function to generate a curve that passes through these discrete points. A feedback linearization controller then ensures that the system accurately tracks this continuous trajectory.

For describing the low-level dynamics of the system, we utilize the unicycle model, which is defined as follows:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} vcos(\theta) \\ vsin(\theta) \\ \omega \\ a \\ \alpha \end{pmatrix} \tag{5-6}$$

In this model, the state vector comprises five components: position along the X and Y directions, orientation θ, linear velocity v, and angular velocity ω. The system inputs are linear acceleration a and angular acceleration α.

Formally, we define the low-level system $S_1 = (X_1, X_{1,0}, U_1, \underset{1}{\longrightarrow}, Y_1, H_1)$ as:

- $X_1 = \begin{pmatrix} x \\ y \\ \theta \\ v \\ \omega \end{pmatrix} \in \mathcal{X}_1 \subseteq \mathbb{R}^5$
- $X_{1,0} = X_1$
- $U_1 = u_1 = \begin{pmatrix} a \\ \alpha \end{pmatrix} : [0, \tau) \to \mathcal{U}_1 \subseteq \mathbb{R}^2$, $u_1(\cdot)$ is piecewise continuous
- $(x, u, x') \in \underset{1}{\longrightarrow} \iff \exists$ a continuously differentiable function $\xi_{u,x} : [0, \tau) \to X_1$ such that $\xi_{u,x}(0) = x$, $\xi_{u,x}(\tau) = x'$, and $\dot{\xi}_{u,x}(t) = f(\xi_{u,x}(t), u(t)) \; \forall t \in [0, \tau)$
- $Y_1 = \{\xi : [0, \tau] \to X_1 \mid \xi(\cdot) \text{ is continuously differentiable}\}$
- $H_1 = (x, u, x') \mapsto \xi_{u,x}(0 : \tau)$

5-2-4 Signal properties

The signal property, or the output property defined at each layer is as follows:

- $P_4 = \top$
- $P_3 := S_3 \parallel \Pi_3 \models \phi$
- $P_2 := y_{2c} \in \mathcal{Q}_P^{-1} \circ \mathcal{E}_N^{-1} \circ \mathcal{E} \circ \mathcal{Q}_{\mathcal{P}}(y_{2c})$
- $P_1 := y_{1c} = \mathcal{S}_{\delta,T}^{-1} \circ \mathcal{S}_T(y_{1c})$

Note that we have defined a higher layer than previously described top layer. But this top layer is more of a placeholder to ensure that the system guarantees are properly met. We also define the model $M_4 = \top$, which is used for proving the model abstraction guarantee of the top layer.

5-3 Controller design

This section delves into the design of controllers for each layer, enabling the composed system to fulfill its respective contracts. The presence of these contracts allows for independent controller design at each layer, regardless of the functioning of other layers.

5-3-1 High-level controller

The high-level controller aims to determine a sequence of states in the Finite State Machine (FSM) that satisfies the given LTL specification. We can express this as $S_3 \parallel \Pi_3 \models \phi$, where $S_3 \parallel \Pi_3$ represents the system composed with the control policy that satisfies the LTL specification ϕ.

To identify a sequence of states (or trajectory) of S_3 that satisfies the LTL formula ϕ, we use graph-based search methods. Algorithm 5.1 outlines this approach.

Algorithm 5.1 LTL Control: Finding shortest path (S, ϕ) [22], [40]

1: Define the system S as a directed graph with states as graph vertices $\mathcal{V}$ and state transitions as graph edges $\mathcal{E}$
2: Translate ϕ into a deterministic Büchi/Rabin automaton B
3: Construct the product automaton $P = S \otimes B$ as a directed graph
4: Determine the shortest path from the initial state to an accepting state of P and store it in σ_{pre}
5: Find the shortest path from a successor state of σ_{pre} to an accepting state of P and store it in σ_{suf}
6: Project the accepting run $\sigma_{\text{pre}}(\sigma_{\text{suf}})^{\omega}$ onto a trajectory of S

This method is applicable to both single and multi-robot systems. In the case of multi-robot systems, the system S becomes a product transition system $S_{\mathcal{P}}$ defined in 3-1.2. We can employ the same graph-based search methods to find an accepting run of the product automaton $P = S_{\mathcal{P}} \otimes B$. Once an accepting run is identified, we can project it onto individual trajectories for each transition system S_i, allowing them to perform their tasks independently.

5-3-2 Mid-level controller

The mid-level controlled system is responsible for generating trajectories that facilitate the robot's transition from one cell to the next. This layer operates under the assumptions of signal property P_3 from the top layer and the model abstraction M_1 from the bottom layer. Given this information, the middle layer must guarantee its model abstraction $M_2 = S_2 \parallel_{F_2} \Pi_2$ and its output property P_2. The output from the top layer, a pair containing the current and the successor partition, is transformed into constraints for solving a MPC problem. The solution to this MPC problem yields a trajectory that guides the robot from the current cell to the successor cell.

In our controller construction, we consider system Π_2 to be trivial, meaning it passes the input (the output from the top layer) to its output without modification. Formally, we define system $\Pi_2 = (X_{\Pi_2}, X_{\Pi_2,0}, U_{\Pi_2}, \underset{\Pi_2}{\longrightarrow}, Y_{\Pi_2}, H_{\Pi_2})$ as:

- $X_{\Pi_2} = U_{\Pi_2}$
- $X_{\Pi_2,0} = X_{\Pi_2}$
- $U_{\Pi_2} = Y_{3c}$
- $(x_{\Pi_2}, u_{\Pi_2}, x'_{\Pi_2}) \in \underset{\Pi_2}{\longrightarrow} \iff x'_{\Pi_2} = u_{\Pi_2}$
- $Y_{\Pi_2} = U_{\Pi_2}$
- $H_{\Pi_2} : (x_{\Pi_2}, u_{\Pi_2}, x'_{\Pi_2}) \mapsto u_{\Pi_2}$

Consequently, the controlled system $S_{2c} = S_2 \parallel \Pi_2 = (X_{2c}, X_{2c,0}, U_{2c}, \underset{2c}{\longrightarrow}, Y_{2c}, H_{2c})$ is defined as:

- $X_{2c} : X_2 \times X_{\Pi_2} \times Y_2$
- $X_{2c,0} = X_{2c}$
- $U_{2c} : u_{2c} = \mathcal{Q}_P^{-1} \circ \mathcal{E}_N^{-1}(u_3)$
- $\underset{2c}{\longrightarrow} = \{((x_2, x_{\Pi_2}, y_2), u_{2c}, (x'_2, x'_{\Pi_2}, y'_2)) \in X_{2c} \times U_{2c} \times X_{2c} \mid (x_2, u_2, x'_2) \in \underset{2}{\longrightarrow}$ $\wedge\ (x_{\Pi_2}, u_{\Pi_2}, x'_{\Pi_2}) \in \underset{\Pi_2}{\longrightarrow} \wedge\ y'_2 = H_2((x_2, u_2, x'_2))$ with $u_2 = F_{2i}(y_2, y_{\Pi_2}), u_{\Pi_2} = u_{2c}\}$
- $Y_{2c} = \begin{pmatrix} x_m[k] \\ y_m[k] \end{pmatrix} \in \mathcal{Y}_{2c}$
- $H_{2c} = F_{2o}$
- $F_{2i} = MPC(y_2, y_{\Pi_2})$, $F_{2o} = \begin{pmatrix} I_2 & 0 \end{pmatrix} y_2$

In definition 4-2.5, we define the input map $F_i = F_{i1} \times F_{i2}$, where $F_{i1} : Y_a \times Y_b \to U_a$, $F_{i2} : Y_a \times U_c \to U_b$. For the middle layer, F_{i2} simplifies to a direct mapping from U_c to U_b. This simplification is represented in the controlled system S_{2c} by setting $u_{\Pi_2} = u_{2c}$. Hence, for notational clarity, we omit the explicit F_{i2} mapping and represent the middle layer's input map as $F_{2i} : Y_2 \times Y_{\Pi_2} \to U_2$, where the subscript '2' denotes the middle layer and 'i' indicates the input map. Figure 5-3 illustrates the block diagram of the feedback composition for the middle layer.

To better understand the middle layer's operation, we introduce additional terms. We define $\mathcal{C}_{p_i}$ as the maximal control invariant set contained in $p_i \ominus B_\delta$, where p_i is a partition from the top layer, $\ominus$ denotes the Pontryagin set difference, and B_δ is a ball of radius δ. The ball B_δ is formulated to accommodate for the tracking error of the bottom layer. Note that the partition p_i is just another way of representing the symbols c_ij. For the top-level system to transition from one state (or partition) to another, the control invariant set of

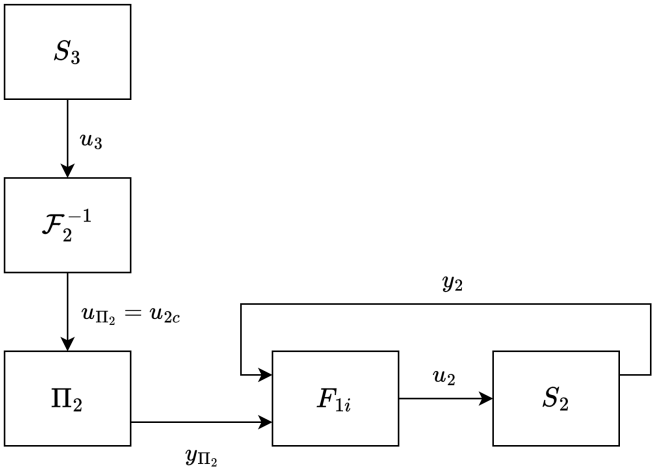

Figure 5-3: Block diagram of feedback composition for the middle layer

the current partition p_i must be N-step backward reachable from the control invariant set of the next partition p_j. Mathematically, this is expressed as:

$$(p_i, (p_i, p_j), p_j) \in \underset{3c}{\longrightarrow} \iff \mathcal{C}_{p_i} \subset \mathcal{R}_N(\mathcal{C}_{p_j})$$

where $\mathcal{R}_N(\cdot)$ computes the N-step backward reachable set. For polytopic p_i, we can compute $\mathcal{C}_{p_i}$ and $\mathcal{R}_N(\mathcal{C}_{p_i})$ using algorithms from Chapter 10 of [50]. This restriction ensures that the high-level system only demands transitions that the mid-level system can satisfy.

Additionally, we require that the initial state of the mid-level system lies within the control invariant set of its partition p_i. For multi-robot systems, we solve multiple MPC problems, each specific to an individual robot. The detailed formulation of these MPC problems is provided in appendix A. For a comprehensive proof of the model abstraction and signal property guarantees of the middle layer, please refer to [1].

5-3-3 Low-level controller

The primary objective of the low-level controller is to accurately track the output from the middle layer, minimizing deviations to allow the mid-level system to function as a discrete-time linear system. This is achieved through a two-step process. First, we employ a nonlinear interpolation function $\mathcal{T}_{\alpha,T}(\cdot)$ to generate a continuous trajectory from the discrete points provided by the middle layer output. Following this, we implement a feedback linearization controller to track this interpolated trajectory with high precision.

In our formulation, we consider the system Π_1 to be identical to system S_2. This design choice allows Π_1 to perform the interpolation of the mid-level trajectory, resulting in a continuous trajectory suitable for controller tracking. To achieve this interpolation of the

middle layer output, we modify H_{Π_1} as follows:

$$H_{\Pi_1} : (x_{\Pi_1}, u_{\Pi_1}, x'_{\Pi_1}) \mapsto \mathcal{T}_{\alpha,T}([x_m[k]]_{k=0}^N)$$

The controlled system $S_{1c} = S_1 \parallel \Pi_1 = (X_{1c}, X_{1c,0}, U_{1c}, \underset{1c}{\longrightarrow}, Y_{1c}, H_{1c})$ is defined as

- $X_{1c} : X_1 \times X_{\Pi_1} \times Y_1$
- $X_{1c,0} = X_{1c}$
- $U_{1c} : u_{1c} = \mathcal{S}_{0,\delta-T}^{-1}(u_2)$
- $\underset{1c}{\longrightarrow} = \{((x_1, x_{\Pi_1}, y_1), u_{1c}, (x'_1, x'_{\Pi_1}, y'_1)) \in X_{1c} \times U_{1c} \times X_{1c} \mid (x_1, u_1, x'_1) \in \underset{1}{\longrightarrow}$ $\wedge\ (x_{\Pi_1}, u_{\Pi_1}, x'_{\Pi_1}) \in \underset{\Pi_1}{\longrightarrow} \wedge\ y'_1 = H_1((x_1, u_1, x'_1))$ with $u_1 = F_{1i}(y_1, y_{\Pi_1}), u_{\Pi_1} = u_{1c}\}$
- $Y_{1c} = \begin{pmatrix} x_l \\ y_l \\ v_{x_l} \\ v_{y_l} \end{pmatrix} \in \mathcal{Y}_{1c}$
- $H_{1c} = F_{1o}$
- $F_{1i} = K_{fbl}(y_1, y_{\Pi_1})$, $F_{1o} = y_{1c}$

In definition 4-2.5, we define the input map $F_i = F_{i1} \times F_{i2}$, where $F_{i1} : Y_a \times Y_b \to U_a$, $F_{i2} : Y_a \times U_c \to U_b$. For the bottom layer, F_{i2} simplifies to a direct mapping from U_c to U_b. This simplification is represented in the controlled system S_{1c} by setting $u_{\Pi_1} = u_{1c}$. Hence, for notational clarity, we omit the explicit F_{i2} mapping and represent the bottom layer's input map as $F_{1i} : Y_1 \times Y_{\Pi_1} \to U_1$, where the subscript '1' denotes the bottom layer and 'i' indicates the input map. The bottom layer's feedback composition follows a structure similar to figure 5-3.

For a detailed formulation of the nonlinear interpolation function $\mathcal{T}_{\alpha,T}$ and the feedback linearization controller K_{fbl}, please refer to appendix B and C respectively. The proof for the guarantees of signal property P_1 and model abstraction M_1 can be found in [1].

5-4 Overall system guarantees

Applying contract composition 5-3 results in a system-wide contract $\mathcal{C}_C = (S_1, \bigwedge_{i=1}^2 M_i \wedge \bigwedge_{i=1}^3 P_i)$ with guaranteed behaviours

$$\begin{aligned} \mathcal{K}\left(\mathcal{S}\left(S_1 \parallel_{F_1} \Pi_1\right) \parallel_{F_2} \Pi_2\right) \parallel_{F_3} \Pi_3 &\models \mathcal{K} \circ \mathcal{K}^{-1}(P_3) \\ \mathcal{S}\left(S_1 \parallel_{F_1} \Pi_1\right) \parallel_{F_2} \Pi_2 &\models \mathcal{S} \circ \mathcal{S}^{-1}(P_2) \\ S_1 \parallel_{F_1} \Pi_1 &\models P_1 \end{aligned} \tag{5-7}$$

where $\mathcal{S} = \mathcal{S}_T$, $\mathcal{S}^{-1} = \mathcal{S}_{\delta,T}^{-1}$, $\mathcal{K} = \mathcal{E} \circ \mathcal{Q}_{\mathcal{P}}$, $\mathcal{K}^{-1} = \mathcal{Q}_P^{-1} \circ \mathcal{E}_N^{-1}$. Breaking down equation 5-7, we can identify all the individual guarantees. The first is the straightforward signal property guarantee P_1. The guarantee $\mathcal{S}\left(S_1 \parallel_{F_1} \Pi_1\right) \parallel_{F_2} \Pi_2 \models \mathcal{S} \circ \mathcal{S}^{-1}(P_2)$ can be understood in two parts. The model abstraction guarantee M_1 ensures that $\mathcal{S}\left(S_1 \parallel_{F_1} \Pi_1\right)$ is simulated by system S_2, while the expression $S_2 \parallel_{F_2} \Pi_2 \models \mathcal{S} \circ \mathcal{S}^{-1}(P_2)$ provides the signal property guarantee for the middle layer.

Chapter 6

Experiment Setup

6-1 Simulation setup

This section details the software implementation for the hierarchical control system. The majority of the system is implemented in Python, chosen for its ease of use and strong library support for solving LTL and MPC problems. We also attempted to implement the controller in MATLAB, but discarded the option as it took more time to solve optimization problems compared to Python. Additionally, Python provides a more natural syntax for object-oriented programming constructs, making it preferable to MATLAB. For few specific tasks, we utilize MATLAB through its Python engine.

The experiments are performed on a machine running Ubuntu 20.04 (dual-boot), equipped with an Intel i7-1165G7, 4-core processor, 8 logical processors, 16 GB of RAM. For the versions of software packages used in the experiments, please refer to table 6-1.

Software	**Version**
Ubuntu	20.04.6 LTS
ROS	Noetic
Python	3.10.13
g++/gcc	Ubuntu 9.4.0-1ubuntu1~20.04.2
cmake	3.20.0
SPOT	2.12.1
MATLAB	2024a

Table 6-1: Software versions used in implementation

6-1-1 Top layer implementation

The top layer of our control hierarchy synthesizes a motion plan from LTL specifications. We represent the top-level finite transition system as a directed graph using the NetworkX library. The Spot library [54] is employed to translate LTL specifications into Büchi automata. We construct the product automaton as a directed graph and determine an accepting run using NetworkX's built-in "shortest_path" algorithm. The output is a sequence of partitions that the robot must traverse to satisfy its corresponding LTL specification.

6-1-2 Middle layer implementation

The middle layer generates trajectories given a pair of states from the top layer. We employ CVXPY as our optimization framework for formulating and solving the MPC problem. System dynamics are described using NumPy arrays. Polytopic constraints for the MPC problem are defined using the polytope package [55] in Python. Position constraints, control invariant sets, and N-step backward reachable sets are formulated using the MPT3 [56] toolbox with MATLAB. We utilize the MATLAB Engine API for Python to execute these MATLAB operations within our Python code. To enhance runtime performance, all these sets are precomputed and stored prior to execution.

6-1-3 Bottom layer implementation

The bottom layer implementation is straightforward. Control inputs are designed as per appendix C, with equations defined symbolically using SymPy. This approach allows for efficient manipulation and evaluation of these equations based on the current state and execution time.

6-2 Hardware setup

Our experimental setup employs Elisa3 robots operating within an environment equipped with a motion capture (MoCap) system. This section details the physical design of the robot and the software architecture based on Robot Operating System (ROS). Our implementation relies on multiple ROS nodes that communicate with each other, facilitating the deployment of our hierarchical control system in a real-world setting.

6-2-1 Physical design of Elisa3 robots

Figure 6-1 shows the physical design of an Elisa3 robot. Some of the important features include [2]

- 8 MHz microprocessor
- A 3-axis accelerometer
- RF radio for communication
- 2 DC motors
- 8 proximity sensors for obstacle detection and avoidance
- 4 ground sensors for line following and cliff detection
- Max speed of 60 cm/s
- Wireless communication throughput of around 250Hz for 4 robots, and 100Hz for 10 robots

Additionally, due to limited memory capabilities of the robot, most of the processing is done on the PC side. The robot is responsible in computing odometry, transmitting the status of its sensors, and receiving actuator commands from the central PC.

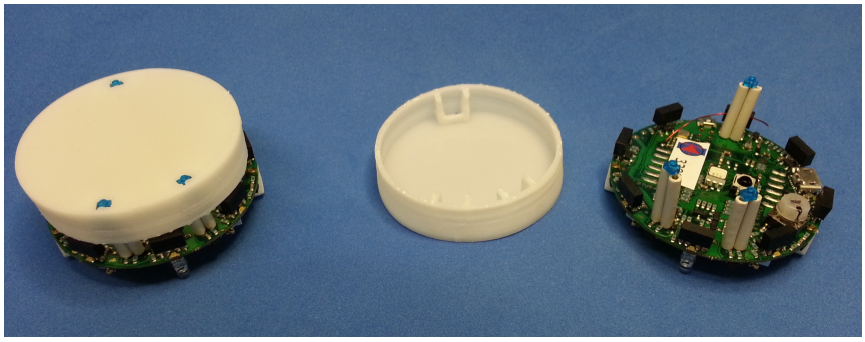

Figure 6-1: Physical design of an Elisa3 robot [2]

6-2-2 MoCap system

Our experiments utilize a camera-based motion capture (MoCap) system consisting of 10 cameras installed in the Delft Center for Systems and Control (DCSC) Laboratory at TU Delft. This high-precision system provides global positioning data for the robots, complementing the relative positioning information from their onboard sensors. The integration of MoCap data significantly enhances the accuracy of robot localization within the experimental environment.

6-2-3 Overall software architecture

Our codebase is built upon the Robot Operating System (ROS) framework, which facilitates communication and data exchange between various components of the system. The implementation uses multiple ROS nodes, each responsible for specific functionalities.

Mocap node

This node interfaces with the motion capture system and publishes information about the robots' pose (position and orientation) obtained from the global camera system.

Elisa3 node

The Elisa3 node manages direct communication with the Elisa3 robots. It handles the transmission of actuation commands to the robots and receives data from their onboard sensors.

Estimator node

The estimator node is responsible for fusing data from multiple sources to provide accurate estimates of the robots' pose. This node implements an Extended Kalman Filter (EKF) to combine robot sensor data and data from the camera system to generate more accurate estimates of the robots' pose. The initial version of this estimator was developed by a former Master student at TU Delft[57, 58]. We have since improved the estimator node's performance, particularly its speed of operation, and this enhanced version is used in our current implementation [59].

Controller node

The controller node implements the hierarchical control system described in the previous section. It receives pose estimates from the estimator node and generates control commands for the Elisa3 robots.

Software integration

The integration of these ROS nodes creates a closed-loop system as seen in figure 6-2. This architecture allows for real-time control and adaptation of our hierarchical control system to physical Elisa3 robots.

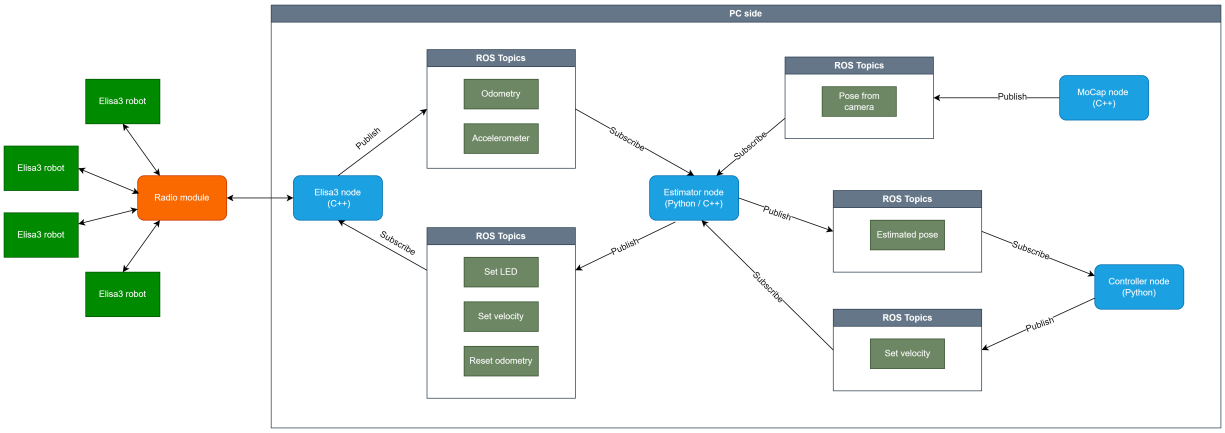

Figure 6-2: Overall software architecture for the hardware implementation

Chapter 7

Simulation Results

In this chapter, we present simulation studies to validate our hierarchical control architecture. We demonstrate the framework's effectiveness by implementing a high-level task specification using LTL formulas and establishing appropriate parameters for both middle and bottom layers. Through these simulations, we show how our architecture generates feasible trajectories at the concrete system level that not only track the mid-level reference trajectories but also satisfy the prescribed high-level mission specifications.

7-1 Simulation results for a single robot

We begin by defining the operational environment for the robot. Based on calculations detailed in appendix B, the ball B_δ has a radius $\delta = \frac{3}{32}\alpha T \Delta v_{max}$, which evaluates to approximately $0.0005m$. As a precautionary measure, we use a more conservative estimate of $\delta = 0.01m(1cm)$. This larger value of B_δ results in a more constrained control invariant set, which restricts the robot's terminal state to a smaller subset of the partition space. This conservative choice helps prevent the robot from approaching cell boundaries when receiving new partition commands from the top layer, thereby reducing the risk of unintended transitions into neighboring partitions

Figures 7-1 and 7-2 illustrate the complete environment and a single partition, respectively. These visualizations highlight the partition boundaries and their associated control invariant sets. We employ the L^∞ norm to represent B_δ as a polytope. Notably, figure 7-2 demonstrates that the control invariant set of a partition closely approximates $c\_41 \ominus B_\delta$.

We define a set of atomic propositions $\mathcal{AP} = \{\text{'b'}, \text{'g'}, \text{'d'}\}$, where

- 'b' denotes base region

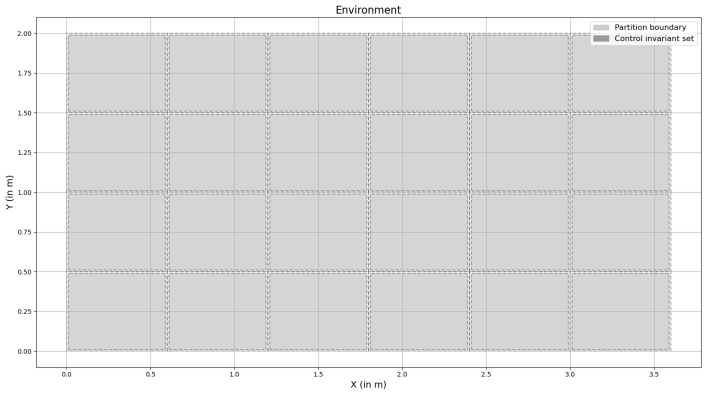

Figure 7-1: The environment where the robots operate

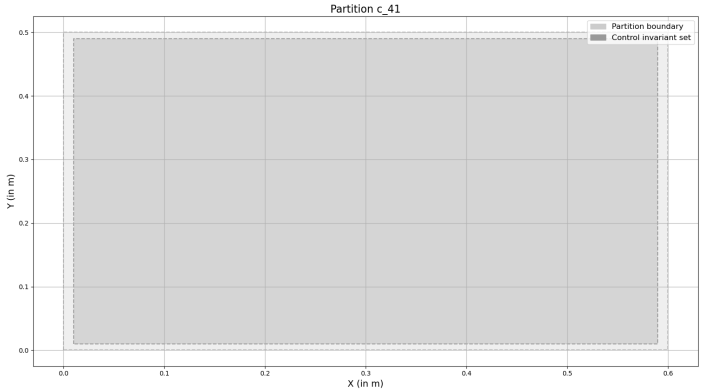

Figure 7-2: Visual representation of a single partition of the environment

- 'g' denotes goal region
- 'd' denotes dangerous region

Each partition is associated with at most one of these propositions or labels. Figure 7-3 depicts the environment with these atomic propositions assigned to specific partitions.

7-1-1 Choosing the LTL formula

Having established the robot's operational environment, we now focus on defining the high-level specification. For this simulation, we've chosen a simple surveillance task where the robot must navigate through the environment, surveying specific regions while avoiding others. The high-level specification is represented by the following LTL formula:

$$\phi = \square\diamond \mathrm{b} \wedge \square(\mathrm{b} \rightarrow \bigcirc\neg \mathrm{b}\,\mathcal{U}\diamond \mathrm{g}) \wedge \square\neg(\mathrm{b} \wedge \mathrm{g}) \wedge \square\neg \mathrm{d}$$

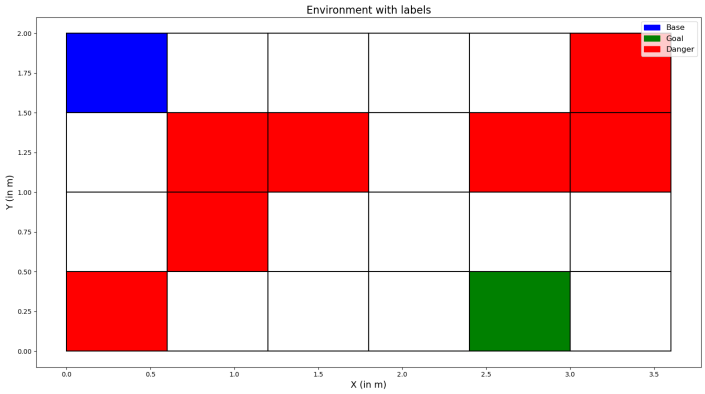

Figure 7-3: Visual representation of the environment marked with labels

Let us break down this LTL formula into multiple parts and analyse their meaning:

- $\square\diamond$b: The robot must always eventually visit the base region
- $\square$(b $\rightarrow \bigcirc\neg$b$\mathcal{U}\diamond$g): When in the base region, the robot must not return to it before visiting the goal region
- $\square\neg$(b $\wedge$ g): The robot cannot simultaneously occupy the base and goal regions
- $\square\neg$d: The robot must always avoid the dangerous region

With our LTL formula defined, we proceed to construct the Büchi automaton using the "translate" command from the Spot library. We employ the options "Büchi", "Deterministic", and "state-based" to generate a deterministic Büchi automaton with state-based acceptance. Figure 7-4 illustrates the Büchi automaton representation of the LTL formula ϕ.

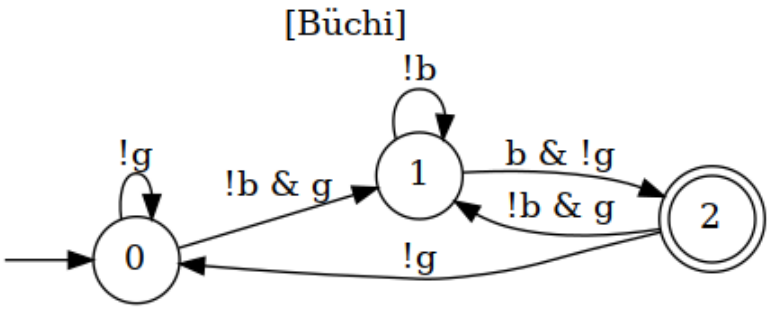

Figure 7-4: LTL formula ϕ translated to a Büchi automaton

The Büchi automaton depicted consists of three states: 0, 1, and 2. State "2" is designated as the accepting state, which implies that our LTL synthesis solution must ensure that state

"2" is visited infinitely often. A key observation is that the only transition leading to the accepting state (state "2") is conditioned by "b & !g", which corresponds to the base region. This structure ensures that our solution will visit the base region infinitely often, aligning with our expected behavior. Moreover, after exiting state "2", the automaton ensures a visit to the goal region, though not necessarily immediately. This can occur either through a direct transition from state "2" to state "1" using the condition "g", or by first transitioning to state "0" and later moving to state "1" with the condition "g". This structure enforces the surveillance pattern we intended: the robot must always visit the goal after leaving the base before it can return to the base again.

The top-level system S_3 is represented as a directed graph, as illustrated in Figure 7-5.

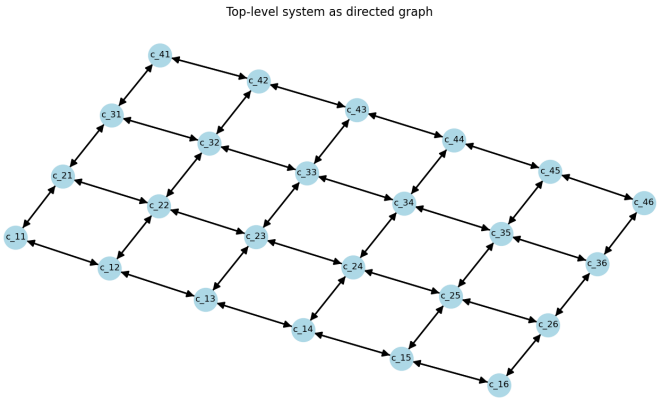

Figure 7-5: Top-level system as a directed graph

Following algorithm 5.1, we construct a product automaton and determine an accepting run using NetworkX's "shortest_path" command. Prior to finding this accepting run, we verify the N-step backward reachability condition for all $p_i, (p_i, p_j), p_j \in \underset{3}{\longrightarrow}$. This step ensures the feasibility of reaching any adjacent cell from the current cell. The entire process—from translating the LTL formula to a Büchi automaton to identifying an accepting run—was completed in $8.5ms$. The accepting run, projected onto the trajectory of S_3, yields the following results:

$$\sigma_{\text{pre}} = [\text{'c\_31'}, \text{'c\_21'}, \text{'c\_11'}, \text{'c\_12'}]$$

$$\sigma_{\text{suf}} = [\text{'c\_13'}, \text{'c\_14'}, \text{'c\_24'}, \text{'c\_34'}, \text{'c\_44'}, \text{'c\_45'}, \text{'c\_35'}, \text{'c\_34'}, \text{'c\_24'},$$

$$\text{'c\_14'}, \text{'c\_13'}, \text{'c\_12'}, \text{'c\_11'}, \text{'c\_12'}]$$

Examining the Büchi automaton in Figure 7-4, we observe that the accepting state "2" can only be reached via the transition condition "b & !g". In our transition system S_3, the sole state associated with proposition "b" is c_11. Notably, state c_11 appears as the

penultimate state in both the prefix and suffix. This occurrence aligns with the Büchi automaton's transition from state 1 to state 2 using the condition "b & !g", which corresponds to the transition from state c_11 to c_12 in S_3, as c_11 carries the atomic proposition "b". Figure 7-6 provides a visual representation of this accepting run as a directed graph, with the initial state highlighted in orange.

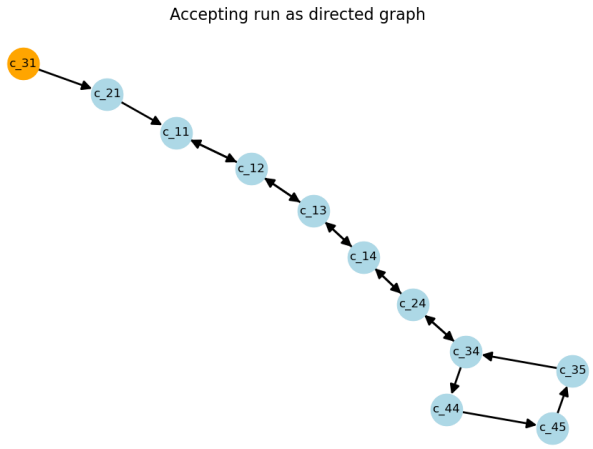

Figure 7-6: Accepting run projected onto a trajectory of S_3

7-1-2 Constraints for the MPC problem

The initial state of the mid-level system S_2 is defined as $S_{2,0} = (0.2, 0.6, 0, 0)^T$. The sampling time for the system is set to $T = 1s$. We set the prediction horizon for the MPC problem to $N = 15$ steps. We define the cost matrices as $Q = Q_f = \text{diag}(10, 10, 1, 1)$ and $R = \text{diag}(1, 1)$. The higher weights in the position components of matrices Q and Q_f prioritize the minimization of positional errors in the optimization. The complete formulation of the MPC problem is detailed in appendix A.

Position constraints for states (x_m, y_m) are derived from the output partitions of the top layer. For velocity constraints, we face a tricky design choice. Ideally, we would prefer constraints of the form $\sqrt{v_{x_m}^2 + v_{y_m}^2} \leq v_{max}$. However, to maintain the structure of polytopic constraints, we opt for $|v_{x_m}| + |v_{y_m}| \leq v_{max}$, with $v_{max} = 0.5m/s$. This design choice is motivated by our attempt to incorporate the physical limitations of the Elisa3 robots into our simulated environment. An alternative approach would be to set $|v_{x_m}| \leq v_{max}$ and $|v_{y_m}| \leq v_{max}$ independently. However, this could potentially allow the robot to move at a speed of $0.7m/s$ (when both $v_{x_m} = v_{y_m} = v_{max} = 0.5m/s$), which exceeds the physical capabilities of the robot. Our more conservative approach ensures that the simulated robot's movement remains within realistic bounds.

In our simulations, the middle layer completes its operations in an average of $9ms$ for a single robot over the entire prediction horizon. If we consider only the top and middle layers, ignoring the bottom layer, the mid-level controlled system produces the output trajectory shown in figure 7-7.

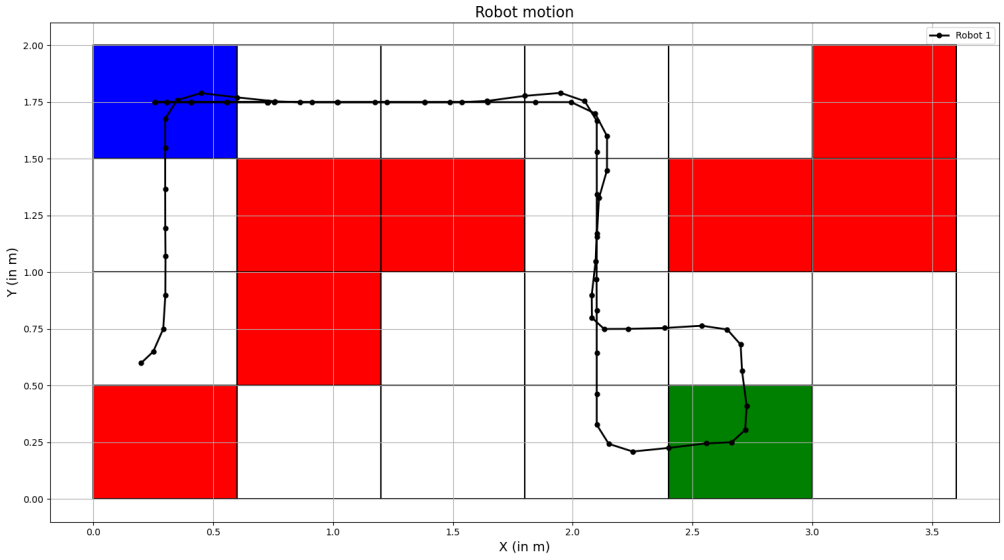

Figure 7-7: Trajectory of the robot commanded by the middle layer

7-1-3 Tuning the feedback linearization controller

The feedback linearization controller formulated in appendix C requires tuning of three controller constants: K_p, K_d, and σ. We employ a trial-and-error approach to determine the optimal set of these constants for our specific problem. This approach involves systematically adjusting one parameter while keeping the others fixed. For instance, we might fix K_d and σ while incrementally adjusting K_p to observe its effect on the system response. Once a satisfactory response is achieved, we then fix K_p and σ, and vary K_d. Through multiple iterations of this process, we converge on a set of constants that provide a sufficiently good response for our system. The final controller constants are $K_p = 17$, $K_d = 1.6$, and $\sigma = 17$. An important consideration in the initialization of system S_1 pertains to the velocity state. The control input calculations incorporate terms where velocity appears in the denominator, necessitating the need to maintain non-zero velocity values to avoid undefined behavior. This situation is particularly relevant at the start of the program, where the initial velocity of the robot is typically zero. To address this, we initialize the low-level system with a small, non-zero velocity of $0.01m/s$ (or $1cm/s$). This initialization serves a dual purpose: it prevents undefined behavior that would arise from a zero initial

velocity, while also avoiding the generation of excessively large control inputs that could result from very small velocities (e.g., $0.00001m/s$).

While true continuous control is not feasible in our digital implementation, we approximate it by operating the low-level controller at a high frequency of $20Hz$ (corresponding to a time period of $50ms$). This high-frequency operation ensures sufficient time for calculating low-level control inputs and occasionally computing the mid-level trajectory, as the middle layer operates at $1Hz$. It is generally beneficial to run the bottom layer multiple times between successive middle layer operations. For instance, if the bottom layer operates only 5 times between two middle layer operations, there might not be adequate time to rectify any errors originating from the bottom layer. Finally, our simulations show that the bottom layer requires an average of $5ms$ to complete its operation.

Figure 7-8 illustrates the interpolation of the mid-level trajectory performed by the function $\mathcal{T}_{\alpha,T}$. This interpolated trajectory closely aligns with the linear interpolation shown in figure 7-7. However, the key distinction lies in the nature of our nonlinear interpolation function. Unlike a linear interpolator, $\mathcal{T}_{\alpha,T}$ enables us to perform the necessary calculations for deriving low-level control inputs. For instance, $\mathcal{T}_{\alpha,T}$ allows us to compute second and third-order derivatives of the trajectory, which are crucial for our low-level controller calculations.

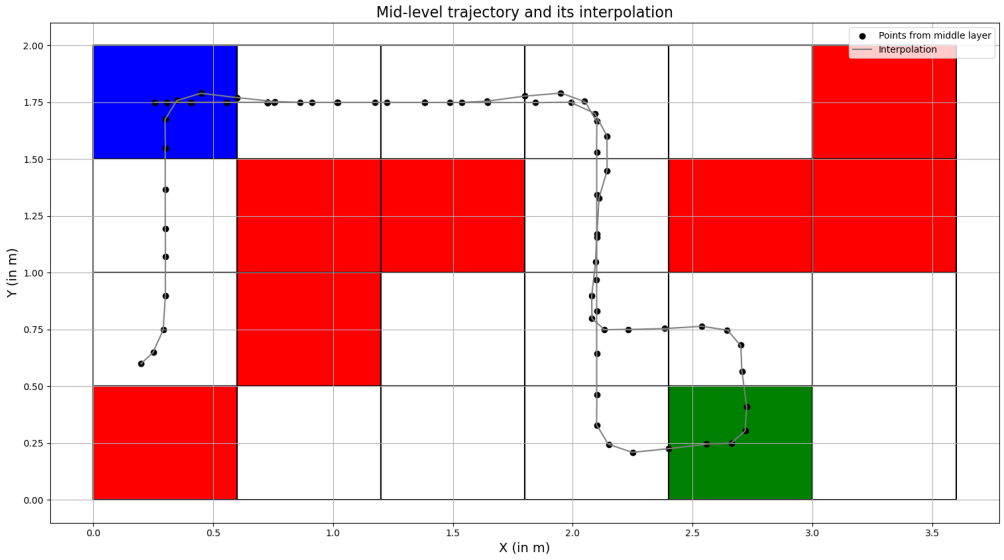

Figure 7-8: Nonlinear interpolation of the mid-level trajectory

7-1-4 Results of the integrated system

With all layers of our hierarchical control system defined and implemented, we now present the results of the full system in operation. Figure 7-9 illustrates the performance of our

integrated control structure. As evident from the figure, the low-level concrete system demonstrates a good degree of accuracy in tracking the interpolated trajectory. This close adherence to the planned path indicates effective coordination between the different layers of our control hierarchy. The trajectory shown in the figure not only represents the robot's

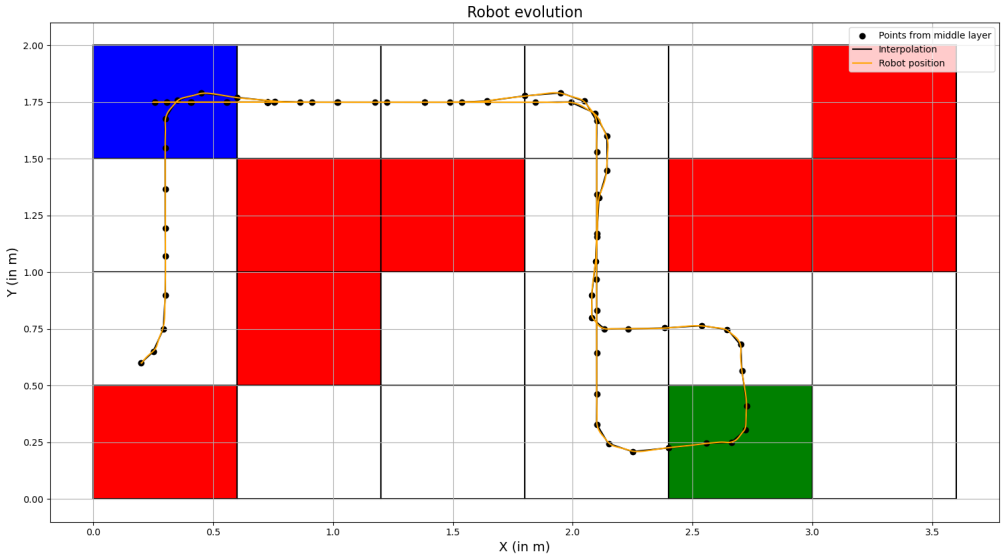

Figure 7-9: Trajectory of robot generated by the hierarchical control system

movement through space but also embodies the successful execution of the high-level task specification encoded in the LTL formula. As the robot traverses its path, it fulfills various aspects of its mission, including visiting designated areas and avoiding restricted regions, all while adhering to the constraints imposed by the hierarchical control system.

The implementation code for the simulation experiments is available in the "simulation" branch of the project repository [60].

7-2 Discussion

The slight deviations observed in the low-level controller's tracking of the desired trajectory can be attributed to the inherent nonlinear nature of the system and the current tuning of the controller constants. These deviations could potentially be reduced through further refinement of the low-level controller parameters. Additionally, exploring alternative nonlinear control strategies might yield improved tracking performance.

7-3 Simulation results for multiple robots

We extend our framework to consider a multi-robot scenario with three robots surveying the environment. Each robot is represented by a transition system S_i^k, where $i = \{1, 2, 3\}$ denotes the hierarchical layer and $k = \{a, b, c\}$ identifies the specific robot. For multi-robot coordination, we first construct a product transition system $S_{\mathcal{P}} = S_3^a \times S_3^b \times S_3^c$. The product automaton $S_{\mathcal{P}} \otimes B$ is then created to find an accepting run that, when projected onto individual systems S_3^a, S_3^b, and S_3^c, provides independent trajectories for each robot. This approach enables the specification of global LTL formulas that can enforce coordination between robots However, our current implementation faces limitations in the multi-robot scenario. While our algorithm successfully finds accepting runs for single-robot systems by comparing Büchi automaton transitions with atomic propositions (both represented as strings), it cannot handle the complexity of the product transition system in the multi-robot case. The primary challenge lies in comparing the Büchi automaton transitions (generated by Spot) with the atomic propositions of the product transition system. We were unable to develop a complete solution for this comparison, and we identify this as an important direction for future work

Chapter 8

Hardware Implementation

In this chapter, we transition from theoretical simulations to physical implementation using Elisa3 robots, discussing the challenges encountered and presenting experimental results with a modified control framework that better accommodates the practical constraints of these robots.

8-1 Challenges in hardware implementation

When implementing our hierarchical control structure on physical systems like Elisa3 robots, we introduce an additional layer beneath our concrete system for robot interfacing. This layer's primary function is to translate control inputs from the bottom layer into commands interpretable by the robots. We illustrate this modified hierarchical structure in figure 8-1.

Our control inputs consist of linear acceleration a and angular acceleration α. To make these usable for the robots, we first integrate these acceleration terms to obtain linear velocity v and angular velocity ω. These velocities are then converted to individual wheel speeds using the following equations:

$$v_r = v + \frac{\omega L}{2}$$

$$v_l = v - \frac{\omega L}{2}$$

where L represents the distance between the two wheels ($4.1cm$ in our case).

However, experimental results revealed several challenges. As is common with physical systems, actual behavior often deviates significantly from theoretical expectations. Factors

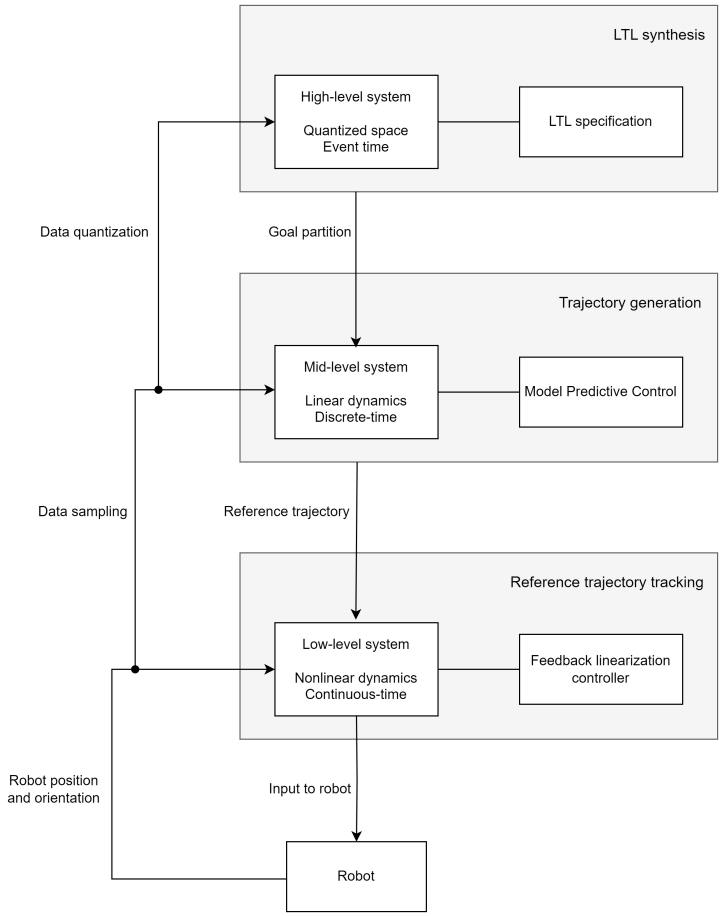

Figure 8-1: Modified block diagram of the hierarchical control system with robot interfacing

such as aging components, improper calibration, battery charge levels, and general wear and tear affect robot performance. We observed that even when commanding equal speeds to both wheels, the robots failed to move in a straight line, indicating asynchronous wheel behavior.

To address these issues, we implemented a proportional controller for velocity tracking. However, two significant problems persisted. First, the velocity controller could not fully compensate for the wheel speed synchronization issues. Second, our bottom layer's design posed challenges: even with successful velocity tracking, positional errors could accumulate between the desired and actual robot positions. Our feedback linearization controller was not designed to handle large errors, as its control inputs increase rapidly with growing error magnitudes. This situation demanded extremely precise velocity control at the robot level for proper trajectory tracking. Given these practical constraints, we developed an alternative control strategy better suited to the robots' capabilities.

8-2 Solution for hardware implementation

Figure 8-2 illustrates the modified control architecture implemented in our robot experiments.

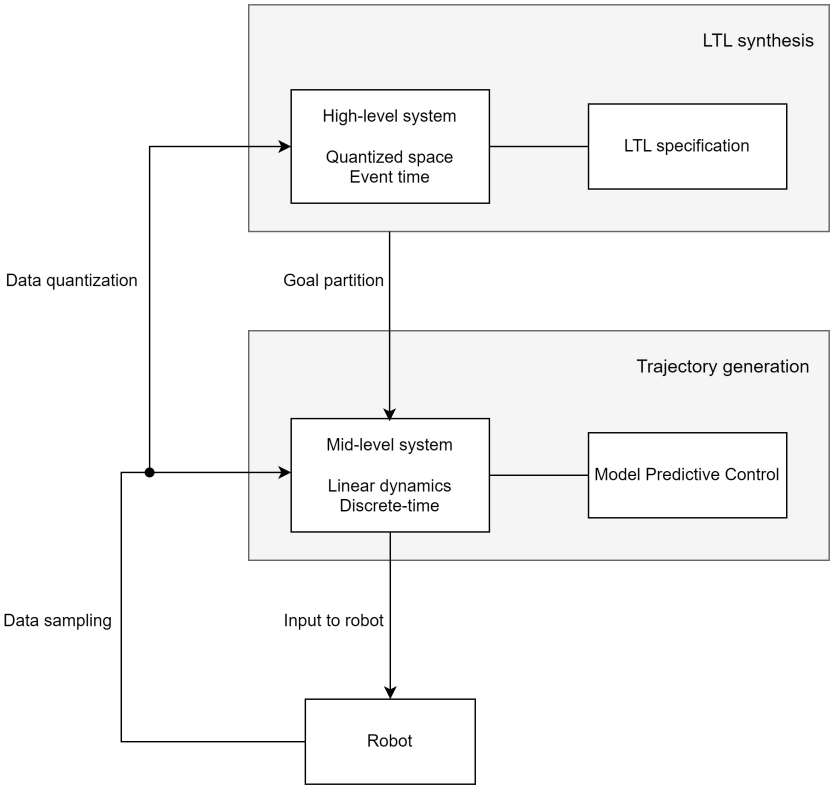

Figure 8-2: Final version of the hierarchical control system with robot interfacing

We simplified our original design by removing the bottom layer containing the nonlinear system with feedback linearization controller, instead creating a direct interface between the middle layer and the robot. On the robot side (estimator node), we implement the proportional controller developed in [59] to get the linear and angular velocity inputs. This controller uses the difference between current position and destination as its error term. The destination coordinates are derived from the MPC optimization problem solved in the middle layer, enabling the robot to maintain sufficient speeds for reaching subsequent partitions.

We implemented an additional control feature where the robot prioritizes rotational movement when the angular velocity exceeds certain thresholds. This enhancement allows the robot to achieve proper orientation before forward movement, resulting in smoother trajectories with reduced curvature (see [59] for more details).

8-3 Experiment: Case 1

In our first experimental scenario, we focused on a subset of the environment, specifically cells c_ij where i, j = $\{1, 2, 3\}$. Figure 8-3 shows this environment with its corresponding labels. The LTL formula remains unchanged:

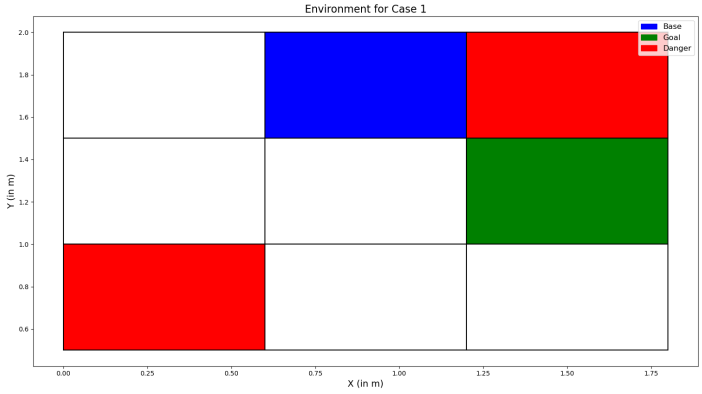

Figure 8-3: Visual representation of the environment for case 1

$$\phi = \square\diamond \mathrm{b} \wedge \square(\mathrm{b} \rightarrow \bigcirc\neg \mathrm{b}\mathcal{U}\diamond \mathrm{g}) \wedge \square\neg(\mathrm{b} \wedge \mathrm{g}) \wedge \square\neg \mathrm{d}$$

Graph search yielded the following accepting run:

$$\sigma_{\text{pre}} = [\text{'c\_11'}, \text{'c\_12'}, \text{'c\_11'}]$$

$$\sigma_{\text{suf}} = [\text{'c\_21'}, \text{'c\_22'}, \text{'c\_23'}, \text{'c\_22'}, \text{'c\_12'}, \text{'c\_11'}]$$

For our experimental implementation, we set the Elisa3 node to run at $100Hz$ and the estimator node at $50Hz$. The middle layer operates at a frequency of $2Hz$ ($T = 0.5s$) with a prediction horizon of $N = 15$. Figure 8-4 displays the resulting robot trajectories, with arrows indicating their evolution. Despite significant deviations from the desired mid-level trajectory, the robot successfully satisfied its high-level LTL specification. These deviations arose from limitations in robot side control and hardware variations. The erratic nature of robot motion could be mitigated by implementing a more sophisticated velocity controller and using well-calibrated robots in better mechanical condition. A video demonstration of the robot's motion can be found in [61].

8-4 Experiment: Case 2

In our second experimental case, we expand our testing to include the entire workspace, resulting in a more extensive motion plan than Case 1. Figure 8-5 illustrates the environment with its corresponding labels.

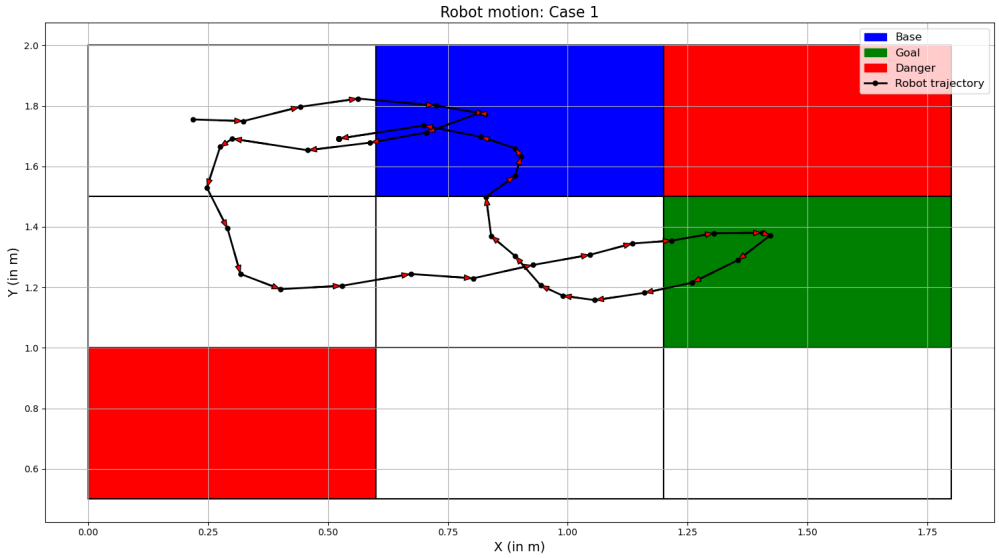

Figure 8-4: Trajectory of robot for Case 1

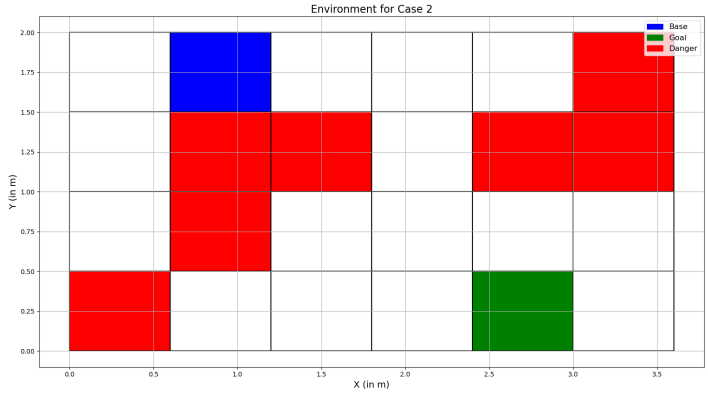

Figure 8-5: Visual representation of the environment for case 2

We maintain the same LTL specification and operational frequencies for all nodes as in Case 1. The accepting run for this case is:

$$\sigma_{\text{pre}} = [\text{'c\_11'}, \text{'c\_12'}, \text{'c\_13'}]$$

$$\sigma_{\text{suf}} = [\text{'c\_14'}, \text{'c\_24'}, \text{'c\_34'}, \text{'c\_35'}, \text{'c\_34'}, \text{'c\_24'}, \text{'c\_14'}, \text{'c\_13'}, \text{'c\_12'}, \text{'c\_11'}]$$

The robot's trajectory for this extended scenario is depicted in Figure 8-6. Similar to case 1, the robot successfully fulfills its LTL specification, albeit in a slightly erratic manner.

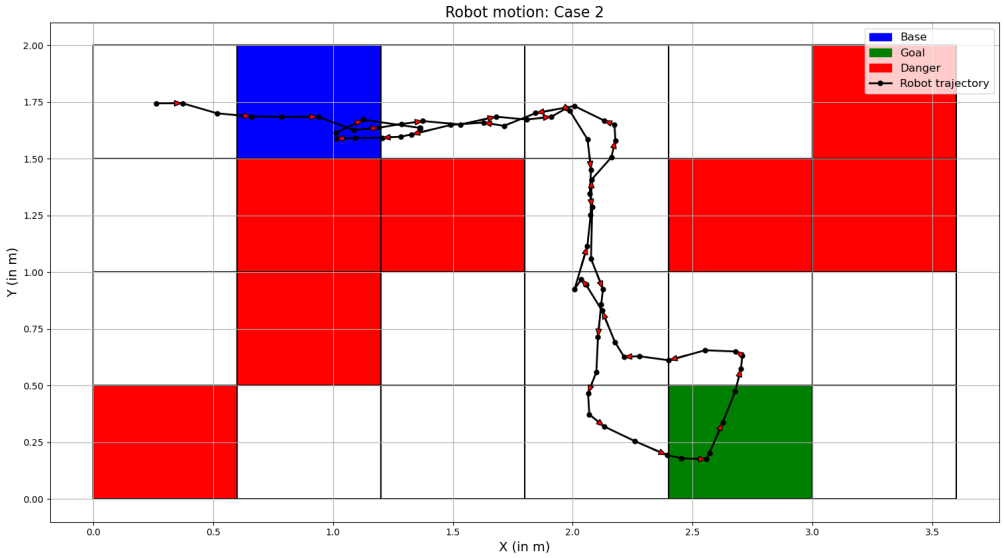

Figure 8-6: Trajectory of robot for Case 2

A video demonstration of the robot's motion can be found in [62]. The implementation code for the hardware experiments is available in the "hardware" branch of the project repository [60].

8-5 Discussion

Although our experiments demonstrated successful completion of the high-level mission specifications in both cases, the robot exhibited irregular motion patterns due to limitations in the velocity controller. In some trials, the experiments had to be terminated when the robot deviated into incorrect partitions, causing the MPC optimization problem to become infeasible. The implementation of a more sophisticated velocity controller would likely result in more consistent performance, characterized by smoother trajectories and fewer deviations from the planned path.

As discussed in Section 7-3, our current limitation in finding accepting runs for product automata in multi-robot systems prevents us from implementing hardware experiments with multiple robots. The development of this capability and corresponding physical implementation remains an important direction for future work.

Chapter 9

Conclusion and Future Work

This thesis presented the implementation of a contract-based hierarchical control framework for robotic systems. The framework addresses several key challenges: providing formal guarantees across different abstraction levels, enabling independent controller development, and offering implementation flexibility where components can be replaced while maintaining system guarantees. Using this hierarchical approach, complex robotic surveillance tasks were decomposed into manageable subtasks, integrating motion planning, trajectory generation, and trajectory tracking into a unified framework. Through both simulation and hardware experiments on Elisa3 robots, the framework demonstrated its capability to execute surveillance tasks specified using Linear Temporal Logic. The modular nature of the framework makes it adaptable to various complex problems through different choices of system models and controller designs.

The implementation revealed several practical challenges and limitations. A primary limitation was the suboptimal velocity controller implementation on the robot side, coupled with the feedback linearization controller being tuned through trial and error rather than systematic methods. The framework's assumption of a disturbance-free environment presented challenges in hardware implementation where external disturbances are inevitable. Furthermore, the current algorithm for finding accepting runs, being a temporary solution, proved inadequate for extension to multi-robot systems, where additional challenges of communication delays and synchronization would need to be addressed.

The framework's evaluation also had several analytical limitations. The testing was restricted to relatively simple LTL specifications, leaving questions about computational feasibility for more complex temporal requirements unanswered. The absence of multi-robot implementation prevented assessment of task completion times in coordinated scenarios. Additionally, the lack of comparison with baseline approaches makes it difficult to quantitatively evaluate the framework's performance advantages, despite its use of simple controllers at each layer.

A key priority for future work is developing enhanced correction controllers for the physical robots to better match simulation performance, addressing issues like wheel synchronization and hardware variations. To enhance the system's real-world applicability, incorporating robust control techniques and obstacle avoidance in one or multiple layers could be a valuable direction for future work. At the middle layer, for example, Tube MPC could be employed to explicitly account for uncertainties and disturbances. For the lower layer, robust nonlinear control methods such as H-infinity control or adaptive control could be considered to maintain performance in the face of model inaccuracies or external perturbations.

The framework could also be extended to handle dynamic environments, including both moving obstacles and changing workspace conditions, through the incorporation of reactive synthesis techniques. The framework's applicability could be extended to more complex scenarios, such as coordinated warehouse operations where one robot performs surveillance of restricted areas while another handles package delivery tasks, with both robots needing to coordinate their movements to avoid conflicts. Additionally, the integration of STL tasks could enable richer specifications with explicit timing constraints, enhancing the framework's expressiveness.

Appendix A

MPC formulation

We formulate our MPC problem following an approach similar to [1]. Consider a transition $p_i \rightarrow p_j$ commanded by the top layer, with an initial state $x_{2,0} \in \mathcal{C}_{p_i}$, where both partitions p_i and p_j are represented as polytopes. The first constraint restricts the robot's movement to the space $(p_i \cup p_j) \ominus B_\delta$, which itself forms a polytope. Defining the position states as $x_{\text{pos}}[k] = \begin{pmatrix} x_m[k] & y_m[k] \end{pmatrix}^T$, we can express this overall position constraint as the polytope $A_{\text{pos}} x_{\text{pos}}[k] \leq b_{\text{pos}}$.

For terminal constraints, the robot must reach the set $\mathcal{C}_{p_j}$ within N steps, where N is the prediction horizon. This can be formulated as $A_{C_{p_j}} x_{\text{pos}}[k] \leq b_{C_{p_j}}$, with the terminal position states of $x_{2,f}$ chosen as the center of the control invariant set $\mathcal{C}_{p_j}$ and terminal velocity states set to zero.

The state velocity and input constraints are defined as $-v_{max} \leq x_{\text{vel}}[k] \leq v_{max}$ and $-\Delta v_{max} \leq u_2[k] \leq \Delta v_{max}$ respectively, where $x_{\text{vel}}[k] = \begin{pmatrix} v_{x_m}[k] & v_{y_m}[k] \end{pmatrix}^T$ are the velocity states.

With cost matrices $Q, Q_f \succeq 0$ and $R \succ 0$, we formulate the optimal control problem over the input u_2^* as:

$$u_2^* = \arg\min_{u_2} \sum_{k=0}^{N-1} \left(\|x_2[k] - x_{2,f}\|_Q^2 + \|u_2[k]\|_R^2 \right) + \|x_2[N] - x_{2,f}\|_{Q_f}^2 \tag{A-1}$$

$$
\begin{aligned}
s.t. \quad & x_2[k+1] = \begin{bmatrix} I & TI \\ 0 & I \end{bmatrix} x_2[k] + \begin{bmatrix} 0 \\ I \end{bmatrix} u_2[k], && \forall k \in \{0, \dots, N-1\} \\
& A_{\text{pos}} x_{\text{pos}}[k] \leq b_{\text{pos}}, && \forall k \in \{0, \dots, N-1-n\} \\
& A_{C_{p_j}} x_{\text{pos}}[k] \leq b_{C_{p_j}}, && \forall k \in \{N-n, \dots, N\} \\
& -v_{max} \leq x_{\text{vel}}[k] \leq v_{\text{max}}, && \forall k \in \{0, \dots, N\} \\
& -\Delta v_{max} \leq u_2[k] \leq \Delta v_{max}, && \forall k \in \{0, \dots, N-1\}
\end{aligned}
\tag{A-2}
$$

where $\|x\|_Q^2 = x^T Q x$ denotes the weighted 2-norm. The variable n tracks the elapsed time steps since receiving new partition commands from the top layer, ensuring the transition completes within N steps.

Appendix B

Interpolation Function

We implement a slightly modified version of the interpolation function presented in [1]. The nonlinear interpolation function $\mathcal{T}_{\alpha,T}([x_m[k]]_{k=0}^N)$ generates a smooth trajectory through the discrete points $x_m[k]$. This function is defined over time intervals $t \in [kT, (k+1)T)$ as:

$$\begin{aligned} x(t) &= \begin{cases} x^1(t) & t \in [kT, (k+0.5)T) \\ x^2(t) & t \in [(k+0.5)T, (k+1)T) \end{cases} \\ x^1(t) &= s_k + v_k(t-kT) - \frac{v_{k+1}-v_k}{\delta T^2} t_{p,1}^3 \left(1 + 2\frac{t_{p,1}}{\delta T}\right) \\ x^2(t) &= s_k + v_k(t-kT) - \frac{v_{k+1}-v_k}{\delta T^2} t_{p,2}^3 \left(1 - 2\frac{t_{p,2}}{\delta T}\right) \\ \delta T &= 2(1-\alpha/2)T \\ t_{p,1} &= t - kT - \alpha T/2 \\ t_{p,2} &= t - kT - \delta T/2 \end{aligned} \tag{B-1}$$

where $s_k = x_m[k]$, $v_k = v_{x_m}[k]$, $v_{k+1} = v_{x_m}[k+1]$, $\alpha = 1$ and T is the sampling time of the middle layer. An identical interpolation can be applied to $y_m[k]$. At the interval endpoints kT and $(k+1)T$, the nonlinear interpolator coincides with the linear interpolator, ensuring the trajectory passes through the discrete points specified by the middle layer. The maximum deviation between the linear interpolator $\bar{x}$ and our nonlinear interpolator $x(t)$ is given by:

$$|x(t) - \bar{x}| = \frac{3}{32}\alpha|v_{k+1} - v_k|T \tag{B-2}$$

This maximum deviation value is considered when designing the polytope B_δ in the middle layer to ensure safe transitions between partitions. Figure 7-8 shows an example case of the nonlinear interpolator. For a further analysis of the interpolation function, please refer to [1].

Appendix C

Feedback Linearization Controller

We design the low-level controller by combining feedback linearization with Lyapunov backstepping techniques [1, 52, 51]. The control design begins by defining the error signal:

$$e = \begin{pmatrix} x_l \\ y_l \end{pmatrix} + x_d(t) \tag{C-1}$$

where x_l, y_l represent the low-level system's position states, and $x_d(t)$ is the reference trajectory obtained by interpolating x_m and y_m using the function $\mathcal{T}_{\alpha,T}$ (detailed in appendix B). Following the approach in [1], the control inputs are given by:

$$\begin{aligned}
u &= \begin{pmatrix} a_d \\ \alpha_d \end{pmatrix} \\
a_d &= \begin{pmatrix} \cos\theta & \sin\theta \end{pmatrix} \ddot{e}_d \\
a_d &= -\frac{1}{2}\sigma(\omega - \omega_d) + \dot{\omega}_d - 2\begin{pmatrix} e^T & \dot{e}^T \end{pmatrix} P \begin{pmatrix} 0 \\ 0 \\ -v\sin\theta \\ v\cos\theta \end{pmatrix} \\
\ddot{e}_d &= \ddot{x}_d(t) - K_p e - K_d \dot{e} \\
\ddot{e} &= -K_p e + -K_d \dot{e} \\
\omega_d &= \frac{1}{v}\begin{pmatrix} -\sin\theta & \cos\theta \end{pmatrix} \ddot{e}_d \\
\dot{\omega}_d &= \begin{pmatrix} \frac{a_d}{v^2}\sin\theta - \frac{\omega}{v}\cos\theta & -\frac{a_d}{v^2}\cos\theta - \frac{\omega}{v}\sin\theta \end{pmatrix} \ddot{e}_d + \\
&\quad \frac{1}{v}\begin{pmatrix} -\sin\theta & \cos\theta \end{pmatrix} (\dddot{x}_d(t) - K_p \dot{e} - K_d \ddot{e})
\end{aligned} \tag{C-2}$$

where K_p, K_d, σ are the controller constants. P is obtained by solving the continuous-time Lyapunov equation

$$\begin{aligned} A^T P + PA &= -I \\ A &= \begin{pmatrix} 0 & I \\ -K_p & -K_d \end{pmatrix} \end{aligned} \tag{C-3}$$

Bibliography

[1] M. Mazo Jr, W. Compton, M. H. Cohen, and A. D. Ames, "A contract theory for layered control architectures," *arXiv preprint arXiv:2409.14902*, 2024.

[2] "Elisa3 - gctronic wiki." https://www.gctronic.com/doc/index.php/Elisa-3. Accessed: 17-10-2024.

[3] N. Smith and A. Sage, "An introduction to hierarchical systems theory," *Computers & Electrical Engineering*, vol. 1, no. 1, pp. 55–71, 1973.

[4] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, K. G. Larsen, *et al.*, "Contracts for system design," *Foundations and Trends® in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[5] P. Nuzzo and A. L. Sangiovanni-Vincentelli, "Hierarchical system design with vertical contracts," *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, pp. 360–382, 2018.

[6] P. Tabuada, *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

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

[8] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and automation*, vol. 14, no. 1, pp. 166–171, 1998.

[9] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE international conference on robotics and automation*, pp. 489–494, IEEE, 2009.

[10] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[11] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2020–2025, IEEE, 2005.

[12] M. Kloetzer and C. Belta, "Distributed implementations of global temporal logic motion specifications," in *2008 IEEE International Conference on Robotics and Automation*, pp. 393–398, IEEE, 2008.

[13] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[14] D. Aksaray, K. Leahy, and C. Belta, "Distributed multi-agent persistent surveillance under temporal logic constraints," *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 174–179, 2015.

[15] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[16] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*, vol. 89. Springer, 2017.

[17] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57, ieee, 1977.

[18] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 4885–4890, IEEE, 2005.

[19] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[20] G. E. Fainekos, A. Girard, and G. J. Pappas, "Hierarchical synthesis of hybrid controllers from temporal logic specifications," in *Hybrid Systems: Computation and Control: 10th International Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007. Proceedings 10*, pp. 203–216, Springer, 2007.

[21] 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.

[22] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning under temporal logic constraints," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3288–3293, IEEE, 2010.

[23] 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, IEEE, 2014.

[24] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," in *Verification, Model Checking, and Abstract Interpretation: 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006. Proceedings 7*, pp. 364–380, Springer, 2006.

[25] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[26] J. Alonso-Mora, J. A. DeCastro, V. Raman, D. Rus, and H. Kress-Gazit, "Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles," *Autonomous Robots*, vol. 42, pp. 801–824, 2018.

[27] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with tulip: The temporal logic planning toolbox," in *2016 IEEE Conference on Control Applications (CCA)*, pp. 1030–1041, IEEE, 2016.

[28] U. Rosolia, A. Singletary, and A. D. Ames, "Unified multirate control: From low-level actuation to high-level planning," *IEEE Transactions on Automatic Control*, vol. 67, no. 12, pp. 6627–6640, 2022.

[29] L. Lindemann, *Planning and control of multi-agent systems under signal temporal logic specifications*. PhD thesis, KTH Royal Institute of Technology, 2020.

[30] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," *Lectures on Runtime Verification: Introductory and Advanced Topics*, pp. 135–175, 2018.

[31] 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.

[32] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE control systems letters*, vol. 3, no. 1, pp. 96–101, 2018.

[33] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[34] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on ltl specifications," in *2004 43rd IEEE conference on decision and control (CDC)(IEEE Cat. No. 04CH37601)*, vol. 1, pp. 153–158, IEEE, 2004.

[35] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2009.

[36] I. Filippidis, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Decentralized multi-agent control from local ltl specifications," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 6235–6240, IEEE, 2012.

[37] J. Tumova and D. V. Dimarogonas, "A receding horizon approach to multi-agent planning from local ltl specifications," in *2014 American Control Conference*, pp. 1775–1780, IEEE, 2014.

[38] J. Tumova and D. V. Dimarogonas, "Decomposition of multi-agent planning under distributed motion and task ltl specifications," in *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 7448–7453, IEEE, 2015.

[39] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[40] Y. Kantaros and M. M. Zavlanos, "Sampling-based control synthesis for multi-robot systems under global temporal specifications," in *Proceedings of the 8th International Conference on Cyber-Physical Systems*, pp. 3–13, 2017.

[41] M. Srinivasan, S. Coogan, and M. Egerstedt, "Control of multi-agent systems with finite time control barrier certificates and temporal logic," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 1991–1996, IEEE, 2018.

[42] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimal multi-robot path planning with temporal logic constraints," in *2011 IEEE/RSJ international conference on intelligent robots and systems*, pp. 3087–3092, IEEE, 2011.

[43] B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[44] B. M. Shali, A. van der Schaft, and B. Besselink, "Behavioural assume-guarantee contracts for linear dynamical systems," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 2002–2007, IEEE, 2021.

[45] B. M. Shali, A. van der Schaft, and B. Besselink, "Composition of behavioural assume-guarantee contracts," *IEEE Transactions on Automatic Control*, 2022.

[46] B. Besselink, K. H. Johansson, and A. Van Der Schaft, "Contracts as specifications for dynamical systems in driving variable form," in *2019 18th European Control Conference (ECC)*, pp. 263–268, IEEE, 2019.

[47] E. S. Kim, M. Arcak, and S. A. Seshia, "A small gain theorem for parametric assume-guarantee contracts," in *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pp. 207–216, 2017.

[48] M. Al Khatib and M. Zamani, "Controller synthesis for interconnected systems using parametric assume-guarantee contracts," in *2020 American Control Conference (ACC)*, pp. 5419–5424, IEEE, 2020.

[49] A. Eqtami and A. Girard, "A quantitative approach on assume-guarantee contracts for safety of interconnected systems," in *2019 18th European Control Conference (ECC)*, pp. 536–541, IEEE, 2019.

[50] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[51] M. Krstic, P. V. Kokotovic, and I. Kanellakopoulos, *Nonlinear and Adaptive Control Design*. John Wiley & Sons, Inc., 1st ed., 1995.

[52] S. Vaidyanathan and A. T. Azar, "An introduction to backstepping control," in *Backstepping Control of Nonlinear Dynamical Systems*, pp. 1–32, Elsevier, 2021.

[53] I. Incer, *The Algebra of Contracts*. PhD thesis, EECS Department, University of California, Berkeley, May 2022.

[54] "Spot: a platform for ltl and ω-automata manipulation." https://gitlab.lre.epita.fr/spot/spot.git. Accessed: 19-10-2024.

[55] "Polytope: a toolbox for geometric operations on polytopes in any dimension." https://github.com/tulip-control/polytope.git. Accessed: 19-10-2024.

[56] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-Parametric Toolbox 3.0," in *Proc. of the European Control Conference*, (Zürich, Switzerland), pp. 502–510, July 17–19 2013. http://control.ee.ethz.ch/~mpt.

[57] Y. Li, "Indoor localization with multi-rate extended kalman filter," Master's thesis, Delft University of Technology, 2023.

[58] "Localization code for elisa3 robots." https://github.com/kemosabe564/ting_msc.git. Accessed: 17-10-2024.

[59] "Improved localization code for elisa3 robots." https://github.com/shashank0911/elisa3_code. Accessed: 17-10-2024.

[60] "Project repository for the hierarchical control system." https://github.com/shashank0911/hierarchical_control.git. Accessed: 23-10-2024.

[61] "Video showing robot motion for case 1 of the hardware implementation." https://drive.google.com/file/d/1CXKxIqBuHV5wtQry0y0Aicbb9BTUs_Rq/view?usp=sharing. Accessed: 22-10-2024.

[62] "Video showing robot motion for case 2 of the hardware implementation." https://drive.google.com/file/d/1CnLQQVmFeji1ABsvptet3X-LdSkU3fIi/view?usp=sharing. Accessed: 22-10-2024.