Formal Methods for Design and Verification of Embedded Control Systems: Application to an Autonomous Vehicle

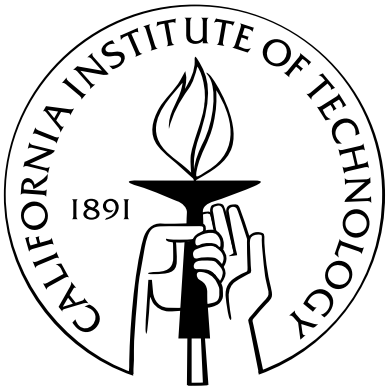
Thesis by

Tichakorn Wongpiromsarn

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2010

(Defended May 27, 2010)

Acknowledgements

I owe my deepest gratitude to my advisor, Richard Murray, for all the support, guidance and encouragement he has provided throughout my PhD studies. Richard has been much more than a thesis advisor for me. He has given me the freedom to pursue my own interests. In the meantime, he has always been available to give valuable insights and advice. His enthusiasm and dedicated efforts have shaped my attitude toward research and work. I am also grateful for the opportunity to work on the DARPA Urban Challenge, which turned out to be not only an invaluable experience but also the main motivation of this work.

It has been a great pleasure and an honor for me to have Joel Burdick, Mani Chandy and Gerard Holzmann on my thesis committee. A significant portion of this work has been inspired by the discussions with them. As my first-year advisor, Joel also helped me through what is believed to be the toughest year of the PhD program and continued to provide support and advice throughout my PhD studies.

I enjoy working with my collaborators, Andy Lamperski, Sayan Mitra and Ufuk Topcu. I would like to thank Mitch Ingham and Bob Rasmussen for their tremendous help and advice regarding the mission management subsystem of Alice and the canonical software architecture. I would also like to thank the Murray group, the robotics group, Team Caltech, SOPS, the Thai club, and the wonderful group of people in CDS and ME for their support and friendship and for making my life at Caltech so enjoyable. A special thanks to Anissa Scott, Charis Brown, Gloria Bain, Maria Cooper, Cheryl Geer and Chris Silva. Without their generous help and professional assistance, my life at Caltech would be much harder.

Finally, I wish to extend my warm appreciation to my family for their unconditional love, support and understanding. Thanks for making my life as beautiful as it could possibly be. Also, to my forever best friend, Gon, for always being everything a single person could be for me. Thanks for giving me the strength, hope and encouragement and for completing the missing part of me. This journey would not have been nearly as wonderful without him.

Abstract

The design of reliable embedded control systems inherits the difficulties involved in designing both control systems and distributed (concurrent) computing systems. Design bugs in these systems may arise from the unforeseen interactions among the computing, communication and control subsystems. Motivated by the difficulties of finding this type of design bugs, this thesis develops mathematical frameworks, based on formal methods, to facilitate the design and analysis of such embedded systems. An expressive specification language of linear temporal logic (LTL) is used to specify the desired system properties. The practicality of the proposed frameworks is demonstrated through autonomous vehicle case studies and autonomous urban driving problems.

Our approach incorporates methodology from computer science and control, including model checking, theorem proving, synthesis of digital designs, reachability analysis, Lyapunov-type methods and receding horizon control. This thesis consists of two complementary parts, namely, verification and design. First, we introduce Periodically Controlled Hybrid Automata (PCHA), a subclass of hybrid automata that abstractly captures a common design pattern in embedded control systems. New sufficient conditions that exploit the structure of PCHAs in order to simplify their invariant verification are presented.

Although the aforementioned technique simplifies invariant verification of PCHAs, finding a proper invariant remains a challenging problem. To complement the verification efforts, in the second part of the thesis, we present a methodology for automatic synthesis of embedded control software that provides a formal guarantee of system correctness, with respect to its desired properties expressed in linear temporal logic. The correctness of the system is guaranteed even in the presence of an adversary (typically arising from changes in the environments), disturbances and modeling errors. A receding horizon framework is proposed to alleviate the associated computational complexity of LTL synthesis. The effectiveness of this framework is demonstrated through the autonomous driving problems.

Contents

Chapter 1

Introduction

1.1 Motivation

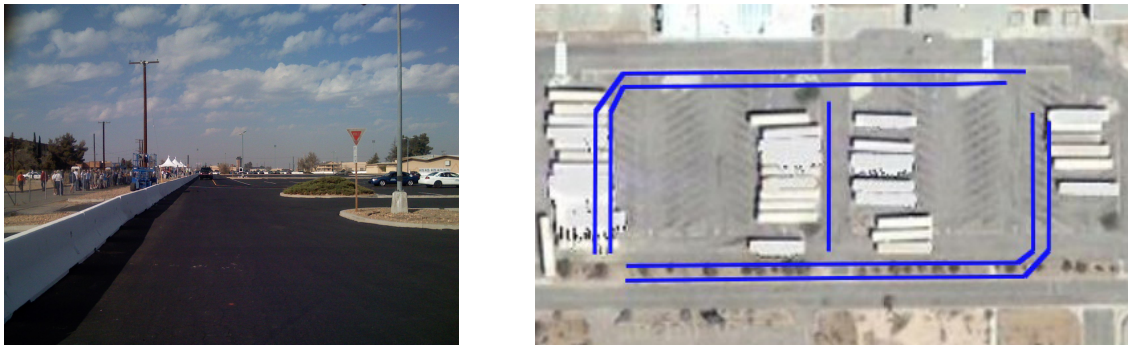
Modern engineering systems often comprise a network of sensors and actuators, equipped with computing and communication capabilities. These systems may also need to readily react to changing environments and operational situations. Such "embedded systems" appear in diverse areas including aerospace, automotive, civil infrastructure, energy, healthcare, manufacturing and transportation. Design bugs in these systems can be fairly subtle and may arise from the unforeseen interactions among the computing, the communication and the control subsystems. Finding this type of design bug is challenging, but is nevertheless important to ensure reliability of the systems. In most cases, traditional techniques for analyzing systems based on testing and simulation are not adequate to ensure the absence of these bugs.

Consider, for example, the autonomous vehicle *Alice* built at Caltech for the 2007 DARPA Urban Challenge [1]. The DARPA Urban Challenge required all the competing vehicles to navigate, in a fully autonomous manner, through a partially known urban-like environment populated with static and dynamic obstacles, including live traffic. These vehicles also had to perform different tasks such as street and off-road driving, parking and visiting certain areas while obeying traffic rules. These tasks were specified by a sequence of checkpoints that the vehicle had to cross. For the vehicles to successfully complete the race, they need to be capable of negotiating an intersection, handling changes in the environment or operating condition (e.g., newly discovered obstacles) and reactively replanning in response to those changes (e.g., making a U-turn and finding a new route when the newly discovered obstacles fully block the road).

Figure 1.1: Alice, Team Caltech's entry in the 2007 DARPA Urban Challenge.

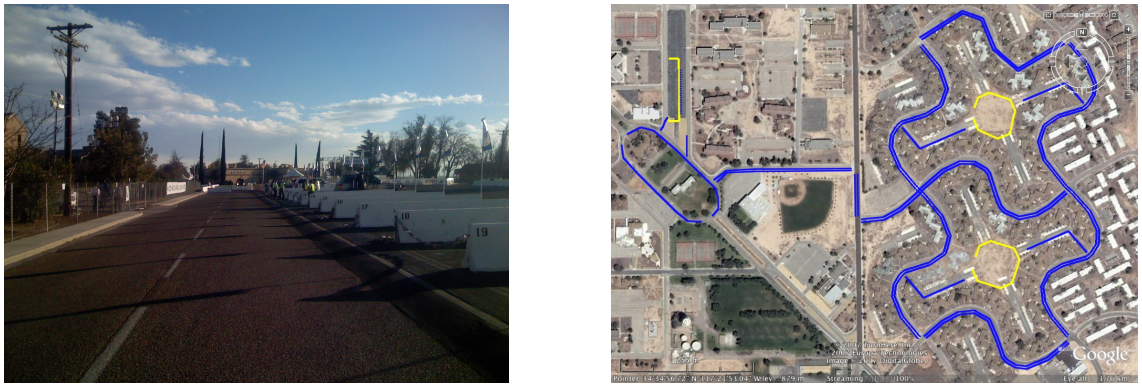
Alice (Figure 1.1) is a modified Ford E350 van, equipped with mechanical actuators (brake, throttle, steering and transmission), sensors (LADARs, RADARs and cameras) and an Applanix INS (for state estimation). The embedded computing system of Alice consists of approximately 25 programs and 200 execution threads to be executed concurrently in the 25 processors onboard. This system can be divided into the sensing and the control subsystems. The sensing subsystem provides a representation of the environment around the vehicle. The control subsystem determines and executes desired motion of the vehicle to satisfy the mission goals, which include crossing GPS waypoints, avoiding obstacles, following traffic rules, etc.

The National Qualifying Event (NQE) of the DARPA Urban Challenge was split into three test areas to assess the vehicle's capabilities in different aspects of urban driving. Each of the competing autonomous vehicles had two chances to perform each test. Test Area A (Figure 1.2) involved making left turns while merging into traffic. There were 10–12 human-operated traffic vehicles involved in this test. These traffic vehicles circled around the outer loop of the figure eight road network (Figure 1.2(b)). The autonomous vehicle started in the middle, single lane road and merged into traffic, which had the right of way, after coming to a complete stop at the bottom T-intersection. It then continually circled around the right loop of the figure eight in the counter-clockwise direction for 30 minutes. Test Area B (Figure 1.3) was designed to test basic navigation, which includes route planning, staying in lanes, parking and obstacle avoidance. There was no additional traffic involved in this



(a) (b)

Figure 1.2: (a) Test Area A. (b) Road network for Test Area A.



(a) (b)

Figure 1.3: (a) Test Area B. (b) Road network for Test Area B.

test. Test Area C (Figure 1.4) was designed to test intersection precedence with 3–4 human-operated traffic vehicles at intersections. In this test area, the autonomous vehicle also had to demonstrate the replanning and U-turn capabilities by finding an alternate route when it encountered a surprise roadblock.

Alice performed reasonably well in Test Area B and Test Area C [18, 121, 30]. It successfully completed the course of Test Area B with only minor errors. It also demonstrated correct intersection and U-turn operations throughout the run of Test Area C even though it could not finish the run within the time limit. In both executions at Test Area A, however, its behavior was unsafe and almost led to a collision. Alice was stuck at the corner of a sharp turn dangerously stuttering in the middle of an intersection.

It was later diagnosed that this behavior was caused by bad interactions between the *reactive obstacle avoidance subsystem (ROA)* and the relatively slowly reacting *path planner*.

Figure 1.4: Road network for Test Area C.

The planner incrementally generates a sequence of waypoints based on the road map, obstacles and the mission goals. The ROA is designed to rapidly decelerate the vehicle when it gets too close to (possibly dynamic) obstacles or when the deviation from the planned path gets too large. Finally, to protect the vehicle steering system, Alice's low-level controller limits the rate of steering at low speeds. Thus, accelerating from a low speed, if the planner produces a path with a sharp left turn, the controller is unable to execute the turn closely. Alice deviates from the path; the ROA activates and slows it down. This cycle continues, leading to stuttering. In order to avoid this type of unsafe behavior, the planner needs to be aware of the constraints imposed by the controller.

The above example illustrates how design of reliable embedded control systems inherits the difficulties involved in designing both control systems and distributed (concurrent) computing systems. The described design bug manifests as undesirable behavior only under a very specific set of conditions and only when the controller, the ROA and the vehicle interact in a very specific manner. Therefore, such a bug had never been discovered in thousands of hours of extensive simulations and over three hundred miles of field testing. Formal methods provide tools and techniques for uncovering such subtle design bugs and mathematically prove correctness of designs. More recently, formal techniques have also been used to automatically generate controllers that are provably correct by construction [64, 33].

Motivated by the failure of Alice, in this thesis, we develop frameworks, based on formal methods, to facilitate design and analysis of such system and other embedded control systems with similar features such as autonomous automotive systems, robots and automatic pilot avionics. This work primarily focuses on the embedded control component that regulates the underlying physical process, which we refer to as the plant. An expressive

and powerful specification language of linear temporal logic (LTL) is used throughout the thesis to specify the desired properties. The practicality of the proposed frameworks is demonstrated through Alice and the autonomous urban driving problem.

1.2 Related Works

Formal methods have been extensively studied in both computer science and control. These approaches rely on applying mathematically-based techniques in proving system correctness. Computer-science oriented approaches are mainly directed towards discrete systems. A large class of properties including deadlocks, livelocks, correctness of system invariants, safety, non-progress execution cycles have been considered. ω-regular languages and temporal logics are widely used to precisely describe such properties [8]. One of the main challenges in this domain lies in dealing with concurrency and proving system correctness for any interleaving of concurrent processes. Model checking and theorem proving are commonly used techniques to enable such proofs. Model checking is attractive because it is fully automatic. However, it is limited to systems with a finite number of states. It also faces a combinatorial blow up of the state space, commonly known as the state explosion problem. Theorem proving, on the other hand, is not limited to finite state systems but it requires a skilled human interaction. Recently, the development of a polynomial-time algorithm to construct finite state automata from their temporal logic specifications enables automatic synthesis of digital designs that satisfy a large class of properties including safety, guarantee and response even in the presence of an adversary (typically arising from changes in the environments) [98].

Control systems, on the other hand, are generally described by a set of differential equations and hence contain an infinite number of states. Reachability analysis and Lyapunov-type approaches are commonly used to verify stability and safety properties of such continuous systems. Optimization-based approaches, receding horizon control (also known as model predictive control) and the abundance of computational resources enable automatic synthesis of continuous controllers that ensure safety and stability even in the presence of disturbances and modeling errors [91, 42, 14].

As previously discussed, embedded control systems usually contain both continuous (physical) and discrete (computational) components. Hybrid system formulation has been

developed to handle such systems. Control of hybrid systems has been studied extensively but properties of interest are typically limited to stability and safety [27, 47]. For systems to perform complex tasks, a wider class of properties such as guarantee (e.g., eventually perform task 1 or task 2 or task 3) and response (e.g., if the system fails, then eventually perform task 1 or perform tasks 1, 2 and 3 infinitely often in any order) need to be considered. Temporal logics have therefore garnered great interest due to their expressive power. Methodology from computer science and control has been integrated to incorporate temporal logics in design and verification of hybrid systems. A model checking tool, HyTech, was developed for automatic verification of hybrid systems [5, 45]. Its successor, PHAVer, was designed to address many limitations of HyTech such as the overflow problem, which prohibits the use of HyTech with complex systems [34]. Both HyTech and PHAVer are symbolic model checkers for linear hybrid automata, a subclass of hybrid automata that are defined by linear predicates and piecewise constant bounds on the derivatives. Hence, the applications of HyTech and PHAVer are limited to systems whose continuous dynamics is defined by linear differential inequalities of the form $A\dot{x} \sim b$ where $\sim \in \{\leq, \geq\}$, A is a constant matrix and b is a constant vector.

Approaches that incorporate temporal logic in control include an approach based on mixed integer linear programming [58]. Reference [68] introduced LTLC, an extension of linear temporal logic for specifying properties of discrete-time linear systems, and described LTLC model checking that allows a sequence of control inputs to be automatically computed such that a complex control objective expressed in LTLC is satisfied.

Digital design synthesis and hybrid system theory has been integrated to allow automatic synthesis of provably correct embedded control software for continuous systems. Such integration is enabled by the development of language equivalence and bisimulation notions, which allows abstraction of the continuous component of the system to a purely discrete model while preserving all the desired properties [6]. This subsequently provides a hierarchical approach to system design. In particular, a two-layer design is common and widely used in the area of planning and control [65, 25, 64, 33, 113, 40]. In the first layer, a discrete planner plans, in the abstracted discrete domain, a set of transitions of the system to ensure the satisfaction of the desired properties, taking into account all the possible behaviors of the environment. This abstract plan is then continuously implemented by a continuous controller in the second layer. Simulations/bisimulations provide the proof that the continuous

execution preserves the desired properties. The planner can be automatically synthesized using the digital design synthesis tool while the controller can be automatically generated using, for example, an optimization-based or other control-theoretic approach.

1.3 Thesis Overview and Contributions

The objective of this thesis is to develop a framework for systematic design and analysis of embedded control systems to provide a formal, mathematical guarantee of the correctness of such a system with respect to its desired properties. The systems of our particular interest are those with both the low-level (continuous) dynamics associated with the physical hardware and the high-level (discrete) logics that govern the overall behavior of the systems. Design and analysis of these systems thus require integration of reasoning about discrete and continuous behaviors within a single framework.

The research presented here consists of three key components—*specification*, *design* and *verification*. Specification refers to a precise description of both the system and its desired properties. This precise description of the system, however, does not need to capture all the details of the actual implementation itself. To simplify the analysis of the system, one may want to capture only the important aspects and abstract the actual implementation in this description.

Verification is the process of checking the correctness of the system. Here, correctness is only defined relative to the desired properties. Hence, specifications of both the system and its desired properties are essential in this process. It is well known that verifying the correctness of complex systems such as autonomous vehicles can be very difficult due to the interleaving between their continuous and their discrete components as previously discussed. Although much work has been done in this domain, verification of such systems remains a time consuming process and requires some level of expertise.

To complement the verification efforts, there has been a growing interest in automatic design of embedded control software that provides a formal guarantee of system correctness. This avenue of research is appealing and promising. Once it is brought to practicality, this type of automatic design can potentially reduce the time and cost of the system development cycle as it helps reduce the number of iterations between redesigning the system and verifying the new design.

The planner-controller subsystem of Alice and the autonomous urban driving problem of the DARPA Urban Challenge are considered as case studies throughout the thesis. Linear temporal logic is used as the main specification language for describing desired properties. A detailed description of the planner-controller subsystem of Alice and an overview of LTL are provided in Chapter 2 as the background for later chapters.

This thesis has two main parts. The first part focuses on the verification aspect while the second part focuses on the design aspect. Specification is mentioned in both parts as a key requirement that enables systematic verification and design. The original contributions of this work cover both theoretical and application aspects as outlined below.

Part I: Verification

Chapters 3–5 focus on the verification aspect of this work. First, Chapter 3 summarizes related work on system verification. A brief overview of approaches from computer science and control to system verification is provided. In particular, formal verification based on model checking and theorem proving is discussed. Reachability analysis and Lyapunov-type approaches which have been proved successful in verifying safety of control systems are presented. This chapter ends with verification of hybrid systems, a formalism that has been utilized to handle both discrete and continuous behaviors. The limitations of existing approaches are identified.

The main contribution of Chapter 4 is in the applications of existing verification approaches to Alice. First, it describes the Canonical Software Architecture that has been implemented to facilitate the coordination of different components in the planner-controller subsystem of Alice. This description is followed by two verification case studies. The first case study illustrates the use of model checking to prove the correctness of the finite state machine implemented in Gcdrive to handle multiple concurrent commands. The second case study demonstrates the provably correct decomposition of the desired system properties into components' properties based on the structure imposed by the Canonical Software Architecture.

Chapter 5 contains the main theoretical contributions of the verification part of the thesis. First, we introduce Periodically Controlled Hybrid Automata (PCHA), a class of hybrid automata for modular specification of embedded control systems. In a PCHA, *control* actions that change the control input to the plant occur roughly periodically, while

other actions that update the state of the controller may occur in the interim, changing the set-point of the system. Such actions could model, for example, sensor updates and information received from higher-level planning modules that change the set-point of the controller. Based on periodicity and subtangential conditions, a new sufficient condition for verifying invariant properties of PCHAs is presented. In principal, for PCHAs with polynomial continuous vector fields, it is possible to check these conditions automatically using, for example, quantifier elimination or sum of squares relaxations. We examine the feasibility of this automatic approach on a small example. The proposed technique is also used to manually verify safety and progress properties of a fairly complex planner-controller subsystem of Alice. Geometric properties of planner-generated paths are derived that guarantee that such paths can be safely followed by the controller. The material presented in this chapter has been reported in [118, 119, 120].

Part II: Design

Chapters 6–7 focus on the design aspect of this work. Specifically, we propose an approach for automatically synthesizing embedded control software that ensures system correctness with respect to its desired properties regardless of the environment in which the system operates. The material presented in this part of the thesis has been published in [122, 123, 124].

Chapter 6 summarizes related work on automatic synthesis of embedded control software, including background on automatic design of digital and control systems. A common approach to automatic synthesis of embedded control software that provably satisfies a given LTL property is to construct a finite transition system that serves as an abstract model of the physical system (which typically has infinitely many states) and synthesize a strategy, represented by a finite state automaton, satisfying the desired properties based on the abstract model. This leads to a hierarchical, two-layer design with a discrete planner computing a strategy based on the abstract model and a continuous controller computing a control signal based on the physical model to continuously implement the strategy. Simulations/bisimulations [6] provide the proof that the continuous execution preserves the desired properties. To increase the robustness of the system against the effects of direct, external disturbances and a mismatch between the actual system and its model, we extend the hierarchical approach to account for exogenous disturbances. Specifically, the contribu-

tion of this part of the thesis is to consider a discrete-time linear time-invariant state space model with exogenous disturbances and provide an approach to automatically compute a finite state abstraction for such a model.

The main practical limitation of the hierarchical approach is the well-known state explosion problem inherent in LTL synthesis. To alleviate this problem, Chapter 7 proposes a receding horizon framework that effectively reduces the synthesis problem into a set of smaller problems while preserving the desired system-level temporal properties. An implementation of the proposed framework leads to a hierarchical, modular design with a goal generator, a trajectory planner and a continuous controller. The goal generator essentially reduces the trajectory generation problem into a sequence of smaller problems of short horizon while preserving the desired system-level temporal properties. Subsequently, in each iteration, the trajectory planner solves the corresponding short-horizon problem with the currently observed state as the initial state and generates a feasible trajectory to be implemented by the continuous controller. Based on the simulation property, we show that the composition of the goal generator, trajectory planner and continuous controller and the corresponding receding horizon framework guarantee the correctness of the system. The effectiveness of this framework is demonstrated through an example of an autonomous vehicle navigating an urban environment.

Chapter 2

Background

This chapter provides background for later chapters. This includes a detailed description of the planner-controller subsystem of Alice and an overview of linear temporal logic.

2.1 Alice: An Autonomous Vehicle

Alice was equipped with 25 CPUs and utilized a networked control system architecture to provide high performance and modular design. The embedded control component of Alice is shown in Figure 2.1. This hierarchical control architecture comprises the following modules [18, 30, 121]:

Mission Planner computes the route, i.e., a sequence of roads the vehicle has to navigate in order to cross a given sequence of checkpoints. A sequence of checkpoints was provided by DARPA approximately 5 minutes before the start of each run. Mission Planner is also capable of recomputing the route when the response from Traffic Planner indicates that the previously computed route cannot be navigated successfully. This type of failure occurs, for example, when the road is blocked.

Traffic Planner makes decisions to guide Alice at a high level. Specifically, based on the traffic rules and the current environment, it determines how Alice should navigate the Mission Planner generated route, that is, whether Alice should stay in the travel lane or perform a passing maneuver, whether it should go or stop and whether it is allowed to reverse. Traffic Planner also implements an obstacle clearance requirement. In addition, it is responsible for intersection handling (e.g., keeping track of whether it is Alice's turn to go through an intersection). Based on these decisions, it sets up

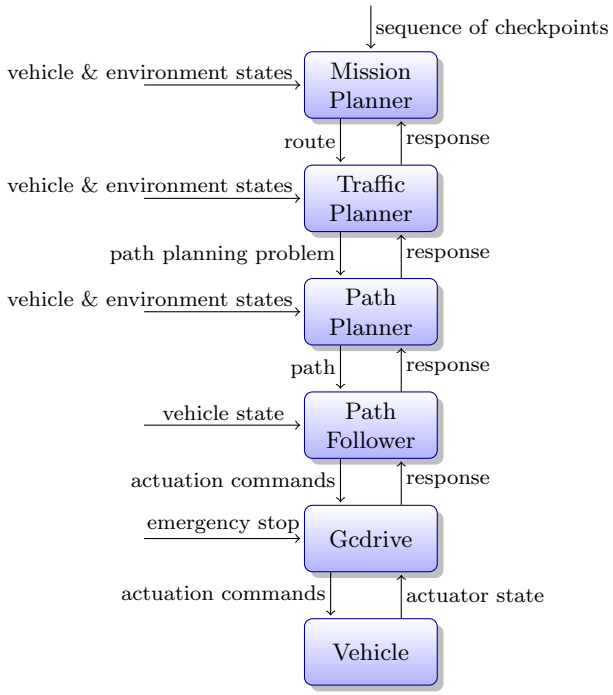


Figure 2.1: The embedded control component of Alice.

the constraints for the path planning problem.

The logic implemented in Traffic Planner can be described by a two-level finite state machine (FSM). First, the high-level mode is determined based on the current status of the vehicle, the current status of Traffic Planner and the current position of the vehicle with respect to the road network. This high-level mode includes road region, zone region, off-road, intersection, U-turn, failed and paused. Each of the high-level modes can be further decomposed to completely specify the planning problem described by the drive state, the allowable maneuvers and the obstacle clearance requirement. The drive state includes DR (drive), BACKUP (reverse) and STO (stop for an obstacle). When the drive state is DR or STO, the allowable maneuvers are specified by the following modes: NP (no passing or reversing allowed), P (passing allowed but reversing not allowed) and PR (both passing and reversing allowed). The obstacle clearance modes include S (the nominal, or safe, mode), A (an aggressive mode) and B (a very aggressive, or bare, mode). When the obstacle clearance mode is A or B, both passing and reversing maneuvers are allowed by default. As an example, the finite state machine associated with the road region mode consists of 11 states and 25 transitions as shown in Figure 2.2.

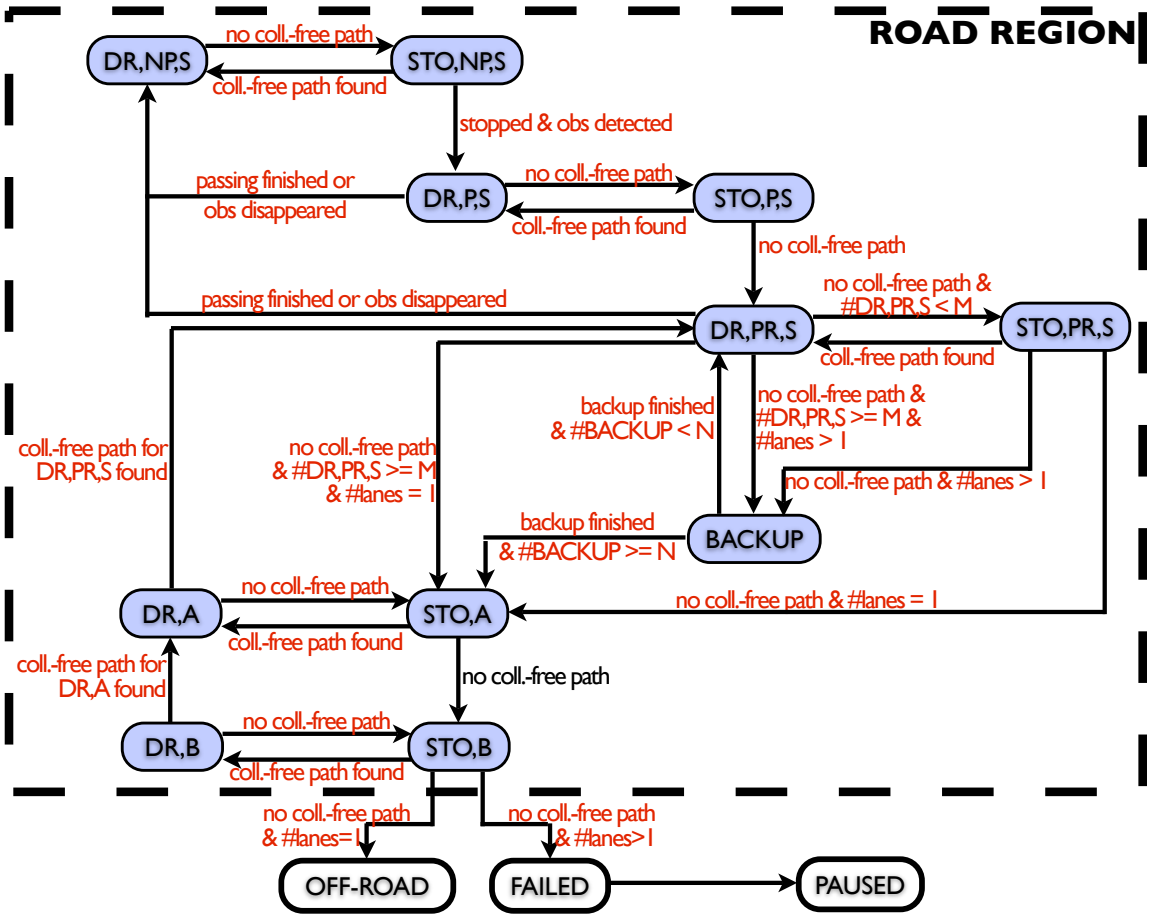


Figure 2.2: Traffic Planner FSM for the road region mode.

Path Planner generates a path that satisfies the constraints determined by Traffic Planner. Since Alice needs to operate in both structured and unstructured regions, three types of path planner have been implemented to exploit the structure of the environment: (1) the *rail planner* (for structured regions such as roads and intersections), (2) the *off-road rail planner* (for unstructured regions such as obstacle fields and sparse waypoint regions), and (3) the *clothoid planner* (for parking lots and unstructured regions). All the maneuvers available to the *rail planner* are pre-computed; thus, the *rail planner* may be too constraining. To avoid a situation where Alice gets stuck in a structured region (e.g., when there is an obstacle between the predefined maneuvers), the *off-road rail planner* or the *clothoid planner* may also be used in a structured region. This decision is made by Traffic Planner.

Path Follower computes control signals (acceleration and steering angle) that keep Alice

on the path generated by Path Planner [73]. Although these paths are guaranteed to be collision-free, since Alice cannot track them perfectly, it may get too close or even collide with an obstacle if the tracking error is too large. To address this issue, a reactive obstacle avoidance (ROA) component was implemented. The ROA component can override the acceleration command to rapidly stop the vehicle if the deviation from the planned path is too large or the projected position of Alice overlaps with an obstacle. The projection distance depends on the velocity of Alice. Path Follower will report failure to Path Planner if the ROA is triggered, in which case Path Planner will recompute the path.

Gcdrive is the overall driving software for Alice. It receives actuation commands from Path Follower, determines if they can be executed and, if so, sends the appropriate commands to the actuators. Gcdrive also performs checking on the health and operational state of the actuators, resets the actuators that fail, and broadcasts the actuator state. Also included in the role of Gcdrive is the implementation of physical protections for the hardware to prevent the vehicle from hurting itself. This includes three functions: (1) limiting the steering rate at low speeds, (2) preventing shifting from occurring while the vehicle is moving, and (3) transitioning to the paused mode in which the brakes are depressed and commands to any actuator are rejected when any of the critical actuators such as steering and brake fail. Furthermore, Gcdrive implements the emergency stop functionality for Alice and stops the vehicle when an externally produced emergency stop command is received.

A Canonical Software Architecture (CSA) has been developed to support a hierarchical decomposition and separation of functionality in this planner-controller subsystem, while maintaining communication and contingency management. More details on CSA can be found in Chapter 4.

Note that this planner-controller subsystem, including the complicated finite state machines implemented in Traffic Planner, was designed and implemented completely by hand in an ad-hoc manner. Furthermore, it was validated only through simulations and tests. Hence, there was absolutely no formal guarantee that the system would work as desired. This thesis develops methods and tools for systematic design and analysis of embedded control system such as Alice and illustrates their applications to the autonomous urban driving

problem. Linear temporal logic provides a precise mathematical language for describing desired system properties.

2.2 Linear Temporal Logic

Temporal logic is a branch of logic that implicitly incorporates temporal aspects and can be used to reason about a time line [8, 32, 50, 82]. Its use as a specification language was introduced by Pnueli [100]. Since then, temporal logic has been demonstrated to be an appropriate specification formalism for reasoning about various kinds of systems, especially those of concurrent programs. It has been utilized to formally specify and verify behavioral properties in various applications [23, 101, 37, 72, 48, 15, 112, 35, 54, 19, 55].

In this thesis, we consider a version of temporal logic, namely linear temporal logic (LTL), which is particularly suitable for describing properties of software systems. Before describing LTL, we need to define an atomic proposition, which is LTL's main building block. An atomic proposition can be defined based on a variable structure of the system as follows.

Definition 2.2.1. A system consists of a set V of variables. The *domain* of V, denoted by $dom(V)$, is the set of valuations of V. A *state* of the system is an element $v \in dom(V)$.

Definition 2.2.2. An *atomic proposition* is a statement on system variables v that has a unique truth value (*True* or *False*) for a given value of v. Let $v \in dom(V)$ be a state of the system and p be an atomic proposition. We write $v \Vdash p$ if p is *True* at the state v. Otherwise, we write $v \nVdash p$.

In this language, an *execution* of a system is described by an infinite sequence of its states. Specifically, for a discrete-time system whose state is only evaluated at time $t \in \{0, 1, \ldots\}$, its execution σ can be written as $\sigma = v_0 v_1 v_2 \ldots$ where for each $t \geq 0$, $v_t \in dom(V)$ is the state of the system at time t.

LTL has two kinds of operators: logical connectives and temporal modal operators. The logic connectives are those used in propositional logic: *negation* ($\neg$), *disjunction* ($\vee$), *conjunction* ($\wedge$) and *material implication* ($\Longrightarrow$). The temporal modal operators include *next* ($\bigcirc$), *always* ($\Box$), *eventually* ($\Diamond$) and *until* ($\mathcal{U}$).

An LTL formula is defined inductively as follows:

(1) any atomic proposition p is an LTL formula; and

(2) given LTL formulas φ and ψ, $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ and $\varphi \,\mathcal{U}\, \psi$ are also LTL formulas.

Other operators can be defined as follows:

- $\varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \neg\psi)$,
- $\varphi \Longrightarrow \psi \triangleq \neg\varphi \vee \psi$,
- $\Diamond\varphi \triangleq \mathit{True}\,\mathcal{U}\,\varphi$, and
- $\Box\varphi \triangleq \neg\Diamond\neg\varphi$.

A *propositional formula* is one that does not include temporal operators. Given a set of LTL formulas $\varphi_1, \ldots, \varphi_n$, their *Boolean combination* is an LTL formula formed by joining $\varphi_1, \ldots, \varphi_n$ with logical connectives.

Semantics of LTL: An LTL formula is interpreted over an infinite sequence of states. Given an execution $\sigma = v_0 v_1 v_2 \ldots$ and an LTL formula φ, we say that φ *holds at position* $i \geq 0$ of σ, written $v_i \vDash \varphi$, if and only if (iff) φ holds for the remainder of the execution σ starting at position i. The semantics of LTL is defined inductively as follows:

(a) For an atomic proposition p, $v_i \vDash p$ iff $v_i \Vdash p$;

(b) $v_i \vDash \neg\varphi$ iff $v_i \nvDash \varphi$;

(c) $v_i \vDash \varphi \vee \psi$ iff $v_i \vDash \varphi$ or $v_i \vDash \psi$;

(d) $v_i \vDash \bigcirc\varphi$ iff $v_{i+1} \vDash \varphi$; and

(e) $v_i \vDash \varphi \,\mathcal{U}\, \psi$ iff there exists $j \geq i$ such that $v_j \vDash \psi$ and $\forall k \in [i, j), v_k \vDash \varphi$.

Based on this definition, $\bigcirc\varphi$ holds at position i of σ iff φ holds at the next state v_{i+1}, $\Box\varphi$ holds at position i iff φ holds at every position in σ starting at position i, and $\Diamond\varphi$ holds at position i iff φ holds at some position $j \geq i$ in σ.

Definition 2.2.3. An execution $\sigma = v_0 v_1 v_2 \ldots$ *satisfies* φ, denoted by $\sigma \vDash \varphi$, if $v_0 \vDash \varphi$.

Definition 2.2.4. Let Σ be the set of all executions of a system. The system is said to be *correct* with respect to its specification φ, written $\Sigma \vDash \varphi$, if all its executions satisfy φ, that is, $(\Sigma \vDash \varphi)$ iff $\big(\forall\sigma, (\sigma \in \Sigma) \implies (\sigma \vDash \varphi)\big)$.

Examples of LTL formulas: Given propositional formulas p and q, important and widely used properties can be defined in terms of their corresponding LTL formulas as follows.

(a) **Safety** (invariance): A safety formula is of the form $\Box p$, which asserts that the property p remains invariantly true throughout an execution. Typically, a safety property ensures that nothing bad happens. A classic example of safety property frequently used in the robot motion planning domain is obstacle avoidance.

(b) **Guarantee** (reachability): A guarantee formula is of the form $\Diamond p$, which guarantees that the property p becomes true at least once in an execution. Reaching a goal state is an example of a guarantee property.

(c) **Obligation**: An obligation formula is a disjunction of safety and guarantee formulas, $\Box p \vee \Diamond q$. It can be shown that any safety and progress property can be expressed using an obligation formula. (By letting $q \equiv \mathit{False}$, we obtain a safety formula and by letting $p \equiv \mathit{False}$, we obtain a guarantee formula.)

(d) **Progress** (recurrence): A progress formula is of the form $\Box\Diamond p$, which essentially states that the property p holds infinitely often in an execution. As the name suggests, a progress property typically ensures that the system makes progress throughout an execution.

(e) **Response**: A response formula is of the form $\Box(p \implies \Diamond q)$, which states that following any point in an execution where the property p is true, there exists a point where the property q is true. A response property can be used, for example, to describe how the system should react to changes in the operating conditions.

(f) **Stability** (persistence): A stability formula is of the form $\Diamond \Box\, p$, which asserts that there is a point in an execution where the property p becomes invariantly true for the remainder of the execution. This definition corresponds to the definition of stability in the controls domain since it ensures that eventually, the system converges to a desired operating point and remains there for the remainder of the execution.

Example 2.2.1. Consider a robot motion planning problem where the robot is moving in an environment that is partitioned into six regions as shown in Figure 2.3.

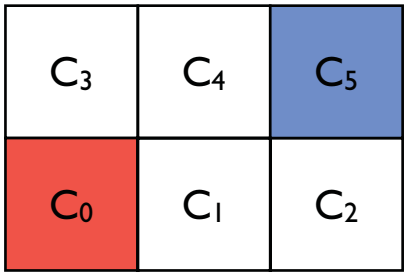


Figure 2.3: The robot environment of Example 2.2.1.

Let s represent the position of the robot and $C_0, \ldots, C_5$ represent the polygonal regions in the robot environment. Suppose the robot receives an externally triggered PARK signal. Consider the following desired behaviors.

(a) Visit region C_5 infinitely often.

(b) Eventually go to region C_0 when a PARK signal is received.

Assuming that infinitely often, a PARK signal is not received, the desired properties of the system can be expressed in LTL as

$$\Box\Diamond(\neg park) \implies \big(\Box\Diamond(s \in C_5) \wedge \Box(park \implies \Diamond(s \in C_0))\big).$$

Here, *park* is a boolean variable that indicates whether a PARK signal is received.

Remark 2.2.1. Properties typically studied in the control and hybrid systems domains are safety (usually in the form of constraints on the system state) and stability (i.e., convergence to an equilibrium or a desired state). LTL thus offers extensions to properties that can be expressed. Not only can it express a more general class of properties, but it also allows more general safety and stability properties than constraints on the system state or convergence to an equilibrium since p in $\Box p$ and $\Diamond\Box p$ can be any propositional formula.

Chapter 3

System Verification

System verification is the process of checking that a system meets its requirements. The requirements specify the allowed behaviors and formalize the desired properties of the system. This chapter reviews existing approaches to system verification that provides a formal guarantee that the system satisfies the desired properties. These approaches rely on a mathematical model of the system and analyze the correctness of the model, instead of the actual implementation itself, with respect to the requirements.

3.1 Formal Methods

Formal methods are mathematically-based techniques that provide a guarantee of system correctness and enable the developers to construct a system that operates reliably despite its complexity. This approach relies on constructing a mathematical model of the system and proving that the model respects the system requirements. There are two key elements involved in this process—*specification* and *verification*.

Formal specification is a precise mathematical representation of a system and its requirements. It helps remove ambiguities from the description of the expected behaviors of the system. Examples of such mathematical objects typically used in modeling systems include finite state machines, differential equations, time automata and hybrid automata.

Formal verification relies on a repertoire of proof techniques by which the correctness of the abstract mathematical model of the system relative to the requirements can be analyzed. It gives a formal guarantee that the desired properties hold over all possible executions of the system, provided that the actual execution of the system respects its model.

System correctness may be formally verified by hand in the style of mathematical proofs.

This approach is usually slow and often error prone. It also requires a high level of mathematical sophistication and expertise. As a result, there have been growing interests in automating or semi-automating such proofs. In this section, we summarize existing approaches for system verification from computer science and control theory.

3.1.1 Computer-Science Oriented Approaches

In computer science, automated proofs typically fall into two categories: algorithmic approaches and deductive approaches. The algorithmic approach relies on exhaustively exploring the state space to check that the desired properties of the system are satisfied. Model checking is a well-established technique that enables such exploration [24, 8]. The key requirement of this technique is that the description of the system and its requirements be formulated in some precise mathematical language. From the description of the system, all of its possible behaviors can be derived. In addition, all the invalid behaviors can be obtained from the system requirements. A model checker then checks whether an intersection of all the possible behaviors of the system and all the invalid behaviors is empty. It terminates with a yes/no answer and provides an error trace in case of a negative result. This technique is very attractive because it is automatic, fast and requires no human interaction. However, to achieve the decidability, model checking is limited to finite state systems. It also faces a combinatorial blow up of the state space, commonly known as the state explosion problem.

Various model checkers have been developed for different specification languages. TLC [125] is a model checker for specifications written in TLA+, which is a specification language based on Temporal Logic of Actions (TLA) [2, 70, 71]. TLA introduces new kinds of temporal assertions to traditional linear temporal logic to make it practical to describe a system by a single formula and to make the specifications simpler and easier to understand. SPIN, on the other hand, is a model checker for specifications written in Process Meta-Language (PROMELA) [49]. This language was influenced by Dijkstra, Hoare's CSP language and C. It emphasizes the modeling of process synchronization and coordination, not computation and is not meant to be analyzed manually. SPIN can be run in two modes—simulation and verification. The simulation mode performs random or iterative simulations of the modeled system's execution while the verification mode generates a C program that performs a fast exhaustive verification of the system state space. SPIN is mainly used for checking for dead-

locks, livelocks, unspecified receptions, unexecutable code, correctness of system invariants and non-progress execution cycles. It also supports the verification of linear time temporal constraints. SPIN has been used in many applications, especially in proving correctness of safety-critical software [44, 41]. Other popular model checkers include Symbolic Model Verifier (SMV) [84] and its successor NuSMV [22].

The deductive approach relies on using axioms and proof rules to prove the correctness of a system. The proofs are typically based on inductive invariants: If property φ holds at the initial stage of the system and all legal successors of every φ-state are φ states, then φ always holds. Theorem proving is a machinery that allows such proofs to be partially automated. This approach is not limited to finite state systems. However, it demands a skilled human interaction. An example of a commonly used theorem prover is Prototype Verication System (PVS) [94, 93, 52].

3.1.2 Control Oriented Approaches

Control systems are described by a set of differential or difference equations, e.g., $\dot{x}(t) = f(x(t), u(t))$ or $x[t+1] = f(x[t], u[t])$ where x represents the state of the system and u represents the control input to the plant. These systems have been a subject of research in the control community for many decades. Parallel to the studies of formal methods in computer science, control theorists have developed a methodology for verifying that such a continuous system remains within a certain set, namely, the safe set. The dual of this safety problem is the reachability problem that concerns proving the existence of a trajectory that starts from an initial set and reaches another given set. This kind of safety and reachability analysis is typically based on two main approaches: direct reachability analysis and Lyapunov-type methods.

The detailed examination of direct reachability analysis can be found in [87] and references therein. In summary, these approaches seek to compute either the set of all states that can be reached along trajectories that start within a given set of initial states (forward reachability), or the set of all states from which trajectories start such that a given set of target or final states can be reached (backward reachability). Lyapunov-type methods, on the other hand, do not require explicit computation of reachable sets. In addition, nonlinearity, uncertainty and constraints can be handled directly. The underlying idea is to search for a Lyapunov-type function that satisfies certain algebraic conditions. This func-

tion, together with the corresponding algebraic conditions, provides a certificate that all trajectories of the system starting from a given initial set remain within the safe set. An example of such function is a barrier certificate [103, 104, 105]. The existence of a feasible solution to the dual of this safety analysis problem provides a certificate for the existence of a trajectory from the initial set to the unsafe set [107]. This dual test can be formulated using the concept of convex duality and density functions. The roles of a barrier certificate and a density function in proving safety and reachability can also be interchanged so that a density function is used to prove safety while a barrier certificate is used to prove reachability. This approach can be extended to analyze the *eventuality* property: All trajectories starting from a given initial set eventually reach a given final set in a finite time. More details regarding this primal-dual approach can be found in [107].

A barrier certificate and a density function can be automatically constructed using sum of squares techniques in conjunction with semidefinite programming, provided that the vector field is polynomial and the sets are semialgebraic (i.e., can be described by polynomial equalities and inequalities) [106]. Such construction relies on using the generalized S-procedure [116] (a special case of the Positivstellensatz) and sum of squares relaxations to translate the set containment constraints to a sum of squares optimization problem. The applicability of this sum of squares technique can be extended to certain non-polynomial systems by using a recasting procedure [95].

S-Procedure Given functions $f_0(x) = x^T F_0 x$ and $f_1(x) = x^T F_1 x$ where $x \in \mathbb{R}^n, n \in \mathbb{N}$ and F_0 and F_1 are $n \times n$ symmetric matrices, if there exists $\alpha \geq 0$ such that

$$f_0(x) - \alpha f_1(x) \geq 0, \forall x \in \mathbb{R}^n,$$

then the following equivalent statements hold:

(i) $f_0(x) \geq 0$ for all x such that $f_1(x) \geq 0$,

(ii) $\{x \in \mathbb{R}^n \mid f_1(x) \geq 0\} \subseteq \{x \in \mathbb{R}^n \mid f_0(x) \geq 0\}$,

(iii) $\{x \in \mathbb{R}^n \mid f_1(x) \geq 0 \text{ and } f_0(x) < 0\} = \varnothing$,

(iv) $\forall x \in \mathbb{R}^n, f_1(x) \geq 0 \implies f_0(x) \geq 0$.

Generalized S-Procedure Given polynomials $f_0, f_1, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$, if there exist

positive semidefinite polynomials $r_1, \ldots, r_m$ such that

$$f_0(x) - \sum_{i=1}^{m} r_i(x) f_i(x) \geq 0, \forall x \in \mathbb{R}^n,$$

then $\{x \in \mathbb{R}^n \mid f_i(x) \geq 0, \forall i \in \{1, \ldots, m\} \subseteq \{x \in \mathbb{R}^n \mid f_0(x) \geq 0\}$.

Sum of Squares Relaxations The sum of squares relaxations simply replace all the non-negativity constraints obtained from the generalized S-procedure with the constraint that they are sum of squares polynomials, i.e., polynomials that can be represented as a sum of squares of finitely many polynomials. This sum of squares constraint is more restrictive but more computationally tractable than the non-negativity constraint.

A constraint-based technique [43] for computing inductive invariants can be thought of as a variant of the Lyapunov-type methods. Similar to Lyapunov-type methods, it requires a template for the unknown inductive invariant $\mathcal{I}$. With this template, sufficient conditions for an invariance of $\mathcal{I}$ can be expressed as satisfiability of a $\exists\forall$ formula over reals, where the existential quantification is over the template variables and the universal quantifier is over the state variables. The universal quantifier can then be eliminated using a special case of the generalized S-procedure known as Farkas Lemma. The following variant of Farkas Lemma is presented in [43] and duplicated here for convenience.

Farkas Lemma Let J and K be finite sets. For each $j \in J$ and $k \in K$, let p_j and r_k be polynomials of bounded degrees over the state variables. The formula

$$\bigwedge_{j \in J} p_j > 0 \wedge \bigwedge_{k \in K} r_k \geq 0$$

is unsatisfiable over reals if there exist non-negative constants μ, $\mu_j, j \in J$ and $\nu_k, k \in K$ such that

$$\mu + \sum_{j \in J} \mu_j p_j + \sum_{k \in K} \nu_k r_k = 0,$$

and at least one of the μ_j or μ is strictly positive.

We refer the reader to [43] for a detailed discussion on converting an arbitrary universally quantified arithmetic formula into an existentially quantified formula using this variant of Farkas Lemma. The resulting satisfiability checking can then be converted to a bit-vector

satisfiability problem. A solution, in a bounded range, of this satisfiability problem can be searched for using the bit-vector decision procedure of a satisfiability modulo theory (SMT) solver.

3.2 Hybrid System Verification

3.2.1 Hybrid Automata

The hybrid system formalism [3, 4, 38] provides a rich mathematical language for specifying embedded systems where computing and control components interact with physical processes. In this framework, a hybrid system is characterized by (a) a set of continuous states, (b) a finite set of locations or discrete states, (c) the set of initial states, (d) an invariant set associated with each location, (e) a set of vector fields, and (f) a set of discrete transitions between two locations. A guard set and a reset map can be derived from the set of discrete transitions between two locations. Reference [38] adds continuous and discrete input sets to this description.

Reachability analysis, Lyapunov-type methods and the constraint-based approach described in Section 3.1.2 can be applied to verify safety properties of systems modeled in this hybrid automata framework. Forward reachability analysis has been implemented in model checkers for hybrid systems such as HyTech [45] and PHAVer [34]. Backward reachability has been applied, for example, in [10] to analyze safety of aircraft autoland systems.

To explicitly capture the concurrency and asynchronous characteristics of distributed algorithms and distributed systems, a family of system modeling frameworks based on interacting infinite-state machines was introduced by Lynch et al. [80, 59, 76, 79, 88]. These frameworks are based on input/output (I/O) automata model and come in many flavors, e.g., basic asynchronous I/O automata, timed I/O automata and hybrid I/O automata. Properties of these automata can be proved by hand or with the assistance of theorem provers. Algorithms and proofs described in this I/O-automata-style modeling framework can be found in [78]. Of our particular interest is hybrid I/O automata (HIOA), which add a set of trajectories to describe the evolution of system state over intervals of time.

In the HIOA framework, the discrete behavior of a system is described by a set of discrete state transitions (actions). The continuous behavior is described by a set of trajectories that specify the behavior of the variables of an automaton with time. An execution of HIOA is

described by a finite or infinite alternating sequence of trajectories and actions. A safety or invariant property $\mathcal{I}$ of an HIOA $\mathcal{A}$ is typically deduced by finding a stronger inductive invariant $\mathcal{I}' \subseteq \mathcal{I}$ and checking, through case analysis, that all the actions and trajectories of $\mathcal{A}$ preserve $\mathcal{I}'$. Specifically, the set $\mathcal{I}$ of states is an invariant of an HIOA $\mathcal{A}$ if

- (Start condition) any initial state of the system $x_0 \in \mathcal{I}$,
- (Transition condition) For any action a, if $x \xrightarrow{a} x'$ and $x \in \mathcal{I}$, then $x' \in \mathcal{I}$,
- (Trajectory condition) For any trajectory τ, if the first state of τ, $\tau.\mathsf{fstate} \in \mathcal{I}$, then the last state of τ, $\tau.\mathsf{lstate} \in \mathcal{I}$.

This technique allows the reasoning about the actions and the trajectories to be decoupled. Theorem prover strategies (PVS programs) that partially automate construction of such proofs can be found, for example, in [88].

HIOA has been applied, for example, to prove the correctness of a vehicle deceleration maneuver part of an automated transportation system [77] and to verify the safety of the automated highway system of the California PATH project and the Traffic Alert and Collision Avoidance System (TCAS) that is used by aircraft to avoid midair collisions [28, 74]. More details of HIOA can be found in Chapter 5 where we introduce its subclass that is suitable for modeling embedded systems with periodic sensing and actuation.

An important aspect of these hybrid and I/O automata frameworks is that they allow composition of automata to make larger ones. This enables modular specification of hybrid and distributed systems as each component of the system can be individually specified. These component specifications can then be composed to describe the whole system.

3.2.2 Other Modeling Frameworks

Computational and Control Language (CCL) is another formalism for expressing systems with a tight link between the computing and the control components [62]. It was influenced by the UNITY formalism [20] and was originally developed for expressing cooperative control systems. The main idea of CCL is to divide an execution of a system into *epochs* during which each process is executed exactly once. The order that the processes are executed in one epoch, however, is arbitrary. This allows us to take into account the small-time interleaving that may occur between processors executing at essentially the same rate. It

is a stronger assumption of the fairness constraint adopted by I/O automata and other distributed system modeling frameworks that only assume that each process gets executed *eventually*. Analysis of a CCL program is typically based on inductive techniques. Summary of distinctions between CCL and other modeling frameworks can be found in [63]. CCL has been applied to analyze a simplified version of a capture-the-flag system [62, 26].

3.2.3 Limitations

The algorithmic verification problem for hybrid systems with general dynamics is known to be computationally hard [46]. Restricted subclasses that are amenable to algorithmic analysis have been identified, such as the rectangular-initialized hybrid automata [46], o-minimal hybrid automata [69], and more recently planar [102] and STORMED [117] hybrid automata. Although these restricted subclasses improve our understanding of the decidability frontier for hybrid systems, the imposed restrictions are artificial, i.e., they are not representative of structures that arise in real-world systems. For example, initialized hybrid automata require the continuous state of the system to be reset every time the automaton enters a new mode. STORMED hybrid automata, on the other hand, require all the vector fields and reset maps to be monotonic with respect to a certain fixed direction.

On the other hand, deductive verification of hybrid systems is typically very time consuming and can be error prone due to the required human interaction. Although some proofs can be partially automated using theorem provers, significant human interaction remains necessary, especially for models with non-linear dynamics. More automation is needed in order to make this approach more practical and attractive.

Chapter 4

Applications of Formal Methods to Alice

This chapter illustrates the applications of formal methods presented in Chapter 3 to the embedded control component of Alice (Figures 2.1). The case studies presented in this chapter focus on the two low-level modules, namely Path Follower and Gcdrive.

4.1 Overview

As described in Section 2.1, the control subsystem of Alice consists of five software modules: Mission Planner, Traffic Planner, Path Planner, Path Follower and Gcdrive. These modules are responsible for reasoning at different levels of abstraction and are executed concurrently in the onboard processors. A Canonical Software Architecture is used to support this hierarchical decomposition and separation of functionality, while maintaining communication and contingency management.

The case studies considered in this chapter focus on the two lower-level modules—Path Follower (or Follower, for short) and Gcdrive. Path Follower receives a planned path and computes commands to throttle, brake and transmission that enable Alice to track this path. Gcdrive receives actuation commands from Path Follower and an externally produced emergency stop (estop) command from DARPA. It then performs checking to make sure that the commands are reasonable. For example, gear can be changed only when Alice is stopped. Based on the received commands and actuators' states, Gcdrive computes appropriate commands to all the actuators.

This chapter is organized as follows. Section 4.2 describes the Canonical Software Ar-

chitecture. Section 4.3 illustrates the use of model checking to prove the correctness of the finite state machine implemented in Gcdrive to handle concurrent commands from Follower and DARPA. Section 4.4 describes the use of the Canonical Software Architecture in systematically decomposing system-level requirements into a set of component-level requirements. It is followed by a case study where we mathematically prove that the component-level requirements of Follower and Gcdrive are sufficient to ensure that certain system-level requirements are satisfied. Section 4.5 concludes the chapter.

4.2 Canonical Software Architecture

One of the main issues with distributed systems is synchronization. A Canonical Software Architecture (CSA)[1] has been developed in order to address this issue. This architecture builds on the state analysis framework developed at the Jet Propulsion Laboratory (JPL) and takes the approach of clearly delineating state estimation and control determination as described in [31, 108, 9, 51].

In CSA, we can think of the entire system as being broken up into "modules," each of which has a separate, dedicated function. There are two types of modules in CSA: estimation modules and control modules. An estimation module estimates the system state or provides an abstraction of the system state for the corresponding control module(s). A control module gets inputs, performs actions based on the inputs, and delivers outputs. This section only discusses CSA control modules, which are the focus of this thesis. For modularity, each software module in the control subsystem may be broken down into multiple CSA modules. An example of the control subsystem in CSA we have implemented on Alice is shown in Figure 4.1.

The CSA imposes a structure on both the interface between control modules and the major operations that happen within a control module. As shown in Figure 4.2, inputs to a CSA control module are restricted to be one of the followings: state information, directives/instructions (from other modules wishing to control this module) and responses/status reports (from other modules receiving instructions from this module). The outputs can only be either status reports from this module or directives/instructions for other control modules.

[1]The idea of this architecture came from discussions with Robert D. Rasmussen and Michel D. Ingham from the Jet Propulsion Laboratory.

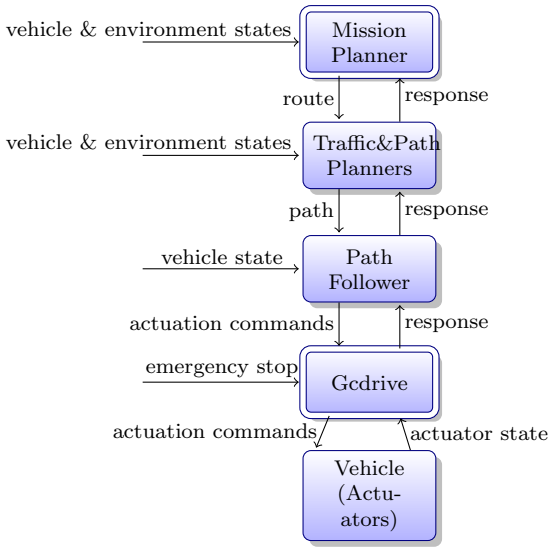


Figure 4.1: The planner-controller subsystem of Alice in the Canonical Software Architecture. Boxes with double lined borders are subsystems that will be broken up into multiple CSA modules.

For each directive that a control module is designed to accept, the following must be (implicitly or explicitly) specified:

(a) Entry (initial) condition: defines what must be true before starting to execute this directive; could result in rejection if not readily achievable;

(b) Exit (end) condition: defines what must be true to complete the execution of this directive; could result in rejection if not readily achievable; could result in failure if deadlines are not met;

(c) Rules: constraints, control objectives, etc. that must be satisfied during the execution of the directive; otherwise, failure is declared;

(d) Performance criteria: performance or other items to be optimized.

For each directive received, a response that indicates rejection, acceptance, failure or completion of the directive and the reason for rejection or failure must be reported to the source of the directive. Rejection or failure of a directive occurs when the entry or exit condition is not readily achievable, the deadlines are not met, or one of the constraints cannot be satisfied. It results in dropping the problem directive and all subsequent directives from that source until an acknowledgement of the failure or rejection is received. This directive-response mechanism allows CSA to support distributed goal and contingency management,

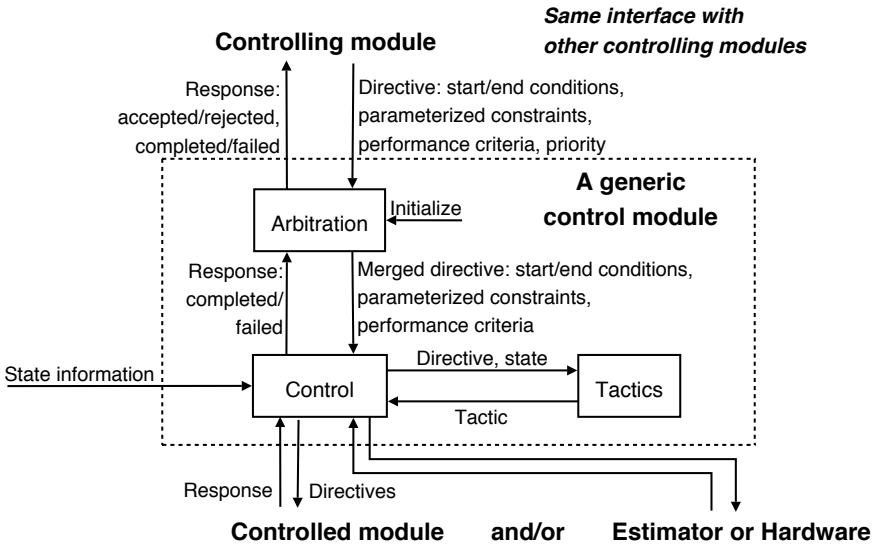


Figure 4.2: A generic control module in the Canonical Software Architecture.

an approach where each of the modules is responsible for handling the faults in its own domain and anything the module is unable to handle is propagated "up the chain" until the correct level has been reached to resolve the fault or conflict.

To separate communication requirements from the given module's core function requirements, the CSA decomposes a module into three components: *Arbitration*, *Control* and *Tactics*. *Arbitration* is responsible for (1) managing the overall behavior of the control module by issuing a merged directive, computed from all the received directives, to *Control*; and (2) reporting rejection, acceptance, failure and completion of a received directive to *Control* of the issuing control module. *Control* is responsible for (1) computing the output directives to the controlled module(s) or the commands to the hardware based on the merged directive, received responses and state information; and (2) reporting failure and completion of a merged directive to *Arbitration*. *Tactics* provides the core functionality of the control module and is responsible for providing the logic used by *Control* for computing output directives.

4.3 Verification of Gcdrive Finite State Machine

Gcdrive consists of five CSA modules: Actuation Interface, Acceleration Module, Steering Module, Transmission Module and Turn Signal Module. Acceleration Module, Steering Module, Transmission Module and Turn Signal Module are the interfaces to the correspond-

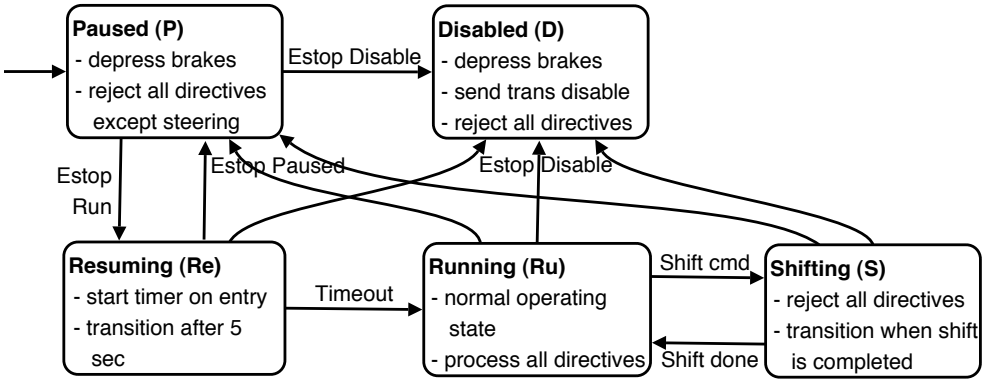


Figure 4.3: Finite state machine implemented in Actuation Interface.

ing actuators. The logic in Gcdrive is contained in Actuation Interface and can be described by a finite state machine. This example illustrates the use of model checking in proving the correctness of the implementation of this finite state machine.

Gcdrive takes independent commands from Path Follower and DARPA and sends appropriate commands to the actuators. Commands from Path Follower include control signals to throttle, brake and transmission. Commands from DARPA include estop pause, estop run and estop disable. An estop pause command should cause the vehicle to be brought quickly and safely to a rolling stop. An estop run command resumes the operation of the vehicle. An estop disable command is used to stop the vehicle and put it in the disable mode. A vehicle that is in the disable mode may not restart in response to an estop run command.

The finite state machine to handle these concurrent commands is shown in Figure 4.3. To prove the correctness of its implementation in Actuation Interface, we model Follower, Actuation Interface and DARPA (see Figure 4.4) in the SPIN model checker with the following global variables.

- $state \in$ {DISABLED (D), PAUSED (P), RUNNING (Ru), RESUMING (Re), SHIFTING (S)} is the state of the finite state machine as described in Figure 4.3.

- $estop \in$ {DISABLE (0), PAUSE (1), RUN (2)} is the emergency stop command sent by DARPA.

- $acc \in [-1, 1]$ and $acc\_cmd \in [-1, 1]$ are the acceleration commands sent from Actuation Interface to Acceleration Module and from Follower to Actuation Interface, respec-

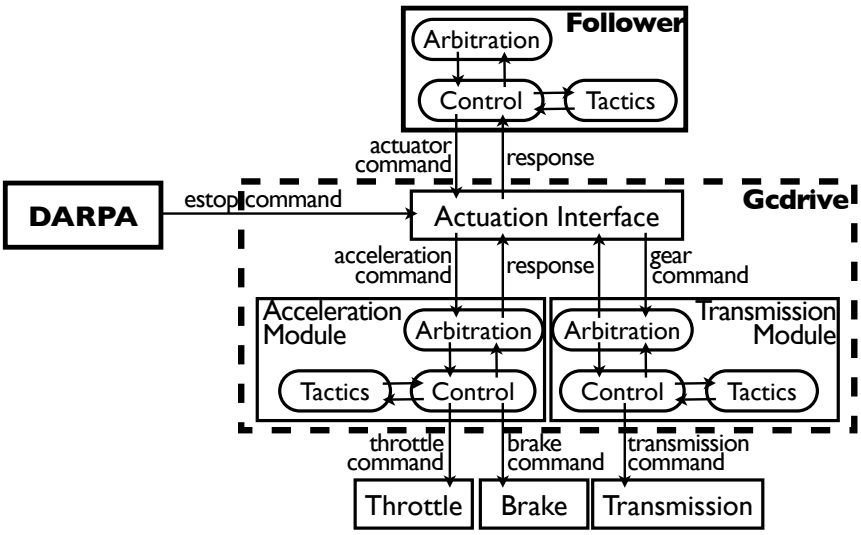


Figure 4.4: The components involved in the Gcdrive FSM example.

tively.

- *gear* $\in \{-1, 0, 1\}$ and *gear_cmd* $\in \{-1, 0, 1\}$ are the gear commands sent from Actuation Interface to Transmission Module and from Follower to Actuation Interface, respectively.

- *timer* $\in \{0, 1, 2, 3, 4, 5\}$ keeps track of the time after which the latest estop run command is received.

We make the following assumptions.

(a) Actuation Interface gets executed at least once each time an estop command is sent.

(b) Actuation Interface reads the current estop status at the beginning of each iteration. It then performs a computation based on this estop status for the rest of the iteration.

(c) All the estop commands are eventually received.

We introduce a global variable *enableEstop* in the PROMELA model to incorporate assumption (a). Assumption (b) is enforced using atomic sequences. Lastly, assumption (c) is enforced by letting the variable *estop* represent the estop command received by Gcdrive as well. With these assumptions, SPIN can verify the correctness of the system with respect to the following desired properties.

(1) If DARPA sends an estop disable command, Gcdrive state will eventually stay at DISABLED and Acceleration Module will eventually command full brake forever.

$$\Box\big((estop = \text{DISABLE}) \implies \Diamond\Box\,(state = \text{DISABLED} \wedge acc = -1)\big) \tag{4.1}$$

(2) If DARPA sends an estop pause command while the vehicle is not disabled, eventually Gcdrive state will be PAUSED.

$$\Box\big((estop = \text{PAUSE} \wedge state \neq \text{DISABLED}) \implies \Diamond(state = \text{PAUSED})\big) \tag{4.2}$$

(3) If DARPA sends an estop run command while the vehicle is not disabled, eventually Gcdrive state will be RUNNING or RESUMING or DARPA will send an estop disable or estop pause command.

$$\begin{aligned}\Box\big((estop = \text{RUN} \wedge state \neq \text{DISABLED}) \implies \\ \Diamond(state \in \{\text{RUNNING}, \text{RESUMING}\} \vee estop \neq \text{RUN})\big)\end{aligned} \tag{4.3}$$

(4) If the current state is RESUMING, eventually the state will be RUNNING or DARPA will send an estop disable or pause command.

$$\Box\big((state = \text{RESUMING}) \implies \Diamond(state = \text{RUNNING} \vee estop \in \{\text{DISABLE}, \text{PAUSE}\})\big) \tag{4.4}$$

(5) The vehicle is disabled only after it receives an estop disable command.

$$\big((state \neq \text{DISABLED})\,\mathcal{U}\,(estop = \text{DISABLE})\big) \vee \Box(state \neq \text{DISABLED}) \tag{4.5}$$

(6) Actuation Interface sends a full brake command to the Acceleration Module if the current state is DISABLED, PAUSED, RESUMING or SHIFTING. In addition, if the vehicle is disabled, then the gear is shifted to 0.

$$\begin{aligned}\Box(state \in \{\text{DISABLED, PAUSED, RESUMING, SHIFTING}\} \implies acc = -1) \wedge \\ \Box(state = \text{DISABLED} \implies gear = 0)\end{aligned} \tag{4.6}$$

(7) After receiving an estop pause command, the vehicle may resume the operation 5 seconds after an estop run command is received.

$$\Box\,(state = \text{RUN} \implies timer \geq 5) \tag{4.7}$$

The PROMELA models of the components involved in this example can be found in Appendix 4.A. Note that assumptions (a)–(c) are necessary for all the above desired properties to be satisfied. In fact, assumptions (a) and (b) were not realized until we model checked the early implementation of Gcdrive. Realizing that these assumptions needed to be enforced, we then modified the implementation of Alice by having Gcdrive store all the estop commands in a queue and process all these commands one by one. If an estop command is not stored but only sampled at the beginning of each iteration, an estop disable or pause command may not be handled appropriately. Consider, for example, the case where an estop pause command is sent while Actuation Interface is in the middle of an iteration and an estop run command is sent immediately after. In this case, Alice will not stop because the estop pause command is not processed, leading to an incorrect, unsafe behavior.

4.4 Composing Requirements

As described in Section 4.2, the inputs, outputs and major operations within a CSA control module have a well-defined structure. To facilitate the design and verification of the system, system requirements should be decomposed into the requirements for the *Arbitration* component and the requirements for the *Control* and *Tactics* components for each of the modules. The requirements for the *Arbitration* component specify the relationship between the received directives and the merged directives while the requirements for the *Control* and *Tactics* components specify the relationship between the merged directives, responses, state knowledge and the output directives as presented later in this section.

Besides module requirements, communication requirements such as bandwidth, packet drop, delay, etc also need to be specified. Modules' requirements and communiation requirements can then be composed either manually or with the assistance of tools such as theorem provers [36, 94] to verify that they are sufficient to ensure that the system requirements are satisfied.

We illustrate this requirement composition idea in the following case study where we prove that the CSA can ensure the state consistency between different software modules. The example considered here focuses on the two lower-level modules: Follower and Gcdrive (see Figure 4.1). Specifically, we want to prove that Follower, the module that commands a gear change, has the right knowledge about the gear Alice is currently in even though it does not talk to the actuator directly and sensors may fail. Otherwise, it will command full brake. This example involves six components—the *Control* of Follower, Actuation Interface, Transmission Module, Acceleration Module, the actuators and the network—as shown in Figure 4.4.

In this example, we are only interested in acceleration and transmission commands. The following variables are involved in this example as shown in Figure 4.5:

- $Trans_{f,s}$: transmission directive sent from Follower;
- $Trans_{f,r}$: transmission directive received by Actuation Interface;
- $Trans_{a,s}$: transmission directive sent from Actuation Interface;
- $Trans_{a,r}$: transmission directive received by Transmission Module;
- $Acc_{f,s}$: acceleration directive sent from Follower;
- $Acc_{f,r}$: acceleration directive received by Actuation Interface;
- $Acc_{a,s}$: acceleration directive sent from Actuation Interface;
- $Acc_{a,r}$: acceleration directive received by Acceleration Module;
- $TransResp_{f,s} \in \{\text{COMPLETED}, \text{FAILED}\}$: response sent from Actuation Interface;
- $TransResp_{f,r} \in \{\text{COMPLETED}, \text{FAILED}\}$: response received by Follower;
- $TransResp_{a,s} \in \{\text{COMPLETED}, \text{FAILED}\}$: response sent from Transmission Module;
- $TransResp_{a,r} \in \{\text{COMPLETED}, \text{FAILED}\}$: response received by Actuation Interface.

Each of these variables is represented by a finite sequence, whose n^{th} element represents its value in the n^{th} iteration, with the following operators:

- $Last(s)$: The last element of sequence s;

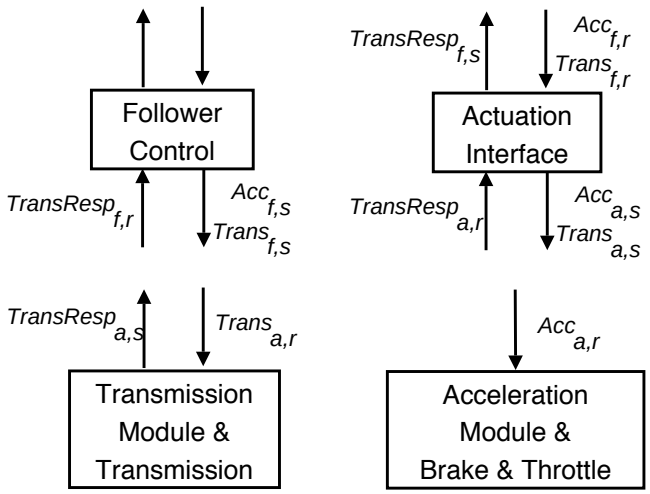


Figure 4.5: The variables involved in the CSA example.

- $Len(s)$: The length of sequence s;
- $s[n]$: The n^{th} element of sequence s.

Let $Trans$ be the actual gear and $Trans_f$ be the gear that Follower thinks Alice is in. Assume that when $Len(Trans_{f,s}) = 0$ (i.e., before any command is sent from Follower), $Trans_f = Trans$, we want to verify the following desired system-level properties:

(1) Follower has the right knowledge about the gear that Alice is currently in, or it commands full brake. Mathematically, this can be written as:

$$\Box\big((Len(TransResp_{f,r}) = Len(Trans_{f,s}) \wedge Last(TransResp_{f,r}) = \text{COMPLETED}) \implies Trans_f = Trans\big) \wedge \Box(Trans_f = Trans \vee Acc_{f,s} = -1). \tag{4.8}$$

(2) At infinitely many instants, Follower has the right knowledge about the gear that Alice is currently in, or a hardware failure (HWF) occurs:

$$\Box \Diamond (Trans_f = Trans \vee HWF). \tag{4.9}$$

We consider the following component-level properties:

A. **Transmission Module and transmission actuator**:

- The number of responses cannot be greater than the number of directives. This can be formalized by the following LTL formula:

$$\Box\,(Len(TransResp_{a,s}) \leq Len(Trans_{a,r})). \tag{4.10}$$

- For each of the directives that Transmission Module receives, a response will eventually be sent. If the gear is successfully changed, the completion of the directive will be reported. Otherwise, a hardware failure occurs and the failure will be reported. This property can be mathematically represented by the conjunction of the following three LTL formulas:

$$\begin{aligned}&\Box\big(n = Len(Trans_{a,r}) \implies \\ &\Diamond\big((Trans = Trans_{a,r}[n] \ \wedge\ TransResp_{a,s}[n] = \text{COMPLETED}) \\ &\vee\ (HWF \ \wedge\ TransResp_{a,s}[n] = \text{FAILED})\big)\big);\end{aligned} \tag{4.11}$$

$$\begin{aligned}&\Box\big(Last(TransResp_{a,s}) = \text{COMPLETED} \implies \\ &Trans = Trans_{a,r}[Len(TransResp_{a,s})]\big);\end{aligned} \tag{4.12}$$

$$\Box\,\big(Last(TransResp_{a,s}) = \text{FAILED} \implies HWF\big). \tag{4.13}$$

B. **Actuation Interface**: All the transmission directives and responses received are always sent (to Transmission Module and to Follower, respectively). This property can be described by the conjunction of the following LTL formulas:

$$\begin{aligned}&\Box\big(Len(Trans_{a,s}) = Len(Trans_{f,r}) \ \wedge \\ &\forall i \in \{1, \ldots, Len(Trans_{f,r})\} : Trans_{a,s}[i] = Trans_{f,r}[i]\big);\end{aligned} \tag{4.14}$$

$$\begin{aligned}&\Box\big(Len(TransResp_{f,s}) = Len(TransResp_{a,r}) \ \wedge \\ &\forall i \in \{1, \ldots, Len(TransResp_{a,r})\} : TransResp_{f,s}[i] = TransResp_{a,r}[i]\big).\end{aligned} \tag{4.15}$$

C. **Network**: All messages are eventually delivered. An example of this assumption for the transmission directive sent from Actuation Interface and received by Transmission

Module, formalized in LTL, is given by

$$\begin{aligned} &\Box(Len(Trans_{a,r}) \leq Len(Trans_{a,s})) \ \wedge \\ &\forall i \in \{1, \ldots, Len(Trans_{a,r})\} : Trans_{a,r}[i] = Trans_{a,s}[i]). \end{aligned} \tag{4.16}$$

D. **The *Control* of Follower**:

- If the response is not yet received, send a brake command:

$$\Box\left(Len(TransResp_{f,r}) \neq Len(Trans_{f,s}\right) \implies Acc_{f,s} = -1). \tag{4.17}$$

- If the last response indicates failure, send a brake command:

$$\Box\left(Last(TransResp_{f,r}) = \text{FAILED} \implies Acc_{f,s} = -1\right). \tag{4.18}$$

- Do not send a new directive until a response for the last directive is received:

$$\Box\left(Len(Trans_{f,s}) \leq Len(TransResp_{f,r}) + 1\right). \tag{4.19}$$

- Infinitely often, the number of the transmission directives is not greater than the number of the responses (i.e., once a response is received, Follower processes it before sending out another directive):

$$\Box\Diamond\left(Len(Trans_{f,s}) \leq Len(TransResp_{f,r})\right). \tag{4.20}$$

- If the last response indicates completion of the directive, Follower updates $Trans_f$ to the corresponding directive:

$$\begin{aligned} \Box\big(&Last(TransResp_{f,r}) = \text{COMPLETED} \implies \\ &Trans_f = Trans_{f,s}[Len(TransResp_{f,r})]\big). \end{aligned} \tag{4.21}$$

To prove the above component-level requirements are sufficient to ensure system-level requirements, we use the following lemmas and proposition.

Lemma 4.4.1. *Any execution of the program satisfies the following properties:*

(1) $\Box\big(Len(TransResp_{a,r}) \leq Len(Trans_{a,s})\big)$;

(2) $\Box\big(Len(TransResp_{f,r}) \leq Len(Trans_{f,s})\big)$;

(3) $\Box\big(Len(Trans_{a,s}) \leq Len(TransResp_{a,r}) + 1\big)$.

Proof. These properties can be derived from the assumptions about the network, (4.10), (4.14), (4.15) and (4.19). □

Lemma 4.4.2. *Any execution of the program satisfies the following properties:*

(1) $\Box\big((Len(TransResp_{a,r}) = Len(Trans_{a,s})) \vee (Len(Trans_{a,s}) = Len(TransResp_{a,r}) + 1\big)$;

(2) $\Box\big((Len(TransResp_{f,r}) = Len(Trans_{f,s})) \vee (Len(Trans_{f,s}) = Len(TransResp_{f,r}) + 1\big)$.

Proof.

1. Let

$$\begin{aligned} A &\equiv Len(Trans_{a,s}) \geq Len(TransResp_{a,r}), \\ B &\equiv Len(Trans_{a,s}) \leq Len(TransResp_{a,r}), \\ C &\equiv Len(Trans_{a,s}) = Len(TransResp_{a,r}) + 1. \end{aligned}$$

   From Lemma 4.4.1, we get that any execution satisfies

$$\Box((A \wedge B) \ \vee \ (A \wedge C)).$$

   Since

$$A \wedge B \equiv Len(TransResp_{a,r}) = Len(Trans_{a,s})$$

   and

$$A \wedge C \equiv Len(Trans_{a,s}) = Len(TransResp_{a,r}) + 1,$$

   this completes the proof.

2. This can be proved using Lemma 4.4.1(2) and property (4.19) and following the same steps as in the previous proof.

□

Lemma 4.4.3. *Any execution of the program satisfies*

$$\Box\big(Len(TransResp_{f,r}) = Len(Trans_{f,s}) \implies$$
$$Len(TransResp_{f,r}) = Len(TransResp_{f,s}) = Len(TransResp_{a,r}) = Len(TransResp_{a,s})$$
$$= Len(Trans_{a,r}) = Len(Trans_{a,s}) = Len(Trans_{f,r}) = Len(Trans_{f,s})\big).$$

Proof. This can be derived from the assumptions about the network, (4.10), (4.14) and (4.15). □

Lemma 4.4.4. *Any execution of the program satisfies*

$$\Box\big((Len(TransResp_{f,r}) \le Len(TransResp_{f,s}) \le Len(TransResp_{a,r}) \le Len(TransResp_{a,s}))$$
$$\wedge\ (\forall i \in \{1, \ldots, Len(TransResp_{f,r})\} : TransResp_{f,r}[i] = TransResp_{f,s}[i] =$$
$$TransResp_{a,r}[i]) = TransResp_{a,s}[i])\big).$$

Proof. This is clear from (4.15) and the assumptions about the network. □

The following proposition can be proved using the truth table.

Proposition 4.4.1. *The following propositional formula is a tautology:*

$$\big((\neg A \vee B) \wedge (A \vee C)\big) \implies (B \vee C).$$

We now prove that the system-level requirements hold, provided that all the component-level requirements are satisfied.

Theorem 4.4.1. *Any execution of the program satisfies*

$$\Box\big((Len(TransResp_{f,r}) = Len(Trans_{f,s}) \ \wedge\ Last(TransResp_{f,r}) = \text{COMPLETED})$$
$$\implies\ Trans_f = Trans\big).$$

Proof. The case where $Len(Trans_{f,s}) = 0$ is trivial so we only consider the case where $Len(Trans_{f,s}) > 0$. Suppose $Len(TransResp_{f,r}) = Len(Trans_{f,s})$ and $Last(TransResp_{f,r}) =$

COMPLETED). Then, we get

$$\begin{aligned} Trans_f &\overset{(4.21)}{=} Trans_{f,s}[Len(TransResp_{f,r})] \\ &\overset{\text{Lemma4.4.3}}{=} Trans_{f,s}[Len(Trans_{a,s})] \\ &\overset{(4.14),\text{network}}{=} Trans_{a,s}[Len(Trans_{a,s})]. \end{aligned}$$

Also, from Lemma 4.4.3 and Lemma 4.4.4, we get

$$Last(TransResp_{a,s}) = \text{COMPLETED}.$$

Using (4.12), we can then conclude that

$$\begin{aligned} Trans &= Trans_{a,s}[Len(TransResp_{a,s})] \\ &\overset{\text{Lemma4.4.3}}{=} Trans_{a,s}[Len(Trans_{a,s})] \\ &= Trans_f. \end{aligned}$$

□

Theorem 4.4.2. *Any execution of the program satisfies*

$$\Box\big(Trans_f = Trans \ \vee \ Acc_{f,s} = -1\big).$$

Proof. From (4.17),

$$\Box\big(Len(TransResp_{f,r}) \neq Len(Trans_{f,s}) \implies Acc_{f,s} = -1\big).$$

Or equivalently,

$$\Box\big(Len(TransResp_{f,r}) = Len(Trans_{f,s}) \ \vee \ Acc_{f,s} = -1\big).$$

Similarly, from (4.18), we get

$$\Box\big(Last(TransResp_{f,r}) = \text{COMPLETED}) \ \vee \ Acc_{f,s} = -1\big).$$

Let

$$\begin{aligned} A &\equiv Len(TransResp_{f,r}) = Len(Trans_{f,s}), \\ B &\equiv Last(TransResp_{f,r}) = \text{COMPLETED}, \\ C &\equiv Trans_f = Trans, \\ D &\equiv Acc_{f,s} = -1. \end{aligned}$$

The system has the following property

$$\Box\big((A \wedge B) \implies C) \wedge (A \vee D) \wedge (B \vee D)\big) \equiv \Box\big((\neg A \vee \neg B \vee C)) \wedge (A \vee D) \wedge (B \vee D)\big).$$

Applying Proposition 4.4.1 twice, we can complete the proof. □

Theorem 4.4.3. *Any execution of the program satisfies*

$$\Box\Diamond(Trans_f = Trans \ \vee \ HWF).$$

Proof. From Lemma 4.4.2(2),

$$\Box\big((Len(TransResp_{f,r}) = Len(Trans_{f,s})) \ \vee \ (Len(Trans_{f,s}) = Len(TransResp_{f,r}) + 1)\big).$$

Consider an arbitrary k_1^{th} cycle. From (4.20) and Lemma 4.4.2, $\exists k > k_1$ such that in the k^{th} cycle, $Len(TransResp_{f,r}) = Len(Trans_{f,s})$.

Consider this k^{th} cycle. The case where $Len(Trans_{f,s}) = 0$ is trivial. (By assumption, $Trans_f = Trans$.) So we only consider the case where $Len(Trans_{f,s}) > 0$. If $Last(TransResp_{f,r}) = \text{COMPLETED}$, then from Theorem 4.4.1, (4.21) and Lemma 4.4.3,

$$Trans_f = Trans = Last(Trans_{f,s}).$$

Otherwise, $Last(TransResp_{f,r}) = Last(TransResp_{a,s}) = \text{FAILED}$, so from (4.13), HWF.

□

4.5 Conclusions

This chapter described the Canonical Software Architecture that supports the hierarchical decomposition and separation of functionality in the control subsystem of Alice, while maintaining communication and contingency management. Two case studies were presented to illustrate the applications of formal methods to the complex embedded control system of Alice. The first case study used an off-the-shelf model checker SPIN to verify the correctness of the implementation of the finite state machine that handled multiple concurrent commands. SPIN uncovered a crucial assumption to ensure system correctness. The second example exploited the structure imposed by the Canonical Software Architecture to verify the desired system-level properties from the components' properties.

Appendix

4.A PROMELA Models for the Gcdrive Finite State Machine Example

System Model

Atomic sequences and an auxiliary variable *enableEstop* are used to ensure that Actuation Interface gets executed at least once each time an estop command is sent.

```
mtype = { D, P, Ru, Re, S };

mtype state = P;
byte estop = P;
short acc = -1;
short acc_cmd = 0;
short gear = 1;
short gear_cmd = 0;
bool enableEstop = true;
byte timer = 0;
```

```
active proctype Follower()
{
        do
        :: acc_cmd = -1
        :: acc_cmd = 0
        :: acc_cmd = 1
        :: gear_cmd = -1
        :: gear_cmd = 0
        :: gear_cmd = 1
        od
}

active proctype sendEstop()
{
        do
        :: atomic{ enableEstop == true ->
                if
                :: estop = 0
                :: estop = 1
                :: estop = 2
                fi;
                enableEstop = false;
           }
        od
}

active proctype ActuationInterface()
{
        do
        :: atomic{ (estop == 0 || state == D) ->
                state = D;
                acc = -1;
```

```
                gear = 0;
                enableEstop = true;
            }
        :: atomic { (estop == 1 && state != D) ->
                state = P;
                acc = -1;
                enableEstop = true;
            }
        :: atomic{ else ->
                if
                :: estop == 2 && state == P ->
                        state = Re;
                        timer = 0;
                :: state == Re && timer == 5 ->
                        state = Ru;
                :: state == Ru && gear_cmd == gear ->
                        acc = acc_cmd;
                :: state == Ru && gear_cmd != gear ->
                        state = S;
                        acc = -1;
                :: state == S ->
                        gear = gear_cmd;
                        state = Ru;
                :: else ->
                        skip;
                fi;
                enableEstop = true;
            }
        od
}

active proctype clock()
```

```
{
        do
        :: atomic{ timer < 5 -> timer = timer + 1 }
        od
}
```

Desired Properties

(1) If DARPA sends an estop disable command, Gcdrive state will eventually stay at DISABLED and Acceleration Module will eventually command full brake forever (see Equation (4.1)).

```
never {
T0_init :
        if
        :: (1) -> goto T1_S1
        :: (estop == 0) -> goto T0_S3
        fi;
T1_S1 :
        if
        :: (1) -> goto T1_S1
        :: (estop == 0) -> goto accept_S3
        fi;
accept_S3 :
        if
        :: (1) -> goto T0_S3
        :: (acc != -1) || (state != D) -> goto accept_S3
        fi;
T0_S3 :
        if
        :: (1) -> goto T0_S3
        :: (acc != -1) || (state != D) -> goto accept_S3
        fi;
}
```

(2) If DARPA sends an estop pause command while the vehicle is not disabled, eventually Gcdrive state will be PAUSED (see Equation (4.2)).

```
never {
T0_init :
        if
        :: (1) -> goto T0_init
        :: (state != D && state != P && estop == 1) -> goto accept_S2
        fi;
accept_S2 :
        if
        :: (state != P) -> goto accept_S2
        fi;
}
```

(3) If DARPA send an estop run command while the vehicle is not disabled, eventually, Gcdrive state will be RUNNING or RESUMING or DARPA will send an estop disable or estop pause command (see Equation (4.3)).

```
never {
T0_init :
        if
        :: (1) -> goto T0_init
        :: (state != D && state != Re && state != Ru && estop == 2 &&
                estop == 2) -> goto accept_S2
        fi;
accept_S2 :
        if
        :: (state != Re && state != Ru && estop == 2) -> goto accept_S2
        fi;
}
```

(4) If the current state is RESUMING, eventually, the state will be RUNNING or DARPA will send an estop disable or pause command (see Equation (4.4)).

```
never {
T0_init :
        if
        :: (1) -> goto T0_init
        :: (estop != 0 && estop != 1 && state != Ru && state == Re) ->
                goto accept_S2
        fi;
accept_S2 :
        if
        :: (estop != 0 && estop != 1 && state != Ru) -> goto accept_S2
        fi;
}
```

(5) The vehicle is disabled only after it receives an estop disable command (see Equation (4.5)).

```
never {
T0_init :
        if
        :: (estop != 0) -> goto T0_init
        :: (state == D && estop != 0) -> goto accept_all
        fi;
accept_all :
        skip
}
```

(6) Actuation Interface sends a full brake command to the Acceleration Module if the current state is DISABLED, PAUSED, RESUMING or SHIFTING. In addition, if the vehicle is disabled, then the gear is shifted to 0 (see Equation (4.6)). After receiving an estop pause command, the vehicle may resume the operation 5 seconds after an estop run command is received (see Equation (4.7)).

```
active proctype invariant()
{
        assert( (state == Ru || acc == -1) &&
                (state != D || gear == 0) &&
                (state != Ru || timer >= 5) )
}
```

Chapter 5

Periodically Control Hybrid Automata

This chapter introduces *Periodically Controlled Hybrid Automata* (PCHA), a subclass of hybrid automata suitable for modeling sampled control systems and embedded systems with periodic sensing and actuation. A sufficient condition for verifying invariance of PCHAs is presented. This technique is then used to analyze the design flaw that caused the failure of Alice at the National Qualifying Event of the DARPA Urban Challenge as described in Section 1.1.

5.1 Overview

While real-world hybrid systems are large and complex, they are also engineered, and hence, have more structure than general hybrid automata [3]. With the motivation of abstractly capturing a common design pattern in embedded control systems, such as Alice, and other motion control systems [89], in this chapter we study a new subclass of hybrid automata. The two main contributions of this chapter are the following:

First, we define a class of hybrid control systems in which certain *control actions* occur roughly periodically. Each control action sets the *controlling output* that drives the underlying physical process, which we refer to as the plant. In the interval between two control actions, the state of the plant evolves continuously with fixed control inputs. Also, in the same interval, other discrete actions may occur, updating the state of the system. Such discrete changes may correspond, for example, to sensor inputs and changes of the waypoint or the set-point of the controller. These changes may in turn influence the compu-

tation of the next control action. For this class of *Periodically Controlled Hybrid Systems*, we present a sufficient condition for verifying invariant properties. The key requirement in applying this condition is to identify a collection of subset(s) C of the candidate invariant set $\mathcal{I}$, such that if the control action occurs when the system state is in C, then the subsequent control output guarantees that the system remains within $\mathcal{I}$ for the next period. The technique does not require solving the differential equations; instead, it relies on checking conditions on the periodicity and the subtangential condition at the boundary of $\mathcal{I}$. For systems with polynomial vector fields, we show how these checks can be automated using sum of squares decomposition and semidefinite programming [106]. These formulations are illustrated by analyzing a simple example in which an invariant is automatically determined using the constraint-based approach presented in [43]. We believe that other techniques for finding invariants, for example those presented in [99, 111], could also be effectively used for computing invariants of PCHAs.

Second, we apply the above technique to manually verify the safety and progress properties of the planner-controller subsystem of Alice. Since the model of Alice involves complex, non-polynomial dynamics, the proposed automatic approach is not directly applicable. Thus, the analysis is done completely by hand. First, we verify a family of invariants $\{\mathcal{I}_k\}_{k\in\mathbb{N}}$ using the above-mentioned technique. This step is fairly simple, requiring only algebraic simplification of expressions defining the vector fields and $\mathcal{I}_k$'s. Then, we determine a sequence of shrinking $\mathcal{I}_k$'s as the vehicle makes progress along the planned path. From these shrinking invariants, we are able to deduce safety. That is, the deviation—distance of the vehicle from the planned path—remains within a certain constant bound. In the process, we also derive geometric properties of planner paths that guarantee that they can be followed safely by the vehicle. Informally, these geometric properties require that sharp turns in the path are only present *after* relatively long segments. In executing a long segment, the vehicle converges to small deviation as well as small disorientation with respect to the path. Thus, the instruction for executing a subsequent sharp turn, does not make the deviation grow too much.

This chapter is organized as follows: In Section 5.2, we briefly present the key definitions for the hybrid I/O automaton framework. In Section 5.3, we present PCHA and a sufficient condition for proving invariance. The formulation of this sufficient condition as a sum of squares optimization problem for automatic verification is also provided. In Sec-

tions 5.4 and 5.5, we present the formal model and verification of Alice's Controller-Vehicle subsystem.

5.2 Preliminaries

We use the Hybrid Input/Output Automata (HIOA) framework of [76, 59] for modelling hybrid systems and the state model-based notations introduced in [88]. A HIOA is a non-deterministic state machine whose state may change instantaneously through a transition, or continuously over an interval of time following a *trajectory*. In this section, we briefly present important terminology and notations that are used throughout the chapter. We refer the reader to [59, 88] and references therein for more details.

A variable structure is used for specifying the states of an HIOA. Let V be a set of variables. Each variable $v \in V$ is associated with a *type* which defines the set of values v can take. The set of valuations of V is denoted by $dom(V)$. For a valuation $\mathbf{v} \in dom(V)$ of set of variables V, its restriction to a subset of variables $Z \subseteq V$ is denoted by $\mathbf{v} \lceil Z$. A variable may be *discrete* or *continuous*. (See [88] for formal definition of these variable dynamic types.) Typically, discrete variables model protocol or software state, and continuous variables model physical quantities such as time, position and velocity.

A *trajectory* for a set of variables V models continuous evolution of the values of the variables over an interval of time. Formally, a *trajectory* τ is a map from a left-closed interval of $\mathbb{R}_{\geq 0}$ with left endpoint 0 to $dom(V)$. The domain of τ is denoted by $\tau.dom$. The *first state* of τ, $\tau.\mathsf{fstate}$, is $\tau(0)$. A trajectory τ is *closed* if $\tau.dom = [0, t]$ for some $t \in \mathbb{R}_{\geq 0}$, in which case we define the *last time* of τ, $\tau.\mathsf{ltime} \triangleq t$, and the *last state* of τ, $\tau.\mathsf{lstate} \triangleq \tau(t)$. For a trajectory τ for V, its restriction to a subset of variables $Z \subseteq V$ is denoted by $\tau \downarrow Z$.

The set of allowed trajectories of all the variables of an HIOA is defined by *state models*, as follows. For given set V of variables, a *state model* $\mathcal{S}$ is a triple $(\mathscr{F}_{\mathcal{S}}, Inv_{\mathcal{S}}, Stop_{\mathcal{S}})$, where

(a) $\mathscr{F}_{\mathcal{S}}$ is a collection of differential equations (DEs) involving the continuous variables in V, and

(b) $Inv_{\mathcal{S}}$ and $Stop_{\mathcal{S}}$ are predicates on V called *invariant condition* and *stopping condition* of $\mathcal{S}$.

$\mathcal{S}$ defines a set of trajectories, denoted by $traj(\mathcal{S})$, for the set V of variables. A trajectory

τ for V is in the set $trajs(\mathcal{S})$ iff

(a) the discrete variables in V remain constant over τ;

(b) the restriction of τ on the continuous variables in V satisfies all the DEs in $\mathscr{F}_{\mathcal{S}}$;

(c) at every point in time $t \in dom(\tau)$, $(\tau \downarrow V)(t) \in Inv_{\mathcal{S}}$; and

(d) if $(\tau \downarrow V)(t) \in Stop_{\mathcal{S}}$ for some $t \in dom(\tau)$, then τ is closed and $t = \tau.\mathsf{ltime}$.

5.3 Periodically Controlled Hybrid Automata

In this section, we define a subclass of HIOAs that is suitable for modeling sampled control systems and embedded systems with periodic sensing and actuation. The main result of this section, Theorem 5.3.1, gives a sufficient condition for proving invariant properties of this subclass.

5.3.1 Definition of Periodically Controlled Hybrid Automata

A Periodically Controlled Hybrid Automaton (PCHA) is an HIOA with a set of (*control*) actions that occur roughly periodically. These *control* actions alter the actual control signal (input) that feeds to the plant and may change the continuous and the discrete state variables of the automaton. The automaton may have other actions that change only the discrete state of the automaton. These actions can model, for example, sensor inputs and the change in the set-point of the controller from higher-level inputs. However, these external commands do not affect the dynamics of the system immediately; they only change the internal variables of the controller. Formally, a PCHA is defined as follows.

Definition 5.3.1. Let $\mathcal{X} \subseteq \mathbb{R}^n$, for some $n \in \mathbb{N}$, and $\mathcal{L}, \mathcal{Z}$ and $\mathcal{U}$ be arbitrary types. A *Periodically Controlled Hybrid Automaton (PCHA)* $\mathcal{A}$ is a tuple $(X, Q, Q_0, A, \mathcal{D}, \mathscr{S})$ where

(a) $X = \{s, loc, z, u, now, next\}$ is a set of *internal* or *state* variables where s is a *continuous state* variable of type $\mathcal{X}$, *loc* is a *discrete state* (*location* or *mode*) variable of type $\mathcal{L}$, z is a *command* variable of type $\mathcal{Z}$, u is a *control* variable of type $\mathcal{U}$, *now* is a real continuous variable and *next* is a real discrete variable;

(b) $Q \subseteq dom(X)$ is a set of *states* and $Q_0 \subseteq Q$ is a non-empty set of *start states*;

(c) A is a set of actions, consisting of a set of update actions and a single control action;

(d) $\mathcal{D} \subseteq Q \times A \times Q$ is a set of *discrete transitions*. A transition $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$ is written in short as $\mathbf{x} \rightarrow^{a}_{\mathcal{A}} \mathbf{x}'$ or as $\mathbf{x} \stackrel{a}{\rightarrow} \mathbf{x}'$ when $\mathcal{A}$ is clear from the context. An action $a \in A$ is said to *enabled* at a state $\mathbf{x} \in Q$ if there exists a state $\mathbf{x}' \in Q$ such that $\mathbf{x} \stackrel{a}{\rightarrow} \mathbf{x}'$; and

(e) $\mathscr{S}$ is a collection of *state models* for X, such that for every $\mathcal{S}, \mathcal{S}' \in \mathscr{S}$, $Inv_{\mathcal{S}} \cap Inv_{\mathcal{S}'} = \varnothing$ and $Q \subseteq \bigcup_{\mathcal{S} \in \mathscr{S}} Inv_{\mathcal{S}}$.

In addition, $\mathcal{A}$ must satisfy the property that every update action is enabled[1] at every state and may only change the value of z, while control actions occur roughly periodically starting from time 0; the time gap between two successive occurrences is within $[\Delta_1, \Delta_1 + \Delta_2]$ where $\Delta_1 > 0$ and $\Delta_2 \geq 0$.

We denote the components of a PCHA $\mathcal{A}$ by $X_{\mathcal{A}}, Q_{\mathcal{A}}$, etc. For a set X of variables, a *state* $\mathbf{x}$ is an element of $dom(X)$. We denote the valuation of a variable $y \in X$ at state $\mathbf{x}$, by the usual (.) notation $\mathbf{x}.y$.

The continuous state typically includes the continuous state of the plant and some internal state of the controller. The discrete state represents the mode of the system. The command variable is used to store externally produced input commands or sensor updates. The control variable stores the control input to the plant. Finally, the *now* and *next* variables are used for triggering the control action periodically.

PCHA $\mathcal{A}$ has two types of actions: (a) through input action update, $\mathcal{A}$ learns about new externally produced input commands such as set-points, waypoints. When an update(z') action occurs, z' is recorded in the command variable z. (b) The control action changes the continuous and discrete state variables s and *loc* and the control variable u. When control occurs, *loc* and s are computed as a function of their current values and that of z, and u is computed as a function of the new values of *loc* and s. Observe, from this definition, that the external commands do not affect the dynamics of the system immediately, i.e., they do not change the location nor the input value of the hybrid system but only the internal variables of the controller. The modification of the dynamics due to the external commands are effective at the next control cycle.

For each value of $l \in \mathcal{L}$, the continuous state s evolves according to the trajectories specified by state model $smodel(l)$. That is, s evolves according to the differential equation

[1] In the terminology of HIOA, an update action is an input action.

signature
  internal control
  input update($z' : Z$)

variables
  internal $s : \mathcal{X} := s_0$
  internal discrete $loc : \mathcal{L} := l_0$,
    $z : \mathcal{Z} := z_0$, $u : \mathcal{U} := u_0$
  internal $now : \mathbb{R}_{\geq 0} := 0$, $next : \mathbb{R} := -\Delta_2$

transitions
  input update(z')
  eff $z := z'$
  internal control
  pre $now \geq next$
  eff $next := now + \Delta_1$;
    $(loc,\ s) := h(loc,\ s,\ z)$;
    $u := g(loc,\ s)$

trajectories
  trajdef $smodel(l : \mathcal{L})$
    invariant $loc = l$
    evolve $d(now) = 1$;
     $d(s) = f_l(s,\ u)$
    stop when $now = next + \Delta_2$

Figure 5.1: PHCA with parameters Δ_1, Δ_2, g, h, $\{f_l\}_{l \in \mathcal{L}}$. See, for example, [88] for the description of the language.

$\dot{s} = f_l(s, u)$. The timing of control behavior is enforced by the precondition of control and the stopping condition of the state models.

Note that as opposed to a general HIOA, a PCHA does not contain *input* and *output* variables. For the sake of simplicity, we consider the PCHAs of the form shown in Figure 5.1 with only one update action and a unique starting state. However, Theorem 5.3.1 generalizes to PCHAs with multiple update actions as illustrated later in Section 5.5.

An execution of a PCHA $\mathcal{A}$ records the valuations of all its variables and the occurrences of all actions over a particular run. An *execution fragment* of $\mathcal{A}$ is a finite or infinite sequence $\alpha = \tau_0 a_1 \tau_1 a_2 \ldots$, such that for all i in the sequence, $a_i \in A_{\mathcal{A}}$, $\tau \in traj s(\mathcal{S})$ for some $\mathcal{S} \in \mathscr{S}_{\mathcal{A}}$, and $\tau_i.\mathsf{lstate} \overset{a_{i+1}}{\rightarrow} \tau_{i+1}.\mathsf{fstate}$. An execution fragment is an *execution* if $\tau_0.\mathsf{fstate} \in Q_0$. An execution is *closed* if it is finite and the last trajectory in it is closed. The first state of an execution fragment α, $\alpha.\mathsf{fstate}$, is $\tau_0.\mathsf{fstate}$, and for a closed α, its last state, $\alpha.\mathsf{lstate}$, is the last state of its last trajectory. The *limit time* of α, $\alpha.\mathsf{ltime}$, is defined to be $\sum_i \tau_i.\mathsf{ltime}$. The set of executions and reachable states of $\mathcal{A}$ are denoted by $\mathsf{Execs}_{\mathcal{A}}$ and $\mathsf{Reach}_{\mathcal{A}}$. A set of states $\mathcal{I} \subseteq Q_{\mathcal{A}}$ is said to be an *invariant* of $\mathcal{A}$ iff $\mathsf{Reach}_{\mathcal{A}} \subseteq \mathcal{I}$.

5.3.2 Invariant Verification

Proving invariant properties of hybrid automata is a central problem in formal verification. Invariants are used for overapproximating the reachable states of a given system, and therefore, can be used for verifying safety properties.

Given a candidate invariant set $\mathcal{I} \subseteq Q$, we are interested in verifying that $\mathsf{Reach}_{\mathcal{A}} \subseteq \mathcal{I}$. For continuous dynamical systems, checking the well-known subtangential condition as in

the Lyapunov-type methods provides a sufficient condition for proving invariance of a set $\mathcal{I}$ that is bounded by a closed surface (see, for example, [13]). Theorem 5.3.1 below provides an analogous sufficient condition for PCHAs. In general, however, invariant sets $\mathcal{I}$ for PCHAs have to be defined by a collection of functions instead of a single function. For each mode $l \in \mathcal{L}$, we assume that the invariant set $I_l \subseteq \mathcal{X}$ for the continuous state is defined by a collection of m *boundary functions* $\{F_{lk}\}_{k=1}^{m}$, where m is some natural number and each $F_{lk} : \mathcal{X} \to \mathbb{R}$ is a differentiable function.[2] Formally,

$$I_l \triangleq \{s \in \mathcal{X} \mid \forall k \in \{1, \ldots, m\}, F_{lk}(s) \geq 0\} \quad \text{and} \quad \mathcal{I} \triangleq \{\mathbf{x} \in Q \mid \mathbf{x}.s \in I_{\mathbf{x}.loc}\}.$$

Note that the overall candidate invariant set $\mathcal{I}$ does not restrict the values of the command or the control variables. In the remainder of this section, we develop a set of sufficient conditions for checking that $\mathcal{I}$ is indeed an invariant of a given PCHA. Lemma 5.3.1 modifies the standard inductive technique for proving invariance, so that it suffices to check invariance with respect to control transitions and control-free execution fragments of length not greater than $\Delta_1 + \Delta_2$. It states that $\mathcal{I}$ is an invariant if it is closed under (a) the discrete transitions of the control actions, and (b) control-free execution fragments of length at most $\Delta_1 + \Delta_2$.

Lemma 5.3.1. *Suppose $Q_0 \subseteq \mathcal{I}$ and the following two conditions hold:*

*(a) (*control *steps) For each state $\mathbf{x}, \mathbf{x}' \in Q$, if $\mathbf{x} \overset{\text{control}}{\to} \mathbf{x}'$ and $\mathbf{x} \in \mathcal{I}$ then $\mathbf{x}' \in \mathcal{I}$.*

*(b) (*control*-free fragments) For each closed execution fragment $\beta = \tau_0$ update(z_1) τ_1 update(z_2) $\ldots$ τ_n starting from a state $\mathbf{x} \in \mathcal{I}$ where each $z_i \in \mathcal{Z}$, if $\mathbf{x}.next - \mathbf{x}.now = \Delta_1$ and β.ltime $\leq \Delta_1 + \Delta_2$, then β.lstate $\in \mathcal{I}$.*

Then Reach$_{\mathcal{A}} \subseteq \mathcal{I}$.

Proof. Consider any reachable state $\mathbf{x}$ of $\mathcal{A}$ and any execution α such that α.lstate $= \mathbf{x}$. We can write α as β_0 control β_1 control $\ldots \beta_k$, where each β_i is control-free execution fragment of $\mathcal{A}$, i.e., execution fragments in which only update actions occur. From condition (a), it follows that for each $i \in \{0, \ldots, k\}$, if β_i.lstate $\in \mathcal{I}$, then β_{i+1}.fstate $\in \mathcal{I}$.

Thus, it suffices to prove that for each $i \in \{0, \ldots, k\}$, if β_i.fstate $\in \mathcal{I}$, then β_i.lstate $\in \mathcal{I}$. We fix an $i \in \{0, \ldots, k\}$ and assume that β_i.fstate $\in \mathcal{I}$. Let $\beta_i = \tau_0$ update(z_1) τ_1 update$(z_2) \ldots \tau_n$,

[2] Identical size m of the collections simplifies our notation; different number of boundary functions for different values of l can be handled by extending the theorem in an obvious way.

where for $j \in \{0, \ldots, n\}$, $z_j \in \mathcal{Z}$ and τ_j is a trajectory of $\mathcal{A}$. If $i = 0$, then $\beta_i.\mathsf{ltime} = 0$ and $\beta_i.\mathsf{lstate} \lceil \{loc, s\} = \beta_i.\mathsf{fstate} \lceil \{loc, s\}$ since the first control action occurs at time 0 and update transitions do not affect the value of loc and s. Therefore, $\beta_i.\mathsf{lstate} \in \mathcal{I}$. Otherwise, $i > 0$ and since β_i starts immediately after a control action, $\beta.\mathsf{fstate} \lceil next - \beta.\mathsf{fstate} \lceil now = \Delta_1$. From periodicity of main actions, we know that $\beta_i.\mathsf{ltime} \leq \Delta_1 + \Delta_2$, and hence from condition (b) it follows that $\beta_i.\mathsf{lstate} \in \mathcal{I}$. □

Invariance of control steps can often be checked through case analysis, which can be partially automated using a theorem prover [93]. The next key lemma provides a sufficient condition for proving invariance of control-free fragments. Since control-free fragments do not change the valuation of the loc variable, for this part, we fix a value $l \in \mathcal{L}$. For each index of the boundary functions $j \in \{1, \ldots m\}$, we define the set ∂I_j to be part of the set I_l where the function F_{lj} vanishes. That is, $\partial I_j \triangleq \{x \in \mathcal{X} \mid F_{lj}(x) = 0\}$. For the sake of brevity, we call ∂I_j the j^{th} boundary of I_l even though strictly speaking, the j^{th} boundary of I_l is only a subset of ∂I_j according to the standard topological definition. Similarly, we say that the *boundary* of I_l, is $\partial I_l = \bigcup_{j \in \{1,\ldots,m\}} \partial I_j$.

Lemma 5.3.2. *Suppose that there exists a collection $\{C_j\}_{j=1}^m$ of subsets of I_l such that the following conditions hold:*

(a) (Subtangential) For each $s_0 \in I_l \setminus C_j$ and $s \in \partial I_j$, $\frac{\partial F_{lj}(s)}{\partial s} \cdot f_l(s, g(l, s_0)) \geq 0$.

(b) (Bounded distance) $\exists\ c_j > 0$ such that $\forall\ s_0 \in C_j, s \in \partial I_j$, $\|s - s_0\| \geq c_j$.

(c) (Bounded speed) $\exists\ b_j > 0$ such that $\forall\ s_0 \in C_j, s \in I_l$, $\|f_l(s, g(l, s_0))\| \leq b_j$,

(d) (Fast sampling) $\Delta_1 + \Delta_2 \leq \min_{j \in \{1,\ldots,m\}} \frac{c_j}{b_j}$.

Then, any control-free execution fragment β, with $\beta.\mathsf{ltime} \leq \Delta_1 + \Delta_2$, starting from a state in I_l where $next - now = \Delta_1$, remains within I_l.

In Figure 5.2, the control and control-free fragments are shown by bullets and lines, respectively. A fragment starting in $\mathcal{I}$ and leaving $\mathcal{I}$, must cross ∂I_1 or ∂I_2. Consider the following four cases.

(1) If u is evaluated outside both C_1 and C_2 (e.g., τ_2, τ_4 and τ_6), then condition (a) guarantees that the fragment does not cross ∂I_j where $j \in \{1, 2\}$ because when it reaches ∂I_j, the vector field governing its evolution points inwards with respect to ∂I_j.

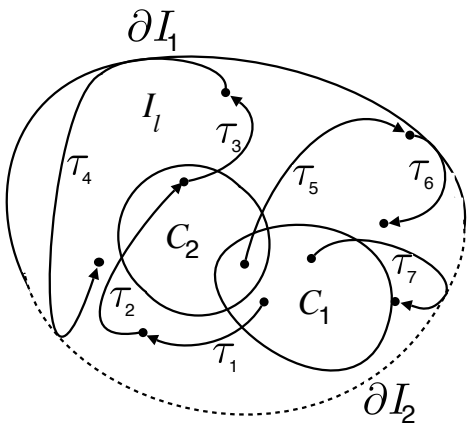


Figure 5.2: A graphical explanation of Lemma 5.3.2 showing an invariant set I_l defined by two boundary functions. The boundary ∂I_1 is drawn in solid line whereas the boundary ∂I_2 is drawn in dotted line. The corresponding sets C_1 and C_2 are also shown.

(2) If u is evaluated inside C_1 but outside C_2 (e.g., τ_1 and τ_7), then by the previous reasoning, condition (a) guarantees that the fragment does not cross ∂I_2. In addition, conditions (b) and (c) guarantee that it takes finite time before the fragment reaches ∂I_1 and condition (d) guarantees that this finite time is at least $\Delta_1 + \Delta_2$; thus, before the fragment crosses ∂I_1, u is evaluated again.

(3) If u is evaluated outside C_1 but inside C_2 (e.g., τ_3), then by a symmetric argument, the fragment does not cross ∂I_1 or ∂I_2.

(4) If u is evaluated inside both C_1 and C_2 (e.g., τ_5), then conditions (b), (c) and (d) guarantee that u is evaluated again before the fragment crosses ∂I_1 or ∂I_2.

Proof. We fix a control-free execution fragment $\beta = \tau_0 \mathsf{update}(z_1)\tau_1 \mathsf{update}(z_2) \ldots \tau_n$ such that at $\beta.\mathsf{fstate}$, $next - now = \Delta_1$. Without loss of generality we assume that at $\beta.\mathsf{fstate}$, $z = z_1$, $loc = l$ and $s = x_1$, where $z_1 \in \mathcal{Z}, l \in \mathcal{L}$ and $x_1 \in I_l$. We have to show that at $\beta.\mathsf{lstate}$, $s \in I_l$.

First, observe that for each $k \in \{0, \ldots, n\}$, $(\tau_k \downarrow s)$ is a solution of the differential equation(s) $d(s) = f_l(s, g(l, x_1))$. Let τ be the pasted trajectory $\tau_0 \frown \tau_1 \frown \ldots \tau_n$.[3] Let $\tau.\mathsf{ltime}$ be T. Since the update action does not change s, $\tau_k.\mathsf{lstate} \lceil s = \tau_{k+1}.\mathsf{fstate} \lceil s$ for each $k \in \{0, \ldots, n-1\}$. As the differential equations are time-invariant, $(\tau \downarrow s)$ is a solution of $d(s) = f_l(s, g(l, x_1))$. We define the function $\gamma : [0, T] \to \mathcal{X}$ as $\forall\ t \in [0, T]$, $\gamma(t) \triangleq (\tau \downarrow s)(t)$.

[3] $\tau_1 \frown \tau_2$ is the trajectory obtained by concatenating τ_2 at the end of τ_1.

We have to show that $\gamma(T) \in I_l$. Suppose, for the sake of contradiction, that there exists $t^* \in [0,T]$, such that $\gamma(t^*) \notin I_l$. By the definition of I_l, there exists i such that $F_{li}(\gamma(0)) \geq 0$ and $F_{li}(\gamma(t^*)) < 0$. We pick one such i and fix it for the remainder of the proof. Since F_{li} and γ are continuous, from intermediate value theorem, we know that there exists a time t_1 before t^* where F_{li} vanishes and that there is some finite time $\epsilon > 0$ after t_1 when F_{li} is strictly negative. Formally, there exists $t_1 \in [0,t^*)$ and $\epsilon > 0$ such that for all $t \in [0,t_1]$, $F_{li}(\gamma(t)) \geq 0$, $F_{li}(\gamma(t_1)) = 0$, and for all $\delta \in (0,\epsilon]$, $F_{li}(\gamma(t_1+\delta)) < 0$.

Case 1: $x_1 \in I_l \setminus C_i$. Since $F_{li}(\gamma(t_1)) = 0$, by definition, $\gamma(t_1) \in \partial I_i$. But from the value of $F_{li}(\gamma(t))$ where t is near to t_1, we get that $\frac{\partial F_{li}}{\partial t}(t_1) = \frac{\partial F_{li}}{\partial s}(\gamma(t_1)) \cdot f_l(\gamma(t_1), g(l,x_1)) < 0$. This contradicts condition (a).

Case 2: $x_1 \in C_i$. Since for all $t \in [0,t_1]$, $F_{li}(\gamma(t)) \geq 0$ and $F_{li}(\gamma(t_1)) = 0$, we get that for all $t \in [0,t_1]$, $\gamma(t) \in I_l$ and $\gamma(t_1) \in \partial I_i$. So from condition (b) and (c), we get $c_i \leq \|\gamma(t_1) - x_1\| = \left\|\int_0^{t_1} f_l(\gamma(t), g(l,x_1))dt\right\| \leq b_i t_1$. That is, $t_1 \geq \frac{c_i}{b_i}$. But we know that $t_1 < t^* \leq T$ and periodicity of control actions $T \leq \Delta_1 + \Delta_2$. Combining these, we get $\Delta_1 + \Delta_2 > \frac{c_i}{b_i}$, which contradicts condition (d). □

For PCHAs with certain properties, the following lemma provides sufficient conditions for the existence of the bounds b_j and c_j, which satisfy the bounded distance and bounded speed conditions of Lemma 5.3.2.

Lemma 5.3.3. *For a given $l \in L$, let $U_l = \{g(l,s) \mid l \in \mathcal{L}, s \in I_l\} \subseteq \mathcal{U}$ and suppose I_l is compact and f_l is continuous in $I_l \times U_l$. The bounded distance and bounded speed conditions (of Lemma 5.3.2) are satisfied if $C_j \subset I_l$ satisfies the following conditions: (a) C_j is closed, and (b) $C_j \cap \partial I_j = \varnothing$*

Proof. From the continuity of F_{lj}, we can assume, without loss of generality, that $\partial I_j \neq \varnothing$. This is because if $\partial I_j = \varnothing$, then for all $s \in \mathcal{X}$, it must be either $F_{lj}(s) > 0$ or $F_{lj}(s) < 0$, that is, F_{lj} is not needed to describe I_l. In addition, the case where $C_j = \varnothing$ is trivial since conditions (b) and (c) of Lemma 5.3.2 are satisfied for any arbitrary large c_j and arbitrary small b_j. So for the rest of the proof, we assume that $\partial I_j \neq \varnothing$ and $C_j \neq \varnothing$. Since I_l is compact and C_j and ∂I_j are closed, C_j and ∂I_j are also compact. Consider a function $G_j : \partial I_j \to \mathbb{R}$ defined by

$$G_j(s) = \min_{s_0 \in C_j} \|s - s_0\|,$$

where $\|\cdot\|$ is a norm on $\mathbb{R}^n$. Due to the continuity of $\|\cdot\|$ and the compactness and non-emptiness of C_j, G_j is continuous and since $C_j \cap \partial I_j = \varnothing$, we get that for all $s \in \partial I_j, G_j(s) > 0$. Since ∂I_j is compact and non-empty, G_j attains its minimum in ∂I_j. So there exists $c_j > 0$ such that $\min_{s \in \partial I_j} G_j(s) \geq c_j$.

Next, consider a function $H_j : I_l \to \mathbb{R}$ defined by

$$H_j(s) = \max_{s_0 \in C_j} \|f_l(s, g(l, s_0))\|.$$

Using the continuity of f_l, the compactness and non-emptiness of C_j and I_l and the same argument as above, we can conclude that there exists $b_j \geq 0$ such that $\max_{s \in I_l} H_j(s) \leq b_j$. □

Theorem 5.3.1 combines the above lemmas and provides sufficient conditions for invariance of $\mathcal{I}$.

Theorem 5.3.1. *Consider a* PCHA $\mathcal{A}$ *and a set* $\mathcal{I} \subseteq Q_{\mathcal{A}}$. *Suppose* $Q_{0\mathcal{A}} \subseteq \mathcal{I}$, $\mathcal{A}$ *satisfies* control *invariance condition of Lemma 5.3.1 and conditions (a)–(d) of Lemma 5.3.2 for each* $l \in \mathcal{L}_{\mathcal{A}}$. *Then* $\mathsf{Reach}_{\mathcal{A}} \subseteq \mathcal{I}$.

Proof. The proof follows directly from Lemma 5.3.1 and Lemma 5.3.2 since if conditions (a)–(d) of Lemma 5.3.2 are satisfied for any $l \in \mathcal{L}$, then condition (b) of Lemma 5.3.1 is satisfied. □

Theorem 5.3.1 essentially exploits the structure of PCHAs in order to simplify their invariant verification. It can be applied to any PCHAs, including those with non-polynomial vector fields such as Alice, as illustrated later in Section 5.5. Although the PCHA of Figure 5.1 has one action of each type, Theorem 5.3.1 can be extended for PCHAs with an arbitrary number of update actions. For PCHAs with polynomial vector fields, given semialgebraic sets I_l and C_j, checking condition (a) and finding c_j and b_j that satisfy conditions (b) and (c) of Lemma 5.3.2 can be formulated as a sum of squares optimization problem (provided that I_l and C_j are basic semialgebraic sets) or proving emptiness of certain semialgebraic sets based on quantifier elimination. The sum of squares formulation is presented in the next section and allows the proof to be automated using, for example, SOSTOOLS [106]. The quantifier elimination problem can also be formulated and allows the proof to be automated using, for example, QEPCAD [17]. Alternatively, in Section 5.3.4,

we show how an invariant set can be automatically computed using the constraint-based approach presented in [43].

5.3.3 Sum of Squares Formulation for Checking the Invariant Conditions

Suppose the candidate invariant set I_l is a basic semialgebraic set, i.e., each of the boundary functions $F_{lk} : \mathcal{X} \to \mathbb{R}$ is a real polynomial. This section presents a sum of squares formulation for (1) checking condition (a) and finding the c_j and b_j that satisfy conditions (b) and (c) of Lemma 5.3.2 for a given basic semialgebraic subset C_j, and (2) finding a subset C_j such that conditions (a)–(c) of Lemma 5.3.2 are satisfied. For the first case, the sum of squares problem is convex and can be solved using, for example, SOSTOOLS [106]. For the second case, however, the problem is not convex but can still be automatically solved using an iterative scheme as presented in [104].

Checking the Invariant Condition for a Given Subset

Suppose C_j is a basic semialgebraic set, that is, there exists a natural number p such that C_j can be written as

$$C_j = \{s \in I_l \mid \forall i \in \{1, \ldots, p\}, G_{ji}(s) \geq 0\} \tag{5.1}$$

where $G_{ji} : \mathcal{X} \to \mathbb{R}$ is a real polynomial for each $i \in \{1, \ldots, p\}$. Using the generalized S-procedure (a special case of the Positivstellensatz) [116], we obtain the following sufficient condition for condition (a) of Lemma 5.3.2.

Proposition 5.3.1. *Suppose for each $k \in \{1, \ldots, p\}$, there exist sums of squares $\kappa_{1,k}(s, s_0)$, $\mu_k(s)$, $\rho_{k,i}(s)$ and $\sigma_{k,i}(s)$ for $i \in \{1, \ldots, m\}$ and a polynomial $\nu_k(s)$ such that*

$$\begin{aligned} \frac{\partial F_{lj}(s)}{\partial s} \cdot f_l(s, g(l, s_0)) \quad = \quad & \kappa_{1,k}(s, s_0) + \textstyle\sum_{i=1}^{m} \rho_{k,i}(s) F_{li}(s) + \nu_k(s) F_{lj}(s) + \sum_{i=1}^{m} \sigma_{k,i}(s_0) F_{li}(s_0) \\ & - \mu_k(s_0) G_{jk}(s_0). \end{aligned} \tag{5.2}$$

Then, For each $s_0 \in I_l \setminus C_j$ and $s \in \partial I_j$,

$$\frac{\partial F_{lj}(s)}{\partial s} \cdot f_l(s, g(l, s_0)) \geq 0.$$

Given arbitrary $s \in \partial I_j$ and $s_0 \in I_l \setminus C_j$, non-negativity of $\kappa_{1,k}(s, s_0)$, $\rho_{k,i}(s)$, $\sigma_{k,i}(s_0)$ and $\mu_k(s_0)$ implies that the derivative term on the left-hand side of the equation (5.2) is non-negative. In other words, the condition in (5.2) ensures that for each $k \in \{1, \ldots, p\}$,

$$\begin{aligned}&\{(s, s_0) \in \mathcal{X} \times \mathcal{X} \mid \forall i \in \{1, \ldots m\}, F_{li}(s) \geq 0, F_{lj}(s) = 0, F_{li}(s_0) \geq 0, G_{jk}(s_0) \leq 0\} \\ &\subseteq \left\{(s, s_0) \in \mathcal{X} \times \mathcal{X} \;\middle|\; \tfrac{\partial F_{lj}(s)}{\partial s} \cdot f_l(s, g(l, s_0)) \geq 0\right\}.\end{aligned}$$

That is, for all $s \in \partial I_j$ and $s_0 \in I_l \setminus C_j$, we have $\frac{\partial F_{lj}(s)}{\partial s} \cdot f_l(s, g(l, s_0)) \geq 0$. Similarly, based on the generalized S-procedure, checking condition (b) and checking condition (c) of Lemma 5.3.2 can be formulated as an optimization problem according to the following propositions.

Proposition 5.3.2. *There exists $c_j > 0$ such that for all $s_0 \in C_j$ and $s \in \partial I_j$, $\|s - s_0\| \geq c_j$ if the solution c_j^* of the following optimization problem is positive.*

Maximize c_j such that there exist sums of squares $\kappa_2(s, s_0)$, $\gamma_i(s)$ for $i \in \{1, \ldots, m\}$ and $\lambda_i(s)$ for $i \in \{1, \ldots p\}$ and a polynomial $\gamma_{m+1}(s)$ such that

$$\|s - s_0\|^2 - c_j^2 \quad = \quad \kappa_2(s, s_0) + \sum_{i=1}^{m} \gamma_i(s) F_{li}(s) + \gamma_{m+1}(s) F_{lj}(s) + \sum_{i=1}^{p} \lambda_i(s_0) G_{ji}(s_0).$$

Proposition 5.3.3. *There exists $b_j > 0$ such that for all $s_0 \in C_j$ and $s \in I_l$, $\|f_l(s, g(l, s_0))\| \leq b_j$ if the solution b_j^* of the following optimization problem is positive.*

Minimize b_j such that there exist sums of squares $\kappa_3(s, s_0)$, $\zeta_i(s)$ for $i \in \{1, \ldots, m\}$ and $\eta_i(s)$ for $i \in \{1, \ldots p\}$ such that

$$b_j^2 - \|f_l(s, g(l, s_0)\|^2 \quad = \quad \kappa_3(s, s_0) + \sum_{i=1}^{m} \zeta_i(s) F_{li}(s) + \sum_{i=1}^{p} \eta_i(s_0) G_{ji}(s_0).$$

Finding a Subset for Checking the Invariant Conditions

Suppose $C_j = \{s \in I_l \mid G_j(s) \geq 0\}$. In this case, we only have to find a polynomial $G_j(s)$. According to the generalized S-procedure, this problem can be formulated as follows: Find sums of squares $\rho_i(s)$, $\sigma_i(s)$, $\mu(s)$, $\gamma_i(s)$, $\lambda(s)$, $\zeta_i(s)$ and $\eta(s)$ for $i \in \{1, \ldots, m\}$ and polynomials $G_j(s)$, $\nu(s)$ and $\gamma_{m+1}(s)$ such that the following are sums of squares:

(a) $\frac{\partial F_{lj}(s)}{\partial s} \cdot f_l(s, g(l, s_0)) - \sum_{i=1}^{m} \rho_i(s) F_{li}(s) - \nu(s) F_{lj}(s) - \sum_{i=1}^{m} \sigma_i(s_0) F_{li}(s_0) + \mu(s_0) G_j(s_0)$,

(b) $\|s - s_0\|^2 - c_j^2 - \sum_{i=1}^{m} \gamma_i(s) F_{li}(s) - \gamma_{m+1}(s) F_{lj}(s) - \lambda(s_0) G_j(s_0)$, and

(c) $b_j^2 - \|f_l(s, g(l, s_0)\|^2 - \sum_{i=1}^{m} \zeta_i(s) F_{li}(s) - \eta(s_0) G_j(s_0)$.

5.3.4 Example

In this section, we illustrate how invariant verification of a PCHA can be partially automated on a simple example. Consider a one-dimensional system whose global state (e.g., position or velocity) needs to be regulated such that it stays within some safe ball with respect to a reference point, given by an external command. The reference point is given as an input to the system and may change throughout an execution. We assume that the distance between the reference point and the global state of the system at the time the reference point is received is not larger than δ. The system has the following variables: (a) a continuous state variable s of type $\mathbb{R}$ that represents the deviation of the system from the current reference point; (b) a discrete state variable *loc* of type $\mathbb{R}$ that represents the current reference point; (c) a command variable z of type $\mathbb{R}$ that stores the last external command, i.e., the reference point for the next control cycle; and (d) a control variable u of type $\mathcal{U} = \{a_1, a_2\}$ where $a_1 \in \mathbb{R}_-$ and $a_2 \in \mathbb{R}_+$ are system parameters.

Figure 5.3 shows the HIOA specification of this state regulator system. The control action occurs once every Δ time starting from time 0 where $\Delta \in \mathbb{R}_+$. This action updates the values of the variables s, *loc* and u as follows.

A. First, set the value of *loc* and s so that they correspond to the new reference point and the deviation of the system from the new reference point, respectively (lines 16–17).

```
signature                                          1   internal control
  internal control                                     pre now ≥ next                    14
  input update(z′ : Z)                             3   eff next := now + Δ;
                                                         s := s − z + loc;               16
variables                                          5     loc := z;
  internal s : ℝ := s0 = 0                               if s > 0 then u := a1           18
  internal discrete loc : ℝ, z : ℝ, u : {a1, a2}   7     else u := a2 fi
  internal now : ℝ≥0 := 0, next : ℝ≥0 := 0                                             20
                                                   9 trajectories
transitions                                              evolve d(now) = 1; d(s) = u     22
  input update(z′)                                 11    stop when now = next
  eff z := z′
```

Figure 5.3: The state regulator system with parameters $a_1 \in \mathbb{R}_-$, $a_2 \in \mathbb{R}_+$, $\Delta \in \mathbb{R}_+$, $\delta \in \mathbb{R}_{\geq 0}$ and $D \in \mathbb{R}$.

B. Based on the updated value of s, u is computed as follows (lines 18–19): If $s > 0$, then u is set to a_1. Otherwise, u is set to a_2.

Along a trajectory, the continuous state s evolves according to the differential equation $\dot{s} = u$ (line 22). That is, for any $l \in \mathcal{L}$, the function f_l of line 24 of Figure 5.1 is defined as $f_l(s, u) = u$.

Invariant For each mode $l \in \mathcal{L}$, we let $I_l = [-\delta + a_1\Delta, \delta + a_2\Delta]$. That is, the candidate invariant set I_l is defined by two boundary functions $F_{l1}(s) = s + \delta - a_1\Delta$ and $F_{l2}(s) = -s + \delta + a_2\Delta$. The overall candidate invariant set is then given by $\mathcal{I} \triangleq \{\mathbf{x} \in Q |\ F_{l1}(\mathbf{x}.s) \geq 0 \text{ and } F_{l2}(\mathbf{x}.s) \geq 0\}$.

Proving Invariance We use Theorem 5.3.1 to show that $\mathcal{I}$ is in fact an invariant of the system. Clearly, the initial state is contained in $\mathcal{I}$. To verify the control invariance condition of Lemma 5.3.1, we define $\hat{s} \triangleq s + loc$ to be the global state of the system. From the assumption on the distance between the reference point and the global state of the system at the time an update action occurs and periodicity control actions, it can be checked that when a control action occurs, $\hat{s} - z \in [-\delta + a_1\Delta, \delta + a_2\Delta]$. Hence, from the update rule of s (line 16), the control invariance condition of Lemma 5.3.1 is satisfied. Finally, define $C_1 \triangleq [0, \delta + a_2\Delta]$ and $C_2 \triangleq [-\delta + a_1\Delta, 0]$. We get that conditions (a)–(d) of Lemma 5.3.2 are satisfied with $c_1 = \delta - a_1\Delta$, $c_2 = \delta + a_2\Delta$, $b_1 = -a_1$, $b_2 = a_2$.

Automatically Finding an Invariant We consider the case where $a_1 = -1$ and $a_2 = 1$. Assume that an invariant I_l for any mode $l \in \mathbb{R}$ has the following form: $I_l = \{s \in \mathbb{R} \mid F_{l1}(s) \geq 0 \text{ and } F_{l2}(s) \geq 0\}$ where $F_{l1}(s) = s - \eta_1$, $F_{l2}(s) = -s + \eta_2$ and $\eta_1 \leq -\delta + a_1\Delta$ and $\eta_2 \geq \delta + a_2\Delta$ are constants that need to be computed such that all the conditions of Lemma 5.3.2 are satisfied. (From the previous proof, these constraints on η_1 and η_2 ensure that the initial state is contained in $\mathcal{I}$ and the control invariance condition of Lemma 5.3.1 are satisfied.)

To prove that I_l is in fact an invariant, we use the sets C_1 and C_2 of the following forms: $C_1 = \{s \in \mathbb{R} \mid G_1(s) \geq 0 \text{ and } F_{l2}(s) \geq 0\}$ and $C_2 = \{s \in \mathbb{R} \mid F_{l1}(s) \geq 0 \text{ and } G_2(s) \geq 0\}$ where $G_1(s) = s - \kappa_1$, $G_2(s) = -s + \kappa_2$ and κ_1 and κ_2 are constants to be determined.

Clearly, for any $s, s_0 \in \mathbb{R}$ and $l \in \mathcal{L}$, $\|f_l(s, g(l, s_0))\| = \|g(l, s_0)\| = 1$. Thus, condition (c) of Lemma 5.3.2 is satisfied with $b_j = 1$ for any sets C_j and I_l. With the particular form of

the sets C_1, C_2 and I_l we have previously chosen, it is straightforward to check that the problem of finding η_1, η_2, κ_1 and κ_2 such that all the conditions of Lemma 5.3.2 are satisfied for $j = 1$ is equivalent to finding η_1, η_2, κ_1 and κ_2 such that for all $s, s_0 \in \mathbb{R}$, the followings are satisfied:

(a) $(F_{l1}(s_0) < 0) \vee (F_{l2}(s_0) < 0) \vee (G_1(s_0) \geq 0) \vee (F_{l1}(s) \neq 0) \vee (F_{l2}(s) < 0) \vee (s_0 \leq 0)$,

(b) $\kappa_1 \leq \eta_2$, $\kappa_1 > \eta_1$ and $\kappa_1 - \eta_1 \geq \Delta$.

Note that condition (a) is obtained from condition (a) of Lemma 5.3.2 while condition (b) is obtained from conditions (b) and (d) of Lemma 5.3.2. Similarly, for $j = 2$, the following conditions need to be satisfied for all $s, s_0 \in \mathbb{R}$:

(c) $(F_{l1}(s_0) < 0) \vee (F_{l2}(s_0) < 0) \vee (G_2(s_0) \geq 0) \vee (F_{l1}(s) < 0) \vee (F_{l2}(s) \neq 0) \vee (s_0 > 0)$,

(d) $\kappa_2 \geq \eta_1$, $\kappa_2 < \eta_2$ and $\eta_2 - \kappa_2 \geq \Delta$.

Applying Farkas Lemma, condition (a) can be proved by finding a constant λ_1 and non-negative constants $\nu_1, \ldots, \nu_3$ and $\mu_1, \ldots, \mu_3$ such that

$$\nu_1 F_{l1}(s_0) + \nu_2 F_{l2}(s_0) - \mu_1 G_1(s_0) + \lambda_1 F_{l1}(s) + \nu_3 F_{l2}(s) + \mu_2 s_0 + \mu_3 = 0 \tag{5.3}$$

and at least one of the μ_1, μ_2, μ_3 is strictly positive. Similarly, the validity of condition (c) can be proved by finding a constant λ_2 and non-negative constants $\nu_4, \ldots, \nu_7$ and μ_4, μ_5 such that

$$\nu_4 F_{l1}(s_0) + \nu_5 F_{l2}(s_0) - \mu_4 G_2(s_0) + \nu_6 F_{l1}(s) + \lambda_2 F_{l2}(s) - \nu_7 s_0 + \mu_5 = 0 \tag{5.4}$$

and either $\mu_4 > 0$ or $\mu_5 > 0$ (or both).

Using the tool presented in [43], the unknowns that satisfy (5.3), (5.4) and conditions (b) and (d) are found for $\delta = 0.08$ and $\Delta = 0.02$ to be: $\eta_1 = -0.2$, $\eta_2 = 0.2$, $\kappa_1 = -0.1$, $\kappa_2 = 0.1$, $\nu_1 = 1$, $\nu_2 = 2$, $\mu_1 = 16$, $\lambda_1 = 0$, $\nu_3 = 0$, $\mu_2 = 17$, $\mu_3 = 1$, $\nu_4 = 0$, $\nu_5 = 0$, $\mu_4 = 20$, $\nu_6 = 0$, $\lambda_2 = 0$, $\nu_7 = 20$ and $\mu_5 = 2$. That is, the invariant set is given by $I_l = [-0.2, 0.2]$ (whereas the invariant set we have verified manually is given by $I_l = [-0.1, 0.1]$).

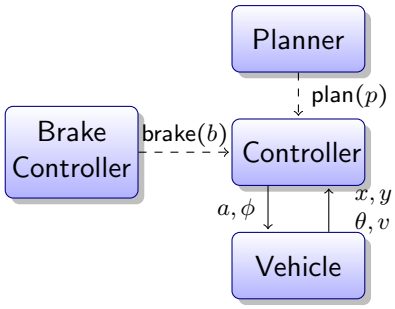


Figure 5.4: Planner-controller system.

5.4 Case Study: Alice

In this section, we describe an autonomous ground vehicle (Alice) consisting of the physical vehicle, modeled by the Vehicle automaton, and the controller, modeled by the Controller automaton (see Figure 5.4). Vehicle captures the position, orientation and velocity of the vehicle in the plane. Controller receives information about the state of the Vehicle and periodically computes the input steering (ϕ) and acceleration (a). It also receives an infinite[4] sequence of waypoints from a Planner and its objective is to compute a and ϕ such that the Vehicle (a) remains within a certain bounded distance e_{max} of the planned path, and (b) makes progress towards successive waypoints at a target speed. Property (a) together with the assumption (possibly guaranteed by Planner) that all planned paths are at least e_{max} distance away from obstacles, imply that the Vehicle does not collide with obstacles. While the Vehicle makes progress towards a certain waypoint, the subsequent waypoints may change owing to the discovery of new obstacles and changes in the mission plan. Finally, the Controller may receive an externally triggered brake input, to which it must react by slowing the Vehicle down. The Vehicle and Controller are modeled as HIOAs, but as we shall see shortly, the composed system has no inputs and in fact is a PCHA.

5.4.1 Vehicle

The Vehicle automaton of Figure 5.5 specifies the dynamics of the autonomous ground vehicle with acceleration (a) and steering angle (ϕ) as inputs. It has two parameters: (a) $\phi_{max} \in (0, \frac{\pi}{2})$ is the physical limit on the steering angle, and (b) L is the wheelbase. The main output variables of Vehicle are (a) x and y coordinates of the vehicle with respect

[4]The verification technique can be extended in an obvious way to handle the case where the Vehicle has to follow a finite sequence of waypoints and halt at the end.

```
variables                                                          1
output x : ℝ := x_0; y: ℝ := y_0;
  θ: ℝ := θ_0; v: ℝ := v_0                                         3
input a, φ: ℝ
                                                                   5
trajectories
evolve d(x) = v cos(θ)                                             7
  d(y) = v sin(θ)
  if |u.φ| ≤ φ_max                                                 9
    then d(θ) = (v/L) tan(φ)
    else d(θ) = (v/L) tan((φ/|φ|) φ_max) fi                        11
  if v > 0 ∨ a ≥ 0
    then d(v) = a                                                  13
    else d(v) = 0 fi
```

Figure 5.5: Vehicle.

to a global coordinate system, (b) orientation θ of the vehicle with respect to the positive direction of the global x axis, and (c) vehicle's velocity v. These variables evolve according to the differential equations of lines 7–14. Two aspects of this Vehicle model are noteworthy: (i) In determining the orientation of the Vehicle, if the input steering angle ϕ is greater than the maximum limit ϕ_{max}, then the maximum steering in the correct direction is applied. (ii) The acceleration can be negative only if the velocity is positive, and therefore the Vehicle cannot move backwards. This Vehicle model requires bounds on minimum and maximum acceleration, however, the Controller ensures that the input acceleration is always within such a bound. It is assumed that the Vehicle can execute any valid command without delay.

5.4.2 Controller

Figure 5.6 shows the HIOA specification of the Controller automaton that reads the state of the Vehicle periodically and issues acceleration and steering outputs to achieve the aforementioned goals. Controller is parameterized by: (a) the sampling period $\Delta \in \mathbb{R}_+$, (b) the target speed $v_T \in \mathbb{R}_{\geq 0}$, (c) proportional control gains $k_1, k_2 > 0$, (d) a constant $\delta > 0$ relating the maximum steering angle and the speed, that is, while the Vehicle is moving at speed v, the maximum steering angle is given by δv, and (e) maximum and braking accelerations $a_{max} > 0$ and $a_{brake} < 0$. Restricting the maximum steering angle instead of the maximum steering rate is a simplifying but conservative assumption. Given a constant relating the maximum steering rate and the speed, there exists δ as defined above that guarantees that the maximum steering rate requirement is satisfied.

A *path* is an infinite sequence of points $p_1, p_2, \ldots$ where $p_i \in \mathbb{R}^2$, for each i. The main

```
signature
  input plan(p:Seq[ℝ]); brake(b : On, Off)
  internal main

variables
  input x, y, θ, v: ℝ
  output a, φ: ℝ := (0, 0)
  internal brake: {On, Off} := Off
    path: Seq[ℝ²] := arbitrary
    new_path: Seq[ℝ²] := path
    seg: ℕ := 1
    e1, e2, d : ℝ := [e1,0, e2,0, d0]
    now: ℝ := 0; next:ℝ≥0 := 0

transitions
  input plan(p)
  eff new_path := p

  input brake(b)
  eff brake := b

  internal main
  pre now = next
  eff next := now + Δ
  if path ≠ new_path ∨ d ≤ 0 then
    if path ≠ new_path
      then seg := 1; path := new_path
    elseif d ≤ 0
      then seg := seg + 1 fi
      let p⃗ = [ path[seg+1].x − path[seg].x ]
              [ path[seg+1].y − path[seg].y ]
      q⃗ = [  path[seg+1].y − path[seg].y  ]
          [ −(path[seg+1].x − path[seg].x) ]
      r⃗ = [ path[seg+1].x − x ]
          [ path[seg+1].y − y ]
    e1 := 1/‖q⃗‖ q⃗ · r⃗
    e2 := θ − ∠p⃗
    d := 1/‖p⃗‖ p⃗ · r⃗
  fi

  let φd = −k1 e1 − k2 e2
  φ = φd/|φd| min(δ × v, |φd|)

  if brake = On then a := abrake
  elseif brake = Off ∧ v < vT
    then a := amax
  else a := 0 fi

trajectories
  evolve d(now) = 1
    d(e1) = v sin(e2)
    d(e2) = v/L tan(φ)
    d(d) = -v cos(e2)
  stop when now = next
```

Figure 5.6: Controller with parameters $v_T, k_1, k_2 \in \mathbb{R}_{\geq 0}$, $\delta, \Delta \in \mathbb{R}_+$ and $a_{brake} < 0$.

state variables of Controller are the following:

(a) *brake* and *new_path* are command variables that store the information received through the most recent brake (*On* or *Off*) and plan (a path) actions.

(b) *path* is the current path being followed by Controller.

(c) *seg* is the index of the last waypoint visited in the current *path*. That is, *seg* + 1 is the index of the current waypoint. The straight line joining $path[seg]$ and $path[seg + 1]$ is called the *current segment*.

(d) *deviation* e_1 is the signed perpendicular distance from the current position of the Vehicle to the current segment (see Figure 5.7).

(e) *disorientation* e_2 is the difference between the current orientation (θ) of the Vehicle and the angle of the current segment.

(f) *waypoint-distance* d is the signed distance of the Vehicle to the current waypoint measured parallel to the current segment.

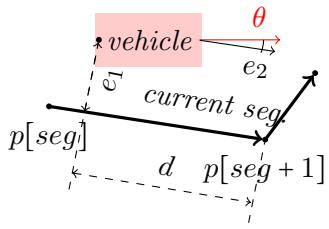


Figure 5.7: Deviation and disorientation.

The brake(b) action is an externally controlled input action that informs the Controller about the application of an external brake (b = *On*) or the removal of the brake (b = *Off*). When brake(b) occurs, b is recorded in the command variable *brake*. The plan(p) action is controlled by the external Planner and it informs the Controller about a newly planned path p. When this action occurs, the path p is recorded in the variable *new_path*. The main action occurs once every Δ time starting from time 0. This action updates the values of the variables $e_1, e_2, d, path, seg, a$ and ϕ as follows:

A. If *new_path* (obtained from the Planner) is different from *path*, then *seg* is set to 1 and *path* is set to *new_path* (line 27).

B. If *new_path* is the same as *path* and the waypoint-distance d is less than or equal to 0, then *seg* is set to $seg + 1$ (line 29).

C. For both of the above cases several temporary variables are computed that are in turn used to update e_1, e_2, d as specified in lines 33–35; otherwise these variables remain unchanged.

D. The steering output to the Vehicle ϕ is computed using a proportional control law saturated at $\delta \times v$ (for the mechanical protection of the steering). That is, the magnitude of the steering output ϕ is set to the minimum of $|-k_1e_1 - k_2e_2|$ and $v \times \delta$ (line 39).

E. The acceleration output a is computed using a "bang-bang" control law. If *brake* is *On*, then a is set to the braking deceleration a_{brake}; otherwise, it executes a_{max} until the Vehicle reaches the target speed, at which point a is set to 0.

Along a trajectory, the evolution of the variables are specified by the differential equations on lines 48–50. These differential equations are derived from the update rules described above and the differential equations governing the evolution of x, y, θ and v.

5.4.3 Complete System

Let $\mathcal{A}$ be the composition of the Controller and the Vehicle automata. The continuous state of $\mathcal{A}$ is defined by the valuations of $x, y, \theta, v, e_1, e_2$ and d of Vehicle and Controller. For convenience, we define a single derived variable s of type $\mathcal{X} = \mathbb{R}^7$ encapsulating all these variables. The discrete state of $\mathcal{A}$ is defined by the valuations of $brake, path$ and seg of Controller. A derived variable loc of type $\mathcal{L} = \texttt{Tuple}[\{On, Off\}, \texttt{Seq}[\mathbb{R}^2], \mathbb{N}]$ is defined encapsulating all these variables. It can be checked easily that the composed automaton $\mathcal{A}$ is a PCHA. Appendix 5.A describes the variables, actions, state transition functions of the corresponding PCHA.

5.5 Analysis of the System

The informally stated goals of the system translate to the following subgoals:

A. (*safety)* At all reachable states of $\mathcal{A}$, the deviation (e_1) of the Vehicle is upperbounded by e_{max}, where e_{max} is determined in terms of system parameters.

B. (*waypoint progress)* The Vehicle reaches successive waypoints.

First, in Section 5.5.1 and 5.5.2, we define a family $\{\mathcal{I}_k\}_{k\in\mathbb{N}}$ of subsets of $Q_{\mathcal{A}}$ and using Lemma 5.3.2 and Lemma 5.3.3, we conclude that they are invariant with respect to the control-free execution fragments of $\mathcal{A}$. From the specification of main action, we see that the discrete state changes only occur if $path \neq new\_path$ or waypoint-distance $d \leq 0$ (i.e., the Vehicle has reached the end of the current segment). Hence, using Theorem 5.3.1, we conclude that any execution fragment starting in $\mathcal{I}_k$ remains within $\mathcal{I}_k$, provided that *path* and current segment do not change.

In Section 5.5.3, we establish the following *segment progress* property: There exist certain threshold values of deviation, disorientation and waypoint-distance such that from any state $\mathbf{x}$ with greater deviation, disorientation and waypoint-distance, the Vehicle reduces its deviation and disorientation with respect to the current segment, while making progress towards its current waypoint. This intermediate result is proved by showing that starting from $\mathcal{I}_k$, $\mathcal{I}_{k+1} \subseteq \mathcal{I}_k$ is reached in a finite amount of time and for k smaller than the threshold value k^*, $\mathcal{I}_{k+1}$ is strictly contained in $\mathcal{I}_k$. Finally, in Section 5.5.4, we prove an invariance of $\mathcal{I}_0$ and derive geometric properties of *planner* paths that can be followed by $\mathcal{A}$

safely. These geometric properties specify the minimum length of a path segment and the relationship between the segment length and the maximum difference between consecutive segment orientations and are derived from the segment progress property. An invariance of $\mathcal{I}_0$ provides a proof certificate that $\mathcal{A}$ satisfies the safety property (A) and the waypoint progress property (B). Since Alice's original parameters violate the sufficient conditions for an invariance of $\mathcal{I}_0$, it is not guaranteed that the behavior of Alice satisfies these subgoals. In fact, during the NQE of the 2007 DARPA Urban Challenge, Alice violated the safety property (A), leading to the stuttering behavior.

5.5.1 Family of Invariants

We define, for each $k \in \mathbb{N}$, the set $\mathcal{I}_k$ that bounds the deviation e_1 of the Vehicle to be within $[-\epsilon_k, \epsilon_k]$. This bound on deviation alone, of course, does not give us an inductive invariant. If the deviation is ϵ_k and the Vehicle is highly disoriented, then it would violate $\mathcal{I}_k$. Thus, $\mathcal{I}_k$ also bounds the disorientation such that the steering angle computed based on the proportional control law is within $[-\phi_k, \phi_k]$. To prevent the Vehicle from not being able to turn at low speed and to guarantee that the execution speed of the Controller is fast enough with respect to the speed of the Vehicle, $\mathcal{I}_k$ also bounds the speed of the Vehicle. Formally, $\mathcal{I}_k$ is defined in terms of $\epsilon_k, \phi_k \geq 0$ as

$$\mathcal{I}_k \triangleq \{\mathbf{x} \in Q \mid \forall i \in \{1, \ldots 6\}, F_{k,i}(\mathbf{x}.s) \geq 0\} \tag{5.5}$$

where $F_{k,1}, \ldots, F_{k,6} : \mathbb{R}^7 \to \mathbb{R}$ are defined as follows:

$$F_{k,1}(s) = \epsilon_k - s.e_1; \qquad F_{k,2}(s) = \epsilon_k + s.e_1; \tag{5.6}$$

$$F_{k,3}(s) = \phi_k + k_1 s.e_1 + k_2 s.e_2; \qquad F_{k,4}(s) = \phi_k - k_1 s.e_1 - k_2 s.e_2; \tag{5.7}$$

$$F_{k,5}(s) = v_{max} - s.v; \qquad F_{k,6}(s) = \delta s.v - \phi_b. \tag{5.8}$$

Here $v_{max} = v_T + \Delta a_{max}$ and $\phi_b > 0$ is an arbitrary constant. As we shall see shortly, the choice of ϕ_b affects the minimum speed of the Vehicle and also the requirements of a brake action. We examine a state $\mathbf{x} \in \mathcal{I}_k$, that is, $F_{k,i}(\mathbf{x}.s) \geq 0$ for any $i \in \{1, \ldots, 6\}$. $F_{k,1}(s), F_{k,2}(s) \geq 0$ means $s.e_1 \in [-\epsilon_k, \epsilon_k]$. $F_{k,3}(s), F_{k,4}(s) \geq 0$ means that the steering angle computed based on the proportional control law is within the range $[-\phi_k, \phi_k]$. Further, if

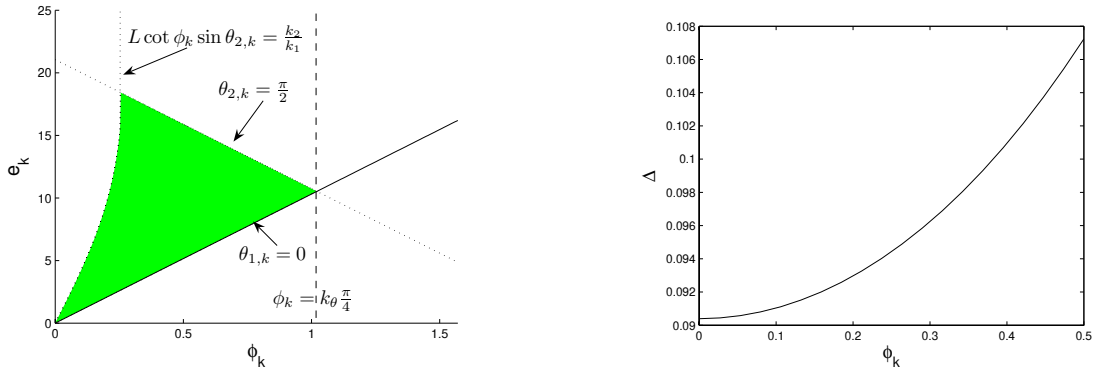
$\phi_k \leq \phi_{max}$, then the computed steering satisfies the physical constraint of the Vehicle. If, in addition, we have $\phi_b \geq \phi_k$ and $F_{k,6}(s) \geq 0$, then the Vehicle actually executes the computed steering command. $F_{k,5}(s) \geq 0$ means that the speed of the Vehicle is at most v_{max}.

For each $k \in \mathbb{N}$, we define

$$\theta_{k,1} = \frac{k_1}{k_2}\epsilon_k - \frac{1}{k_2}\phi_k \quad \text{and} \quad \theta_{k,2} = \frac{k_1}{k_2}\epsilon_k + \frac{1}{k_2}\phi_k. \tag{5.9}$$

That is, $\theta_{k,1}$ and $\theta_{k,2}$ are the values of e_2 at which the proportional control law yields the steering angle of ϕ_k and $-\phi_k$, respectively, given that the value of e_1 is $-\epsilon_k$. From the above definitions, we make the following observations about the boundary of the $\mathcal{I}_k$ sets: for any $k \in \mathbb{N}$ and $\mathbf{x} \in \mathcal{I}_k$, (a) $\mathbf{x}.e_2 \in [-\theta_{k,2}, \theta_{k,2}]$, (b) $F_{k,1}(\mathbf{x}.s) = 0$ implies $\mathbf{x}.e_2 \in [-\theta_{k,2}, -\theta_{k,1}]$, (c) $F_{k,2}(\mathbf{x}.s) = 0$ implies $\mathbf{x}.e_2 \in [\theta_{k,1}, \theta_{k,2}]$, (d) $F_{k,3}(\mathbf{x}.s) = 0$ implies $\mathbf{x}.e_2 \in [-\theta_{k,2}, \theta_{k,1}]$, and (e) $F_{k,4}(\mathbf{x}.s) = 0$ implies $\mathbf{x}.e_2 \in [-\theta_{k,1}, \theta_{k,2}]$.

We assume that ϕ_b and all the $\epsilon_k's$ and ϕ_k's satisfy the following assumptions that are derived from physical and design constraints on the Controller. The region in the ϕ_k,ϵ_k plane that satisfies Assumption 5.5.1 is shown Figure 5.8.



(a) (b)

Figure 5.8: (a) The set of (ϵ_k, ϕ_k) that satisfies Assumptions 5.5.1 (c) and (d) and are represented by the green region. (b) The relationship between the maximum bound on Δ and ϕ_k for $\epsilon_k = \frac{1}{k_1}\phi_k$.

Assumption 5.5.1. (Vehicle and controller design) (a) $\phi_k \leq \phi_b \leq \phi_{max}$ and $\phi_k < \frac{\pi}{2}$, (b) $0 \leq \theta_{k,1} \leq \theta_{k,2} < \frac{\pi}{2}$, (c) $L\cot\phi_k \sin\theta_{k,2} < \frac{k_2}{k_1}$, (d) $\Delta \leq \frac{c}{b}$ where $c = \frac{1}{\sqrt{k_1^2+k_2^2}}(\phi_k - \tilde{\phi})$, $b = v_{max}\sqrt{\sin^2\theta_{k,2} + \frac{1}{L^2}\tan^2(\tilde{\phi})}$ and $\tilde{\phi} = \cot^{-1}\left(\frac{k_2}{k_1 L \sin\theta_{k,2}}\right)$,[5] and (e) $\frac{\tan\phi_k}{2L} v_{max}\Delta \leq \frac{\pi}{2}$.

[5]Using Assumption 5.5.1(c), it can be shown that $\tilde{\phi} < \phi_k$ so $\frac{c}{b} > 0$.

If the Vehicle is forced to slow down too much at the boundary of an $\mathcal{I}_k$ by the brakes, then it may not be able to turn enough to remain inside $\mathcal{I}_k$. Thus, in verifying the above properties we need to restrict our attention to executions in certain *good* brake controllers in which brake inputs do not occur at low speeds and are not too persistent. This is formalized by the next definition.

Definition 5.5.1. A brake controller is *good* if its composition with Controller gives rise to an execution $\alpha = \tau_0 a_1 \tau_1 a_2 \ldots$ that satisfies: If a brake(On) action occurs at time t, then for any $i \in \mathbb{N}$ such that $t \in dom(\tau_i)$, (a) $(\tau_i \downarrow v)(t) > \frac{\phi_b}{\delta} + \Delta|a_{brake}|$, and (b) brake($Off$) must occur within time $t + \frac{1}{|a_{brake}|}((\tau_i \downarrow v)(t) - \frac{\phi_b}{\delta} - \Delta|a_{brake}|)$.

We assume that the brake controller satisfies the above assumption and for the remainder of this section, we only consider executions in *good* brake controllers. A state $\mathbf{x} \in Q_{\mathcal{A}}$ is reachable if there exists an execution α in a *good* brake controller with $\alpha.\mathsf{lstate} = \mathbf{x}$.

5.5.2 Invariance Property

We fix a $k \in \mathbb{N}$ for the remainder of the section and denote $\mathcal{I}_k, F_{k,i}$ as $\mathcal{I}$ and F_i, respectively, for $i \in \{1, \ldots, 6\}$. As in Lemma 5.3.2, we define $I = \{s \in \mathcal{X} \mid F_i(s) \geq 0\}$ and for each $i \in \{1, \ldots, 6\}$, $\partial I_i = \{s \in \mathcal{X} \mid F_i(s) = 0\}$ and let the functions $f_1, f_2, \ldots, f_7 : \mathbb{R}^7 \times \mathbb{R}^2 \to \mathbb{R}$ describe the evolution of x, y, θ, v, e_1, e_2 and d, respectively. We prove that I satisfies the control-free invariance condition of Lemma 5.3.1 by applying Lemma 5.3.2.

First, we check that the conditions in Lemma 5.3.2 are satisfied. This analysis appears in Appendix 5.B. It does not involve solving differential equations but relies on algebraic simplification of the expressions defining the vector fields and the boundaries $\{\partial I_i\}_{i \in \{1,\ldots 6\}}$ of the invariant set.

The next lemma shows that the subtangential, bounded distance and bounded speed conditions (of Lemma 5.3.2) are satisfied. The proof for $j = 5$ is presented here as an example. The rest of the proof is provided in Appendix 5.B.

Lemma 5.5.1. *For each $l \in \mathcal{L}$ and $j \in \{1, \ldots, 6\}$, the subtangential, bounded distance and bounded speed conditions (of Lemma 5.3.2) are satisfied.*

Proof. Define $C_5 \triangleq \{s \in I \mid s.v \leq v_T\}$. We apply Lemma 5.3.3 to prove the bounded distance and the bounded speed conditions. First, note that the projection of I onto the (e_1, e_2, v)

space is compact and C_5 is closed. Let $\mathcal{U}_I = \{g(l,s) \mid l \in \mathcal{L},\ s \in I\}$. From the definition of I, it can be easily checked that f is continuous in $I \times \mathcal{U}_I$. In addition, $s.v = v_{max}$ for any $s \in \partial I_5$. Since $a_{max}, \Delta > 0$, $v_{max} = v_T + \Delta a_{max} > v_T$. Therefore, $C_5 \cap \partial I_5 = \varnothing$. Hence, from Lemma 5.3.3, the bounded distance and bounded speed conditions are satisfied. To prove the subtangential condition, we pick an arbitrary $s \in \partial I_5$ and $s_0 \in I \setminus C_5$. From the definitions of I and C_5, $v_T < s_0.v \le v_{max}$. Therefore, for any $l \in \mathcal{L}$, either $f_4(s, g(l, s_0)) = 0$ or $f_4(s, g(l, s_0)) = a_{brake}$ and we can conclude that $\frac{\partial F_5}{\partial s} \cdot f(s, g(l, s_0)) = -f_4(s, g(l, s_0)) \ge 0$. □

From the definition of each C_j, we can derive the lower bound c_j on the distance from C_j to ∂I_j and the upper bound b_j on the length of the vector field f where the control variable u is evaluated when the continuous state $s \in C_j$. Using these bounds and Assumption 5.5.1(d), we prove the sampling rate condition.

Lemma 5.5.2. *For each $l \in \mathcal{L}$, the sampling rate condition is satisfied.*

Thus, conditions (a)–(d) of Lemma 5.3.2 are satisfied. From Theorem 5.3.1, we obtain that good execution fragments of $\mathcal{A}$ preserve invariance of $\mathcal{I}$, provided that the path and current segment do not change over the fragment.

Proposition 5.5.1. *For any* plan*-free execution fragment β starting at a state $\mathbf{x} \in \mathcal{I}$ and ending at $\mathbf{x}' \in Q_{\mathcal{A}}$, if $\mathbf{x}.path = \mathbf{x}.new\_path$ and $\mathbf{x}.seg = \mathbf{x}'.seg$, then $\mathbf{x}' \in \mathcal{I}$.*

Proof. From Lemma 5.5.1 and 5.5.2, we see that all the conditions in Lemma 5.3.2 are satisfied. Thus, we can conclude that the control-free invariance condition of Lemma 5.3.1 is satisfied. In addition, from the specification of main action, we see that a discrete transition in the continuous state s only occurs when $path \neq new\_path$ (i.e., a new path is received) or $s.d \le 0$ (i.e., the Vehicle has reached the end of the current segment). Hence, if a closed execution β does not contain a plan action, $\beta.\mathsf{fstate}\lceil path = \beta.\mathsf{fstate}\lceil new\_path$ and $\beta.\mathsf{lstate}\lceil seg = \beta.\mathsf{fstate}\lceil seg$, then a discrete transition in the continuous state s does not occur in β. Applying Theorem 5.3.1, we get the desired result. □

5.5.3 Segment Progress

In this section, we establish the segment progress property, i.e., there exist certain threshold values of deviation, disorientation and waypoint-distance such that from any state $\mathbf{x}$ with greater deviation, disorientation and waypoint-distance, the Vehicle reduces its deviation

and disorientation with respect to the current segment, while making progress towards its current waypoint. First, we prove the progress property over a pasted trajectory τ between any two main actions. That is, suppose right after an occurrence of a main action, $\mathbf{x} \in \mathcal{I}_k$ for some $k \in \mathbb{N}$. Then, right before an occurrance of the next main action, $\mathbf{x} \in \mathcal{I}_{k+1}$ where $\mathcal{I}_{k+1} \subseteq \mathcal{I}_k$ and if k is less than some threshold k^*, then $\mathcal{I}_{k+1}$ is strictly contained in $\mathcal{I}_k$.

Next, in Lemma 5.5.4, we compute the bound d^* on the maximum change in the value of the waypoint-distance d over τ. Given the progress property over τ and the bound d^*, we can then establish the segment progress property defined at the beginning of Section 5.5. That is, starting from a state $\mathbf{x}$ and ending at $\mathbf{x}'$, if $\mathbf{x} \in \mathcal{I}_k$, then $\mathbf{x}' \in \mathcal{I}_{k+n}$ where an integer $n \geq 0$ depends on $\mathbf{x}.d - \mathbf{x}'.d$ and the system parameters, provided that path and current segment do not change. Furthermore, if $\mathbf{x}.d - \mathbf{x}'.d$ is large enough, then n is strictly positive.

By solving the differential equation that describes the evolution of e_1 and e_2 along τ and using the periodicity of main actions, the next lemma provides the desired progress property over τ. The complete proof appears in Appendix 5.C.

Lemma 5.5.3. *Suppose* $\tau.\mathsf{fstate} \in \mathcal{I}_k$ *for some* $k \in \mathbb{N}$. *Then* $\tau.\mathsf{lstate} \in \mathcal{I}_{k+1}$ *whose parameters* ϵ_{k+1} *and* ϕ_{k+1} *are given by* $\epsilon_{k+1} = \epsilon_k - \hat{\epsilon}_k$ *and* $\phi_{k+1} = \phi_k - \hat{\phi}_k$ *for some* $\hat{\epsilon}_k, \hat{\phi}_k \geq 0$. *In addition, there exists a natural number* k^* *such that for any* $k < k^*$, $\hat{\epsilon}_k$ *and* $\hat{\phi}_k$ *are strictly positive, that is,* $I_{k+1} \subsetneq I_k$.

The precise definitions of $\hat{\epsilon}_k$, $\hat{\phi}_k$ and k^* are given in Appendix 5.C. The plots showing the progress in the deviation and disoriantation are shown in Figure 5.9(a) and Figure 5.9(b), respectively.

The following lemma provides the value of the bound d^* on the maximum change in the value of d over τ.

Lemma 5.5.4. *Suppose* $\tau.\mathsf{fstate} \in \mathcal{I}_k$ *for some* $k \in \mathbb{N}$. *For any* $t \in dom(\tau)$, $|(\tau \lceil d)(t) - \tau.\mathsf{fstate} \lceil d| \leq d^*$ *where* $d^* = v_{max}\Delta$.

Proof. From Proposition 5.5.1, the definitions of F_5 and F_6 and the definition of f_7 that describes the evolution of d, we get that $\max_{s,s_0 \in I} \|f_7(s, g(l, s_0))\| \leq v_{max}$. Since $dom(\tau) = [0, \Delta]$, we get $|(\tau \downarrow d)(t) - \tau.\mathsf{fstate} \lceil d| \leq \max_{s,s_0 \in I} \|f_7(s, g(l, s_0))\|\Delta \leq v_{max}\Delta$. □

Using Lemma 5.5.3 and Lemma 5.5.4, we establish the relationship between the progress of $\mathcal{I}_k$'s and the decrease in the value of d. The complete proof can be found in Appendix 5.C.

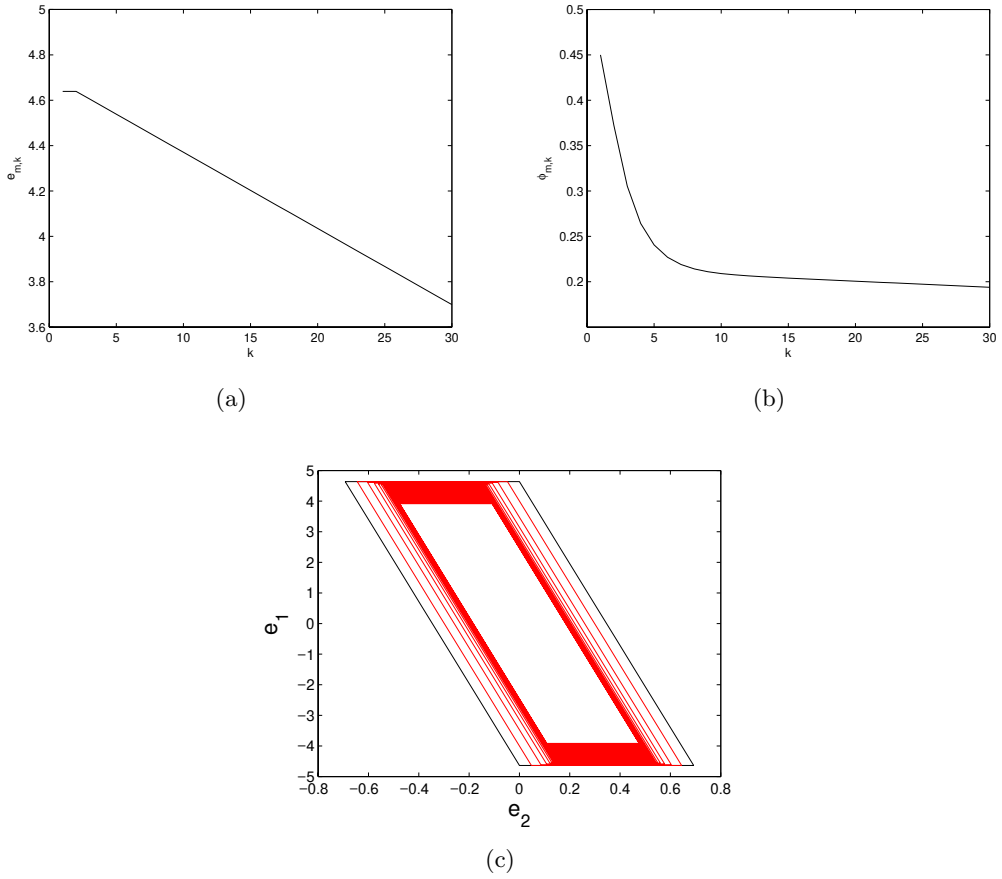


Figure 5.9: The progress in (a) deviation and (b) disorientation. (c) A sequence of shrinking $\mathcal{I}_k$'s visited by $\mathcal{A}$ in making progress towards a waypoint.

Lemma 5.5.5. *For each $k \in \mathbb{N}$, starting from any reachable state $\mathbf{x} \in \mathcal{I}_k$ such that $\mathbf{x}.d > v_{max}\Delta$, $\mathbf{x}.path = \mathbf{x}.new\_path$ and $\mathbf{x}.next = \mathbf{x}.now$, any* plan*-free execution fragment β with $\beta.\mathsf{ltime} = \Delta$ satisfies $\beta.\mathsf{lstate} \in \mathcal{I}_{k+1}$ and $\beta.\mathsf{lstate} \lceil d \geq \mathbf{x}.d - v_{max}\Delta$.*

Finally, we conclude the section by establishing the segment progress property defined at the beginning of Section 5.5.

Proposition 5.5.2. *For each $k \in \mathbb{N}$, starting from any reachable state $\mathbf{x} \in \mathcal{I}_k$, any reachable state $\mathbf{x}'$ is in $\mathcal{I}_{k+n}$ where $n = \max(\lfloor \frac{\mathbf{x}.d - \mathbf{x}'.d}{v_{max}\Delta} \rfloor - 1, 0)$, provided that path and current segment do not change.*

Proof. Consider an arbitrary closed execution fragment β starting at $\mathbf{x}$ and ending at $\mathbf{x}'$. Since by assumption, β is a plan-free execution fragment such that $\beta.\mathsf{lstate} \lceil path = \beta.\mathsf{fstate} \lceil new\_path$ and $\beta.\mathsf{lstate} \lceil seg = \beta.\mathsf{fstate} \lceil seg$, from Proposition 5.5.1, we know that $\beta.\mathsf{lstate} \in$

$\mathcal{I}_k$. This completes the proof for the case where $\lfloor \frac{\mathbf{x}.d - \mathbf{x}'d}{v_{max}\Delta} \rfloor - 1 \leq 0$.

Next, consider the case where $\lfloor \frac{\mathbf{x}.d - \mathbf{x}'.d}{v_{max}\Delta} \rfloor - 1 > 0$. From the structure of a PCHA, we see that $next = now$ every Δ time. So, the first state in β such that $next = now$ occurs no later than time Δ. Using Lemma 5.5.4, we see that at this state, $d \geq \mathbf{x}.d - v_{max}\Delta$. Applying Lemma 5.5.5 and using an invariance of $\mathcal{I}_k$ for any k proved in Proposition 5.5.1, we get that $\beta_1.\mathsf{lstate} \in \mathcal{I}_{k+n}$ where $n = \lfloor \frac{\mathbf{x}.d - v_{max}\Delta - \mathbf{x}'.d}{v_{max}\Delta} \rfloor$. □

A sequence of shrinking $\mathcal{I}_k$'s visited by $\mathcal{A}$ in making progress towards a waypoint is shown in Figure 5.9(c).

5.5.4 Safety and Waypoint Progress: Identifying Safe *Planner* Paths

In this section, we derive a sufficient condition on *planner* paths that can be safely followed with respect to a candidate invariant set $\mathcal{I}_0$ whose parameters $\epsilon_0 \in [0, e_{max}]$ and $\phi_0 \in [0, \phi_{max}]$ satisfy Assumption 5.5.1 and are chosen such that $\mathcal{I}_0$ contains the initial state $Q_{0\mathcal{A}}$. Then, we prove an invariance of $\mathcal{I}_0$ and conclude that the safety and waypoint progress properties (A) and (B) defined at the beginning of Section 5.5 are satisfied.

The proof is structured as follows. First, we consider an execution fragment where path does not change and starting with waypoint-distance not shorter than some threshold D^*. Lemma 5.5.6 uses the segment progress property established in Section 5.5.3 to prove that this execution fragment preserves an invariance of $\mathcal{I}_0$. Then, in Lemma 5.5.7 and Lemma 5.5.8, we show that right after the path changes, the waypoint-distance is not shorter than D^* and the state of $\mathcal{A}$ remains in $\mathcal{I}_0$. Using these results, Lemma 5.5.9 concludes that an execution fragment that updates the path exactly once by the first main action preserves an invariance of $\mathcal{I}_0$. Finally, we use Lemma 5.5.6 and Lemma 5.5.9 to conclude the section that $\mathcal{I}_0$ is in fact an invariant of $\mathcal{A}$ and with this result, we conclude that the system satisfies the safety and waypoint progress properties (A) and (B) defined at the beginning of Section 5.5.

The following assumption provides sufficient conditions for *planner* paths that can be safely followed. The key idea in the condition is: *Longer path segments can be succeeded by sharper turns*. Following a long segment, the Vehicle reduces its deviation and disorientation by the time it reaches the end; thus, it is possible for the Vehicle to turn more sharply at the end without breaking an invariance of $\mathcal{I}_0$.

Assumption 5.5.2. (*Planner* paths) Let $p_0, p_1, \ldots$ be a *planner* path; for $i \in \mathbb{N}$, let λ_i be the length of the segment $\overline{p_i p_{i+1}}$ and σ_i be the difference in orientation of $\overline{p_i p_{i+1}}$ and that of $\overline{p_{i+1} p_{i+2}}$. Then, for each $i \in \{0, 1, \ldots\}$,

(a) $\lambda_i \geq 2v_{max}\Delta + \epsilon_0$.

(b) Let $n = \lfloor \frac{\lambda_i - \epsilon_0 - 2v_{max}\Delta}{v_{max}\Delta} \rfloor$. Then, λ_i and σ_i satisfy the following conditions:

$$\epsilon_n \leq \frac{1}{|\cos\sigma_i|}(\epsilon_0 - v_{max}\Delta|\sin\sigma_i|), \tag{5.10}$$

$$\phi_n \leq \phi_0 - k_1 v_{max}\Delta \sin|\sigma_i| - k_1\epsilon_n(1 - \cos\sigma_i) - k_2|\sigma_i|, \tag{5.11}$$

where, given ϵ_0 and ϕ_0, ϵ_n and ϕ_n are defined recursively for any $n > 0$ by $\epsilon_n = \epsilon_{n-1} - \hat{\epsilon}_{n-1}$ and $\phi_n = \phi_{n-1} - \hat{\phi}_{n-1}$ where for each $k \in \mathbb{N}$, $\hat{\epsilon}_k$ and $\hat{\phi}_k$ are defined in Lemma 5.5.3.

The relationship between λ and the maximum value of σ which satisfies this assumption is shown in Figure 5.10.

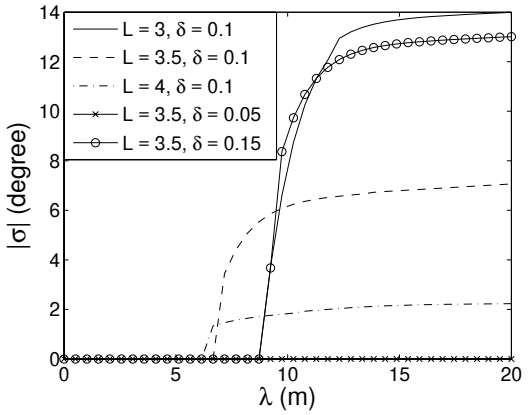


Figure 5.10: Segment length vs. maximum difference between consecutive segment orientations, for different values of L and δ.

Remark 5.5.1. The choice of ϵ_0's and ϕ_0's affects both the requirements on a safe path (Assumption 5.5.2) and the definition of a *good* brake controller (Definition 5.5.1). Larger ϵ_0's and ϕ_0's allow sharper turns in planned paths but force brakes to occur only at higher speeds. That is, relaxing the constraint on a path results in the tighter constraint on a brake action. This tradeoff is related to the design flaw of Alice as discussed in Section 1.1. Without having quantified the tradeoff, we inadvertently allowed a path to have sharp turns and also brakes at low speeds—thus violating safety.

To establish that $\mathcal{I}_0$ is an invariant of $\mathcal{A}$, we further assume that (a) new *planner* paths begin at the current position, (b) Vehicle is not too disoriented with respect to new paths, and (c) Vehicle speed is not too high as stated in Assumption 5.5.3.

Assumption 5.5.3. (plan action and new path)

(a) Any new path $p = p_1 p_2 \ldots$ satisfies $p_1 = [x_p, y_p]$ where x_p and y_p are the values of the variable x and y, respectively, when the path is received (i.e., when the plan action occurs). That is, for any new input path, the path must begin at the current position of the Vehicle.

(b) Let v_p and θ_p be the speed and the orientation of the Vehicle, respectively, when a plan action occurs. Then, $v_p < \frac{\epsilon_0}{\Delta\sqrt{1+\sin^2\theta_{0,2}}} - a_{max}\Delta$ where given ϵ_0 and ϕ_0, $\theta_{0,2}$ is defined as in (5.9). In addition, let $p = p_1 p_2 \ldots$ be the received path and let $\vec{p}$ be the vector that represents a straight line defined by p_1 and p_2. Then,

$$|\measuredangle \vec{p} - \theta_p| \leq \frac{\phi_0}{k_2} - (v_p + a_{max}\Delta)\Delta\left(\frac{k_1}{k_2}\sqrt{1+\sin^2\theta_{0,2}} + \frac{\tan\phi_0}{L}\right).$$

First, we consider an execution fragment where path does not change and starting with a large enough waypoint-distance. Using the progress property established in Section 5.5.3, the update rule of the variable *seg* and Lemma 5.5.4, we can show that before switching to the next segment, $\mathbf{x} \in \mathcal{I}_n$ where $n \geq 0$ depends on the segment length. (See Appendix 5.D for the complete proof.) Since we restrict the sharpness of the turn with respect to segment length (Assumption 5.5.2), we can then conclude that this execution fragment preserves an invariance of $\mathcal{I}_0$.

Lemma 5.5.6. *Consider a* plan*-free execution fragment β starting at a state $\mathbf{x} \in \mathcal{I}_0$. Suppose $\mathbf{x}.path = \mathbf{x}.new\_path$ and $\mathbf{x}.d \geq D^*$ where $D^* = \lambda_1 - \epsilon_0 - v_{max}\Delta$ and λ_1 is the length of the segment $\mathbf{x}.seg$. Then $\beta.\mathsf{lstate} \in \mathcal{I}_0$.*

The next two lemmas show that Assumption 5.5.3 is sufficient to guarantee that if the path is changed, then all the assumptions in the Lemma 5.5.6 are satisfied. All the proofs appear in Appendix 5.D.

Lemma 5.5.7. *For each state $\mathbf{x}, \mathbf{x}' \in Q$ such that $\mathbf{x}.path \neq \mathbf{x}.new\_path$, if $\mathbf{x} \in \mathcal{I}_0$ and $\mathbf{x} \overset{\mathsf{main}}{\rightarrow} \mathbf{x}'$, then $\mathbf{x}'.d \geq \lambda - v_{max}\Delta > 0$ where λ is the length of the first segment of $\mathbf{x}.new\_path$.*

Lemma 5.5.8. *For each state* $\mathbf{x}, \mathbf{x}' \in Q$ *such that* $\mathbf{x}.path \neq \mathbf{x}.new\_path$*, if* $\mathbf{x} \in \mathcal{I}_0$ *and* $\mathbf{x} \xrightarrow{\mathsf{main}} \mathbf{x}'$*, then* $\mathbf{x}' \in \mathcal{I}_0$*.*

Using the previous three lemmas, the following lemma concludes that an execution fragment that updates the path exactly once by the first main action preserves an invariance of $\mathcal{I}_0$.

Lemma 5.5.9. *Consider a* plan*-free execution fragment* β *starting at a state* $\mathbf{x} \in \mathcal{I}_0$*. If* $\mathbf{x}.path \neq \mathbf{x}.new\_path$*, then* $\beta.\mathsf{lstate} \in \mathcal{I}_0$*.*

Proof. β can be written as $\beta = \beta_1 \mathsf{main} \beta_2$ where $\beta_1 = \tau_0 \mathsf{brake} \tau_1 \mathsf{brake} \ldots \tau_n$ and β_2 is a plan-free execution fragment with $\beta_2.\mathsf{fstate} \lceil path = \beta_2.\mathsf{fstate} \lceil new\_path$. Clearly, $\beta_1.\mathsf{lstate} \lceil path \neq \beta_1.\mathsf{lstate} \lceil new\_path$. In addition, $\beta_1.\mathsf{fstate} \in \mathcal{I}_0$ and thus, from Proposition 5.5.1, $\beta_1.\mathsf{lstate} \in \mathcal{I}_0$. Applying Lemma 5.5.7 and Lemma 5.5.8, we see that $\beta_2.\mathsf{fstate} \lceil d \geq \lambda_1 - v_{max}\Delta \geq \lambda_1 - \epsilon_0 - v_{max}\Delta$ and $\beta_2.\mathsf{fstate} \in \mathcal{I}_0$ where λ_1 is the length of the first segment of $\mathbf{x}.new\_path$. Therefore, from Lemma 5.5.6, $\beta.\mathsf{lstate} \in \mathcal{I}_0$. □

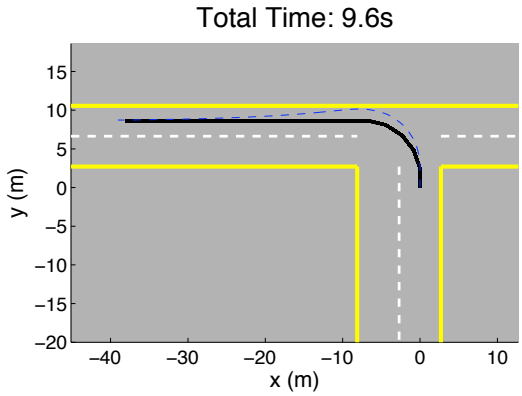
Finally, we conclude that $\mathcal{I}_0$ is an invariant of $\mathcal{A}$.

Theorem 5.5.1. *Suppose the initial state* $\mathbf{x}_0 \in \mathcal{I}_0$ *and* $\mathbf{x}_0.d \geq \lambda_1 - \epsilon_0 - v_{max}\Delta$ *where* λ_1 *is the length of the first segment of the initial path. Then,* $\mathcal{I}_0$ *is an invariant of* $\mathcal{A}$*.*

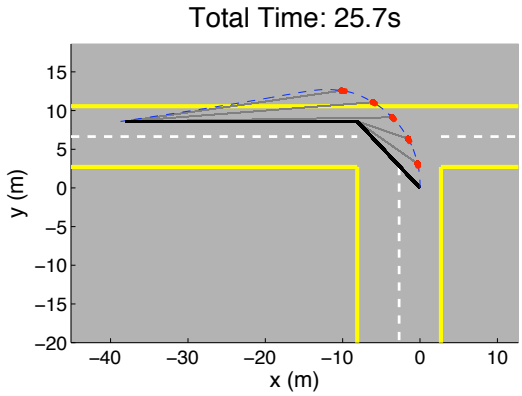
Proof. Any execution α can be written as $\alpha = \beta_1 \mathsf{plan} \beta_2 \mathsf{plan} \ldots$ where β_1 is a plan-free execution fragment with $\beta_1.\mathsf{fstate} \lceil path = \beta_1.\mathsf{fstate} \lceil new\_path$ and for any $i \geq 2$, β_i is a plan-free execution fragment with $\beta_i.\mathsf{fstate} \lceil path \neq \beta_i.\mathsf{fstate} \lceil new\_path$. Since plan action does not affect the variable s, if $\beta_1.\mathsf{lstate} \in \mathcal{I}_0$, then $\beta_2.\mathsf{fstate} \in \mathcal{I}_0$ and using Lemma 5.5.9, we get that for any $i \geq 2$, $\beta_i.\mathsf{lstate} \in \mathcal{I}_0$. Thus, we only need to show that $\beta_1.\mathsf{lstate} \in \mathcal{I}_0$. But this is true from Lemma 5.5.6 since $\beta_1.\mathsf{fstate} \lceil d = \mathbf{x}_0.d \geq \lambda_1 - \epsilon_0 - v_{max}\Delta$ and $\beta_1.\mathsf{fstate} \in \mathcal{I}_0$. □

Since for any state $\mathbf{x} \in \mathcal{I}_0$, $|\mathbf{x}.e_1| \leq \epsilon_0 \leq e_{max}$, invariance of $\mathcal{I}_0$ guarantees the safety property (A). For property (B), we note that for any state $\mathbf{x} \in \mathcal{I}_0$, there exists $v_{min} > 0$ such that $\mathbf{x}.v \geq v_{min} > 0$ and $|\mathbf{x}.e_2| \leq \theta_{0,2} < \frac{\pi}{2}$, that is, $\dot{d} = f_7(\mathbf{x}.s, u) \leq -v_{min} \cos\theta_{0,2} < 0$ for any $u \in \mathcal{U}$. Thus, it follows that the waypoint-distance decreases and the Vehicle makes progress towards its waypoint.

The simulation results are shown in Figure 5.11, which illustrate that the Vehicle is capable of making a sharp left turn, provided that the path satisfies Assumption 5.5.2. In



(a)



(b)

Figure 5.11: The positions of Alice (dashed line) as it follows a path (solid line) to execute a sharp left turn. The initial path is drawn in thick solid (black) line. When *brake* is triggered, a thick dashed (red) line is drawn on the position of Alice. (a) The path satisfies Assumption 5.5.2. (b) The path does not satisfy the assumption and the replan occurs due to excessive deviation. The replanned paths are drawn in thin solid (grey) line.

addition, we are able to replicate the stuttering behavior described in Section 1.1 when Assumption 5.5.2 is violated.

5.6 Conclusions

Motivated by a design bug that caused an undesirable behavior of Alice, an autonomous vehicle built at Caltech for the 2007 DARPA Urban Challenge, this chapter introduced Periodically Controlled Hybrid Automata (PCHA), a subclass of Hybrid I/O Automata that is suitable for modeling embedded control systems with periodic sensing and actuation.

New sufficient conditions for verifying invariant properties of PCHAs were presented. For PCHAs with polynomial continuous vector fields, it is possible to check these conditions automatically using, for example, quantifier elimination or sum of squares relaxations. The intuition behind these conditions is that for an execution fragment to leave an invariant set $\mathcal{I}$, it needs to cross the boundary $\partial\mathcal{I}$ of $\mathcal{I}$. Hence, to verify invariance of $\mathcal{I}$, it is sufficient to identify a subset C of $\mathcal{I}$ such that: (a) there is enough separation between C and $\partial\mathcal{I}$ to ensure that if a control law is evaluated when the state is inside C, then it is evaluated again before an execution fragment reaches $\partial\mathcal{I}$, and (b) if the control law is evaluated when the state is outside C, then the vector field on $\partial\mathcal{I}$ points inwards with respect to $\partial\mathcal{I}$. These conditions can be generalized to the case where a collection of subsets C's corresponding to different parts of $\partial\mathcal{I}$ is needed to prove invariance of $\mathcal{I}$. An example that illustrates automatic construction of an invariant set using the constraint-based approach was provided.

We then applied the proposed technique to manually verify a sequence of invariant properties of the planner-controller subsystem of Alice. Geometric properties of planner-generated paths were derived that guarantee that such paths can be safely followed by the controller. The analysis revealed that the software design was not inherently flawed; the undesirable behavior was caused by an unfortunate choice of certain parameters. The simulation results verified that with the proper choice of parameters, the observed failure does not occur.

Appendix

5.A Vehicle‖Controller as a PCHA

Here we show that the composed automaton $\mathcal{A} = \mathsf{Vehicle}\|\mathsf{Controller}$ is a Periodically Controlled Hybrid Automaton. We define an automaton $\mathcal{A}'$ that is identical to $\mathcal{A}$ except that its variables, actions and transition functions are renamed to match the definition of the generic PCHA of Figure 5.1.

Variables.

$\mathcal{A}'$ has the following variables.

(a) a continuous state variable $s \triangleq \langle x, y, \theta, v, e_1, e_2, d \rangle$ of type $\mathcal{X} = \mathbb{R}^7$.

(b) a discrete state variable $loc \triangleq \langle brake, path, seg \rangle$ of type $\mathcal{L} = \texttt{Tuple}[\{On, Off\}, \texttt{Seq}[\mathbb{R}^2], \mathbb{N}]$.

(c) a control variable is $u = \langle a, \phi \rangle$ of type $\mathcal{U} = \mathbb{R}^2$.

(d) two command variables $z_1 \triangleq brake$ of type $\mathcal{Z}_1 = \{On, Off\}$ and $z_2 = new\_path$ of type $\mathcal{Z}_2 = \texttt{Seq}[\mathbb{R}^2]$.

Actions and transitions.

$\mathcal{A}$ has two input update actions, $\mathsf{brake}(b)$ and $\mathsf{plan}(p)$, and the command variables z_1 and z_2 store the values b and p, respectively, when these actions occur.

An internal control action main occurs every Δ time, starting from time 0. That is, values of Δ_1 and Δ_2 as defined in a generic PCHA are $\Delta_1 = \Delta$ and $\Delta_2 = 0$. The control law function g and the state transition function h of $\mathcal{A}$ can be derived from the specification of main action in Figure 5.6. Let $g = \langle g_a, g_\phi \rangle$ where $g_a : \mathcal{L} \times \mathcal{X} \to \mathbb{R}$ and $g_\phi : \mathcal{L} \times \mathcal{X} \to \mathbb{R}$ represent the control law for a and ϕ, respectively, and are given by

$$
\begin{aligned}
g_a(l, s) &= \begin{cases} a_{brake} & \text{if } l.brake = On \\ a_{max} & \text{if } l.brake = Off \ \wedge \ s_0.v < v_T \\ 0 & \text{otherwise} \end{cases} \\
g_\phi(l, s) &= \frac{\phi_d}{|\phi_d|} \min(\delta \times s.v, |\phi_d|)
\end{aligned}
$$

where $\phi_d = -k_1 s.e_1 - k_2 s.e_2$. Let $h = \langle h_{s,1}, \ldots, h_{s,7}, h_{l,1}, h_{l,2}, h_{l,3} \rangle$ where $h_{s,1}, \ldots, h_{s,7} : \mathcal{L} \times \mathcal{X} \times \mathcal{Z}_1 \times \mathcal{Z}_2 \to \mathbb{R}$ describe the discrete transition of x, y, θ, v, e_1, e_2 and d components of s, respectively, and $h_{l,1} : \mathcal{L} \times \mathcal{X} \times \mathcal{Z}_1 \times \mathcal{Z}_2 \to \{On, Off\}$, $h_{l,2} : \mathcal{L} \times \mathcal{X} \times \mathcal{Z}_1 \times \mathcal{Z}_2 \to \texttt{Seq}[\mathbb{R}^2]$ and $h_{l,3} : \mathcal{L} \times \mathcal{X} \times \mathcal{Z}_1 \times \mathcal{Z}_2 \to \mathbb{N}$ describe the discrete transition of $brake$, $path$ and seg, respectively.

Then, the function h is given by

$$
\begin{aligned}
h_{s,1}(l,s,z_1,z_2) &= s.x \\
h_{s,2}(l,s,z_1,z_2) &= s.y \\
h_{s,3}(l,s,z_1,z_2) &= s.v \\
h_{s,4}(l,s,z_1,z_2) &= s.\theta \\
h_{s,5}(l,s,z_1,z_2) &= \begin{cases} s.e_1 & \text{if } l.path = z_2 \ \wedge \ s.d > 0 \\ \frac{1}{\|\vec{q}\|}\vec{q}\cdot\vec{r} & \text{otherwise} \end{cases} \\
h_{s,6}(l,s,z_1,z_2) &= \begin{cases} s.e_2 & \text{if } l.path = z_2 \ \wedge \ s.d > 0 \\ s.\theta - \angle\vec{p} & \text{otherwise} \end{cases} \\
h_{s,7}(l,s,z_1,z_2) &= \begin{cases} s.d & \text{if } l.path = z_2 \ \wedge \ s.d > 0 \\ \frac{1}{\|\vec{p}\|}\vec{p}\cdot\vec{r} & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
h_{l,1}(l,s,z_1,z_2) &= z_1 \\
h_{l,2}(l,s,z_1,z_2) &= z_2 \\
h_{l,3}(l,s,z_1,z_2) &= \begin{cases} 1 & \text{if } l.path \neq z_2 \\ l.seg + 1 & \text{if } l.path = z_2 \ \wedge \ s.d \leq 0 \\ l.seg & \text{otherwise} \end{cases}
\end{aligned}
$$

where the temporary variable $\vec{p}$, $\vec{q}$ and $\vec{r}$ are computed as in the Controller specification based on the updated value of *path* and *seg*.

Trajectories.

From the the state models of Vehicle and Controller automata specified on line 14 of Figure 5.5 and lines 48–50 of Figure 5.6, we see that $\mathcal{A}$ only has one state model. For any value of $l \in \mathcal{L}$, the continuous state s evolves according to the differential equation $\dot{s} = f(s,u)$ where $f = \langle f_1, f_2, \ldots, f_7 \rangle$ and $f_1, \ldots, f_7 : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ are associated with the evolution of the x, y, θ, v, e_1, e_2 and d components of s, respectively. Using the definition of the control

law function g defined above, we can derive the following components of $f(s, g(l, s_0))$:

$$
\begin{aligned}
f_1(s, g(l, s_0)) &= s.v\cos(s.\theta),\ f_2(s, g(l, s_0)) = s.v\sin(s.\theta) \\
f_3(s, g(l, s_0)) &= f_6(s, g(l, s_0)) = \frac{s.v}{L}\tan(\frac{\phi_d}{|\phi_d|}\min(|\phi_d|, \delta s_0.v, \phi_{max})) \\
f_4(s, g(l, s_0)) &= \begin{cases} a_{brake} & \text{if } l.brake = On \ \wedge\ s.v > 0 \\ a_{max} & \text{if } l.brake = Off \ \wedge\ s_0.v < v_T \\ 0 & \text{otherwise} \end{cases} \\
f_5(s, g(l, s_0)) &= s.v\sin(s.e_2) \\
f_7(s, g(l, s_0)) &= -s.v\cos(s.e_2)
\end{aligned}
$$

where $\phi_d = -k_1 s_0.e_1 - k_2 s_0.e_2$.

5.B Invariant Verification

From the definition of a good execution (Definition 5.5.1), we show that when the value of the variable *brake* is *On*, the speed of the Vehicle is at least $\frac{\phi_b}{\delta} + \Delta|a_{brake}|$.

Lemma 5.B.1. *At any reachable state* $\mathbf{x}$ *of* $\mathcal{A}$*, if* $\mathbf{x}.brake = On$ *then* $\mathbf{x}.v \geq \frac{\phi_b}{\delta} + \Delta|a_{brake}|$.

Proof. Consider an arbitrary execution fragment, $\alpha = \tau_0 a_1 \tau_1 a_2 \ldots$ and an arbitrary $i \in \mathbb{N}$ such that $(\tau_i \downarrow brake)(0) = On$. Since the initial value of the variable *brake* is *Off*, there must exists $j \leq i$ such that a_j is a brake(*On*) action and for any natural number $m \in [j, i]$, a_m is not a brake(*Off*) action. Let $(\tau_{j-1}.\mathsf{lstate}) \lceil v = v_b$. Since a_j is a brake(*On*) action which does not affect v, we get $(\tau_j.\mathsf{fstate}) \lceil v = v_b$. From Definition 5.5.1, $v_b > \frac{\phi_b}{\delta} + \Delta|a_{brake}|$ and there must exists $k > i$ such that a_k is a brake(*Off*) action and $\sum_{m=j}^{k-1} \tau_m.\mathsf{ltime} \leq \frac{1}{|a_{brake}|}(v_b - \frac{\phi_b}{\delta} - \Delta|a_{brake}|)$. So for any $t \in dom(\tau_i)$, we get

$$
\begin{aligned}
(\tau_i \downarrow v)(t) &\geq v_b + \min_{s, s_0 \in \mathcal{X}, l \in \mathcal{L}} f_4(s, g(l, s_0))(t + \sum_{m=j}^{i-1} \tau_m.\mathsf{ltime}) \\
&\geq v_b + a_{brake}(\sum_{m=j}^{k-1} \tau_m.\mathsf{ltime}) = \frac{\phi_b}{\delta} + \Delta|a_{brake}|.
\end{aligned}
$$

□

The next lemma shows that the subtangential, bounded distance and bounded speed

conditions (of Lemma 5.3.2) are satisfied. The proof utilizes Lemma 5.3.3. The knowledge about the reachable state $\mathbf{x}$ of $\mathcal{A}$ with $\mathbf{x}.brake = On$, provided in Lemma 5.B.1, is needed to prove the subtangential condition for $j = 6$.

Lemma 5.5.1. For each $l \in \mathcal{L}$ and $j \in \{1, \ldots, 6\}$, the subtangential, bounded distance and bounded speed conditions (of Lemma 5.3.2) are satisfied.

Proof. First, we define the sets $\{C_j\}_{j\in\{1,\ldots,6\}}$ as follows:

$$
\begin{aligned}
C_1 &\triangleq C_2 \triangleq \varnothing, \\
C_3 &\triangleq \{s \in I \mid -k_1 s.e_1 - k_2 s.e_2 \le 0 \vee L\cot(-k_1 s.e_1 - k_2 s.e_2)\sin\theta_{k,2} \ge \frac{k_2}{k_1}\}, \\
C_4 &\triangleq \{s \in I \mid -k_1 s.e_1 - k_2 s.e_2 \ge 0 \vee L\cot(k_1 s.e_1 + k_2 s.e_2)\sin\theta_{k,2} \ge \frac{k_2}{k_1}\}, \\
C_5 &\triangleq \{s \in I \mid s.v \le v_T\}, \\
C_6 &\triangleq \{s \in I \mid s.v \ge \frac{\phi_b}{\delta} + \Delta|a_{brake}|\}.
\end{aligned}
$$

Since $C_1, C_2 = \varnothing$, we see that the bounded distance and bounded speed conditions are automatically satisfied for $j = 1, 2$ with any arbitrary large c_j and arbitrary small b_j. Now, consider an arbitrary $s_0 \in I$ and $s \in \partial I_1$. By definition, $F_1(s) = 0$. From the definition of $\theta_{k,1}$ and $\theta_{k,2}$ and Assumption 5.5.1(b), $s.e_2 \in [-\theta_{k,2}, -\theta_{k,1}] \subset (-\frac{\pi}{2}, 0]$. In addition, since $s \in I$, $F_6(s) = \delta s.v - \phi_b \ge 0$ and since $\delta > 0$ and $\phi_b \ge 0$, $s.v \ge 0$. Thus,

$$\frac{\partial F_1}{\partial s}(s) \cdot f(s, g(l, s_0)) = -\frac{de_1}{dt} = -s.v\sin(s.e_2) \ge 0.$$

For $j = 2$, the subtangential condition can be proved in a similar way.

To prove the bounded distance and the bounded speed conditions for $j = 3, \ldots, 6$, we apply Lemma 5.3.3. Let $\mathcal{U}_I = \{g(l, s) \mid l \in \mathcal{L},\ s \in I\}$. From the definition of I, we get that for any $s_0 \in I$, $-k_1 s_0.e_1 - k_2 s_0.e_2 \in [-\phi_k, \phi_k] \subset (-\frac{\pi}{2}, \frac{\pi}{2})$. Therefore, f is continuous in $I \times \mathcal{U}_I$.

In addition, it can be easily checked that the projection of I onto the (e_1, e_2, v) space is compact and for any $j \in \{3, \ldots, 6\}$, C_j is closed. Since the only variables involved in proving the control-free invariance condition of Lemma 5.3.1 are e_1, e_2 and v whose evolution along a trajectory can be described without other variables, from the proof of Lemma 5.3.2 and Lemma 5.3.3, we see that the requirement that I is compact can be relaxed to the requirement that the projection of I onto the (e_1, e_2, v) space is compact. Hence, from

Lemma 5.3.3, to prove that conditions (a)–(c) of Lemma 5.3.2 hold, we only need to show that for any $l \in \mathcal{L}$, the following conditions are satisfied for each $j \in \{3, \ldots, 6\}$:

1. $C_j \cap \partial I_j = \varnothing$,
2. For any $s_0 \in I \setminus C_j$ and $s \in \partial I_j$, $\frac{\partial F_j}{\partial s} \cdot f(s, g(l, s_0)) \geq 0$.

Consider an arbitrary $s \in \partial I_3$. From the definition of I_3, $-k_1 s.e_1 - k_2 s.e_2 = \phi_k > 0$. So from Assumption 5.5.1(c), $L\cot(-k_1 s.e_1 - k_2 s.e_2)\sin\theta_{k,2} < \frac{k_2}{k_1}$. Therefore, $C_3 \cap \partial I_3 = \varnothing$. Pick an arbitrary $s_0 \in I \setminus C_3$. From the definition of I and C_3, $0 < -k_1 s_0.e_1 - k_2 s_0.e_2 \leq \phi_k$ and $L\cot(-k_1 s_0.e_1 - k_2 s_0.e_2)\sin\theta_{k,2} < \frac{k_2}{k_1}$. Combining this with Assumption 5.5.1(a), we get $0 < -k_1 s_0.e_1 - k_2 s_0.e_2 \leq \frac{\pi}{2}$ and $|-k_1 s_0.e_1 - k_2 s_0.e_2| \leq \phi_{max}$. In addition, since $s_0 \in I$, $F_6(s_0) \geq 0$ and so $\delta s_0.v \geq \phi_b \geq \phi_k \geq |-k_1 s_0.e_1 - k_2 s_0.e_2|$, and since $s \in I$, $s.v \geq 0$. Therefore, we can conclude that

$$\frac{ds.e_2}{dt} = \frac{s.v}{L}\tan(-k_1 s_0.e_1 - k_2 s_0.e_2) \geq 0$$

and from Assumption 5.5.1(b), $s.e_2 \in [-\theta_{k,2}, \theta_{k,1}] \subset (-\frac{\pi}{2}, 0]$. So we get

$$\begin{aligned}\frac{ds.e_1}{ds.e_2} &= L\cot(-k_1 s_0.e_1 - k_2 s_0.e_2)\sin(s.e_2) \\ &\geq -L\cot(-k_1 s_0.e_1 - k_2 s_0.e_2)\sin\theta_{k,2} \\ &> -\frac{k_2}{k_1}.\end{aligned}$$

Thus,

$$\frac{\partial F_3}{\partial s} \cdot f(s, g(l, s_0)) = k_2\frac{ds.e_2}{dt} + k_1\frac{ds.e_1}{dt} = \frac{ds.e_2}{dt}\left(k_2 + k_1\frac{ds.e_1}{ds.e_2}\right) \geq 0.$$

This completes the proof for $j = 3$.

For $j = 4$, we can follow the previous proof to show that $C_4 \cap \partial I_4 = \varnothing$, $\frac{ds.e_2}{dt} \leq 0$ and $\frac{ds.e_1}{ds.e_2} > -\frac{k_2}{k_1}$, and so

$$\forall s_0 \in I \setminus C_4, \frac{\partial F_4}{\partial s} \cdot f(s, g(l, s_0)) \geq 0.$$

Next, consider an arbitrary $s \in \partial I_5$. From the definition of ∂I_5, $s.v = v_{max}$. Since $a_{max}, \Delta > 0$, $v_{max} = v_T + \Delta a_{max} > v_T$. Therefore, $C_5 \cap \partial I_5 = \varnothing$. Pick an arbitrary $s_0 \in I \setminus C_5$. From the definition of I and C_5, $v_T < s_0.v \leq v_{max}$. Therefore, we can conclude that

$$\frac{\partial F_5}{\partial s} \cdot f(s, g(l, s_0)) = \begin{cases} -a_{brake} & \geq 0. \\ 0 & \end{cases}$$

This completes the proof for $j = 5$.

Finally, consider an arbitrary $s \in \partial I_6$. From the definition of ∂I_6, $s.v = \frac{\phi_b}{\delta}$. Since $\Delta, |a_{brake}| > 0$, $\frac{\phi_b}{\delta} < \frac{\phi_b}{\delta} + \Delta|a_{brake}|$. Therefore, $C_6 \cap \partial I_6 = \varnothing$. Consider an arbitrary $s_0 \in I \setminus C_6$. From Lemma 5.B.1 and the definition of f_4, we see that $f_4(s, g(l, s_0)) = a_{brake}$ only if $s_0.v \geq \frac{\phi_b}{\delta} + \Delta|a_{brake}|$. But since $s_0 \in I \setminus C_6$, from the definition of I and C_6, $s_0.v < \frac{\phi_b}{\delta} + \Delta|a_{brake}|$. Therefore, $f_4(s, g(l, s_0))$ is either 0 or a_{max} and so we can conclude that

$$\frac{\partial F_6}{\partial s} \cdot f(s, g(l, s_0)) = f_4(s, g(l, s_0)) \geq 0.$$

□

Now, we prove that Assumption 5.5.1(d) provides the bound on Δ such that the sampling rate condition of Lemma 5.3.2 is satisfied.

Lemma 5.5.2. For each $l \in \mathcal{L}$, the sampling rate condition is satisfied.

Proof. For each $j \in \{1, \ldots, 6\}$, we want to find c_j and b_j that satisfy conditions (b) and (c) of Lemma 5.3.2. First, we note that for $j = 1, 2$, $C_j = \varnothing$, so c_j can be arbitrary large and b_j can be arbitrary small and therefore any $\Delta \in \mathbb{R}_+$ satisfies the sampling rate condition of Lemma 5.3.2. For $j = 5, 6$, it can be easily shown that $c_5 = \Delta a_{max}$, $b_5 = a_{max}$, $c_6 = \Delta|a_{brake}|$ and $b_6 = |a_{brake}|$; thus, $\frac{c_j}{b_j} = \Delta$. That is, Δ can be an arbitrary large number if we only consider $j = 1, 2, 5, 6$. So we only have to consider $j = 3, 4$. From Assumption 5.5.1(c), there exists

$$\tilde{\phi} = \cot^{-1}\left(\frac{k_2}{k_1 L \sin\theta_{k,2}}\right) < \phi_k.$$

Using symmetry, we get that for $j = 3$ and $j = 4$, the shortest distance between $\mathcal{U}_j$ and ∂I_j is then given by

$$c_j = \min_{s \in \partial I_j, s_0 \in \mathcal{U}_j} \|s - s_0\| = \frac{1}{\sqrt{k_1^2 + k_2^2}}(\phi_k - \tilde{\phi}).$$

Since $\forall s \in I, s.e_2 \in [-\theta_{k,2}, \theta_{k,2}] \subset (-\frac{\pi}{2}, \frac{\pi}{2})$, we have

$$\begin{aligned} b_j &= \max_{s \in I, s_0 \in \mathcal{U}_j} \|f(s, g(l, s_0))\| \\ &\leq v_{max}\sqrt{\sin^2\theta_{k,2} + \frac{1}{L^2}\tan^2(\tilde{\phi})} \end{aligned}$$

From Assumption 5.5.1(d), we see that $\Delta \leq \min_{j \in \{1,\ldots,6\}} \frac{c_j}{b_j}$. □

Having proved that all the conditions of Lemma 5.3.2 are satisfied, it follows that the control-free invariance condition of Lemma 5.3.1 holds. Applying Theorem 5.3.1, we can conclude the following invariance property of $\mathcal{I}$.

5.C Proofs for Segment Progress

First, we solve the differential equation that describes the evolution of e_1 and e_2 along τ. From periodicity of main actions we see that $dom(\tau) = [0, \Delta]$. Define the functions $e_1, e_2, v, v_{avg} : dom(\tau) \to \mathbb{R}$ as follows: $e_1(t) = (\tau \downarrow e_1)(t)$, $e_2(t) = (\tau \downarrow e_2)(t)$, $v(t) = (\tau \downarrow v)(t)$ and $v_{avg}(t) = \frac{1}{t}\int_0^t v(t')dt'$. From the state models of the Vehicle and the Controller specified in Figure 5.5 and Figure 5.6, since ϕ and a are constant along τ, the solution to the differential equations can be solved analytically and are given by

$$\begin{aligned} e_1(t) &= \begin{cases} e_1(0) + L\cot\phi\cos e_2(0) - L\cot\phi\cos e_2(t) & \text{if } \phi \neq 0 \\ e_1(0) + v_{avg}(t)t\sin e_2(0) & \text{otherwise} \end{cases}, \\ e_2(t) &= e_2(0) + \tfrac{\tan\phi}{L} v_{avg}(t)t, \end{aligned} \tag{5.12}$$

where $\phi = \tau.\mathsf{fstate} \lceil \phi$ and $a = \tau.\mathsf{fstate} \lceil a$.

The following lemma provides a bound on the change in e_1 over τ and on the change in ϕ between two consecutive main actions assuming that a discrete transition in the continuous state s does not occur.

Lemma 5.C.1. *Suppose* $\tau.\mathsf{fstate} \in \mathcal{I}_k$ *for some* $k \in \mathbb{N}$. *Then,* $|e_1(0) - e_1(\Delta)| \leq \Delta_e$ *and* $|(k_1e_1(0)+k_2e_2(0))-(k_1e_1(\Delta)+k_2e_2(\Delta))| \leq \Delta_\phi$ *where* $\Delta_e = v_{max}\Delta$ *and* $\Delta_\phi = v_{max}\Delta\left(k_1 + k_2\frac{\tan\phi_k}{L}\right)$.

Proof. From (5.12), we see that $|e_1(\Delta) - e_1(0)| \leq v_{max}\Delta$ and $|e_2(\Delta) - e_1(0)| \leq \frac{\tan\phi_k}{L} v_{max}\Delta$. So

$$\begin{aligned} |(k_1e_1(0) + k_2e_2(0)) - (k_1e_1(\Delta) + k_2e_2(\Delta))| &\leq k_1|e_1(\Delta) - e_1(0)| + k_2|e_2(\Delta) - e_2(0)| \\ &\leq k_1v_{max}\Delta + k_2\frac{\tan\phi_k}{L}v_{max}\Delta. \end{aligned}$$

□

The next lemma proves the desired progress property over τ.

Lemma 5.5.3. Suppose $\tau.\mathsf{fstate} \in \mathcal{I}_k$ for some $k \in \mathbb{N}$. Then $\tau.\mathsf{lstate} \in \mathcal{I}_{k+1}$ whose parameters ϵ_{k+1} and ϕ_{k+1} are given by

$$\epsilon_{k+1} = \epsilon_k - \hat{\epsilon}_k, \tag{5.13}$$

$$\phi_{k+1} = \phi_k - \hat{\phi}_k, \tag{5.14}$$

where $\hat{\epsilon}_k, \hat{\phi}_k \geq 0$ and are given by

$$\hat{\epsilon}_k = \epsilon_k - \max\left(\epsilon'_{k+1}, \frac{1}{k_1}\phi'_{k+1}\right), \tag{5.15}$$

$$\hat{\phi}_k = \phi_k - \max(\phi'_{k+1}, \varphi), \tag{5.16}$$

$$\epsilon'_{k+1} = \begin{cases} \max\left(\epsilon_k - \xi_k, \epsilon^*_k\right) & \text{if } \epsilon_k > \epsilon^*_k \\ \epsilon_k & \text{otherwise} \end{cases}, \tag{5.17}$$

$$\phi'_{k+1} = \begin{cases} \max\left(\phi_k - \psi_k, \phi^*_k\right) & \text{if } \phi_k > \phi^*_k \\ \phi_k & \text{otherwise} \end{cases}, \tag{5.18}$$

$$\epsilon^*_k = \epsilon'_k + v_{max}\Delta, \tag{5.19}$$

$$\phi^*_k = \phi'_k + k_1 v_{max}\Delta + k_2 \frac{\tan\phi_k}{L} v_{max}\Delta, \tag{5.20}$$

$$\xi_k = -2L \max_{\phi \in [-\phi_k, \phi_k]} \cot\phi \sin\left(-\frac{k_1\epsilon^*_k}{k_2} - \frac{\phi}{k_2} + \frac{v_{max}\Delta\tan\phi}{2L}\right)\sin\left(\frac{\phi_b\Delta\tan\phi}{2L\delta}\right), \tag{5.21}$$

$$\psi_k = \frac{k_2}{L}\tan\phi^*_k\frac{\phi_b}{\delta}\Delta - 2k_1 L\cot\phi^*_k \sin\theta_{k,2}\sin\left(\frac{\tan\phi_k}{2L}v_{max}\Delta\right), \tag{5.22}$$

$$\epsilon'_k = \max_{\tilde{\phi} \in [-\phi_k, \phi_k]} \left(-\frac{1}{k_1}\tilde{\phi} + \frac{k_2}{k_1}\frac{\tan\tilde{\phi}}{2L}v_{max}\Delta\right), \tag{5.23}$$

$$\phi'_k = \max\left(\tan^{-1}\sqrt{\frac{2k_1L^2\delta}{k_2\phi_b\Delta}\sin\theta_{k,2}\sin\left(\frac{\tan\phi_k}{2L}v_{max}\Delta\right)}, \Delta_\phi\right), \tag{5.24}$$

where φ is the minimum value of ϕ_{k+1} such that ϵ'_{k+1} and ϕ_{k+1} satisfy Assumption 5.5.1(c). In addition, define k^* to be the minimum value of k such that $\epsilon_k \leq \epsilon^*_k$ or $\phi_k \leq \phi^*_k$. (If for any k, $\epsilon_k > \epsilon^*_k$ and $\phi_k > \phi^*_k$, just pick an arbitrary natural number k^*.) Then, for any $k < k^*$, $\hat{\epsilon}_k$ and $\hat{\phi}_k$ are strictly positive, that is, $I_{k+1} \subsetneq I_k$.

Proof. Since by definition $\epsilon_{k+1} \geq \epsilon'_{k+1}$ and $\phi_{k+1} \geq \phi'_{k+1}$, we see that if $|\tau.\mathsf{lstate}\lceil e_1| \leq \epsilon'_{k+1}$ and $|k_1(\tau.\mathsf{lstate}\lceil e_1) + k_2(\tau.\mathsf{lstate}\lceil e_2)| \leq \phi'_{k+1}$, then $\tau.\mathsf{lstate} \in \mathcal{I}_{k+1}$. To show that ϵ_{k+1} and ϕ_{k+1} satisfy Assumption 5.5.1 and that $\hat{\epsilon}_k, \hat{\phi}_k \geq 0$, we use the following observations: (a) $\psi_k \geq 0$ and $\xi_k \geq 0$ and thus, $\epsilon'_{k+1} \leq \epsilon_k$ and $\phi'_{k+1} \leq \phi_k$, (b) given ϕ'_{k+1}, $\frac{1}{k_1}\phi'_{k+1}$ is the minimum value

of ϵ_{k+1} such that ϵ_{k+1} and ϕ'_{k+1} satisfies Assumption 5.5.1, (c) given ϵ'_{k+1}, φ is the minimum value of ϕ_{k+1} such that ϵ'_{k+1} and ϕ_{k+1} satisfies Assumption 5.5.1, and (d) φ decreases as ϵ'_{k+1} decreases. With these observations and the assumption that ϵ_k and ϕ_k satisfy Assumption 5.5.1, it can be easily checked that (a) $\epsilon_{k+1} \leq \epsilon_k$ and $\phi_{k+1} \leq \phi_k$, (b) if $\epsilon_k > \epsilon_k^*$ and $\phi_k > \phi_k^*$, then $\epsilon'_{k+1} < \epsilon_k$ and $\phi'_{k+1} < \phi_k$, and (c) if $\epsilon_{k+1} \neq \epsilon'_{k+1}$, then $\phi_{k+1} = \phi'_{k+1}$ and if $\phi_{k+1} \neq \phi'_{k+1}$, then $\epsilon_{k+1} = \epsilon'_{k+1}$. Thus, we can conclude that ϵ_{k+1} and ϕ_{k+1} satisfy Assumption 5.5.1 and that if $\epsilon_k > \epsilon_k^*$ and $\phi_k > \phi_k^*$, then $\epsilon_{k+1} < \epsilon_k$ and $\phi_{k+1} < \phi_k$.

So what remains to be proved are $|\tau.\mathsf{lstate}\lceil e_1| \leq \epsilon'_{k+1}$ and $|k_1(\tau.\mathsf{lstate}\lceil e_1) + k_2(\tau.\mathsf{lstate}\lceil e_2)| \leq \phi'_{k+1}$. From Theorem 5.5.1, $\tau.\mathsf{lstate} \in \mathcal{I}_k$. Thus, we can conclude that $\phi'_{k+1} \leq \phi_k$ and $\epsilon'_{k+1} \leq \epsilon_k$. This completes the proof for the second case of (5.17) and (5.18).

Next, we prove the first case of (5.18). Let $\phi_f = -k_1 e_1(0) - k_2 e_2(0)$ and $\phi_l = -k_1 e_1(\Delta) - k_2 e_2(\Delta)$. Suppose $|\phi_f| \geq \Delta_\phi$. From (5.12), we get that

$$\phi_l \quad = \quad -k_1 \left(e_1(0) + L\cot\phi_1 \cos(e_2(0)) - L\cot\phi_1 \cos(e_2(\Delta))\right) - k_2\left(e_2(0) + \frac{\tan\phi_f}{L} v_{avg}\Delta\right)$$

where v_{avg} is the average speed of the $\mathsf{Vehicle}$ over τ. Substituting $e_1(0) = -\frac{k_2}{k_1}e_2(0) - \frac{1}{k_1}\phi_f$, we get

$$\phi_l \quad = \quad \phi_f - \left(\frac{k_2}{L}\tan\phi_f v_{avg}\Delta + 2k_1 L\cot\phi_f \sin(\frac{1}{2}(e_2(0) + e_2(\Delta)))\sin\left(\frac{\tan\phi_f}{2L} v_{avg}\Delta\right)\right).$$

Since $\tau.\mathsf{fstate}, \tau.\mathsf{lstate} \in \mathcal{I}_k$, from the definition of $\theta_{k,2}$, we see that $|e_2(0)|, |e_2(\Delta)| \leq \theta_{k,2}$. So $\frac{1}{2}|e_2(0) + e_2(\Delta)| \leq \theta_{k,2}$. In addition, from Theorem 5.5.1 and the definition of F_5 and F_6, we know that $\frac{\phi_b}{\delta} \leq v_{avg} \leq v_{max}$. From Lemma 5.5.3, we get that ϕ_f and ϕ_l have the same sign. So it is easy to show that

$$|\phi_l| \quad \leq \quad |\phi_f| - \left(\frac{k_2}{L}\tan|\phi_f|\frac{\phi_b}{\delta}\Delta - 2k_1 L\cot|\phi_f|\sin\theta_{k,2}\sin\left(\frac{\tan\phi_k}{2L} v_{max}\Delta\right)\right).$$

Define the function $\Psi : [0, \phi_k] \to \mathbb{R}$ by

$$\Psi(\phi) \quad = \quad \frac{k_2}{L}\tan\phi\frac{\phi_b}{\delta}\Delta - 2k_1 L\cot\phi\sin\theta_{k,2}\sin\left(\frac{\tan\phi_k}{2L} v_{max}\Delta\right).$$

That is $\psi_k = \Psi(\phi_k^*)$. It can be easily checked that with Assumption 5.5.1(e), $\Psi(\phi)$ increases with ϕ and vanishes when $\phi = \tan^{-1}\sqrt{\frac{2k_1L^2\delta}{k_2\phi_b\Delta}\sin\theta_{k,2}\sin\left(\frac{\tan\phi_k}{2L} v_{max}\Delta\right)}$, which does not ex-

ceed ϕ'_k defined in (5.24). For $\phi > \phi'_k$, $\Psi(\phi) > 0$. From Lemma 5.C.1, we also know that for any $\phi_f \in [-\phi_k, \phi_k]$,

$$|\phi_l| \quad \leq \quad |\phi_f| + k_1 v_{max}\Delta + k_2 \frac{\tan\phi_k}{L} v_{max}\Delta.$$

Since $\phi^*_k > \phi'_k$, we arrive at the following conclusion:

$$|\phi_l| \leq \begin{cases} |\phi_f| - \psi_k & \text{if } |\phi_f| > \phi^*_k \\ \phi^*_k & \text{if } \phi'_k \leq |\phi_f| \leq \phi^*_k \\ |\phi_f| + k_1 v_{max}\Delta + k_2 \frac{\tan\phi_k}{L} v_{max}\Delta & \text{if } |\phi_f| < \phi'_k \end{cases}.$$

Thus, $|\phi_l| \leq \max(\phi_k - \psi_k, \phi^*_k)$.

Finally, we prove the first case of (5.17). From (5.12), we get that

$$e_1(\Delta) \quad = \quad e_1(0) + 2L\cot\phi_1 \sin\left(e_2(0) + \frac{\tan\phi_f}{2L} v_{avg}\Delta\right) \sin\left(\frac{\tan\phi_f}{2L} v_{avg}\Delta\right).$$

Note that the case where $\phi_f = 0$ is also captured by this equation as

$$\lim_{\phi_f \to 0} 2L\cot\phi_f \sin\left(\frac{\tan\phi_f}{2L} v_{avg}\Delta\right) = v_{avg}\Delta.$$

Define the function $\Xi : [0, \epsilon_k] \to \mathbb{R}$ by

$$\Xi(\epsilon) \quad = \quad -2L \max_{\phi \in [-\phi_k, \phi_k]} \cot\phi \sin\left(-\frac{k_1}{k_2} e - \frac{1}{k_2}\phi + \frac{\tan\phi}{2L} v_{max}\Delta\right) \sin\left(\frac{\tan\phi}{2L}\frac{\phi_b}{\delta}\Delta\right).$$

That is $\xi_k = \Xi(\epsilon^*_k)$. It can be easily checked that with Assumption 5.5.1(e), $\Xi(\epsilon) > 0$ for any $\epsilon > \epsilon'_k$ and that if $e_1(0) \geq \epsilon'_k$, then $e_2(0) \leq -\frac{k_1}{k_2}\epsilon'_k - \frac{1}{k_2}\phi_f$. So

$$2L\cot\phi_f \sin\left(e_2(0) + \frac{\tan\phi_f}{2L} v_{avg}\Delta\right) \sin\left(\frac{\tan\phi_f}{2L} v_{avg}\Delta\right) \leq -\xi_k.$$

Using symmetry, we can derive a similar lower bound for the case where $e_1(0) \leq -\epsilon'_k$. From Lemma 5.C.1, we also know that

$$|e_1(\Delta)| \quad \leq \quad |e_1(0)| + v_{max}\Delta.$$

So we arrive at the following conclusion:

$$|e_1(\Delta)| \leq \begin{cases} |e_1(0)| - \xi_k & \text{if } |e_1(0)| > \epsilon_k^* \\ \epsilon_k^* & \text{if } \epsilon_k' \leq |e_1(0)| \leq \epsilon_k^* \\ |e_1(0)| + v_{max}\Delta & \text{if } |e_1(0)| < \epsilon_k' \end{cases} .$$

Thus, $|e_1(\Delta)| \leq \max(\epsilon_k - \xi_k, \epsilon_k^*)$. □

Using Lemma 5.5.3 and Lemma 5.5.4, we establish the relationship between the progress of $\mathcal{I}_k$'s and the decrease in the value of d.

Lemma 5.5.5. For each $k \in \mathbb{N}$, starting from any reachable state $\mathbf{x} \in \mathcal{I}_k$ such that $\mathbf{x}.d > v_{max}\Delta$, $\mathbf{x}.path = \mathbf{x}.new\_path$ and $\mathbf{x}.next = \mathbf{x}.now$, any plan-free execution fragment β with $\beta.\mathsf{ltime} = \Delta$ satisfies $\beta.\mathsf{lstate} \in \mathcal{I}_{k+1}$ and $\beta.\mathsf{lstate} \lceil d \geq \mathbf{x}.d - v_{max}\Delta$.

Proof. Since $\mathbf{x}.next = \mathbf{x}.now$ and $\beta.\mathsf{ltime} = \Delta$, we see that β can be written as $\beta = \beta'$ or $\beta = \beta'\mathsf{main}\tau_j\mathsf{brake}(b_j)\tau_{j+1}\mathsf{brake}(b_{j+1})\ldots\tau_n$ where β' is an execution fragment with exactly one main action a_i that occurs at time 0 and is immediately followed by a main action in the execution, $\beta'.\mathsf{ltime} = \Delta$ and $\tau_j, \ldots \tau_n$ are point trajectories. Let τ be the pasted trajectory of all the trajectories after a_i in β'. Then, τ is a pasted trajectory of all the trajectories between two main actions and so Lemma 5.5.3 and Lemma 5.5.4 apply. Since the main action a_i occurs at time 0 in β and brake action does not affect the value of s, we see that $\tau_{i-1}.\mathsf{lstate} \lceil s = \mathbf{x}.s$. So $\tau_{i-1}.\mathsf{lstate} \lceil d > v_{max}\Delta > 0$ and hence a_i does not change the value of s. That is, $\tau.\mathsf{fstate} = \mathbf{x} \in \mathcal{I}_k$. From Lemma 5.5.3, we get that $\beta'.\mathsf{lstate} \in \mathcal{I}_{k+1}$. In addition, from Lemma 5.5.4, we see that $\beta'.\mathsf{lstate} \lceil d \geq \mathbf{x}.d - v_{max}\Delta$. Since $\mathbf{x}.d > v_{max}\Delta$, we get $\beta'.\mathsf{lstate} \lceil d > 0$. Therefore, the main action following β' does not change the value of s. In addition, since brake action only affects the *brake* variable, we see that $\beta.\mathsf{lstate} \lceil s = \beta'.\mathsf{lstate} \lceil s$. Hence, we can conclude that $\beta.\mathsf{lstate} \in \mathcal{I}_{k+1}$ and $\beta.\mathsf{lstate} \lceil d \geq \mathbf{x}.d - v_{max}\Delta$. □

5.D Proofs for Safety and Waypoint Progress

Lemma 5.5.6. Consider a plan-free execution fragment β starting at a state $\mathbf{x} \in \mathcal{I}_0$. Suppose $\mathbf{x}.path = \mathbf{x}.new\_path$ and $\mathbf{x}.d \geq D^*$ where $D^* = \lambda_1 - \epsilon_0 - v_{max}\Delta$ and λ_1 is the length of the segment $\mathbf{x}.seg$. Then $\beta.\mathsf{lstate} \in \mathcal{I}_0$.

Proof. First, observe that β can be written as $\beta = \beta_1 a_1 \beta_2 a_2 \ldots \beta_m$ where for any i, a_i is a main action and β_i is a plan-free execution fragment such that $\beta_i.\mathsf{lstate} \lceil path = \beta_i.\mathsf{fstate} \lceil new\_path$ and $\beta_i.\mathsf{lstate} \lceil seg = \beta_i.\mathsf{fstate} \lceil seg$. From Theorem 5.5.1, we get that for any i, if $\beta_i.\mathsf{fstate} \in \mathcal{I}_0$, then $\beta.\mathsf{lstate} \in \mathcal{I}_0$. So, suppose $\beta_1.\mathsf{fstate} \in \mathcal{I}_0$, $\beta_1.\mathsf{fstate} \lceil path = \beta_1.\mathsf{fstate} \lceil new\_path$ and $\beta_1.\mathsf{fstate} \lceil d \geq \lambda_1 - \epsilon_0 - v_{max}\Delta$. We only need to show that for any $i > 1$, $\beta_i.\mathsf{fstate} \in \mathcal{I}_0$.

Consider the base case $i = 2$. If $\beta_2.\mathsf{fstate} \lceil seg = \beta_1.\mathsf{lstate} \lceil seg$, then a_1 does not change the continuous state s, and so $\beta_2.\mathsf{fstate} \in \mathcal{I}_0$. Otherwise, $\beta_2.\mathsf{fstate} \lceil seg = \beta_1.\mathsf{fstate} \lceil seg + 1$. But from the update rule of the variable seg and Lemma 5.5.4, it can be easily shown that $-v_{max}\Delta < \beta_1.\mathsf{lstate} \lceil d \leq 0$. Applying Theorem 5.5.2, we get that $\beta_1.\mathsf{lstate} \in \mathcal{I}_n$ where $n = \lfloor \frac{\lambda_1 - \epsilon_0 - 2v_{max}\Delta}{v_{max}\Delta} \rfloor$ because by Assumption 5.5.2(a), $\lambda_1 - \epsilon_0 - 2v_{max}\Delta > 0$.

Let $\mathbf{x}_1 = \beta_1.\mathsf{lstate}$ and $\mathbf{x}_2 = \beta_2.\mathsf{fstate}$ and let σ_1 be the difference between the orientation of $\beta_1.\mathsf{fstate} \lceil seg$ and $\beta_1.\mathsf{fstate} \lceil seg + 1$. From the update rule for e_1 and the definition of $\vec{p}$, $\vec{q}$ and $\vec{r}$ in Figure 5.6, it can be shown that $\mathbf{x}_2.e_1 = \mathbf{x}_1.d \sin\sigma_1 + \mathbf{x}_1.e_1 \cos\sigma_1$. But since $\beta_1.\mathsf{lstate} \in \mathcal{I}_n$, from the definition of $\mathcal{I}_n$, $|\mathbf{x}_1.e_1| \leq \epsilon_n$. Therefore, using the bounds on $\mathbf{x}_1.d$ provided earlier in the proof, we get $|\mathbf{x}_2.e_1| \leq v_{max}\Delta|\sin\sigma_1| + \epsilon_n|\cos\sigma_1|$. Hence, from Assumption 5.5.2(b), $|\mathbf{x}_2.e_1| \leq \epsilon_0$, that is, $F_1(\mathbf{x}_2.s), F_2(\mathbf{x}_2.s) \geq 0$.

Next, we prove that $F_3(\mathbf{x}_2.s), F_4(\mathbf{x}_2.s) \geq 0$. From the definition of $\mathcal{I}_n$, we know that $-\frac{k_1}{k_2}\mathbf{x}_1.e_1 - \frac{1}{k_2}\phi_n \leq \mathbf{x}_1.e_2 \leq -\frac{k_1}{k_2}\mathbf{x}_1.e_1 + \frac{1}{k_2}\phi_n$. From the update rule for e_2, it can be easily shown that $\mathbf{x}_2.e_2 = \mathbf{x}_1.e_2 - \sigma_1$. Thus, we get that $-\frac{k_1}{k_2}\mathbf{x}_1.e_1 - \frac{1}{k_2}\phi_n - \sigma_1 \leq \mathbf{x}_2.e_2 \leq -\frac{k_1}{k_2}\mathbf{x}_1.e_1 + \frac{1}{k_2}\phi_n - \sigma_1$. Using the bounds on $\mathbf{x}_2.e_1$, $\mathbf{x}_2.e_2$ and $\mathbf{x}_1.d$, we can derive that $k_1\mathbf{x}_2.e_1 + k_2\mathbf{x}_2.e_2 \leq k_1 v_{max}\Delta \sin|\sigma_1| + k_1\epsilon_n(1 - \cos\sigma_1) + \phi_n + k_2|\sigma_1|$ and $k_1\mathbf{x}_2.e_1 + k_2\mathbf{x}_2.e_2 \geq -k_1 v_{max}\Delta \sin|\sigma_1| - k_1\epsilon_n(1 - \cos\sigma_1) - \phi_n - k_2|\sigma_1|$. That is,

$$|k_1\mathbf{x}_2.e_1 + k_2\mathbf{x}_2.e_2| \quad \leq \quad k_1 v_{max}\Delta \sin|\sigma_1| + k_1\epsilon_n(1 - \cos\sigma_1) + \phi_n + k_2|\sigma_1|.$$

Therefore, Assumption 5.5.2(b) guarantees that $|k_1\mathbf{x}_2.e_1 + k_2\mathbf{x}_2.e_2| \leq \phi_0$. That is, $F_3(\mathbf{x}_2.s) \geq 0$ and $F_4(\mathbf{x}_2.s) \geq 0$. In addition, since a main action does not affect v, $F_5(\mathbf{x}_2.s) = F_5(\mathbf{x}_1.s)$ and $F_6(\mathbf{x}_2.s) = F_6(\mathbf{x}_1.s)$, so $F_5(\mathbf{x}_2.s), F_6(\mathbf{x}_1.s) \geq 0$.

Therefore, by definition of $\mathcal{I}_0$, we get $\beta_2.\mathsf{fstate} \in \mathcal{I}_0$. In addition, from the bounds on $\mathbf{x}_1.d$ and $\mathbf{x}_1.e_1$, it can be easily shown that $\beta_2.\mathsf{fstate} \lceil d \geq \lambda_2 - \epsilon_0 - v_{max}\Delta$ where λ_2 is the length of the segment $\beta_2.\mathsf{fstate} \lceil seg$.

Next, consider an arbitrary $i \geq 2$ and assume that $\beta_{i-1}.\mathsf{fstate} \in \mathcal{I}_0$ and if $i = 2$ or $i > 2$ and $\beta_{i-1}.\mathsf{fstate} \lceil seg \neq \beta_{i-2}.\mathsf{lstate} \lceil seg$, then $\beta_{i-1}.\mathsf{fstate} \lceil d \geq \lambda_{i-1} - \epsilon_0 - v_{max}\Delta$ where λ_{i-1} is the length of the segment $\beta_{i-1}.\mathsf{fstate} \lceil seg$. Simply following the previous proof for $i = 2$, we get $\beta_i.\mathsf{fstate} \in \mathcal{I}_0$ and if $\beta_i.\mathsf{fstate} \lceil seg \neq \beta_{i-1}.\mathsf{lstate} \lceil seg$, then $\beta_i.\mathsf{fstate} \lceil d \geq \lambda_i - \epsilon_0 - v_{max}\Delta$ where λ_i is the length of the segment $\beta_i.\mathsf{fstate} \lceil seg$.

By mathematical induction, we conclude the proof that for any $i > 1$, $\beta_i.\mathsf{fstate} \in \mathcal{I}_0$. □

Lemma 5.5.7. For each state $\mathbf{x}, \mathbf{x}' \in Q$ such that $\mathbf{x}.path \neq \mathbf{x}.new\_path$, if $\mathbf{x} \in \mathcal{I}_0$ and $\mathbf{x} \overset{\mathsf{main}}{\rightarrow} \mathbf{x}'$, then $\mathbf{x}'.d \geq \lambda - v_{max}\Delta > 0$ where λ is the length of the first segment of $\mathbf{x}.new\_path$.

Proof. Consider an arbitrary execution $\alpha = \tau_0 a_1 \tau_1 a_2 \ldots$. Pick an arbitrary natural number i such that a_i is a main action and let $\mathbf{x} = \tau_{i-1}.\mathsf{lstate}$ and $\mathbf{x}' = \tau_i.\mathsf{fstate}$. We want to show that if $\mathbf{x} \lceil path \neq \mathbf{x} \lceil new\_path$, then $\mathbf{x}'.d \geq \lambda - v_{max}\Delta > 0$. Notice that $\mathbf{x}.path \neq \mathbf{x}.new\_path$ if and only if there exists a natural number $j < i$ such that a_j is a plan action and for any natural number $k \in \{j+1, \ldots, i-1\}$, a_k is not a main action. Using Assumption 5.5.3(a), we get $\langle \tau_j.\mathsf{fstate} \lceil x, \tau_j.\mathsf{fstate} \lceil y \rangle = p_{i,1}$ where $p_{i,1}$ is the first waypoint in $\mathbf{x}.new\_path$. Since main action occurs every Δ time, the time between a_i and a_j is at most Δ. Therefore, from Theorem 5.5.1, the definition of F_5 and F_6 and the definition of f_1 and f_2 which describe the evolution of x and y, we see that $\|\langle \mathbf{x}.x, \mathbf{x}.y \rangle - p_{i,1}\| \leq v_{max}\Delta$. Furthermore, from Assumption 5.5.2(a), we know that $\lambda = \|p_{i,2} - p_{i,1}\| > v_{max}\Delta + \epsilon_0$ where $p_{i,2}$ is the second waypoint in p_i. Thus, $\mathbf{x}.d \geq \|p_{i,2} - p_{i,1}\| - \|\langle \mathbf{x}.x, \mathbf{x}.y \rangle - p_{i,1}\| \geq \lambda - v_{max}\Delta > 0$. □

Lemma 5.5.8. For each state $\mathbf{x}, \mathbf{x}' \in Q$ such that $\mathbf{x}.path \neq \mathbf{x}.new\_path$, if $\mathbf{x} \in \mathcal{I}_0$ and $\mathbf{x} \overset{\mathsf{main}}{\rightarrow} \mathbf{x}'$, then $\mathbf{x}' \in \mathcal{I}_0$.

Proof. Consider an arbitrary execution $\alpha = \tau_0 a_1 \tau_1 a_2 \ldots$. Pick an arbitrary natural number i such that a_i is a main action and let $\mathbf{x} = \tau_{i-1}.\mathsf{lstate}$ and $\mathbf{x}' = \tau_i.\mathsf{fstate}$. We want to show that if $\mathbf{x} \in \mathcal{I}_0$ and $\mathbf{x}.path \neq \mathbf{x}.new\_path$, then $\mathbf{x}' \in \mathcal{I}_0$. So suppose $\mathbf{x} \in \mathcal{I}_0$. Notice that $\mathbf{x}.path \neq \mathbf{x}.new\_path$ if and only if there exists a natural number $j < i$ such that a_j is a plan action and for any natural number $k \in \{j+1, \ldots, i-1\}$, a_k is not a main action. Let p_{j1} and p_{j2} be the first two waypoints of the new path. Consider a closed execution fragment $\beta = \tau_j a_{j+1} \ldots \tau_{i-1}$. From Assumption 5.5.3(a), we get that $p_{j1} = \tau_j.\mathsf{fstate} \lceil \langle x, y \rangle$. Since main action occurs every Δ time, we see that $\beta.\mathsf{ltime} \leq \Delta$. From the differential equations

describing the evolution of x and y, we get that

$$\begin{aligned} |(\tau_j.\mathsf{fstate}\lceil x) - (\mathbf{x}.x)| &\leq ((\tau_j.\mathsf{fstate}\lceil v) + a_{max}\Delta)\Delta, \\ |(\tau_j.\mathsf{fstate}\lceil y) - (\mathbf{x}.y)| &\leq \sin\theta_{0,2}((\tau_j.\mathsf{fstate}\lceil v) + a_{max}\Delta)\Delta. \end{aligned}$$

So from the definition of $\vec{r}$ in Figure 5.6, we get that

$$\|\vec{r}\| \leq (\tau_j.\mathsf{fstate}\lceil v) + a_{max}\Delta)\Delta\sqrt{1+\sin^2\theta_{0,2}}.$$

Using Assumption 5.5.3(b), we can conclude that $\|\vec{r}\| \leq \epsilon_0$. So from the update rule for e_1, $|\mathbf{x}'.e_1| \leq \|\vec{r}\|$ and so

$$|\mathbf{x}'.e_1| \leq (\tau_j.\mathsf{fstate}\lceil v) + a_{max}\Delta)\Delta\sqrt{1+\sin^2\theta_{0,2}} \leq \epsilon_0, \tag{5.25}$$

that is $F_1(\mathbf{x}'.s), F_2(\mathbf{x}'.s) \geq 0$.

Similarly, from the differential equation describing the evolution of θ, we get that

$$|(\tau_j.\mathsf{fstate}\lceil \theta) - (\mathbf{x}.\theta)| \leq \frac{1}{L}\tan\phi_0((\tau_j.\mathsf{fstate}\lceil v) + a_{max}\Delta)\Delta.$$

Using Assumption 5.5.3(b), we can conclude that

$$\begin{aligned} |\measuredangle\vec{p} - (\mathbf{x}.\theta)| &= |(\measuredangle\vec{p} - (\tau_j.\mathsf{fstate}\lceil \theta)) + ((\tau_j.\mathsf{fstate}\lceil \theta) - (\mathbf{x}.\theta))| \\ &\leq |(\measuredangle\vec{p}_i - (\tau_j.\mathsf{fstate}\lceil \theta))| + |((\tau_j.\mathsf{fstate}\lceil \theta) - (\mathbf{x}.\theta))| \\ &\leq \frac{\phi_0}{k_2} - \frac{k_1}{k_2}((\tau_j.\mathsf{fstate}\lceil v) + a_{max}\Delta)\Delta\sqrt{1+\sin^2\theta_{0,2}}. \end{aligned}$$

So we get

$$|k_2\mathbf{x}'.e_2| \leq \phi_0 - k_1((\tau_j.\mathsf{fstate}\lceil v) + a_{max}\Delta)\Delta\sqrt{1+\sin^2\theta_{0,2}}.$$

Combining this with (5.25), we get that

$$|k_1(\mathbf{x}'.e_1) + k_2(\mathbf{x}'.e_2)| \leq |k_1(\mathbf{x}'.e_1)| + |k_2(\mathbf{x}'.e_2)| \leq \phi_0,$$

that is, $F_3(\mathbf{x}'.s), F_4(\mathbf{x}'.s) \geq 0$.

In addition, since main action does not affect v, we see that $F_5(\mathbf{x}'.s) = F_5(\mathbf{x}.s)$ and $F_6(\mathbf{x}'.s) = F_6(\mathbf{x}.s)$, so $F_5(\mathbf{x}'.s), F_6(\mathbf{x}'.s) \geq 0$. Therefore, by definition of $\mathcal{I}_0$, we get that $\mathbf{x}' \in \mathcal{I}_0$. □

Chapter 6

Automatic Synthesis of Embedded Control Software

The design flaw in Alice as described in Section 1.1 was, in fact, partially known shortly before the second run of Test Area A. However, it was difficult to modify and verify the design during the National Qualifying Event due to the complexity of the system and the lack of sufficient time. Although it might be impossible to make such a system simple, part of the complexity could have been avoided if the system had been designed in a systematic way. As an effort towards this systematic design direction, this chapter presents an approach that allows embedded control software such as the planner-controller subsystem of Alice to be automatically synthesized such that the system is provably correct with respect to its requirements expressed in linear temporal logic.

6.1 Overview

In this chapter, we investigate the problem of automatically synthesizing an embedded control component to provide a formal guarantee that, by construction, the system satisfies the desired properties, which we also refer to as the specification. We assume that the desired properties are expressed in linear temporal logic.

A common approach to such a synthesis problem is to construct a finite transition system that serves as an abstract model of the physical system (which typically has infinitely many states) [57, 65, 25, 64, 113, 40, 122, 124]. Then based on this abstract model, synthesize a strategy, represented by a finite state automaton, satisfying the specification. This leads to a hierarchical, two-layer design with a discrete planner computing a discrete plan based

on the abstract model and a continuous controller computing a sequence of control signals based on the physical model to continuously implement the plan. Simulations/bisimulations [6] provide the proof that the continuous execution preserves the correctness of the discrete plan.

The correctness of this hierarchical approach relies on the abstraction of systems evolving on a continuous domain into equivalent (in the simulation sense) finite state models. If the abstraction is done properly such that the continuous controller is capable of implementing any discrete plan computed by the discrete planner, then it is guaranteed that the correctness of the plan is preserved in the continuous execution.

Several abstraction methods have been proposed based on a fixed abstraction. For example, a continuous-time, time-invariant model was considered in [65], [25] and [64] for special cases of fully actuated ($\dot{s}(t) = u(t)$), kinematic ($\dot{s}(t) = A(s(t))u(t)$) and piecewise affine dynamics, respectively, while a discrete-time, time-invariant model was considered in [122] and [113] for special cases of piecewise affine and controllable linear systems, respectively. Reference [40] deals with more general dynamics by relaxing the bisimulation requirement and using the notions of approximate simulation and simulation functions [39]. More recently, a sampling-based method has been proposed for μ-calculus specifications [57]. However, these approaches do not take into account the presence of exogenous disturbances and the resulting system may fail to satisfy its specification if its evolution does not exactly match its model.

To increase the robustness of the system against the effects of direct, external disturbances and a mismatch between the actual system and its model, in this chapter, we extend the existing abstraction approaches to deal with a discrete-time linear time-invariant state space model with exogenous disturbances and provide an approach to automatically compute a finite state abstraction for such a model.

The remainder of the chapter is organized as follows. In Section 6.2, we present the key definitions and notations. A brief description of a digital design synthesis tool that we use for automatically synthesizing a discrete planner is also provided. The planner-controller synthesis problem is formulated in Section 6.3. The hierarchical approach is described in detail in Section 6.4. Section 6.5 provides an approach to automatically compute a finite state abstraction for a discrete-time linear time-invariant system, taking into account exogenous disturbances.

6.2 Preliminaries

We use a variable structure to specify the states of the system (as described in Definition 2.2.1) and linear temporal logic to specify properties of a system (see Section 2.2).

As previously discussed, a finite transition system is used as a mathematical object that represents an abstraction of the physical system.

Definition 6.2.1. A *finite transition system* is a tuple $\mathbb{T} := (\mathcal{V}, \mathcal{V}_0, \rightarrow)$ where $\mathcal{V}$ is a finite set of states, $\mathcal{V}_0 \subseteq \mathcal{V}$ is a non-empty set of inital states, and $\rightarrow \subseteq \mathcal{V} \times \mathcal{V}$ is a transition relation. Given states $\nu_i, \nu_j \in \mathcal{V}$, we write $\nu_i \rightarrow \nu_j$ if there is a transition from ν_i to ν_j in $\mathbb{T}$.

Observe that we use ν to represent a state of a finite transition system and v to represent a state of a general, possibly non-finite state system.

6.2.1 Synthesis of a Digital Design: A Two-Player Game Approach

In many applications, systems need to interact with their environments and whether they satisfy the desired properties depends on the behavior of the environments. For example, whether an autonomous car exhibits the correct behavior at an intersection depends on the behavior of other cars at the intersection, e.g., which car gets to the intersection first, etc. In this section, we informally describe the work of Piterman, et al. [98]. We refer the reader to [98] and references therein for the detailed discussion of automatic synthesis of a finite state automaton from its specification.

From Definition 2.2.4, for a system to be correct, its specification φ must be satisfied in all of its executions regardless of the behavior of the environment in which it operates. Thus, the environment can be treated as an adversary and the synthesis problem can be viewed as a two-player game between the system and the environment: the environment attempts to falsify φ while the system attempts to satisfy φ. We say that φ is *realizable* if the system can satisfy φ no matter what the environment does.

For a specification of the form

$$(\bigwedge_{i \in I} \Box\Diamond\varphi_i) \implies (\bigwedge_{j \in J} \Box\Diamond\psi_j),$$

known as *Generalized Reactivity(1)*, Piterman et al. shows that checking its realizability and synthesizing the corresponding automaton can be performed in polynomial time. In

particular, we are interested in a specification of the form

$$\varphi = (\varphi_e \implies \varphi_s)$$

where roughly speaking, φ_e characterizes the initial states of the system and the assumptions on the environment and φ_s describes the correct behavior of the system, including the valid transitions the system can make. We refer the reader to [98] for precise definitions of φ_e and φ_s. Note that since $\varphi_e \implies \varphi_s$ is satisfied whenever φ_e is *False*, if the assumptions on the environment or the initial state of the system violate φ_e, then the correct behavior φ_s of the system is not ensured, even though the specification φ is satisfied.

If the specification is realizable, the digital design synthesis tool presented in [98] generates a finite state automaton that represents a set of transitions the system should follow in order to satisfy φ. Assuming that the environment and the initial state of the system satisfy φ_e, then at any instance of time, there exists a node in the automaton that represents the current state of the system and the system can follow the transition from this node to the next based on the current knowledge about the environment. However, if φ_e is violated, the automaton is no longer valid, meaning that there may not exist a node in the automaton that represents the current state of the system, or even though such a node exists and the system follows the transitions in the automaton, the correct behavior φ_s is not guaranteed.

If the specification is not realizable, the synthesis tool provides an initial state of the system starting from which there exists a set of moves of the environment such that the system cannot satisfy φ. The knowledge of the realizability of the specification is useful since it provides information about the conditions under which the system will fail to satisfy its desired properties.

The main limitation of the synthesis of finite state automata is the state explosion problem. In the worst case, the resulting automaton may contain all the possible states of the system. For example, if the system has N variables, each can take any value in $\{1, \ldots, M\}$, then there may be as many as M^N nodes in the automaton.

6.2.2 Synthesis of a Continuous Controller: An Optimization-Based Approach

Control systems are usually described by a set of differential or difference equations. These continuous systems typically contain an infinite number of states. Hence, the digital design synthesis tool, which relies on the finiteness of the number of system states as described in Section 6.2, cannot be directly applied. For certain classes of systems, however, a provably correct controller can be automatically constructed using an optimization-based approach, provided the desired system properties are restricted to a certain class of safety and guarantee properties.

Consider the discrete-time linear control system

$$s[t+1] = As[t] + Bu[t], \quad t \in \{0, 1, \ldots\},$$

where $s[t] \in \mathbb{R}^n$ represents the state of the system, $u[t] \in \mathbb{R}^m$ represents the control input to the plant and $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$.

Given a fixed horizon $N \in \{0, 1, \ldots\}$ and a cost function $J : (s[0], \ldots, s[N], u[0], \ldots, u[N-1]) \mapsto \mathbb{R}$, the problem of finding a sequence of control signals $u[0], \ldots, u[N-1]$ that optimizes the given cost function subject to linear inequality constraints on the states $s[0], \ldots, s[N]$ and control signals $u[0], \ldots, u[N-1]$ can be formulated as a convex optimization problem, provided that the cost function is convex. An example of such convex cost function is

$$J(s[0], \ldots, s[N], u[0], \ldots, u[N-1]) \triangleq \|Ps[N]\|_2 + \sum_{t=0}^{N-1} (\|Qs[t]\|_2 + \|Ru[t]\|_2)$$

where P and Q are positive semidefinite matrices and R is a positive definite matrix. l_1-norm and l_∞-norm can also be used with some extra assumptions on P, Q and R.

In particular, we are interested in the problem of controlling the state of the system from a given initial state $s[0] \in \mathbb{R}^n$ to a given goal region $G \subseteq \mathbb{R}^n$. We also require that for all $t \in \{0, \ldots, N-1\}$, $s[t]$ stays within a given safe set $S \subseteq \mathbb{R}^n$ and $u[t]$ stays within the set $U \subseteq \mathbb{R}^m$ of admissible control inputs. The corresponding finite horizon constrained optimal

control problem is given by

$$\begin{aligned} \min_{u[0],\ldots,u[N-1]} \quad & \|P\hat{s}[N]\|_2 + \sum_{t=0}^{N-1} (\|Q\hat{s}[t]\|_2 + \|Ru[t]\|_2) \\ \text{such that} \quad & s[N] \in G, \\ & s[t+1] = As[t] + Bu[t], \\ & u[t] \in U, \\ & s[t] \in S, \\ & \forall t \in \{0,\ldots,N-1\}, \end{aligned} \tag{6.1}$$

where for any $t \in \{0,\ldots,N\}$, $\hat{s}[t] = s[t] - s_j$ for some chosen $s_j \in G$ (e.g., s_j may be the center of the goal region G). As previously discussed, for the case where P and Q are positive semidefinite matrices, R is a positive definite matrix and G, S and U are polyhedral sets, i.e., sets defined by affine inequalities, the explicit solution $u[0],\ldots,u[N-1]$ of (6.1) can be computed using convex optimization [16].

This optimization-based approach can be extended to handle a linear time-invariant state space model with bounded exogenous disturbances

$$s[t+1] = As[t] + Bu[t] + Ed[t], \quad d[t] \in D, t \in \{0,1,\ldots\},$$

provided that the set $D \subseteq \mathbb{R}^p$ of exogenous disturbances is polyhedral. In this case, the constraints on the states $s[0],\ldots,s[N]$ and control signals $u[0],\ldots,u[N-1]$ need to be ensured for any sequence of exogenous disturbances $d[0],\ldots,d[N-1] \in D$.

More details on solving such constrained optimal control problems can be found in [14]. Off-the-shelf software such as MPT [67], YALMIP [75] or NTG [92] provides a computational tool for solving such a constrained optimal control problem.

6.3 Problem Formulation

We consider a system that comprises the physical component, which we refer to as the plant, and the (potentially dynamic and not a priori known) environment in which the plant operates. Assuming that the system specification φ is expressed in LTL, we are interested in automatically synthesizing a planner-controller subsystem that generates control signals to the plant in order to ensure that φ is satisfied in the presence of exogenous disturbances

for any initial condition and any environment in which the plant operates. Specifically, we define the system model $\mathbb{S}$ and the specification φ as follows.

System Model: Consider a system model $\mathbb{S}$ with a set $V = S \cup E$ of variables where S and E are disjoint sets that represent, respectively, the set of plant variables that are regulated by the planner-controller subsystem and the set of environment variables whose values may change arbitrarily throughout an execution. The domain of V is given by $dom(V) = dom(S) \times dom(E)$ and a state of the system can be written as $v = (s, e)$ where $s \in dom(S) \subseteq \mathbb{R}^n$ and $e \in dom(E)$. In this thesis, we call s the *controlled* state and e the *environment* state.

Assume that the controlled state evolves according to the following discrete-time linear time-invariant state space model: for $t \in \{0, 1, 2, \ldots\}$,

$$\begin{aligned} s[t+1] &= As[t] + Bu[t] + Ed[t], \\ u[t] &\in U, \\ d[t] &\in D, \\ s[0] &\in dom(S), \end{aligned} \tag{6.2}$$

where $U \subseteq \mathbb{R}^m$ is the set of admissible control inputs, $D \subseteq \mathbb{R}^p$ is the set of exogenous disturbances and $s[t]$, $u[t]$ and $d[t]$ are the controlled state, the control signal and the exogenous disturbance, respectively, at time t.

Example 6.3.1. Consider a robot motion planning problem where a robot needs to navigate an environment populated with (potentially dynamic) obstacles and explore certain areas of interest. S typically includes the state (e.g., position and velocity) of the robot while E typically includes the positions of obstacles (which are generally not known a priori and may change over time). The evolution of the controlled state (i.e., the state of the robot) is governed by its equations of motion, which can be written in the form of (6.2) (after linearization, if necessary).

System Specification: We assume that the specification φ consists of the following components:

(a) the assumption φ_{init} on the initial condition of the system,

(b) the assumption φ_e on the environment, and

(c) the desired behavior φ_s of the system.

Specifically, we assume that φ can be written as

$$\varphi = \big(\varphi_{init} \wedge \varphi_e\big) \implies \varphi_s. \tag{6.3}$$

Let Π be a finite set of atomic propositions of variables from V. Each of the atomic propositions in Π essentially captures the states of interest. We assume that the desired behavior φ_s is an LTL specification built from Π and can be expressed as a conjunction of safety, guarantee, obligation, progress, response and stability properties as follows:

$$\begin{aligned} \varphi_s = \; & \textstyle\bigwedge_{j\in J_1} \Box p^s_{1,j} \wedge \bigwedge_{j\in J_2} \Diamond p^s_{2,j} \wedge \\ & \textstyle\bigwedge_{j\in J_3} (\Box p^s_{3,j} \vee \Diamond q^s_{3,j}) \wedge \bigwedge_{j\in J_4} \Box\Diamond p^s_{4,j} \wedge \\ & \textstyle\bigwedge_{j\in J_5} \Box(p^s_{5,j} \implies \Diamond q^s_{5,j}) \wedge \bigwedge_{j\in J_6} \Diamond\Box\, p^s_{6,j}, \end{aligned} \tag{6.4}$$

where $J_1, \ldots, J_6$ are finite sets and for any i and j, $p^s_{i,j}$ and $q^s_{i,j}$ are propositional formulas of variables from V that are built from Π.

Furthermore, we assume that φ_{init} is a propositional formula built from Π and φ_e can be expressed as a conjunction of safety and justice requirements as follows

$$\varphi_e = \bigwedge_{i\in I_1} \Box p^e_{f,i} \wedge \bigwedge_{i\in I_2} \Box\Diamond p^e_{s,i}, \tag{6.5}$$

where $p^e_{f,i}$ and $p^e_{s,i}$ are propositional formulas built from Π and only contain variables from E.

Example 6.3.2. Consider the robot motion planning problem described in Example 6.3.1. Suppose the workspace of the robot is partitioned into cells $C_1, \ldots, C_M$ and the robot needs to explore (i.e., visit) the cells C_1 and C_2 infinitely often. In addition, we assume that one of the cells $C_1, \ldots, C_M$ may be occupied by an obstacle at any given time and this obstacle-occupied cell may change arbitrarily throughout an execution but infinitely often, C_1 and C_2 are not occupied. Let s and o represent the position of the robot and the obstacle, respectively. In this case, the desired behavior of the system can be written as

$$\begin{aligned} \varphi_s = \; & \Box\Diamond(s \in C_1) \wedge \Box\Diamond(s \in C_2) \wedge \Box\big((o \in C_1) \implies (s \notin C_1)\big) \wedge \\ & \Box\big((o \in C_2) \implies (s \notin C_2)\big) \wedge \ldots \wedge \Box\big((o \in C_M) \implies (s \notin C_M)\big). \end{aligned}$$

Assuming that initially, the robot does not occupy the same cell as the obstacle, we simply let

$$\varphi_{init} = \big((o \in C_1) \implies (s \notin C_1)\big) \wedge \big((o \in C_2) \implies (s \notin C_2)\big) \wedge \ldots \wedge \big((o \in C_m) \implies (s \notin C_m)\big).$$

Finally, the assumption on the environment can be expressed as

$$\varphi_e = \Box\Diamond(o \notin C_1) \ \wedge \ \Box\Diamond(o \notin C_2).$$

Planner-Controller Synthesis Problem: Given the system model $\mathbb{S}$ and the system specification φ, synthesize a planner-controller subsystem that generates a sequence of control signals $u[0], u[1], \ldots \in U$ to the plant to ensure that starting from any initial condition, φ is satisfied for any sequence of exogenous disturbances $d[0], d[1], \ldots \in D$ and any sequence of environment states.

Remark 6.3.1. We restrict φ_s and φ_e to be of the form (6.4) and (6.5), respectively, for the clarity of presentation. Our framework only requires that the specification (6.3) can be reduced to the form of Equation (6.7), presented later.

Remark 6.3.2. The specification φ has to be satisfied for any initial condition and environment, including those that violate the assumptions φ_{init} and φ_e. However, according to (6.3), satisfying φ ensures that the system exhibits the desired behavior φ_s only when φ_{init} and φ_e are satisfied.

6.4 Hierarchical Approach

As described in Section 6.1, we follow a hierarchical approach to attack the Planner-Controller Synthesis Problem defined in Section 6.3. First, we construct a finite transition system $\mathbb{D}$ (e.g., a Kripke structure) that serves as an abstract model of $\mathbb{S}$ (which typically has infinitely many states). With this abstraction, the problem is then decomposed into

- synthesizing a discrete planner that computes a discrete plan satisfying the specification φ based on the abstract, finite-state model $\mathbb{D}$, and
- designing a continuous controller that implements the discrete plan.

The success of this abstraction-based approach thus relies on the following two critical steps:

(a) an abstraction of an infinite-state system into an equivalent (in the simulation sense) finite state model such that any discrete plan generated by the discrete planner can be implemented (i.e., *simulated*; see, for example, [114] for the exact definition of *simulation*) by the continuous controller, provided that the evolution of the controlled state satisfies (6.2), and

(b) synthesis of a discrete planner (i.e., a strategy), represented by a finite state automaton, that ensures the correctness of the discrete plan.

In Section 6.5, we present an approach to handle step (a), assuming that the physical system is modeled as described in Section 6.3. To handle step (b) and ensure the system correctness for any initial condition and environment, we apply the two-player game approach presented in [98] to synthesize a discrete planner as in [65, 122]. In summary, consider a class of LTL formulas of the form

$$\left(\psi_{init} \wedge \Box\psi_e \wedge \bigwedge_{i\in I_f} \Box\Diamond\psi_{f,i}\right) \Longrightarrow \left(\bigwedge_{i\in I_s} \Box\psi_{s,i} \wedge \bigwedge_{i\in I_g} \Box\Diamond\psi_{g,i}\right), \tag{6.6}$$

known as *Generalized Reactivity[1]* (GR[1]) formulas. Here, ψ_{init}, $\psi_{f,i}$ and $\psi_{g,i}$ are propositional formulas of variables from V; ψ_e is a Boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_e^t$ where ψ_e^t is a propositional formula of variables from E that describes the assumptions on the transitions of environment states; and $\psi_{s,i}$ is a Boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_s^t$ where ψ_s^t is a propositional formula of variables from V that describes the constraints on the transitions of controlled states. The approach presented in [98] allows checking the realizability of this class of specifications and synthesizing the corresponding finite state automaton to be performed in time $O(|\mathcal{V}|^3)$ where $|\mathcal{V}|$ is the size of the state space of the finite state abstraction $\mathbb{D}$ of the system. We refer the reader to [98] and references therein for a detailed discussion.

Proposition 6.4.1. *A specification of the form (6.3) can be reduced to a subclass of GR[1] formula of the form*

$$\left(\psi_{init} \wedge \Box\psi_e^e \bigwedge_{i\in I_f} \Box\Diamond\psi_{f,i}^e\right) \Longrightarrow \left(\bigwedge_{i\in I_s} \Box\psi_{s,i} \wedge \bigwedge_{i\in I_g} \Box\Diamond\psi_{g,i}\right), \tag{6.7}$$

where ψ_{init}, $\psi_{s,i}$ and $\psi_{g,i}$ are as defined above and ψ^e_e and $\psi^e_{f,i}$ are propositional formulas of variables from E.

In this thesis, we call the left-hand side and the right-hand side of (6.7) the "assumption" part and the "guarantee" part, respectively.

The proof of Proposition 6.4.1 is based on the fact that all safety, guarantee, obligation and response properties are special cases of progress formulas $\Box\Diamond p$, provided that p is allowed to be a past formula [82]. Hence, these properties can be reduced to the "guarantee" part of (6.7) by introducing auxiliary Boolean variables. For example, a guarantee property $\Diamond p^s_{2,j}$ can be reduced to the "guarantee" part of (6.7) by introducing an auxiliary Boolean variable x, initialized to $p^s_{2,j}$. $\Diamond p^s_{2,j}$ can then be equivalently expressed as a conjunction of $\Box\big((x \vee p^s_{2,j}) \implies \bigcirc x\big)$, $\Box\big(\neg(x \vee p^s_{2,j}) \implies \bigcirc(\neg x)\big)$ and $\Box\Diamond x$. Obligation and response properties can be reduced to the "guarantee" part of (6.7) using a similar idea. In addition, a stability property $\Diamond\Box p^s_{6,j}$ can be reduced to the "guarantee" part of (6.7) by introducing an auxiliary Boolean variable y, initialized to *False*. $\Diamond\Box p^s_{6,j}$ can then be equivalently expressed as a conjunction of $\Box(y \implies p^s_{6,j})$, $\Box(y \implies \bigcirc y)$, $\Box\big(\neg y \implies (\bigcirc y \vee \bigcirc(\neg y))\big)$ and $\Box\Diamond y$. Note that these reductions lead to equivalent specifications. However, for the case of stability, the reduction may lead to an unrealizable specification even though the original specification is realizable. Roughly speaking, this is because the auxiliary Boolean variable y needs to make clairvoyant (prophecy), non-deterministic decisions. For other properties, the realizability remains the same after the reduction since the synthesis algorithm [98] is capable of handling past formulas. The detail of this discussion is beyond the scope of this thesis and we refer the reader to [98] for more detailed discussion on the synthesis of GR[1] specification.

6.5 Computing Finite State Abstraction

To construct a finite transition system $\mathbb{D}$ from the physical model $\mathbb{S}$, we first partition $dom(S)$ and $dom(E)$ into finite sets $\mathcal{S}$ and $\mathcal{E}$, respectively, of equivalence classes or cells such that the partition is *proposition preserving* [6]. Roughly speaking, a partition is said to be proposition preserving if for any atomic proposition $\pi \in \Pi$ and any states v_1 and v_2 that belong to the same cell in the partition, v_1 satisfies π iff v_2 also satisfies π. We denote the resulting discrete domain of the system by $\mathcal{V} = \mathcal{S} \times \mathcal{E}$. We call $v \in dom(V)$ a *continuous state*

and $\nu \in \mathcal{V}$ a *discrete state* of the system. For a discrete state $\nu \in \mathcal{V}$, we say that ν satisfies an atomic proposition $\pi \in \Pi$, denoted by $\nu \Vdash_d \pi$, if and only if there exists a continuous state v contained in the cell labeled by ν such that v satisfies π. Given an infinite sequence of discrete states $\sigma_d = \nu_0\nu_1\nu_2\ldots$ and LTL formula φ built from Π, we say that φ holds at position $i \geq 0$ of σ_d, written $\nu_i \vDash_d \varphi$, if and only if φ holds for the remainder of σ_d starting at position i. With these definitions, the semantics of LTL for a sequence of discrete states can be derived from the general semantics of LTL.

Next, we need to determine the transition relations $\rightarrow$ of $\mathbb{D}$. In Section 6.5.1, we use a variant of the notion of *reachability* that is sufficient to guarantee that if a discrete controlled state ς_j is reachable from ς_i, the transition from ς_i to ς_j can be continuously *implemented* or *simulated* by a continuous controller. A computational scheme that provides a sufficient condition for reachability between two discrete controlled states and subsequently refines the state space partition is also presented in Section 6.5.3 and 6.5.4.

6.5.1 Finite Time Reachability

Let $\mathcal{S} = \{\varsigma_1, \varsigma_2, \ldots, \varsigma_l\}$ be a set of discrete controlled states. We define a map $T_s : dom(S) \rightarrow \mathcal{S}$ that sends a continuous controlled state to a discrete controlled state of its equivalence class. That is, $T_s^{-1}(\varsigma_i) \subseteq dom(S)$ is the set of all the continuous controlled states contained in the cell labeled by ς_i and $\{T_s^{-1}(\varsigma_i), \ldots, T_s^{-1}(\varsigma_n)\}$ is the partition of $dom(S)$. We define the reachability relation, denoted by $\rightsquigarrow$, as follows.

Definition 6.5.1. A discrete state ς_j is *reachable* from a discrete state ς_i, written $\varsigma_i \rightsquigarrow \varsigma_j$, only if starting from any point $s[0] \in T_s^{-1}(\varsigma_i)$, there exists a horizon length $N \in \{0, 1, \ldots\}$ and a sequence of control signals $u[0], u[1], \ldots, u[N-1] \in U$ that takes the system (6.2) to a point $s[N] \in T_s^{-1}(\varsigma_j)$ satisfying the constraint $s[t] \in T_s^{-1}(\varsigma_i) \cup T_s^{-1}(\varsigma_j), \forall t \in \{0, \ldots, N\}$ for any sequence of exogenous disturbances $d[0], d[1], \ldots, d[N-1] \in D$. We write $\varsigma_i \not\rightsquigarrow \varsigma_j$ if ς_j is not reachable from ς_i.

In general, for two discrete states ς_i and ς_j, verifying the reachability relation $\varsigma_i \rightsquigarrow \varsigma_j$ is hard because it requires searching for a proper horizon length N. Therefore, we consider the restricted case where the horizon length is fixed and given and U, D and $T_s^{-1}(\varsigma_i), i \in \{1, \ldots, l\}$ are polyhedral sets. Our approach relies on solving the following problem.

Reachability Problem: Given an initial continuous controlled state $s[0] \in \mathbb{R}^n$, discrete

controlled states $\varsigma_i, \varsigma_j \in \mathcal{S}$, the set of admissible control inputs $U \subseteq \mathbb{R}^m$, the set of exogenous disturbances $D \subseteq \mathbb{R}^p$, the matrices A, B and E as in (6.2), a horizon length $N \geq 0$, determine a sequence of control signals $u[0], u[1], \ldots, u[N-1] \in \mathbb{R}^m$ such that for all $t \in \{0, \ldots, N-1\}$ and $d[t] \in D$,

$$\begin{aligned} s[t+1] &= As[t] + Bu[t] + Ed[t], \\ s[t] &\in T_s^{-1}(\varsigma_i), \\ u[t] &\in U, \\ s[N] &\in T_s^{-1}(\varsigma_j). \end{aligned} \tag{6.8}$$

6.5.2 Preliminaries on Polyhedral Convexity

We consider the case where U, D and $T_s^{-1}(\varsigma_i), i \in \{1, \ldots, l\}$ are polyhedral sets defined as follows.

Definition 6.5.2. A subset P of $\mathbb{R}^n$ is said to be a *polyhedral set* if it is non-empty and has the form $P = \{p \mid Gp \leq h\}$ for some $G \in \mathbb{R}^{r \times n}$ and $h \in \mathbb{R}^r$, where the inequality $\leq$ is evaluated elementwise.

Definition 6.5.3. Let P be a non-empty convex set. A point $p \in P$ is an *extreme point* of P if and only if it does not lie strictly between the endpoints of any line segment contained in the set, i.e., for an extreme point p,

$$p = \lambda p_1 + (1-\lambda)p_2,\ p_1, p_2 \in P,\ \lambda \in (0,1) \implies p = p_1 = p_2.$$

To compute the set $\mathcal{S}_0$ of initial states for which the Reachability Problem is feasible, we apply the following results on polyhedral convexity. The proofs for the next three propositions can be found in [11].

Proposition 6.5.1. *Let P be a polyhedral subset of $\mathbb{R}^n$. If P has the form $P = \{p \in \mathbb{R}^n \mid g_j' p \leq h_j, j = 1, \ldots, r\}$ where $g_j \in \mathbb{R}^n$, g_j' is the transpose of g_j and $h_j \in \mathbb{R}$, then a point $p \in P$ is an extreme point of P if and only if the set $G_p \triangleq \{g_j \mid g_j' p = h_j, j \in \{1, \ldots, r\}\}$ contains n linearly independent vectors.*

Proposition 6.5.2. *Let P be a non-empty convex subset of $\mathbb{R}^n$. If P is closed, then P has at least one extreme point if and only if it does not contain a line, i.e., a set of the form $\{p + \lambda h \mid \lambda \in \mathbb{R}\}$, where $h \in \mathbb{R}^n$ is non-zero and $p \in P$.*

Proposition 6.5.3 (Fundamental Theorem of Linear Programming)**.** *Let P be a polyhedral set that has at least one extreme point. A linear function that is bounded below over P attains a mininum at some extreme point of P.*

Using Proposition 6.5.1, we can derive the following proposition.

Proposition 6.5.4. *Let P be a polyhedral subset of $\mathbb{R}^n$ and let $\overline{P}$ be the set of all its extreme points. For any natural number N, $P^N \triangleq P \times \ldots \times P$ (N times) is a polyhedral subset of $\mathbb{R}^{nN}$ and $\overline{P}^N \triangleq \overline{P} \times \ldots \times \overline{P}$ (N times) is the set of all its extreme points.*

Proof. Since P is a polyhedral set, by definition, it can be written as $P = \{p \in \mathbb{R}^n \mid Cp \leq D\}$ for some matrix $C \in \mathbb{R}^{r \times n}$ and vector $D \in \mathbb{R}^r$. Clearly, P^N is a polyhedral subset of $\mathbb{R}^{nN}$ since it can be written as $P^N = \{p \in \mathbb{R}^{nN} \mid \tilde{C}p \leq \tilde{D}\}$ where $\tilde{C} \in \mathbb{R}^{rN \times nN}$ is a block diagonal matrix whose diagonal blocks are C and $\tilde{D} = [D', \ldots, D']' \in \mathbb{R}^{rN}$.

Let $\overline{Q}$ be the set of all the extreme points of P^N. First, we will show that $\overline{P}^N \subseteq \overline{Q}$. Consider any point $q \in \overline{P}^N$. Then, $q = [p_1', \ldots, p_N']'$ for some $p_1, \ldots, p_N \in \overline{P}$. Let c_j' be the j^{th} row of C and let d_j be the j^{th} element of D. From Proposition 6.5.1, we get that for each $i \in \{1, \ldots, N\}$, the set $C_{p_i} \triangleq \left\{c_j \mid c_j' p_i = d_j, j \in \{1, \ldots, r\}\right\}$ contains n linearly independent vectors. Let $\tilde{c}_j'$ be the j^{th} row of $\tilde{C}$ and let $\tilde{d}_j$ be the j^{th} element of $\tilde{D}$. Define $C_q \triangleq \left\{\tilde{c}_j \mid \tilde{c}_j' q = \tilde{d}_j, j \in \{1, \ldots, rN\}\right\}$. From the block diagonal structure of $\tilde{C}$, it can be easily verified that

$$C_q = \bigcup_{i=1}^{N} \mathbf{0}^{n(i-1)} \times C_{p_i} \times \mathbf{0}^{n(N-i)},$$

where for any natural number m, $\mathbf{0}^m$ is a zero vector of length m. Since for each $i \in \{1, \ldots, N\}$, C_{p_i} contains n linearly independent vectors, it can be easily shown that C_q contains nN linearly independent vectors. Thus, using Proposition 6.5.1, we can conclude that $q \in \overline{Q}$.

Next, we will show that $\overline{Q} \subseteq \overline{P}^N$. Consider any point $q \in \overline{Q}$. Then, $q = [p_1', \ldots, p_N']'$ for some $p_1, \ldots, p_N \in P$. From Proposition 6.5.1, we get that the set $C_q \triangleq \left\{\tilde{c}_j \mid \tilde{c}_j' q = \tilde{d}_j, j \in \{1, \ldots, rN\}\right\}$ contains nN linearly independent vectors. From the block diagonal structure of $\tilde{C}$, it can be easily verified that for each $i \in \{1, \ldots, N\}$, the set $C_{p_i} \triangleq \left\{c_j \mid c_j' p_i = d_j, j \in \{1, \ldots, r\}\right\}$ must contain n linearly independent vectors. Using Proposition 6.5.1, we can conlude that for each $i \in \{1, \ldots, N\}$, $p_i \in \overline{P}$. □

In addition, the following proposition can be proved using Proposition 6.5.2.

Proposition 6.5.5. *Let P be a polyhedral subset of $\mathbb{R}^n$. If P is closed and bounded, then P has at least one extreme point.*

Proof. Assume, to arrive at a contradiction, that P does not have an extreme point. Then, from Proposition 6.5.2, P contains a line $L = \{p + \lambda h \mid \lambda \in \mathbb{R}\}$ where $h \in \mathbb{R}^n$ is non-zero and $p \in P$. This contradicts the assumption that P is bounded. □

Finally, the next three propositions can be found in standard textbooks on topology, e.g., [109].

Proposition 6.5.6 (Heine-Borel Theorem)**.** *A subset of Euclidean space $\mathbb{R}^n$ is compact if and only if it is closed and bounded.*

Proposition 6.5.7 (Tychonoff's Theorem)**.** *The product of any collection of compact topological spaces is compact.*

Proposition 6.5.8 (Extreme Value Theorem)**.** *A continuous real-valued function on a non-empty compact space is bounded and attains its supremum.*

6.5.3 Verifying the Reachability Relation

Given two discrete controlled states $\varsigma_i, \varsigma_j \in \mathcal{S}$, to determine whether $\varsigma_i \rightsquigarrow \varsigma_j$, we essentially have to verify that $T_s^{-1}(\varsigma_i) \subseteq S_0$ where S_0 is the set of $s[0]$ starting from which the Reachability Problem defined in Section 6.5.1 is feasible. In this section, we describe how S_0 can be computed using an idea from constrained robust optimal control [14].

We assume that U, D and $T_s^{-1}(\varsigma_i), i \in \{1, \ldots, l\}$ are polyhedral sets, i.e., there exist matrices L_1, L_2 and L_3 and vectors M_1, M_2 and M_3 such that $T_s^{-1}(\varsigma_i) = \{s \in \mathbb{R}^n \mid L_1 s \leq M_1\}$, $U = \{u \in \mathbb{R}^m \mid L_2 u \leq M_2\}$ and $T_s^{-1}(\varsigma_j) = \{s \in \mathbb{R}^n \mid L_3 s \leq M_3\}$. Then, by substituting

$$s[t] = A^t s[0] + \sum_{k=0}^{t-1} \left(A^k B u[t-1-k] + A^k E d[t-1-k] \right)$$

and replacing $s[t] \in T_s^{-1}(\varsigma_i)$, $u[t] \in U$ and $s[N] \in T_s^{-1}(\varsigma_j)$ with $L_1 s[t] \leq M_1$, $L_2 u[t] \leq M_2$ and $L_3 s[N] \leq M_3$, respectively, in (6.8), it can be easily checked that equation (6.8) can be rewritten in the form

$$L \begin{bmatrix} s[0] \\ \hat{u} \end{bmatrix} \leq M - G\hat{d}, \tag{6.9}$$

where $\hat{u} \triangleq [u[0]', \ldots, u[N-1]']' \in \mathbb{R}^{mN}$, $\hat{d} \triangleq [d[0]', \ldots, d[N-1]']' \in D^N$ and the matrices $L \in \mathbb{R}^{r \times n+mN}$ and $G \in \mathbb{R}^{r \times pN}$ and the vector $M \in \mathbb{R}^r$ can be obtained from L_1, L_2, L_3, M_1, M_2, M_3, A, B and E.

Using properties of polyhedral convexity, we can prove the following result.

Theorem 6.5.1. *Suppose D is a closed and bounded polyhedral subset of $\mathbb{R}^p$ and $\overline{D}$ is the set of all its extreme points. Let $P \triangleq \{y \in \mathbb{R}^{n+mN} \mid Ly \leq M - G\hat{d}, \forall \hat{d} \in \overline{D}^N\}$ and let S_0 be the projection of P onto its first n coordinates, i.e.,*

$$S_0 = \left\{ s \in \mathbb{R}^n \;\middle|\; \exists \hat{u} \in \mathbb{R}^{mN} \text{ s.t. } L \begin{bmatrix} s \\ \hat{u} \end{bmatrix} \leq M - G\hat{d}, \forall \hat{d} \in \overline{D}^N \right\}.$$

Then, the Reachability Problem defined in Section 6.5.1 is feasible for any $s[0] \in S_0$.

Proof. From the Heine-Borel theorem and the Tychonoff's theorem, we get that D^N is compact. For each $j \in \{1, \ldots, r\}$, let m_j be the j^{th} element of M and g_j' be the j^{th} row of G and define a linear function $f_j : D^N \to \mathbb{R}$ by $f_j(\hat{d}) = m_j - g_j'\hat{d}$. Since f_j is continuous and D^N is compact, from the extreme value theorem, f_j is bounded below over D^N. In addition, since D^N is compact, from the Heine-Borel theorem and Proposition 6.5.5, D^N has at least one extreme point. Using the fundamental theorem of linear programming, we can conclude that f_j attains a minimum at some extreme point of D^N.

Assume, for the sake of contradiction, that there exists $s_0 \in S_0$ and $\hat{d}_0 \in D^N$ such that for any $\hat{u} \in \mathbb{R}^{mN}$, $L \begin{bmatrix} s_0 \\ \hat{u} \end{bmatrix} > M - G\hat{d}_0$. Then, there exists $j \in \{1, \ldots, r\}$ such that $l_j' \begin{bmatrix} s_0 \\ \hat{u} \end{bmatrix} > f_j(\hat{d}_0)$ where l_j' is the j^{th} row of L. But since f_j attains a minimum at some extreme point of D^N, there exists $\hat{d} \in \overline{D}^N$ such that $l_j' \begin{bmatrix} s_0 \\ \hat{u} \end{bmatrix} > f_j(\hat{d})$. This contradicts the assumption that $s_0 \in S_0$. □

Using Theorem 6.5.1, the problem of computing the set S_0 such that the Reachability Problem is feasible for any $s[0] \in S_0$ is reduced to computing a projection of the intersection of finite sets and can be automatically solved using off-the-shelf software, for example, the Multi-Parametric Toolbox (MPT) [67].

6.5.4 State Space Discretization and Correctness of the System

In general, given the previous partition of $dom(S)$ and any $i, j \in \{1, \ldots, n\}$, the reachability relation between ς_i and ς_j may not be established through the set S_0 of $s[0]$ starting from which the Reachability Problem defined in Section 6.5.1 is feasible since $T_s^{-1}(\varsigma_i)$ is not necessarily covered by S_0 (due to the constraints on u and a specific choice of the finite horizon N). To partially alleviate this limitation, we refine the partition based on the reachability relation defined earlier to increase the number of valid discrete state transitions of $\mathbb{D}$. The underlying idea is that starting with an arbitrary pair of ς_i and ς_j, we determine the set S_0 of feasible $s[0]$ for the Reachability Problem. Then, we partition $T_s^{-1}(\varsigma_i)$ into $T_s^{-1}(\varsigma_i) \cap S_0$, labeled by $\varsigma_{i,1}$, and $T_s^{-1}(\varsigma_i) \backslash S_0$, labeled by $T_s^{-1}(\varsigma_{i,2})$, and obtain the following reachability relations: $\varsigma_{i,1} \rightsquigarrow \varsigma_j$ and $\varsigma_{i,2} \not\rightsquigarrow \varsigma_j$. This process is continued until some pre-specified termination criteria are met. Table 6.1 shows the pseudo-code of the algorithm where a prescribed lower bound Vol_{min} on the volume of each cell in the new partition is used as a termination criterion. The algorithm terminates when no cell can be partitioned such that the volumes of the two resulting new cells are both greater than Vol_{min}. Larger Vol_{min} causes the algorithm to terminate sooner. Other termination criteria such as the maximum number of iterations can be used as well. Note that the point at which the algorithm terminates affects the reachability between discrete controlled states of the new partition and as a result, affects the realizability of the specification. Generally, a coarse partition makes the specification unrealizable but a fine partition causes state space explosion. A way to decide when to terminate the algorithm is to start with a coarse partition and keep refining it until the specification is realizable.

We denote the set of all the discrete controlled states corresponding to the resulting partition of $dom(S)$ after applying the discretization algorithm by $\mathcal{S}'$. Since the partition obtained from the proposed algorithm is a refined partition of $\{T_s^{-1}(\mathcal{S}_1), \ldots, T_s^{-1}(\mathcal{S}_n)\}$ and $\mathcal{V} = \mathcal{S} \times \mathcal{E}$ is proposition preserving, it is trivial to show that $\mathcal{V}' = \mathcal{S}' \times \mathcal{E}$ is also proposition preserving. For simplicity of notation, we call $\mathcal{S}'$ as $\mathcal{S}$ and $\mathcal{V}'$ as $\mathcal{V}$ for the rest of the chapter. We define the finite transition system $\mathbb{D}$ that serves as the abstract model of $\mathbb{S}$ as follows: $\mathcal{V} = \mathcal{S} \times \mathcal{E}$ is the set of states of $\mathbb{D}$ and for any two states $\nu_i = (\varsigma_i, \epsilon_i)$ and $\nu_j = (\varsigma_j, \epsilon_j)$, $\nu_i \rightarrow \nu_j$ (i.e., there exists a transition from ν_i to ν_j) only if $\varsigma_i \rightsquigarrow \varsigma_j$. Using the abstract model $\mathbb{D}$, a discrete planner that guarantees the satisfaction of φ while ensuring that the discrete plans

Table 6.1: Discretization Algorithm.

Discretization Algorithm

input: The lower bound on cell volume (Vol_{min}), the parameters A, B, E, U, D, N of the Reachability Problem, and the original partition ($\{T_s^{-1}(\varsigma_i) \mid i \in \{1,\ldots,n\}\}$)
output: The new partition sol

$sol = \{T_s^{-1}(\varsigma_i) \mid i \in \{1,\ldots,n\}\}$; $IJ = \{(i,j) \mid i,j \in \{1,\ldots,n\}\}$;
while $(size(IJ) > 0)$
  Pick arbitrary ς_i and ς_j where $(i,j) \in IJ$;
  Compute the set S_0 of $s[0]$ starting from which the Reachability Problem is feasible for the previously chosen ς_i and ς_j;
  if $(volume(sol[i] \cap S_0) > Vol_{min}$ **and** $volume(sol[i] \backslash S_0 > Vol_{min})$ **then**
    Replace $sol[i]$ with $sol[i] \cap S_0$ and append $sol[i] \backslash S_0$ to sol;
    For each $k \in \{1,\ldots,size(sol)\}$, add (i,k), (k,i), $(size(sol),k)$ and $(k,size(sol))$ to IJ;
  else
    Remove (i,j) from IJ;
  endif
endwhile

are restricted to those satisfying the reachability relations can be automatically constructed using the digital design synthesis tool [98].

From the stutter-invariant property of φ [97], the formulation of the Reachability Problem and the proposition preserving property of $\mathcal{V}$, it is straightforward to prove the following proposition.

Proposition 6.5.9. *Let $\sigma_d = \nu_0\nu_1\ldots$ be an infinite sequence of discrete states of $\mathbb{D}$ where for each natural number k, $\nu_k \to \nu_{k+1}$, $\nu_k = (\varsigma_k, \epsilon_k)$, $\varsigma_k \in \mathcal{S}$ is the discrete controlled state and $\epsilon_k \in \mathcal{E}$ is the discrete environment state. If $\sigma_d \vDash_d \varphi$, then by applying a sequence of control signals, each corresponding to a solution of the Reachability Problem with $\varsigma_i = \varsigma_k$ and $\varsigma_j = \varsigma_{k+1}$, the infinite sequence of continuous states $\sigma = v_0v_1v_2\ldots$ satisfies φ.*

Proof. From the formulation of the Reachability Problem, the (continuous) execution of the system (i.e., the infinite sequence of continuous states) can be written as

$$\sigma = v_{0,0}v_{0,1}\ldots v_{0,N-1}v_{1,0}v_{1,1}\ldots v_{1,N-1}\ldots,$$

where N is the horizon length of the Reachability Problem and for each $i \in \{0,1,\ldots,\}$ and $j \in \{0,\ldots,N-1\}$, the continuous state $v_{i,j}$ belongs to the cell labeled by ν_i. Thus, for

each atomic proposition $\pi \in \Pi$, $v_{i,j} \Vdash \pi$ if and only if $\nu_i \Vdash_d \pi$. Since $\sigma_d \vDash_d \varphi$, from the stutter-invariant property of φ, we can conclude that $\sigma \vDash \varphi$. □

As described in Section 6.2.2, a solution $u[0], \ldots, u[N-1]$ of the Reachability Problem can be computed by formulating a constrained optimal control problem, which can be solved using off-the-shelf software such as MPT [67], YALMIP [75] or NTG [92].

6.5.5 Example

We consider a point-mass omnidirectional vehicle navigating a straight road while avoiding obstacles and obeying certain traffic rules. It was shown in [56] that the non-dimensional equations of motion of the vehicle are given by

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} \dot{x} \\ \dot{y} \\ \frac{2mL^2}{J}\dot{\theta} \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_\theta \end{bmatrix}, \tag{6.10}$$

with the following constraints on the control efforts:

$$\forall t, q_x^2(t) + q_y^2(t) \leq \left(\frac{3 - |q_\theta(t)|}{2}\right)^2 \text{ and } |q_\theta(t)| \leq 3. \tag{6.11}$$

Conservatively, we can set $|q_x(t)| \leq \sqrt{0.5}$, $|q_y(t)| \leq \sqrt{0.5}$ and $|q_\theta(t)| \leq 1$ so that the constraints (6.11) are decoupled.

In this section, we are only interested in the translational (x and y) components of the vehicle state. Discretizing the dynamics (6.10) with time step 0.1, we obtain the following discrete-time linear time-invariant state space model

$$\begin{bmatrix} z[t+1] \\ v_z[t+1] \end{bmatrix} = \begin{bmatrix} 1 & 0.0952 \\ 0 & 0.9048 \end{bmatrix} \begin{bmatrix} z[t] \\ v_z[t] \end{bmatrix} + \begin{bmatrix} 0.0048 \\ 0.0952 \end{bmatrix} q_z \tag{6.12}$$

where z represents either x or y and v_z represents the rate of change in z. Let C_z be the domain of the vehicle state projected onto the (z, v_z) coordinates. We restrict the domain C_z to $[zmin, zmax] \times [-1, 1]$ and partition C_z as $C_z = \bigcup_{i \in \{zmin+1, \ldots, zmax\}} C_{z,i}$ where $C_{z,i} = [i-1, i] \times [-1, 1]$ as shown in Figure 6.1. Throughout the section, we call this partition the *original* partition of the domain C_z.

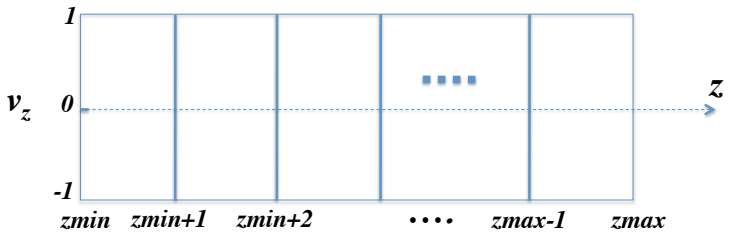


Figure 6.1: The original partition of the domain C_z.

We consider a road with two lanes, each of width 1, so we set $ymin = 0$ and $ymax = 2$. Since the vehicle dynamics are translationally invariant, without loss of generality we set $xmin = 0$ and $xmax = L$ where L is the length of the road.

For each $i \in \{1, \ldots, L\}$ and $j \in \{1, 2\}$, we define a Boolean variable $O_{i,j}$ that is assigned the value *True* if and only if an obstacle is detected at some position $(x_o, y_o) \in [i-1, i] \times [j-1, j]$. The state of the system is therefore a tuple $(x, v_x, y, v_y, O_{1,1}, O_{1,2}, \ldots, O_{L,1}, O_{L,2})$ where $(x, v_x, y, v_y) \in [0, L] \times [-1, 1] \times [0, 2] \times [-1, 1]$ is the vehicle state or the controlled state and $(O_{1,1}, O_{1,2}, \ldots, O_{L,2}) \in \{0, 1\}^{2L}$ is the environment state.

State Space Discretization

Since the dynamics and the constraints on the control efforts for the x and y components of the vehicle state are decoupled, we apply the discretization algorithm presented in Section 6.5.4 for the x and y components separately for the sake of computational efficiency.[1] Since the vehicle dynamics (6.10) are translationally invariant, we can use similar partitions for all $C_{z,i}$. The discretization algorithm with horizon length $N = 10$ and $Vol_{min} = 0.1$ yields a partition with 11 cells $\{C^1_{z,i}, C^2_{z,i}, \ldots, C^{11}_{z,i}\}$ for each $C_{z,i}$ as shown in Figure 6.2. For each $i \in \{zmin + 1, \ldots, zmax\}$ and $j \in \{1, \ldots, 11\}$, we let $\mathcal{C}^j_{z,i}$ be the state label of cell $C^j_{z,i}$ and let $\mathcal{C}_{z,i} = \{\mathcal{C}^1_{z,i}, \ldots, \mathcal{C}^{11}_{z,i}\}$. A discrete state is therefore a tuple $(\nu_x, \nu_y, O_{1,1}, \ldots, O_{L,2})$ where $(\nu_x, \nu_y) \in \mathcal{C}_{x,i} \times \mathcal{C}_{y,i}$ is the discrete controlled state. The reachability between discrete controlled states can be determined by checking set inclusion and applying Thereom 6.5.1 as described in Section 6.5.3. By solving the associated constrained optimal control problem using off-the-shelf software such as MPT, a controller associated with each reachable pair of them can be generated such that the resulting continuous execution implements the discrete

[1] Before performing the discretization, we partition each $C_{z,i}$ into $\left(C^+_{z,i} \cup C^-_{z,i}\right)$ where $C^+_{z,i} = [i-1, i] \times [0, 1]$ and $C^-_{z,i} = [i-1, i] \times [-1, 0]$ to allow the possibility of enforcing other traffic laws such as disallowing reverse motion of the vehicle.

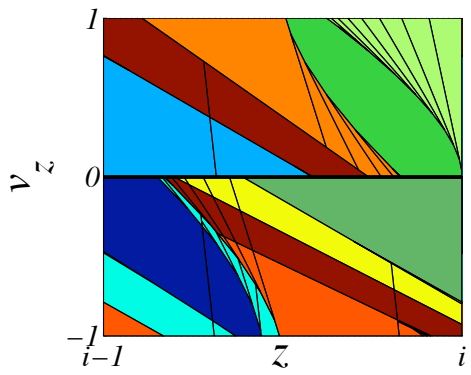


Figure 6.2: The partition of each cell $C_{z,i}$ in the original partition of the domain C_z.

transition between them.

6.6 Conclusions

This chapter proposed an approach to automatically compute a finite state abstraction of a discrete-time linear time-invariant system, taking into account the presence of exogenous disturbances. This, together with the digital design synthesis tool, allows us to automatically synthesize embedded control software for the system that is guaranteed, by construction, to satisfy its specification regardless of the environment in which it operates (subject to certain assumptions on the environment that need to be stated in the specification). The resulting system is provably robust with respect to bounded exogenous disturbances.

Chapter 7

Receding Horizon Framework for Temporal Logic Specifications

This chapter describes a receding horizon framework that satisfies a class of linear temporal logic specifications sufficient to describe a wide range of properties including safety, stability, progress, obligation, response and guarantee. The resulting embedded control software consists of a goal generator, a trajectory planner and a continuous controller. The goal generator reduces the trajectory generation problem to a sequence of smaller problems of short horizon while preserving the desired system-level temporal properties. Subsequently, in each iteration, the trajectory planner solves the corresponding short-horizon problem with the currently observed state as the initial state and generates a feasible trajectory to be implemented by the continuous controller. Based on the simulation property, we show that the composition of the goal generator, trajectory planner and continuous controller and the corresponding receding horizon framework guarantee the correctness of the system. To handle failures that may occur due to a mismatch between the actual system and its model, we propose a response mechanism and illustrate, through an example, how the system is capable of responding to certain failures and continues to exhibit a correct behavior.

7.1 Overview

Synthesis of correct-by-construction embedded control software based on temporal logic specifications has attracted considerable attention in the recent years due to the increasing frequency of systems with tight integration between computational and physical elements and the complexity in designing and verifying such systems. The hierarchical approach

presented in Chapter 6 is a commonly used to attack this synthesis problem. One of the main challenges of this approach, which is the focus of Chapter 6, is in the abstraction of systems evolving on a continuous domain into equivalent (in the simulation sense) finite state models.

Another challenge of this hierarchical approach that remains an open problem and has received less attention in the literature is the computational complexity in the synthesis of finite state automata. In particular, the synthesis problem becomes significantly harder when the interaction with the (potentially dynamic and not a priori known) environment has to be taken into account. Piterman et al. [98] treated this problem as a two-player game between the system and the environment and proposed an algorithm for the synthesis of a finite state automaton that satisfies its specification regardless of the environment in which it operates (subject to certain assumptions on the environment that need to be stated in the specification). Although for a certain class of properties, known as *Generalized Reactivity[1]*, such an automaton can be computed in polynomial time, the applications of the synthesis tool are limited to small problems due to the state explosion issue.

Similar computational complexity is also encountered in the area of constrained optimal control. In the control domain, an effective and well-established technique to address this problem is to design and implement control strategies in a receding horizon manner, i.e., optimize over a *shorter* horizon, starting from the currently observed state, implement the initial control action, move the horizon one step ahead, and reoptimize. This approach reduces the computational complexity by essentially solving a sequence of *smaller* optimization problems, each with a specific initial condition (as opposed to optimizing with *any* initial condition in traditional optimal control). Under certain conditions, receding horizon control strategies are known to lead to closed-loop stability [83, 92, 53]. See, e.g., [91, 42] for a detailed discussion on constrained optimal control, including finite horizon optimal control and receding horizon control.

This chapter describe an extension of traditional receding horizon control to incorporate linear temporal logic specification of the form (6.7) in order to reduce the computational complexity of the synthesis problem. Specifically, we develop the receding horizon framework for executing finite state automata while ensuring system correctness with respect to a given temporal logic specification. This essentially allows the synthesis problem to be reduced to a set of smaller problems of short horizon.

The remainder of the chapter is organized as follows. Section 7.2 provides a brief description of traditional receding horizon control. Section 7.3 formulates the planner synthesis problem. To reduce the complexity of the synthesis problem, in Section 7.4, we propose the receding horizon framework for executing finite state automata while ensuring system correctness with respect to a given linear temporal logic specification. The proposed framework allows the synthesis problem to be reduced to a set of smaller problems of short horizon. Its implementation, presented in Section 7.5, leads to the decomposition of the discrete planner into a goal generator and a trajectory planner. The goal generator reduces the synthesis problem to a sequence of short horizon problems while preserving the desired system-level temporal properties. Subsequently, in each iteration, the trajectory planner solves the corresponding short-horizon problem with the currently observed state as the initial state and generates a feasible trajectory to be implemented by the continuous controller. This design corresponds to Alice's planner-controller subsystem (Figure 2.1) with the goal generator having similar functionality as Mission Planner, the trajectory planner having similar functionality as the composition of Traffic Planner and Path Planner, and the continuous controller having similar functionality as Path Follower. Also presented in Section 7.5 is a response mechanism that potentially increases the robustness of the system with respect to a mismatch between the actual system and its model and violation of the environment assumptions. Finally, in Section 7.6, we demonstrate the effectiveness of the proposed technique through an example of an autonomous vehicle navigating an urban environment. This example also illustrates that the system is not only robust with respect to exogenous disturbances but is also capable of handling violation of the environment assumptions.

7.2 Preliminaries on Receding Horizon Control

Computational complexity is an inherent problem in the area of constrained optimal control. Consider, for example, the two degree of freedom controller design [7] of Figure 7.1 commonly used in motion control systems to handle non-linear designs, including global non-linearities, input saturation and state space constraints, taking into account noise and unmodeled dynamics. This control architecture separates the control component into the trajectory generation (feedforward compensator) and the local control (feedback compen-

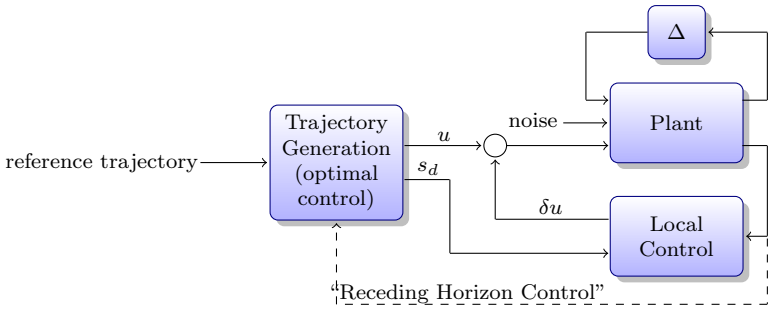


Figure 7.1: A typical control system with trajectory generation implemented in a receding horizon manner. Δ models uncertainties in the plant model.

sator) components. Given a reference trajectory, the trajectory generation component computes a feasible state space trajectory s_d and the nominal control signals u that enable the system to track s_d based on the differential equations that govern the evolution of the plant state. The local control is implemented to account for the effect of the noise and unmodeled dynamics captured by Δ.

In traditional constrained optimal control [91, 42], the trajectory generation component solves a constrained optimization problem to generate an optimal state space trajectory s_d. This component is typically run in an open-loop manner, i.e., there is no dashed arrow labeled "Receding Horizon Control" and s_d is computed offline, taking into account all the possible initial conditions. An effective and well-established approach to deal with computational complexity pertaining to this problem is to "close the loop" at the trajectory generation level as shown in Figure 7.1 and allow control strategies to be designed and implemented in a receding horizon manner. In summary, at each sampling instant, the current state of the plant is sampled and a finite horizon optimal control problem (see Section 6.2.2) is solved using the current plant state as the initial state. Only an initial part of the resulting control strategy is then implemented. The plant state is sampled again and the computation is repeated starting from the new current plant state. Hence, this strategy essentially reduces the original constrained optimal control problem into a sequence of smaller problems, each with a specific initial condition.

To ensure closed-loop stability, certain terminal state constraints need to be imposed [60, 85, 83]. A more sophisticated approach is to use an appropriate control Lyapunov function (CLF) as a terminal cost [53, 92]. The absence of terminal constraints in this CLF approach results in a significant speedup in computation time and even allows a trajectory generation

problem to be solved in real-time [86].

Receding horizon control is known to not only reduce computational complexity but also increase the robustness of the system with respect to exogenous disturbances and modeling uncertainties [92]. With disturbances and modeling uncertainties, an actual execution of the system usually deviates from a state space trajectory s_d. Receding horizon control allows the current state of the system to be continually reevaluated so s_d can be adjusted accordingly based on the externally received reference if the actual execution of the system does not match it. Receding horizon control has been successfully demonstrated in many applications [96, 29, 92, 61].

7.3 Problem Formulation

We consider the Planner-Controller Synthesis Problem formulated in Section 6.3. The hierarchical approach described in Section 6.4 is commonly used to attack this problem. In this chapter, we focus on the computational complexity issue of the discrete planner synthesis problem (step (b) as described in Section 6.4). We assume that a finite state abstraction $\mathbb{D}$ of the physical model $\mathbb{S}$ of the system has been constructed using, for example, the discretization algorithm presented in Section 6.5.4. We denote the (finite) set of states of $\mathbb{D}$ by $\mathcal{V}$.

We consider a specification of the form (6.7) since, from Proposition 6.4.1, the specification (6.3) can be reduced to this form. The following aspects of specification (6.7) are noteworthy:

(i) The desired properties include the safety properties, $\bigwedge_{i \in I_s} \Box \psi_{s,i}$, and the progress properties, $\bigwedge_{i \in I_g} \Box \Diamond \psi_{g,i}$.

(ii) Each progress property, $\Box \Diamond \psi_{g,i}, i \in I_g$, specifies the set of states that the system needs to visit infinitely often. In other words, it represents a system goal. The conjunction of these progress properties allows multiple goals to be specified. Each of these goals has to be achieved infinitely often. However, the order in which they are achieved is irrelevant.

Discrete Planner Synthesis Problem: Given a finite state abstraction $\mathbb{D}$ of the physical system and the system specification φ of the form (6.7), synthesize a discrete planner that

computes a discrete plan to ensure that starting from any initial condition, φ is satisfied for any sequence of environment states.

7.4 Receding Horizon Framework

The digital design synthesis tool presented in Section 6.2.1 was designed to solve the Discrete Planner Synthesis Problem. However, in many cases, this tool fails because of the state explosion problem. In the worst case, the resulting automaton may contain all the possible states of the system. For example, if the system has N variables, each can take any value in $\{1, \ldots, M\}$, then there may be as many as M^N nodes in the automaton. This type of computational complexity limits the application of the systhesis to relatively small problems.

To reduce computational complexity in the synthesis of finite state automata, we apply an idea similar to the traditional receding horizon control described in Section 7.2. First, we observe that in many applications, it is not necessary to plan for the whole execution, taking into account all the possible behaviors of the environment since a state that is very far from the current state of the system typically does not affect the near future plan. Consider, for example, the robot motion planning problem described in Example 6.3.2. Suppose C_1 or C_2 is very far away from the initial position of the robot. Under certain conditions, it may be sufficient to only plan out an execution for only a short segment ahead and implement it in a receding horizon fashion, i.e., recompute the plan as the robot moves, starting from the currently observed state (rather than from all initial conditions satisfying φ_{init} as the original specification (6.3) suggests). In this section, we present a sufficient condition and a receding horizon strategy that allows the synthesis to be performed on a smaller domain; thus, substantially reduces the number of states (or nodes) of the automaton while still ensuring the system correctness with respect to the LTL specification (6.3).

We use the notion of partial order to provide a measure of closeness to the goal states.

Definition 7.4.1. A *partially ordered set* $(V, \preceq)$ consists of a set V and a binary relation $\preceq$ over the set V satisfying the following properties: for any $v_1, v_2, v_3 \in V$, (a) $v_1 \preceq v_1$; (b) if $v_1 \preceq v_2$ and $v_2 \preceq v_1$, then $v_1 = v_2$; and (c) if $v_1 \preceq v_2$ and $v_2 \preceq v_3$, then $v_1 \preceq v_3$.

First, for each progress property $\square\diamond\psi_{g,i}$, $i \in I_g$, we construct a collection of subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ of $\mathcal{V}$ such that

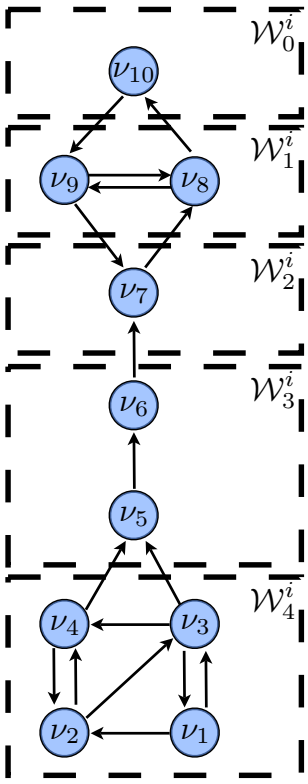


Figure 7.2: Illustration of the receding horizon framework showing the relationships between the states of $\mathcal{V}$ and between the subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$.

(a) $\mathcal{W}_0^i \cup \mathcal{W}_1^i \cup \ldots \cup \mathcal{W}_p^i = \mathcal{V}$,

(b) $\psi_{g,i}$ is satisfied for any $\nu \in \mathcal{W}_0^i$, i.e., $\mathcal{W}_0^i$ is the set of goal states associated with the progress property $\square\diamond\psi_{g,i}$, and

(c) $\mathcal{P}^i \triangleq (\{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}, \preceq_{\psi_{g,i}})$ is a partially ordered set defined such that $\mathcal{W}_0^i \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$.

Define a function $\mathcal{F}^i : \{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\} \to \{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}$ such that $\mathcal{F}^i(\mathcal{W}_j^i) \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$. As we shall see shortly, the function $\mathcal{F}^i(\mathcal{W}_j^i)$ defines an intermediate goal for starting from a state in $\mathcal{W}_j^i$.

Consider a simple case where $\{\nu_1, \ldots, \nu_{10}\}$ is the set $\mathcal{V}$ of states, ν_{10} satisfies $\psi_{g,i}$, and the states in $\mathcal{V}$ are organized into five subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_4^i$. The relationships between the states in $\mathcal{V}$ and the subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_4^i$ are illustrated in Figure 7.2. The partial order may be defined as $\mathcal{W}_0^i \prec \mathcal{W}_1^i \prec \ldots \prec \mathcal{W}_4^i$ and the mapping $\mathcal{F}^i$ may be defined as $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_{j-2}^i, \forall j \geq 2$, $\mathcal{F}^i(\mathcal{W}_1^i) = \mathcal{W}_0^i$ and $\mathcal{F}^i(\mathcal{W}_0^i) = \mathcal{W}_0^i$. Suppose ν_1 is the initial state of the system. The key

idea of the receding horizon framework, as described later, is to plan from the initial state $\nu_1 \in \mathcal{W}_4^i$ to any state in $\mathcal{F}^i(\mathcal{W}_4^i) = \mathcal{W}_2^i$, rather than planning from the initial state ν_1 to the goal state ν_{10} in one shot, taking into account all the possible behaviors of the environment. Once a state in $\mathcal{W}_3^i$, i.e., ν_5 or ν_6 is reached, we then replan from that state to a state in $\mathcal{F}^i(\mathcal{W}_3^i) = \mathcal{W}_1^i$. We repeat this process until ν_{10} is reached. Under certain sufficient conditions presented later, this strategy ensures the correctness of the overall execution of the system.

Formally, with the above definitions of $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ and $\mathcal{F}^i$, we define a short-horizon specification Ψ_j^i associated with $\mathcal{W}_j^i$ for each $i \in I_g$ and $j \in \{0, \ldots, p\}$ as

$$\begin{aligned} \Psi_j^i \;\triangleq\; & \big((\nu \in \mathcal{W}_j^i) \wedge \Phi \wedge \Box\psi_e^e \wedge \textstyle\bigwedge_{k\in I_f} \Box\Diamond\psi_{f,k}^e\big) \\ & \implies \big(\textstyle\bigwedge_{k\in I_s} \Box\psi_{s,k} \wedge \Box\Diamond\big(\nu \in \mathcal{F}^i(\mathcal{W}_j^i)\big) \wedge \Box\Phi\big). \end{aligned} \tag{7.1}$$

Here, ν is the state of the system and ψ_e^e, $\psi_{f,k}^e$ and $\psi_{s,k}$ are defined as in (6.7). Φ is a propositional formula of variables from V such that $\psi_{init} \implies \Phi$ is a tautology, i.e., any state $\nu \in \mathcal{V}$ that satisfies ψ_{init} also satisfies Φ. The role of Φ is to add assumptions on the initial states that need to be considered when synthesizing an automaton satisfying Ψ_j^i. These assumptions may need to be added in order to make Ψ_j^i realizable. For example, Φ may be used to exclude an unsafe state from the set of initial states. A more detailed discussion on the role of Φ can be found in Remark 7.4.2.

An automaton $\mathcal{A}_j^i$ satisfying Ψ_j^i defines a strategy for going from a state $\nu_1 \in \mathcal{W}_j^i$ to a state $\nu_2 \in \mathcal{F}^i(\mathcal{W}_j^i)$ while satisfying the safety requirements $\bigwedge_{i\in I_s} \Box\psi_{s,i}$ and maintaining the invariant Φ (see Remark 7.4.2 for the role of Φ in this framework). Roughly speaking, the partial order $\mathcal{P}^i$ provides a measure of "closeness" to the states satisfying $\psi_{g,i}$. Since each specification Ψ_j^i asserts that the system eventually reaches a state that is smaller in the partial order, it ensures that each automaton $\mathcal{A}_j^i$ brings the system "closer" to the states satisfying $\psi_{g,i}$. The function $\mathcal{F}^i$ thus defines the horizon length for these short-horizon problems. In general, the size of $\mathcal{A}_j^i$ increases with the horizon length. However, with too short a horizon, the specification Ψ_j^i is typically not realizable. A good practice is to choose the shortest horizon, subject to the realizability of Ψ_j^i, so that the automaton $\mathcal{A}_j^i$ contains as small a number of states as possible.

Receding Horizon Strategy: For each $i \in I_g$ and $j \in \{0, \ldots, p\}$, construct an automa-

ton $\mathcal{A}_j^i$ satisfying Ψ_j^i, defined in (7.1). Let $I_g = \{i_1, \ldots, i_n\}$ and define a corresponding ordered set $(i_1, \ldots, i_n)$. Note that this order only affects the sequence of progress properties $\psi_{g,i_1}, \ldots, \psi_{g,i_n}$ that the system tries to satisfy. Hence, it can be picked arbitrarily without affecting the correctness of the receding horizon strategy.

(1) Determine the index j_1 such that the current state $\nu_0 \in \mathcal{W}_{j_1}^{i_1}$. If $j_1 \neq 0$, then execute the automaton $\mathcal{A}_{j_1}^{i_1}$ until the system reaches a state $\nu_1 \in \mathcal{W}_k^{i_1}$ where $\mathcal{W}_k^{i_1} \prec_{\psi_{g,i_1}} \mathcal{W}_{j_1}^{i_1}$. Note that since the union of $\mathcal{W}_1^{i_1}, \ldots, \mathcal{W}_p^{i_1}$ is the set $\mathcal{V}$ of all the states, given any $\nu_0, \nu_1 \in \mathcal{V}$, there exist $j_1, k \in \{0, \ldots, p\}$ such that $\nu_0 \in \mathcal{W}_{j_1}^{i_1}$ and $\nu_1 \in \mathcal{W}_k^{i_1}$. This step corresponds to going from $\mathcal{W}_{j_1}^{i_1}$ to $\mathcal{W}_{j_1-1}^{i_1}$ in Figure 7.3.

(2) If the current state $\nu_1 \notin \mathcal{W}_0^{i_1}$, switch to the automaton $\mathcal{A}_k^{i_1}$ where the index k is chosen such that the current state $\nu_1 \in \mathcal{W}_k^{i_1}$. Execute $\mathcal{A}_k^{i_1}$ until the system reaches a state that is smaller in the partial order $\mathcal{P}^{i_1}$. Repeat this step until a state $\nu_2 \in \mathcal{W}_0^{i_1}$ is reached. Note that it is guaranteed that a state $\nu_2 \in \mathcal{W}_0^{i_1}$ is eventually reached because of the finiteness of the set $\{\mathcal{W}_0^{i_1}, \ldots, \mathcal{W}_p^{i_1}\}$ and its partial order. See the proof of Theorem 7.4.1 for more details. This step corresponds to going all the way down the leftmost column in Figure 7.3.

(3) Switch to the automaton $\mathcal{A}_{j_2}^{i_2}$ where the index j_2 is chosen such that the current state $\nu_2 \in \mathcal{W}_{j_2}^{i_2}$. Repeat step (2) with i_1 replaced by i_2 for the partial order $\mathcal{P}^{i_2}$ until a state $\nu_3 \in \mathcal{W}_0^{i_2}$ is reached. Repeat this step with i_2 replaced by $i_3, i_4, \ldots, i_n$, respectively, until a state $\nu_n \in \mathcal{W}_0^{i_n}$ is reached. In Figure 7.3, this step corresponds to moving to the next column, going all the way down this column and repeating this process until we reach the bottom of the rightmost column.

(4) Repeat steps (1)–(3).

Theorem 7.4.1. *Suppose Ψ_j^i is realizable for each $i \in I_g$, $j \in \{0, \ldots, p\}$. Then the proposed receding horizon strategy ensures that the system is correct with respect to the specification (6.7), i.e., any execution of the system satisfies (6.7).*

Proof. Consider an arbitrary execution σ of the system that satisfies the "assumption" part of (6.7). We want to show that the safety properties $\psi_{s,i}, i \in I_s$, hold throughout the execution and for each $i \in I_g$, a state satisfying $\psi_{g,i}$ is reached infinitely often.

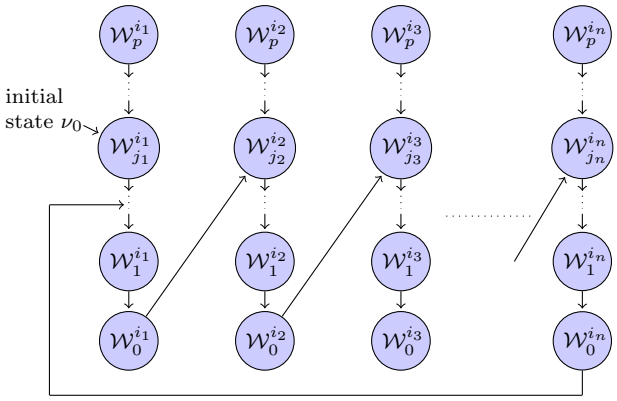


Figure 7.3: A graphical description of the receding horizon strategy for a special case where for each $i \in I_g$, $\mathcal{W}_j^i \prec_{\psi_{g,i}} \mathcal{W}_k^i, \forall j < k$, $F^i(\mathcal{W}_j^i) = \mathcal{W}_{j-1}^i, \forall j > 0$ and $F^i(\mathcal{W}_0^i) = \mathcal{W}_0^i$. Starting from a state ν_0, the system executes the automaton $\mathcal{A}_{j_1}^{i_1}$ where the index j_1 is chosen such that $\nu_0 \in \mathcal{A}_{j_1}^{i_1}$. Repetition of step (2) ensures that a state $\nu_2 \in \mathcal{W}_0^{i_1}$ (i.e., a state satisfying ψ_{g,i_1}) is eventually reached. This state ν_2 belongs to some set, say, $\mathcal{W}_{j_2}^{i_2}$ in the partial order $\mathcal{P}^{i_2}$. The system then works through this partial order until a state $\nu_3 \in \mathcal{W}_0^{i_2}$ (i.e., a state satisfying ψ_{g,i_2}) is reached. This process is repeated until a state ν_n satisfying ψ_{g,i_n} is reached. At this point, for each $i \in I_g$, a state satisfying $\psi_{g,i}$ has been visited at least once in the execution. In addition, the state ν_n belongs to some set in the partial order $\mathcal{P}^{i_1}$ and the whole process is repeated, ensuring that for each $i \in I_g$, a state satisfying $\psi_{g,i}$ is visited infinitely often in the execution.

Let $\nu_0 \in \mathcal{V}$ be the initial state of the system and let the index j_1 be such that $\nu_0 \in \mathcal{W}_{j_1}^{i_1}$. From the tautology of $\psi_{init} \implies \Phi$, it is easy to show that σ satisfies the "assumption" part of $\Psi_{j_1}^{i_1}$ as defined in (7.1). Thus, if $j_1 = 0$, then $\mathcal{A}_0^{i_1}$ ensures that a state ν_2 satisfying ψ_{g,i_1} is eventually reached and the safety properties $\psi_{s,i}, i \in I_s$ hold at every position of σ up to and including the point where ν_2 is reached. Otherwise, $j_1 \neq 0$ and $\mathcal{A}_{j_1}^{i_1}$ ensures that eventually, a state $\nu_1 \in \mathcal{W}_k^{i_1}$ where $\mathcal{W}_k^{i_1} \prec_{\psi_g} \mathcal{W}_{j_1}^{i_1}$ is reached, i.e., ν_1 is the state of the system at some position l_1 of σ. In addition, the invariant Φ and all the safety properties $\psi_{s,i}, i \in I_s$, are guaranteed to hold at all the positions of σ up to and including the position l_1. According to the receding horizon strategy, the planner switches to the automaton $\mathcal{A}_k^{i_1}$ at position l_1 of σ. Since $\nu_1 \in \mathcal{W}_k^{i_1}$ and ν_1 satisfies Φ, the "assumption" part of $\Psi_k^{i_1}$ as defined in (7.1) is satisfied. Using the previous argument, we get that $\Psi_k^{i_1}$ ensures that all the safety properties $\psi_{s,i}, i \in I_s$, hold at every position of σ starting from position l_1 up to and including position l_2 at which the planner switches the automaton (from $\mathcal{A}_k^{i_1}$) and Φ holds at position l_2. By repeating this procedure and using the finiteness of the set $\{\mathcal{W}_0^{i_1}, \ldots, \mathcal{W}_p^{i_1}\}$ and its partial order condition, eventually the automaton $\mathcal{A}_0^{i_1}$ is executed which ensures that σ contains a

state ν_2 satisfying ψ_{g,i_1} and step (2) terminates.

Applying the previous argument to step (3), we get that step (3) terminates and before this step terminates, the safety properties $\psi_{s,i}, i \in I_s$, and the invariant Φ hold throughout the execution and for each $i \in I_g$, a state satisfying $\psi_{g,i}$ has been reached at least once. By continually repeating steps (1)–(3), the receding horizon strategy ensures that $\psi_{s,i}, i \in I_s$, hold throughout the execution and for each $i \in I_g$, a state satisfying $\psi_{g,i}$ is reached infinitely often. □

Remark 7.4.1. As discussed in Section 7.2, by continually evaluating the current plant state and adjusting the state space trajectory s_d based on the actual execution of the system, traditional receding horizon control is known to increase the robustness of the system with respect to exogenous disturbances and modeling uncertainties. Such an effect may be expected in our extension of the traditional receding horizon control. Verifying this property is subject to current study.

Remark 7.4.2. The propositional formula Φ can be viewed as an invariant of the system. It adds an assumption on the initial state of each automaton $\mathcal{A}_j^i$ and is introduced in order to make Ψ_j^i realizable. Without Φ, the set of initial states of $\mathcal{A}_j^i$ includes all states $\nu \in \mathcal{W}_j^i$. However, starting from some "bad" state (e.g., unsafe state) in $\mathcal{W}_j^i$, there may not exist a strategy for the system to satisfy Ψ_j^i. Φ is essentially used to eliminate the possibility of starting from these "bad" states. Given a partially ordered set $\mathcal{P}^i$ and a function $\mathcal{F}^i$, one way to determine Φ is to start with $\Phi \equiv \mathit{True}$ and check the realizability of the resulting Ψ_j^i. If there exist $i \in I_g$ and $j \in \{0, \ldots, p\}$ such that Ψ_j^i is not realizable, the synthesis process provides the initial state ν^* of the system starting from which there exists a set of moves of the environment such that the system cannot satisfy Ψ_j^i. This information provides guidelines for constructing Φ, i.e., we can add a propositional formula to Φ that prevents the system from reaching the state ν^*. This procedure can be repeated until Ψ_j^i is realizable for any $i \in I_g$ and $j \in \{0, \ldots, p\}$ or until Φ excludes all the possible states, in which case either the original specification is unrealizable or the proposed receding horizon strategy cannot be applied with the given partially ordered set $\mathcal{P}^i$ and function $\mathcal{F}^i$.

Remark 7.4.3. For each $i \in I_g$ and $j \in \{0, \ldots, p\}$, checking the realizability of Ψ_j^i requires considering all the initial conditions in $\mathcal{W}_j^i$ satisfying Φ. However, as will be further discussed in Section 7.5, when a strategy (i.e., a finite state automaton satisfying Ψ_j^i) is to be

extracted, only the currently observed state needs to be considered as the initial condition. Typically, the realizability can be checked symbolically and enumeration of states is only required when a strategy needs to be extracted [98]. Symbolic methods are known to handle large number of states, in practice, significantly better than enumeration-based methods. Hence, state explosion usually occurs in the synthesis (i.e., strategy extraction) stage rather than the realizability checking stage. By considering only the currently observed state as the initial state in the synthesis stage, the receding horizon strategy delays state explosion both by considering a short-horizon problem and a specific initial state.

Remark 7.4.4. The proposed receding horizon approach is not complete. Even if there exists a control strategy that satisfies the original specification in (6.7), there may not exist an invariant Φ or a collection of subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ that allow the receding horizon strategy to be applied since the corresponding Ψ_j^i may not be realizable for all $i \in I_g$ and $j \in \{0, \ldots, p\}$.

Example

We consider the point-mass omnidirectional vehicle as in Section 6.5.5. The embedded control software has to be designed such that the vehicle is capable of navigating a straight road while avoiding obstacles and obeying certain traffic rules. In this section, we precisely describe these desired properties using linear temporal logic and illustrate the application of the receding horizon framework.

System Specification

We assume that at the initial configuration, the vehicle is at least d_{obs} away from any obstacle and that the vehicle starts in the right lane. That is, ψ_{init} in (6.7) is defined as: for any $i \in \{1, \ldots, L\}$,

$$\left(x \in \bigcup_{k=i-d_{obs}}^{i+d_{obs}} C_{x,k} \implies (\neg O_{i,1} \ \wedge \ \neg O_{i,2}) \right) \wedge \ y \in C_{y,1}. \tag{7.2}$$

The following properties are assumed for the environment.

1. An obstacle is detected before the vehicle gets too close to it. That is, there is a lower bound $d_{popup} \geq 0$ on the distance from the vehicle for which obstacle is allowed to

instantly pop up. An LTL formula corresponding to this assumption is a conjunction of the following formula: for all $i \in \{1, \ldots, L\}$ and $k \in \{1, 2\}$,

$$\Box\left(\left(x \in \bigcup_{j=i-d_{popup}}^{i+d_{popup}} C_{x,j} \ \wedge \ \neg O_{i,k}\right) \implies \Box(\neg O_{i,k})\right). \tag{7.3}$$

2. Sensing range is limited. That is, the vehicle cannot detect an obstacle that is farther away from it than $d_{sr} > d_{popup} \geq 0$. An LTL formula corresponding to this assumption is a conjunction of the following formula: for all $i \in \{1, \ldots, L\}$,

$$\Box\left(x \in C_{x,i} \implies \bigwedge_{j>i+d_{sr}} (\neg O_{j,1} \ \wedge \ \neg O_{j,2})\right). \tag{7.4}$$

3. The road is not blocked. That is, for any $i \in \{1, \ldots, L\}$,

$$\Box\left(\neg O_{i,1} \ \vee \ \neg O_{i,2}\right). \tag{7.5}$$

4. To make sure that the stay-in-lane requirement (see below) is achievable, we assume that an obstacle in the right lane does not disappear while the vehicle is in its vicinity. That is, for any $i \in \{1, \ldots, L\}$,

$$\Box\left(\left(x \in \bigcup_{j=i-1}^{i+1} C_{x,j} \ \wedge \ O_{i,1}\right) \implies \Box(O_{i,1})\right). \tag{7.6}$$

These assumptions can be relaxed so that they have the form of the "assumption" part of (6.6) by replacing the inner $\Box$ in (7.3) and (7.6) with $\bigcirc$. Note that the resulting formula is still not in the form of $\Box\psi_e^e$ in (6.7). However, it can be shown that the resulting system specification φ is stutter-invariant; thus, Proposition 6.5.9 can be applied to ensure the correctness of the hierarchical approach. In addition, it can be shown that the proof of Theorem 7.4.1 is valid for this specification; hence, the receding horizon framework can be utilized to reduce the computational complexity of the synthesis problem while ensuring the correctness of the system.

Next, we define the desired safety property, $\bigwedge_{i \in I_s} \Box\psi_{s,i}$, as the conjunction of the following properties:

1. No collision, i.e., for any $i \in \{1, \ldots, L\}$ and $j \in \{1, 2\}$,

$$\Box\,(O_{i,j} \implies \neg(x \in C_{x,i} \ \wedge \ y \in C_{y,j})). \tag{7.7}$$

2. The vehicle stays in the right lane unless there is an obstacle blocking the lane. That is, for any $i \in \{1, \ldots, L\}$,

$$\Box\,((\neg O_{i,1} \ \wedge \ x \in C_{x,i}) \implies (y \in C_{y,1})). \tag{7.8}$$

Finally, we define the desired progress property $\Box\Diamond(x \in C_{x,L})$, i.e., we want to ensure that the vehicle reaches the end of the road.

Receding Horizon Formulation

Based on the new partition of the vehicle state space, there are the total of $242 \times L$ discrete vehicle states and $2^{2\times L}$ discrete environment states. Thus, in the worst case, the resulting automaton may have as many as $242 \times L \times 2^{2\times L}$ nodes. To avoid state explosion, we apply the receding horizon strategy presented earlier. The partially ordered set is defined by $\mathcal{P} = (\{\mathcal{W}_0, \ldots, \mathcal{W}_{L-1}\}, \preceq_{\psi_g})$ where for each $j \in \{0, \ldots, L-1\}$, $\mathcal{W}_j = \{(\nu_x, \nu_y, O_{1,1}, \ldots, O_{L,2}) \mid \nu_x \in \mathcal{C}_{x,L-j}\}$ and $\mathcal{W}_j \prec_{\psi_g} \mathcal{W}_k$ for any $j < k$.

Next, we follow the scheme in Remark 7.4.2 to find an invariant Φ. Starting with $\Phi = \mathit{True}$, we iteratively add, until Ψ_j as defined in (7.1) is realizable, a propositional formula to exclude the initial states starting from which there exists a set of moves of the environment such that the system cannot satisfy Ψ_j. A close examination of the resulting Φ reveals that Φ is essentially the conjunction of the following logics:

1. To ensure the progress property $\Box\Diamond(x \in C_{x,L})$, we need to assume that $\nu_x \notin \mathcal{X}_{notrans}$ and $\nu_y \notin \mathcal{Y}_{notrans}$ where $\mathcal{Z}_{notrans}$ is defined as: for any $\nu_z \in \mathcal{Z}_{notrans}$, $i \in \{zmin + 1, \ldots, zmax\}$ and $j \in \{1, \ldots, 11\}$, $\nu_z \nrightarrow \mathcal{C}^j_{z,i}$ and $\mathcal{Z}$ represent either $\mathcal{X}$ or $\mathcal{Y}$.

2. To ensure no collision, the vehicle cannot collide with an obstacle at the initial state.

3. Suppose $\nu_x \in \mathcal{C}_{x,i}$. To ensure no collision, if ν_y can only transition to $\nu'_y \in \mathcal{C}_{y,1}$, then either $O_{i,1}$ or $O_{i+1,1}$ is *False*. Similarly, if ν_y can only transition to $\nu'_y \in \mathcal{C}_{y,2}$, then either $O_{i,2}$ or $O_{i+1,2}$ is *False*. Similar reasoning can be derived for the case where

$\nu_x \in \mathcal{C}_{x,i}$ such that it can only transition to $\nu'_x \in \mathcal{C}_{x,i+1}$ and for the case where it can only transition to $\nu'_x \in \mathcal{C}_{x,i}$.

4. To ensure the stay-in-lane property, the vehicle cannot be in the left lane unless there is an obstacle blocking the right lane at the initial state. In addition, the vehicle is never in the state $(\nu_x, \nu_y) \in \mathcal{C}_{x,i} \times \mathcal{C}_{y,1}$ that can only transition to $(\nu'_x, \nu'_y) \in \mathcal{C}_{x,i} \times \mathcal{C}_{y,2}$.

5. Suppose $\nu_x \in \mathcal{C}_{x,i}$ and $O_{i+1,1}$ is *False*. To ensure that the vehicle does not go to the left lane when the right lane is not blocked, it is not the case that $\nu_y \in \mathcal{C}_{y,1}$ that can only transition to $\nu'_y \in C_{y,2}$. In addition, it is not the case that ν_x can only transition to $\nu'_x \in C_{x,i+1}$ and $\nu_y \in \mathcal{C}_{y,2}$ that can only transition to $\nu'_y \in \mathcal{C}_{y,2}$.

With $d_{popup} = 1$ and the horizon length 2 (i.e., $\mathcal{F}(\mathcal{W}_j) = \mathcal{W}_{j-2}, \forall j \geq 2$), the specification (7.1) is realizable. In addition, if we let d_{obs} be greater than 1 and restrict the initial state of the system such that $\nu_x \notin \mathcal{X}_{notrans}$ and $\nu_y \notin \mathcal{Y}_{notrans}$, we get that $\psi_{init} \implies \Phi$ is a tautology.

Results

The synthesis was performed on a MacBook with a 2 GHz Intel Core 2 Duo processor. Using JTLV [98], the computation time was approximately 20 seconds. The resulting automaton contains 2845 nodes.

A simulation result with the road length of 30 is shown in Figure 7.4. The polygons drawn in red are obstacles, which are not known a priori. Notice that when there is no obstacle blocking the lane, the vehicle tries to stay as close to the lane boundary ($y = 1$) as possible. This is expected since to be able to avoid a pop up obstacle, due to the constraint on the admissible control inputs, the vehicle needs to stay close to the lane boundary to be

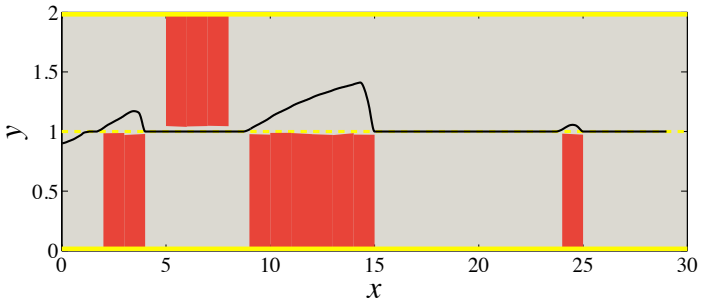


Figure 7.4: Simulation result. The solid line is the trajectory of the vehicle. The polygons are obstacles discovered during the execution when the vehicle gets close enough to them.

able to change lane. To force the vehicle to stay close to the center of the lane, we need a finer partition of the road and extra LTL formula to ensure this property needs to be added to the system specification.

7.5 Implementation of the Receding Horizon Framework

In order to implement the receding horizon strategy described in Section 7.4, a partial order $\mathcal{P}^i$ and the corresponding function $\mathcal{F}^i$ need to be defined for each $i \in I_g$. In the previous example, we show how these elements can be manually defined for a simple case. For a more complex case, the problem of determining $\mathcal{P}^i$ and $\mathcal{F}^i$ may not be as straightforward. In this section, we present an implementation of the receding horizon strategy, allowing $\mathcal{P}^i$ and $\mathcal{F}^i$ to be automatically determined for each $i \in I_g$ while ensuring that all the short-horizon specifications $\Psi^i_j, i \in I_g, j \in \{0, \ldots, p\}$, as defined in (7.1) are realizable.

Given an invariant Φ and subsets $\mathcal{W}^i_0, \ldots, \mathcal{W}^i_p$ of $\mathcal{V}$ for each $i \in I_g$, we first construct a finite transition system $\mathbb{T}^i$ with the set of states $\{\mathcal{W}^i_0, \ldots, \mathcal{W}^i_p\}$. For each $j, k \in \{0, \ldots, p\}$, there is a transition $\mathcal{W}^i_j \to \mathcal{W}^i_k$ in $\mathbb{T}^i$ only if $j \neq k$ and the specification in (7.1) is realizable with $\mathcal{F}^i(\mathcal{W}^i_j) = \mathcal{W}^i_k$. The finite transition system $\mathbb{T}^i$ can be regarded as an abstraction of the finite state model $\mathbb{D}$ of the physical system $\mathbb{S}$, i.e., a higher-level abstraction of $\mathbb{S}$.

Suppose Φ is defined such that there exists a path in $\mathbb{T}^i$ from $\mathcal{W}^i_j$ to $\mathcal{W}^i_0$ for all $i \in I_g$, $j \in \{1, \ldots, p\}$. (Verifying this property is basically a graph search problem. If a path does not exist, Φ can be recomputed using a procedure described in Remark 7.4.2.) We propose a planner-controller subsystem with three components (see Figure 7.5): goal generator, trajectory planner, and continuous controller.

Goal generator: Pick an order[1] $(i_1, \ldots, i_n)$ for the elements of the unordered set $I_g = \{i_1, \ldots, i_n\}$ and maintain an index $k \in \{1, \ldots, n\}$ throughout the execution. Starting with $k = 1$, in each iteration, the goal generator performs the following tasks.

(a1) Receive the currently observed state of the plant (i.e., the controlled state) and environment.

[1] As discussed in the description of the receding horizon strategy in Section 7.4, this order can be picked arbitrarily. In general, its definition affects a strategy the system chooses to satisfy the specification (6.7) as it corresponds to the sequence of progress properties $\psi_{g,i_1}, \ldots, \psi_{g,i_n}$ the system tries to satisfy.

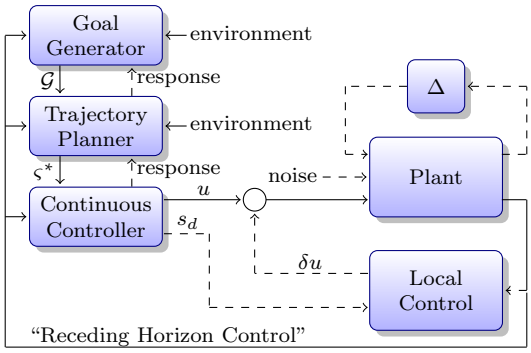


Figure 7.5: A system with the planner-controller subsystem implemented in a receding horizon manner. In addition to the components discussed in this section, Δ, which captures uncertainties in the plant model, may be added to make the model more realistic. In addition, the local control may be implemented to account for the effect of noise, disturbances and unmodeled dynamics. The inputs and outputs of these two components are drawn dashed since they are not considered in this thesis.

(a2) If the abstract state corresponding to the currently observed state belongs to $\mathcal{W}_0^{i_k}$, update k to $(k \mod n) + 1$.

(a3) If k was updated in step (a2) or this is the first iteration, then based on the higher-level abstraction $\mathbb{T}^{i_k}$ of the physical system $\mathbb{S}$, compute a path from $\mathcal{W}_j^{i_k}$ to $\mathcal{W}_0^{i_k}$ where the index $j \in \{0, \ldots, p\}$ is chosen such that the abstract state corresponding to the currently observed state belongs to $\mathcal{W}_j^{i_k}$.

(a4) If a new path is computed in step (a3), then issue this path (i.e., a sequence $\mathcal{G} = \mathcal{W}_{l_0}^{i_k}, \ldots, \mathcal{W}_{l_m}^{i_k}$ for some $m \in \{0, \ldots, p\}$ where $l_0, \ldots l_m \in \{0, \ldots, p\}$, $l_0 = j$, $l_m = 0$, $l_\alpha \neq l_{\alpha'}$ for any $\alpha \neq \alpha'$, and there exists a transition $\mathcal{W}_{l_\alpha}^{i_k} \to \mathcal{W}_{l_{\alpha+1}}^{i_k}$ in $\mathbb{T}^{i_k}$ for any $\alpha < m$) to the trajectory planner.

Note that the problem of finding a path in $\mathbb{T}^{i_k}$ from $\mathcal{W}_j^{i_k}$ to $\mathcal{W}_0^{i_k}$ can be efficiently solved using any graph search or shortest-path algorithm [110], such as Dijkstra's, A*, etc. To reduce the original synthesis problem to a set of problems with short horizon, the cost on each edge $(\mathcal{W}_{l_\alpha}^{i_k}, \mathcal{W}_{l_{\alpha'}}^{i_k})$ of the graph built from $\mathbb{T}^{i_k}$ may be defined, for example, as an exponential function of the "distance" between the sets $\mathcal{W}_{l_\alpha}^{i_k}$ and $\mathcal{W}_{l_{\alpha'}}^{i_k}$ so that a path with smaller cost contains segments of shorter "distance."

Trajectory planner: The trajectory planner maintains the latest sequence $\mathcal{G} = \mathcal{W}_{l_0}^{i_k}, \ldots, \mathcal{W}_{l_m}^{i_k}$ of goal states received from the goal generator, an index $q \in \{1, \ldots, m\}$ of the current goal

state in $\mathcal{G}$, a strategy $\mathbb{F}$ represented by a finite state automaton, and the next abstract state ν^* throughout the execution. Starting with $q = 1$, $\mathbb{F}$ being an empty finite state automaton and ν^* being a null state, in each iteration, the trajectory planner performs the following tasks.

(b1) Receive the currently observed state of the plant and environment.

(b2) If a new sequence of goal states is received from the goal generator, update $\mathcal{G}$ to this latest sequence of goal states, update q to 1, and update ν^* to null. Otherwise, if the abstract state corresponding to the currently observed state belongs to $\mathcal{W}_{l_q}^{i_k}$, update q to $q+1$ and ν^* to null.

(b3) If ν^* is null, then based on the abstraction $\mathbb{D}$ of the physical system $\mathbb{S}$, synthesize a strategy that satisfies the specification in (7.1) with $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_{l_q}^{i_k}$, starting from the abstract state ν_0 corresponding to the currently observed state, i.e., replace the assumption $\nu \in \mathcal{W}_j^i$ with $\nu = \nu_0$. Assign this strategy to $\mathbb{F}$ and update ν^* to the state following the initial state in $\mathbb{F}$ based on the current environment state.

(b4) If the controlled state ς^* component of ν^* corresponds to the currently observed state of the plant, update ν^* to the state following the current ν^* in $\mathbb{F}$ based on the current environment state.

(b5) If ν^* was updated in step (b3) or (b4), then issue the controlled state ς^* to the continuous controller.

Continuous controller: The continuous controller maintains the most recent (abstract) final controlled state ς^* from the trajectory planner. In each iteration, it receives the currently observed state s of the plant. Then, it computes a control signal u such that the continuous execution of the system eventually reaches the cell of $\mathbb{D}$ corresponding to the abstract controlled state ς^* while always staying in the cell corresponding to the abstract controlled state ς^* and the cell containing s. Essentially, the continuous execution has to *simulate* the abstract plan computed by the trajectory planner. As discussed at the end of Section 6.5.4, such a control signal can be computed by formulating a constrained optimal control problem, which can be solved using off-the-shelf software such as MPT [67], YALMIP [75] or NTG [92].

From the construction of $\mathbb{T}^i, i \in I_g$, it can be verified that the composition of the goal generator and the trajectory planner correctly implements the receding horizon strategy described in Section 7.4. Roughly speaking, the path $\mathcal{G}$ from $\mathcal{W}_j^i$ to $\mathcal{W}_0^i$ computed by the goal generator essentially defines the partial order $\mathcal{P}^i$ and the corresponding function $\mathcal{F}^i$. For a set $\mathcal{W}_{l_\alpha}^i \neq \mathcal{W}_0^i$ contained in $\mathcal{G}$, we simply let $\mathcal{W}_{l_{\alpha+1}}^i \prec \mathcal{W}_{l_\alpha}^i$ and $\mathcal{F}^i(\mathcal{W}_{l_\alpha}^i) = \mathcal{W}_{l_{\alpha+1}}^i$ where $\mathcal{W}_{l_{\alpha+1}}^i$ immediately follows $\mathcal{W}_{l_\alpha}^i$ in $\mathcal{G}$. In addition, since, by assumption, for any $i \in I_g$ and $l \in \{0, \ldots, p\}$, there exists a path in $\mathbb{T}^i$ from $\mathcal{W}_l^i$ to $\mathcal{W}_0^i$, it can be easily verified that the specification Ψ_l^i is realizable with $\mathcal{F}(\mathcal{W}_l^i) = \mathcal{W}_0^i$. Thus, to be consistent with the previously described receding horizon framework, we assign $\mathcal{W}_l^i \succ \mathcal{W}_0^i$ and $\mathcal{F}(\mathcal{W}_l^i) = \mathcal{W}_0^i$ for a set $\mathcal{W}_l^i$ not contained in $\mathcal{G}$. Note that any $\mathcal{W}_l^i$ that is not in the path $\mathcal{G}$ does not affect the computational complexity of the synthesis algorithm. With this definition of the partial order $\mathcal{P}^i$ and the corresponding function $\mathcal{F}^i$, we can apply Theorem 7.4.1 to conclude that the abstract plan generated by the trajectory planner ensures the correctness of the system with respect to the specification in (6.7). In addition, since the continuous controller *simulates* this abstract plan, the continuous execution is guaranteed to preserve the correctness of the system.

The resulting system is depicted in Figure 7.5. Note that since it is guaranteed to satisfy the specification in (6.7), the desired behavior (i.e., the "guarantee" part of (6.7)) is ensured only when the environment and the initial condition respect their assumptions. To moderate the sensitivity to violation of these assumptions, the trajectory planner may send a response to the goal generator, indicating the failure of executing the last received sequence of goals as a consequence of assumption violation. The goal generator can then remove the problematic transition from the corresponding finite transition system $\mathbb{T}^i$ and recompute a new sequence $\mathcal{G}$ of goals. This procedure will be illustrated in the example presented in Section 7.6. Similarly, a response may be sent from the continuous controller to the trajectory planner to account for the mismatch between the actual system and its model. In addition, a local control may be added in order to account for the effect of the noise and unmodeled dynamics captured by Δ.

7.6 Case Study: Autonomous Urban Driving

Motivated by the challenges faced in the design and verification of a DARPA Urban Challenge vehicle such as Alice, we consider an autonomous vehicle navigating an urban envi-

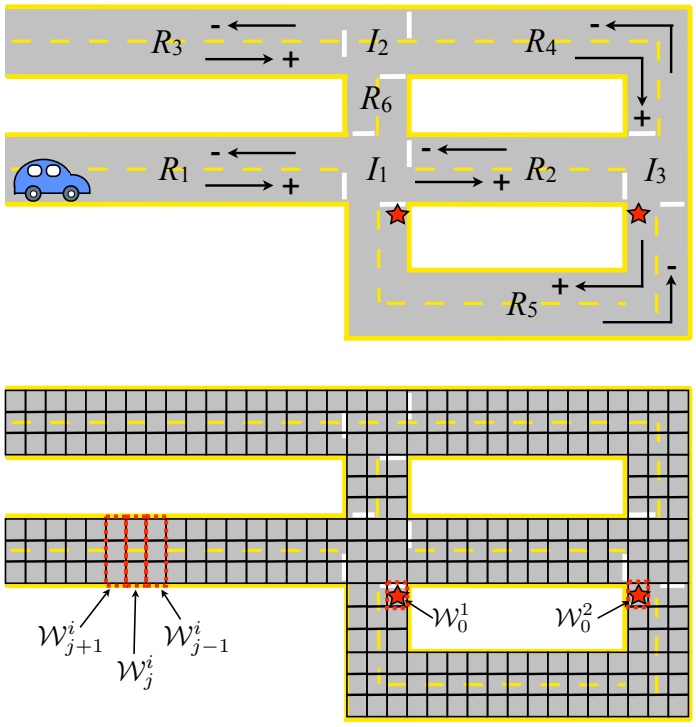


Figure 7.6: The road network and its partition for the autonomous vehicle example. The solid (black) lines define the states in the set $\mathcal{V}$ of the finite state model $\mathbb{D}$ used by the trajectory planner. Examples of subsets $\mathcal{W}^i_j$ are drawn in dotted (red) rectangles. The stars indicate the positions that need to be visited infinitely often.

ronment while avoiding obstacles and obeying certain traffic rules. The state of the vehicle is the position (x, y) whose evolution is governed by

$$\dot{x}(t) = u_x(t) + d_x(t) \quad \text{and} \quad \dot{y}(t) = u_y(t) + d_y(t)$$

where $u_x(t)$ and $u_y(t)$ are control signals and $d_x(t)$ and $d_y(t)$ are external disturbances at time t. The control effort is subject the constraints $u_x(t), u_y(t) \in [-1, 1], \forall t \geq 0$. We assume that the disturbances are bounded by $d_x(t), d_y(t) \in [-0.1, 0.1], \forall t \geq 0$.

We consider the road network shown in Figure 7.6 with three intersections, I_1, I_2 and I_3, and six roads, R_1, R_2 (joining I_1 and I_3), R_3, R_4 (joining I_2 and I_3), R_5 (joining I_1 and I_3) and R_6 (joining I_1 and I_2). Each of these roads has two lanes going in opposite directions. The positive and negative directions for each road are shown in Figure 7.6. We partition the roads and intersections into $N = 282$ cells (see Figure 7.6), each of which may be occupied by an obstacle.

As described in Section 2.1, a planner-controller subsystem of Alice is implemented in a

hierarchical fashion with Mission Planner computing a route (i.e., a sequence of roads to be navigated) to achieve the given tasks, the composition of Traffic Planner and Path Planner computing a path (i.e., a sequence of desired positions) that describes how the vehicle should navigate the route generated by Mission Planner while satisfying the traffic rules, and Path Follower computing a control signal such that the vehicle closely follows the path generated by Traffic Planner. This hierarchical approach follows our general framework for designing a planner-controller subsystem (see Figure 7.5) with Mission Planner being an instance of a goal generator and each of the sets $\mathcal{W}_1^i, \ldots, \mathcal{W}_p^i$ being an entire road. However, these components are typically designed by hand and validated through extensive simulations and field tests. Although a correct-by-construction approach has been applied in [66], it is based on building a finite state abstraction of the physical system and synthesizing a planner that computes a strategy for the whole execution, taking into account all the possible behaviors of the environment. As discussed in Section 6.4, this approach fails to handle even modest size problems due to the state explosion issue. In this section, we apply the receding horizon scheme to substantially reduce computational complexity of the correct-by-construction approach.

Given this system model, we want to design a planner-controller subsystem for the vehicle based on the following desired behavior and assumptions.

Desired Behavior: Following the terminology and notations used in Section 6.3, the desired behavior φ_s in (6.3) includes the following properties.

(P1) Each of the two cells marked by star needs to be visited infinitely often.

(P2) No collision, i.e., the vehicle cannot occupy the same cell as an obstacle.

(P3) The vehicle stays in the travel lane (i.e., right lane) unless there is an obstacle blocking the lane.

(P4) The vehicle can only proceed through an intersection when the intersection is clear.

Assumptions: We assume that the vehicle starts from an obstacle-free cell on R_1 with at least one obstacle-free cell adjacent to it. This constitutes the assumption φ_{init} on the initial condition of the system. The environment assumption φ_e encapsulates the following statements, which are assumed to hold throughout each execution.

(A1) Obstacles may not block a road.

(A2) An obstacle is detected before the vehicle gets too close to it, i.e., an obstacle may not instantly pop up right in front of the vehicle.

(A3) Sensing range is limited, i.e., the vehicle cannot detect an obstacle that is farther away from it than a certain distance. In this example, we let this sensing range be two cells ahead in the driving direction.

(A4) To make sure that the stay-in-lane property is achievable, we assume that an obstacle does not disappear while the vehicle is in its vicinity.

(A5) Obstacles may not span more than a certain number of consecutive cells in the middle of the road.

(A6) Each of the intersections is clear infinitely often.

(A7) Each of the cells marked by star and its adjacent cells are not occupied by an obstacle infinitely often.

The LTL formulas for expressing properties (P2) and (P3) and assumptions (A1)–(A4) can be found in Section 6.5.5. Property (P4) can be expressed as a safety formula and property (P1) is a progress property. Finally, assumption (A5) can be expressed as a safety assumption on the environment while assumptions (A6) and (A7) can be expressed as justice requirements on the environment.

We follow the hierarchical approach described in Section 6.4. First, we compute a finite state abstraction $\mathbb{D}$ of the system. Following the scheme in Section 6.5, a state ν of $\mathbb{D}$ can be written as $\nu = (\varsigma, \rho, o_1, o_2, \ldots, o_M)$ where $\varsigma \in \{1, \ldots, M\}$ and $\rho \in \{+, -\}$ are the controlled state components of ν, specifying the cell occupied by the vehicle and the direction of travel, respectively, and for each $i \in \{1, \ldots, M\}$, $o_i \in \{0, 1\}$ indicates whether the i^{th} cell is occupied by an obstacle. This leads to the total of $2M2^M$ possible states of $\mathbb{D}$. With the horizon length $N = 12$, it can be shown that based on the Reachability Problem defined in Section 6.5.1, there is a transition $\nu_1 \to \nu_2$ in $\mathbb{D}$ if the controlled state components of ν_1 and ν_2 correspond to adjacent cells (i.e., they share an edge in the road network of Figure 7.6).

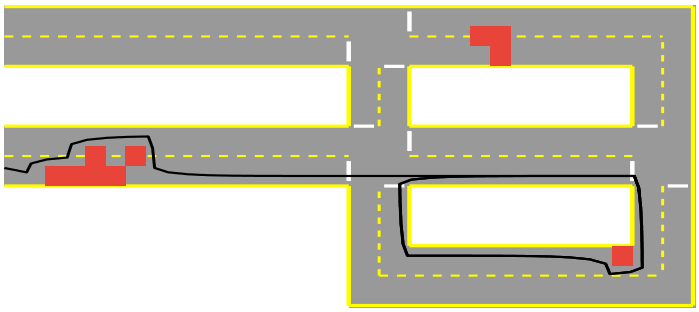
Since the only progress property is to visit the two cells marked by star infinitely often, the set I_g in (6.7) has two elements, say, $I_g = \{1, 2\}$. We let $\mathcal{W}_0^1$ be the set of abstract states

whose ς component corresponds to one of these two cells and define $\mathcal{W}_0^2$ similarly for the other cell as shown in Figure 7.6. Other $\mathcal{W}_j^i$ is defined such that it includes all the abstract states whose ς component corresponds to cells across the width of the road (see Figure 7.6).

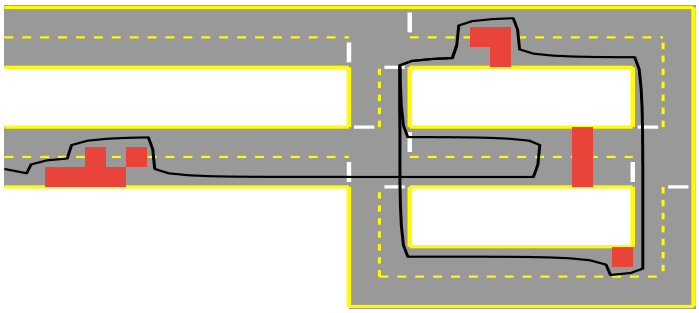
Next, we define Φ such that it excludes states where the vehicle is not in the travel lane while there is no obstacle blocking the lane and states where the vehicle is in the same cell as an obstacle or none of the cells adjacent to the vehicle are obstacle-free. Using this Φ, we can show that for each $i \in I_g$, the specification in (7.1) is realizable with $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_k^i$ for any j, k, provided that $\mathcal{W}_j^i$ and $\mathcal{W}_k^i$ correspond to adjacent dotted (red) rectangles in Figure 7.6. The finite transition system $\mathbb{T}^i$ used by the goal planner can then be constructed such that there is a transition $\mathcal{W}_j^i \to \mathcal{W}_k^i$ for any adjacent $\mathcal{W}_j^i$ and $\mathcal{W}_k^i$. With this transition relation, for any $i \in I_g$ and $j \in \{0, \ldots, p\}$, there exists a path in $\mathbb{T}^i$ from $\mathcal{W}_j^i$ to $\mathcal{W}_0^i$ and the trajectory planner essentially only has to plan one step ahead. Thus, the size of finite state automata synthesized by the trajectory planner to satisfy the specification in (7.1) is completely independent of M. Using JTLV [98], each of these automata has less than 900 states and only takes approximately 1.5 seconds to compute on a MacBook with a 2 GHz Intel Core 2 Duo processor. In addition, with an efficient graph search algorithm, the computation time required by the goal generator is on the order of milliseconds. Hence, with a real-time implementation of optimization-based control such as NTG [86] at the continuous controller level, our approach can be potentially implemented in real-time.

A simulation result is shown in Figure 7.7(a), illustrating a correct execution of the vehicle even in the presence of exogenous disturbances when all the assumptions on the environment and initial condition are satisfied. Note that without the receding horizon strategy, there can be as many as 10^{87} states in the automaton, making this problem practically impossible to solve.

To illustrate the benefit of the response mechanism, we add a road blockage on R_2 to violate the assumption (A1). The result is shown in Figure 7.7(b). Once the vehicle discovers the road blockage, the trajectory planner cannot find the current state of the system in the finite state automaton synthesized from the specification in (7.1) since the assumption on the environment is violated. The trajectory planner then informs the goal generator of the failure to satisfy the corresponding specification with the associated pair of $\mathcal{W}_j^i$ and $\mathcal{F}(\mathcal{W}_j^i)$. Subsequently, the goal generator removes the transition from $\mathcal{W}_j^i$ to $\mathcal{F}(\mathcal{W}_j^i)$ in $\mathbb{T}^i$ and recomputes a path to $\mathcal{W}_0^i$. As a result, the vehicle continues to exhibit a



(a)



(b)

Figure 7.7: Simulation results with (a) no road blockage, (b) a road blockage on R_2.

correct behavior by making a U-turn and completing the task using a different path.

The result with exactly the same setup is also shown in Figure 7.8 where the presence of exogenous disturbances is not incorporated in the planner-controller subsystem synthesis. Once the vehicle overtakes the obstacles on R_1, the continuous controller computes the sequence of control inputs that is expected to bring the vehicle back to its travel lane as commanded by the trajectory planner. However, due to the disturbance, the vehicle remains in the opposite lane. As a consequence, the trajectory planner keeps sending commands to the continuous controller to bring the vehicle back to its travel lane but even though the control inputs computed by the continuous controller are supposed to bring the vehicle back to its travel lane, the vehicle remains in the opposite lane due to the disturbance. In the meantime, the disturbance also causes the vehicle to drift slowly to the right. This cycle continues, leading to violation of the desired property that the vehicle has to stay in the travel lane unless there is an obstacle blocking the lane.

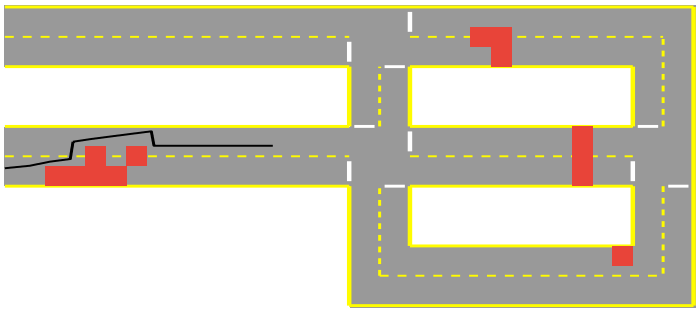


Figure 7.8: Simulation result with the presence of disturbances not incorporated in the planner-controller subsystem synthesis.

7.7 Conclusions

This chapter described a receding-horizon-based framework that allows a computationally complex synthesis problem to be reduced to a set of significantly smaller problems. The implementation of the proposed framework led to a hierarchical, modular design with a goal generator, a trajectory planner and a continuous controller. A response mechanism that increases the robustness of the system with respect to a mismatch between the system and its model and between the actual behavior of the environment and its assumptions was discussed. The example illustrated that the system is capable of exhibiting a correct behavior even if some of the assumptions on the environment do not hold in the actual execution.

Chapter 8

Conclusions and Future Work

8.1 Summary

Design and analysis of embedded control systems are complicated by the tight interactions among their computing, communication and control components. Simulations and tests are not sufficient to guarantee correctness of these systems. Motivated by a subtle design bug that caused an unsafe behavior of the autonomous vehicle, Alice, built at Caltech for the 2007 DARPA Urban Challenge, this thesis develops methods and tools for systematic design and analysis of embedded control software that regulates the overall behavior of such system. The control component of Alice and the autonomous urban driving problem were considered throughout the thesis as motivating, real-world examples.

Existing verification methods were applied to prove correctness of various components of Alice. Periodically Controlled Hybrid Automaton (PCHA), a class of hybrid automata for modular specication of embedded control systems was introduced. Sufficient conditions that simplify invariant verification of PCHAs by exploiting their structure were presented. For PCHAs with polynomial continuous vector fields, it is possible to check these conditions automatically using, for example, quantifier elimination or sum of squares relaxations. The proposed technique is then applied to provide a detailed analysis of the design bug that caused the failure of Alice.

Despite several efforts in simplifying the verification process, verification of embedded control systems such as Alice remains time consuming and requires some level of expertise. Existing verification techniques have their limitations. Algorithmic verification based on model checking is limited to finite state systems. It also faces a combinatorial blow up of the state space, commonly known as the state explosion problem. On the other hand,

deductive verification based on theorem proving is applicable to a more general class of systems. However, it requires a skilled human interaction so the theorem proving process is slow and often error prone. Most of the control oriented verification techniques are limited to linear or polynomial vector fields with relatively small dimension. As a consequence, automatic verification is not applicable to the majority of embedded control systems.

Motivated by these limitations, the second part of the thesis focused on a correct-by-construction approach to system design to complement the verification efforts. We illustrated how off-the-shelf tools from computer science and control can be integrated to allow automatic synthesis of embedded control software that, by construction, ensures the correctness of the system with respect to its desired properties even in the presence of an adversary (typically arising from changes in the environments). This avenue of research is appealing and promising. Once it is brought to practicality, this type of automatic design can potentially reduce the time and cost of the system development cycle as it helps reduce the number of iterations between redesigning the system and verifying the new design.

As an effort towards making correct-by-construction design more practical, a receding horizon framework was presented. This approach reduces computational complexity of the synthesis problem by effectively reducing the original synthesis problem into a set of smaller ones. An implementation of the proposed framework leads to a hierarchical, modular design similar to the one that was manually designed for Alice. A response mechanism that increases the robustness of the system with respect to a mismatch between the system and its model and between the actual behavior of the environment and its assumptions was discussed. This mechanism was motivated by the Canonical Software Architecture implemented on Alice. By taking into account the presence of exogenous disturbances in the synthesis process, the resulting system is provably robust with respect to bounded exogenous disturbances. We conjecture that this architecture also increases the robustness of the system with respect to modeling uncertainties.

The proposed receding-horizon-based approach is not complete in the sense that even if there exists a control strategy that satisfies the system requirements, certain conditions that allow the receding horizon method to be applied may not be satisfied. It also requires some human involvement in defining proper elements necessary to apply this approach. Although the receding horizon framework substantially reduces the size of the automatically generated finite state automaton, this automaton still contains many more states than the manually

generated one (Figure 2.2). However, the correctness of the automatically generated one is guaranteed by construction, while the manually generated one needs to be formally verified for correctness.

8.2 Future Work

On the verification side, an interesting direction for future research is towards automatic invariant proofs of PCHAs combining the proofs for invariance of control steps and for invariance of control-free fragments based on the results of Lemma 5.3.1. Invariance of control steps can be partially automated using a theorem prover (e.g., PVS [93]) while invariance of control-free fragments can be automated using software tools for solving sum of squares problems (e.g., SOSTOOLS [106]) or software tools for quantifier elimination (e.g., QEPCAD [17], the constraint-based approach [43]). We are currently examining a collection of PCHAs with polynomial dynamics for which this direction is promising.

Another direction of future research is related to the progress property. Although the basic principle is straightforward, the details of the progress proof in Section 5.5.3 and 5.5.4 are quite involved. This is partly owing to the difficulty of finding the appropriate Lyapunov functions. In the future, we plan on investigating this further and using ideas from [21] for the progress proof. A longer term goal is to integrate all these proof techniques within the TEMPO [115] environment.

The correct-by-construction approach to system design provides a promising complementary approach to system verification. One of the main limitations of the correct-by-construction approach lies in the computational complexity associated with LTL synthesis, similar to that faced by LTL model checking. The proposed receding horizon framework alleviates this state explosion problem and allows more complex embedded control software synthesis problems to be solved. The key steps in this framework include organizing the discrete states obtained from the state space discretization process into a partially ordered set and defining the receding horizon invariant Φ. A systematic approach for defining these elements needs to be further investigated.

Further investigation of the robustness of the receding horizon framework is also of interest. Specifically, we want to formally identify the types of properties and faults/failures that can be correctly handled using the proposed response mechanism. This type of mech-

anism has been implemented on Alice for distributed mission and contingency management [121]. Based on extensive simulations and field tests, it has been shown to handle many types of failures and faults at different levels of the system, including inconsistency of the states of different software modules and hardware and software failures. A similar response mechanism also appears in the multi-layered synergistic approach presented in [12].

Online computation of the discrete plan generated by the trajectory planner is also promising. As presented in the thesis, the trajectory planner synthesizes an automaton satisfying each short-horizon specification Ψ_j^i offline, stores all the resulting automata and executes the one associated with the current intermediate goal. Since the computation time for synthesizing each of these automata is only 1.5 seconds for the case studies presented in Section 7.6, online synthesis is promising, provided that enough onboard computational power is available. Online synthesis also potentially reduces the computation time and the number of states in the resulting automaton since the synthesis can be performed based on the current state of the system and the environment, as opposed to having to take into account all the possible states of the environment as in offline synthesis.

Another direction of research is to study an asynchronous execution of the goal generator, the trajectory planner and the continuous controller. As described in Chapter 7, these components are to be executed sequentially. However, with certain assumptions on the communication channels, a distributed, asynchronous implementation of these components may still guarantee the correctness of the system.

Introducing optimization in LTL synthesis is also of interest since it allows the trajectory planner to compute an optimal correct plan, instead of any correct plan. Optimal LTL synthesis allows the possibility of considering soft constraints such as the maximum amount of time or energy needed to achieve the goal. Considering multiple goals in this case may lead to exponential computational complexity since all the permutations of the goals need to be consider. Hence, we may have to limit its application to the case where only a small number of goals are specified and consider, for example, the worst case scenario for the evolution of the environment.

We believe that system verification needs to be integrated in the correct-by-construction approach in order to handle large scale systems. For instance, finite state abstraction based on state space discretization often leads to a large number of states. Together with the use of the receding horizon framework, a higher level abstraction such as one based on

the set of allowable maneuvers as implemented on Alice may be needed in order to deal with large scale systems. This type of higher level abstraction requires incorporating a certain degree of system verification in proving properties that each maneuver satisfies. Longer term research directions following this work include incorporating such proofs in the receding horizon framework and illustrating its application in larger scale systems such as smart grids and transportation systems. The challenges in applying the methodology presented in this thesis to these different application domains include identifying the right level of abstraction and the desired properties of these systems. In addition, the notion of "closeness" to the goal states captured by the partially ordered set $\mathcal{P}^i$, which is one of the key elements in applying the receding horizon framework, may not be as obvious as in the examples presented in the thesis.

Bibliography

[1] DARPA Urban Challenge. http://www.darpa.mil/grandchallenge/index.asp, 2007.

[2] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[4] R. Alur, T. Dang, and F. Ivancic. Progress on reachability analysis of hybrid systems using predicate abstraction. In Maler and Pnueli [81], pages 4–19.

[5] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *IEEE Transactions on Software Engineering*, pages 181–201, 1996.

[6] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[7] K. J. Astrom and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, illustrated edition, 2008.

[8] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[9] A. Barrett, R. Knight, R. Morris, and R. Rasmussen. Mission planning and execution within the mission data system. In *Proceedings of the International Workshop on Planning and Scheduling for Space*, 2004.

[10] A. Bayen, I. Mitchell, M. Osihi, and C. Tomlin. Aircraft autolander safety analysis through optimal control-based reach set computation. *Journal of Guidance, Control, and Dynamics*, 30(1):68–77, 2007.

[11] D. P. Bertsekas, A. Nedić, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003.

[12] A. Bhatia, L. Kavraki, and V. Moshe. Sampling-based motion planning with temporal goals. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[13] N. P. Bhatia and G. P. Szegö. *Dynamical Systems: Stability Theory and Applications*, volume 35 of *Lecture notes in mathematics*. Springer-Verlag, Berlin; New York, 1967.

[14] F. Borrelli. *Constrained Optimal Control of Linear and Hybrid Systems*, volume 290 of *Lecture Notes in Control and Information Sciences*. Springer, 2003.

[15] W. Bouma, W. Levelt, A. Melisse, K. Middelburg, and L. Verhaard. Formalization of properties for feature interaction detection: Experience in real-life situation. In H.-J. Kugler, A. Mullery, and N. Niebert, editors, *Towards a Pan-European Telecommunication Servcie Infrastructure*, number 851 in Lecture Notes in Computer Science, pages 393–405. Springer-Verlag, 1994.

[16] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, 2004.

[17] C. W. Brown. QEPCAD b: a program for computing with semi-algebraic sets using cads. *SIGSAM Bull.*, 37(4):97–108, 2003.

[18] J. W. Burdick, N. E. DuToit, A. Howard, C. Looman, J. Ma, R. M. Murray, and T. Wongpiromsarn. Sensing, navigation and reasoning technologies for the DARPA Urban Challenge. Technical report, DARPA Urban Challenge Final Report, 2007.

[19] S. Cerrito and M. C. Mayer. Using linear temporal logic to model and solve planning problems. In *AIMSA*, pages 141–152, 1998.

[20] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[21] K. M. Chandy, S. Mitra, and C. Pilotto. Convergence verification: From shared memory to partially synchronous systems. In *Proceedings of Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 217–231. Springer-Verlag, 2008.

[22] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.

[23] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[24] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[25] D. Conner, H. Kress-Gazit, H. Choset, A. Rizzi, and G. Pappas. Valet parking without a valet. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 572–577, 2007.

[26] R. D'Andrea and R. M. Murray. The RoboFlag competition. In *American Control Conference*, Denver, CO, 2003.

[27] A. J. der Schaft and J. M. Schumacher. *Introduction to Hybrid Dynamical Systems*. Springer-Verlag, London, UK, 1999.

[28] E. Dolginova and N. A. Lynch. Safety verification for automated platoon maneuvers: A case study. In *International Workshop on Hybrid and Real-Time Systems*, pages 154–170, 1997.

[29] M. D. Donahue. Implementation of an active suspension, preview controller for improved ride comfort. Master's thesis, University of California at Berkeley, 2001.

[30] N. E. DuToit, T. Wongpiromsarn, J. W. Burdick, and R. M. Murray. Situational reasoning for road driving in an urban environment. In *International Workshop on Intelligent Vehicle Control Systems (IVCS)*, 2008.

[31] D. Dvorak, R. D. Rasmussen, G. Reeves, and A. Sacks. Software architecture themes in JPL's mission data system. In *Proceedings of 2000 IEEE Aerospace Conference*, 2000.

[32] E. A. Emerson. Temporal and modal logic. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, Cambridge, MA, 1990.

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

[34] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past hytech. In Morari and Thiele [90], pages 258–273.

[35] D. M. Gabbay, C. J. Hogger, and J. A. Robinson. *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 4): Epistemic and Temporal Reasoning.* Oxford University Press, Oxford, UK, 1995.

[36] J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Number 5 in Harper & Row Computer Science and Technology Series. Harper & Row, New York, 1986.

[37] A. Galton, editor. *Temporal Logics and Their Applications.* Academic Press Professional, Inc., San Diego, CA, 1987.

[38] Y. Gao, J. Lygeros, and M. Quincampoix. The reachability problem for uncertain hybrid systems revisited: A viability theory perspective. In J. P. Hespanha and A. Tiwari, editors, *HSCC*, volume 3927 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2006.

[39] A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2):163–179, 2008.

[40] A. Girard and G. J. Pappas. Hierarchical control system design using approximate simulation. *Automatica*, 45(2):566–571, 2009.

[41] P. Gluck and G. Holzmann. Using spin model checking for flight software verification. In *Proc. of IEEE Aerospace Conference*, volume 1, pages 1–105–1–113, 2002.

[42] G. C. Goodwin, M. M. Seron, and J. A. D. Doná. *Constrained Control and Estimation: An Optimisation Approach*. Springer, 2004.

[43] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *20th International Conference on Computer Aided Verification (CAV)*, 2008.

[44] K. Havelund, M. Lowry, and J. Penix. Formal analysis of a space-craft controller using spin. *IEEE Transactions on Software Engineering*, 27:749–765, 2001.

[45] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1:110–122, 1997.

[46] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *ACM Symposium on Theory of Computing*, pages 373–382, 1995.

[47] J. Hespanha. Stabilization through hybrid control. In *Encyclopedia of Life Support Systems (EOLSS)*, volume Control Systems, Robotics, and Automation, 2004.

[48] G. Holzmann. The theory and practice of a formal method: NewCoRe. In *Proc. of the IFIP World Computer Congress*, volume 1, pages 35–44. North-Holland Publ., 1994.

[49] G. J. Holzmann. *The Spin Model Checker*. Addison-Wesley, 2004.

[50] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.

[51] M. Ingham, R. Rasmussen, M. Bennett, and A. Moncada. Engineering complex embedded systems with state analysis and the mission data system. *J. Aerospace Computing, Information and Communication*, 2005.

[52] S. International. The PVS bibliography, 2006. http://pvs.csl.sri.com/papers/pvs-bib/pvs-bib.html.

[53] A. Jadbabaie. *Nonlinear Receding Horizon Control: A Control Lyapunov Function Approach*. PhD thesis, California Institute of Technology, 2000.

[54] L. J. Jagadeesan, C. Puchol, and J. E. Von Olnhausen. A formal approach to reactive systems software: a telecommunications application in esterel. *Formal Methods in System Design*, 8(2):123–151, 1996.

[55] S. Jiang and R. Kumar. Failure diagnosis of discrete event systems with linear-time temporal logic fault specifications. In *IEEE Transactions on Automatic Control*, pages 128–133, 2001.

[56] T. Kalmár-Nagy, R. D'Andrea, and P. Ganguly. Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle. *Robotics and Autonomous Systems*, 46(1):47–64, 2004.

[57] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2009.

[58] S. Karaman, R. G. Sanfelice, and E. Frazzoli. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *Proc. of the IEEE Conference on Decision and Control (CDC)*, pages 2117–2122, 2008.

[59] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool, November 2005. Also available as Technical Report MIT-LCS-TR-917.

[60] S. S. Keerthi and E. G. Gilbert. Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: stability and moving-horizon approximations. *J. Optim. Theory Appl.*, 57(2):265–293, 1988.

[61] G. Keviczky, T. Balas. Flight test of a receding horizon controller for autonomous UAV guidance. In *Proceedings of the American Control Conference*, volume 5, pages 3518–3523, 2005.

[62] E. Klavins. A formal model of a multi-robot control and communication task. In *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2003.

[63] E. Klavins and R. M. Murray. Distributed algorithms for cooperative control. *IEEE Pervasive Computing*, 3(1):56–65, 2004.

[64] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[65] H. Kress-Gazit, G. Fainekos, and G. Pappas. Where's Waldo? Sensor-based temporal logic motion planning. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 3116–3121, April 2007.

[66] H. Kress-Gazit and G. J. Pappas. Automatically synthesizing a planning and control subsystem for the DARPA Urban Challenge. In *IEEE International Conference on Automation Science and Engineering*, pages 766–771, 2008.

[67] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004. Software available at http://control.ee.ethz.ch/~mpt/.

[68] Y. Kwon and G. Agha. LTLC: Linear temporal logic for control. In *HSCC '08: Proc. of the International Workshop on Hybrid Systems*, pages 316–329, Berlin, Heidelberg, 2008. Springer-Verlag.

[69] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F. W. Vaandrager and J. H. van Schuppen, editors, *HSCC*, volume 1569 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 1999.

[70] L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, April 1983.

[71] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[72] F. Lin. Analysis and synthesis of discrete event systems using temporal logic. *Control Theory and Advanced Technologies*, 9(1):341–350, 1993.

[73] M. Linderoth, K. Soltesz, and R. M. Murray. Nonlinear lateral control strategy for nonholonomic vehicles. In *Proceedings of the American Control Conference*, 2008. Submitted.

[74] C. Livadas, J. Lygeros, and N. Lynch. High-level modeling and analysis of TCAS. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 115–125, Phoenix, AZ, December 1999.

[75] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004. Software available at http://control.ee.ethz.ch/~joloef/yalmip.php.

[76] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, August 2003.

[77] N. Lynch and H. Weinberg. Proving correctness of a vehicle maneuver: Deceleration. In *the Second European Workshop on Real-Time and Hybrid Systems*, June 1995.

[78] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1996.

[79] N. A. Lynch. Input/output automata: Basic, timed, hybrid, probabilistic, dynamic, .. In R. M. Amadio and D. Lugiez, editors, *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 187–188. Springer, 2003.

[80] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.

[81] O. Maler and A. Pnueli, editors. *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3–5, 2003, Proceedings*, volume 2623 of *Lecture Notes in Computer Science*. Springer, 2003.

[82] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[83] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000.

[84] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[85] H. Michalska and D. Mayne. Robust receding horizon control of constrained nonlinear systems. *IEEE Transactions on Automatic Control*, 38:1623–1633, 1993.

[86] M. B. Milam, K. Mushambi, and R. M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *Proc. of IEEE Conference on Decision and Control (CDC)*, pages 845–851, 2000.

[87] I. M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. In A. Bemporad, A. Bicchi, and G. C. Buttazzo, editors, *Hybrid Systems: Computation and Control*, volume 4416 of *Lecture Notes in Computer Science*, pages 428–443. Springer, 2007.

[88] S. Mitra. *A Verification Framework for Hybrid Systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, September 2007.

[89] S. Mitra, Y. Wang, N. Lynch, and E. Feron. Safety verification of model helicopter controller using hybrid Input/Output automata. In Maler and Pnueli [81], pages 343–358.

[90] M. Morari and L. Thiele, editors. *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9–11, 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*. Springer, 2005.

[91] R. M. Murray. Optimization-based control. Preprint: http://www.cds.caltech.edu/~murray/amwiki/Supplement:_Optimization-Based_Control, 2008.

[92] R. M. Murray, J. Hauser, A. Jadbabaie, M. B. Milam, N. Petit, W. B. Dunbar, and R. Franz. Online control customization via optimization-based control. In G. B. Tariq Samad, editor, *Software-Enabled Control: Information Technology for Dynamical Systems*. Wiley-IEEE Press, 2003. Software available at http://www.cds.caltech.edu/~murray/software/2002a_ntg.html.

[93] S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[94] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*,

volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[95] A. Papachristodoulou and S. Prajna. Analysis of non-polynomial systems using the sum of squares decomposition. In D. Henrion and A. Garulli, editors, *Positive Polynomials in Control*, pages 23–43. Springer Berlin/Heidelberg, 2005.

[96] G. Papageorgiou, M. Huzmezan, K. Glover, and J. Maciejowski. A combined $MBPC/\mathcal{H}_\infty$ automatic pilot for a civil aircraft. In *Proceedings of the American Control Conference*, pages 118–122, 1997.

[97] D. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Inf. Process. Lett.*, 63(5):243–246, 1997.

[98] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Verification, Model Checking and Abstract Interpretation*, volume 3855 of *Lecture Notes in Computer Science*, pages 364 – 380. Springer-Verlag, 2006. Software available at http://jtlv.sourceforge.net/.

[99] A. Platzer and E. M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In A. Gupta and S. Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 176–189. Springer, 2008.

[100] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[101] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. *Current Trends in Concurrency. Overviews and Tutorials*, pages 510–584, 1986.

[102] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud. A decidable class of planar linear hybrid systems. In M. Egerstedt and B. Mishra, editors, *HSCC*, volume 4981 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2008.

[103] S. Prajna. *Optimization-based methods for nonlinear and hybrid systems verification.* PhD thesis, California Institute of Technology, 2005.

[104] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2004.

[105] S. Prajna, A. Jadbabaie, and S. Member. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Transactions on Automatic Control*, 52:1415–1429, 2005.

[106] S. Prajna, A. Papachristodoulou, and P. A. Parrilo. Introducing SOSTOOLS: A general purpose sum of squares programming solver. In *Proc. of the IEEE Conference on Decision and Control (CDC)*, pages 741–746, 2002.

[107] S. Prajna and A. Rantzer. Primal-dual tests for safety and reachability. In Morari and Thiele [90], pages 542–556.

[108] R. D. Rasmussen. Goal based fault tolerance for space systems using the mission data system. In *Proceedings of the 2001 IEEE Aerospace Conference*, 2001.

[109] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill Book Co., New York, third edition, 1976.

[110] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[111] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32(1):25–55, 2008.

[112] K. T. Seow and R. Devanathan. A temporal framework for assembly sequence representation and analysis. *IEEE Transactions on Robotics and Automation*, 10(2):220–229, 1994.

[113] P. Tabuada and G. J. Pappas. Linear time logic control of linear systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.

[114] H. Tanner and G. J. Pappas. Simulation relations for discrete-time linear systems. In *Proc. of the IFAC World Congress on Automatic Control*, pages 1302–1307, 2002.

[115] Tempo Project. Tempo toolset, version 0.2.2 beta, January 2008. http://www.veromodo.com/tempo/.

[116] U. Topcu, A. Packard, and P. Seiler. Local stability analysis using simulations and sum-of-squares programming. *Automatica*, 44:2669 – 2675, 2008.

[117] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G. E. Dullerud. Stormed hybrid systems. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2008.

[118] T. Wongpiromsarn, S. Mitra, R. Murray, and A. Lamperski. Periodically controlled hybird systems: Verifying a controller for an autonomous vehicle. Technical Report CaltechCDSTR:2008.003, California Institute of Technology, 2008. Full version: http://resolver.caltech.edu/CaltechCDSTR:2008.003.

[119] T. Wongpiromsarn, S. Mitra, R. M. Murray, and A. Lamperski. Periodically controlled hybrid systems: Verifying a controller for an autonomous vehicle. In R. Majumdar and P. Tabuada, editors, *Hybrid Systems: Computation and Control*, volume 5469 of *Lecture Notes in Computer Science*, pages 396–410. Springer, 2009.

[120] T. Wongpiromsarn, S. Mitra, R. M. Murray, and A. Lamperski. Verification of periodically controlled hybrid systems: Application to an autonomous vehicle. *ACM TECS Special Issue on the Verification of Cyber-Physical Software*, 2010. submitted.

[121] T. Wongpiromsarn and R. M. Murray. Distributed mission and contingency management for the DARPA Urban Challenge. In *International Workshop on Intelligent Vehicle Control Systems (IVCS)*, 2008.

[122] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning for dynamical systems. In *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2009.

[123] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Automatic synthesis of robust embedded control software. In *AAAI Spring Symposium on Embedded Reasoning: Intelligence in Embedded Systems*, pages 104–111, 2010.

[124] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. In K. H. Johansson and W. Yi, editors, *HSCC*, pages 101–110. ACM ACM, 2010.

[125] Y. Yu, P. Manolios, and L. Lamport. Model checking TLA+ specifications. In *Conference on Correct Hardware Design and Verification Methods*, pages 54–66, 1999.