Verification of Periodically Controlled Hybrid Systems: Application to An Autonomous Vehicle

TICHAKORN WONGPIROMSARN
California Institute of Technology
SAYAN MITRA
University of Illinois at Urbana Champaign
and
ANDREW LAMPERSKI and RICHARD M. MURRAY
California Institute of Technology

This paper introduces Periodically Controlled Hybrid Automata (PCHA) 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. 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. For PCHAs with polynomial continuous vector fields, it is possible to check these conditions automatically using, for example, quantifier elimination or sum of squares decomposition. 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 an autonomous ground vehicle. Geometric properties of planner-generated paths are derived which guarantee that such paths can be safely followed by the controller.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification—*Correctness proofs*; *Formal methods*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Invariants*; *Specification techniques*; I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*

General Terms: Verification

Additional Key Words and Phrases: Embedded systems, Invariants

1. INTRODUCTION

Subtle design bugs in embedded systems may arise from the unforeseen interactions among the computing, the communication, and the control subsystems. Consider, for example, the embedded computing system of the autonomous vehicle *Alice* built at Caltech. Alice successfully accomplished two of the three tasks at the National Qualifying Event (NQE) of the 2007 DARPA Urban Challenge [Burdick et al.

Author's address: T. Wongpiromsarn, MC 104-44, 1200 E. California Blvd., Pasadena, CA 91125.
This work is partially supported by AFOSR through the MURI program.

2007; Wongpiromsarn and Murray 2008; DuToit et al. 2008]. During the third task, which involved making left-turns while merging into traffic, 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*. 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 an obstacle 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.

The above example illustrates how design of reliable embedded 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 our 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 correct by construction [Kloetzer and Belta 2006; Fainekos et al. 2009].

The hybrid system formalism [Alur et al. 1995; Kaynar et al. 2005] provides a rich mathematical language for specifying embedded systems where computing and control components interact with physical processes. The algorithmic verification problem for hybrid systems with general dynamics is known to be computationally hard [Henzinger et al. 1995]. Restricted subclasses that are amenable to algorithmic analysis have been identified, such as the rectangular-initialized hybrid automata [Henzinger et al. 1995], o-minimal hybrid automata [Lafferriere et al. 1999], and more recently planar [Prabhakar et al. 2008] and STORMED [Vladimerou et al. 2008] 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.

While real world embedded control systems are large and complex, they are also implemented on hardware platforms that have, at the fundamental level, similar architectures. Hence, hybrid automaton models that capture the commonality in such systems have more structure than general hybrid automata [Alur et al. 1995]. With the motivation of abstractly capturing a common design pattern in embedded control systems, such as Alice, and other motion control systems [Mitra et al. 2003], in this paper we study a new subclass of hybrid automata.

Two main contributions of this paper 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 computation 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 [Prajna et al. 2002]. These formulations are illustrated by analyzing a simple example in which an invariant is automatically determined using the constraint-based approach presented in [Gulwani and Tiwari 2008]. We believe that other techniques for finding invariants, for example those presented in [Platzer and Clarke 2008; Sankaranarayanan et al. 2008], could also be effectively used for computing invariants of PCHAs. The findings from this direction of research will be reported in a future paper.

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, nonpolynomial 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. The preliminary results of this paper were published in [Wongpiromsarn et al. 2009].

The remainder of the paper is organized as follows: In Section 2, we briefly present the key definitions for the hybrid I/O automaton framework. In Section 3, we present PCHA and a sufficient condition for proving invariance. In this section, we also present the formulation of the sufficient condition as a sum of squares optimization problem for automatic verification. In Sections 4 and 5, we present the formal model and verification of Alice's Controller-Vehicle subsystem.

2. PRELIMINARIES

We use the Hybrid Input/Output Automata (HIOA) framework of [Lynch et al. 2003; Kaynar et al. 2005] for modelling hybrid systems and the state model-based notations introduced in [Mitra 2007]. 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 paper. We refer the reader to [Kaynar et al. 2005; Mitra 2007] 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 $\text{val}(V)$. For a valuation $\mathbf{v} \in \text{val}(V)$ 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*. 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 $\text{val}(V)$. 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 $t = \tau.\mathsf{ltime}$.

3. PERIODICALLY CONTROLLED HYBRID SYSTEMS

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 3.5, gives a sufficient condition for proving invariant properties of this subclass.

3.1 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* 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) $Q \subseteq \text{val}(X)$ is a set of *states* and $Q_0 \subseteq Q$ is a nonempty set of *start states*;

(c) A is a set of actions, consisting of a set of update actions and a 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 $\mathbf{x} \overset{a}{\rightarrow}_{\mathcal{A}} \mathbf{x}'$ or as $\mathbf{x} \overset{a}{\rightarrow} \mathbf{x}'$ when $\mathcal{A}$ is clear from the context. An action $a \in A$ is said to be *enabled* at a state $\mathbf{x} \in Q$ if there exists a state $\mathbf{x}' \in Q$ such that $\mathbf{x} \overset{a}{\rightarrow} \mathbf{x}'$; and

(e) $\mathscr{S}$ is a collection of *state models* for X, such that for every $\mathcal{S}, \mathcal{S}' \in \mathscr{S}$, $Inv_{\mathcal{S}} \cap Inv_{\mathcal{S}'} = \emptyset$ 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 of control actions 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 val(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 stores 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. Initializing *next* to $-\Delta_2$ ensures that the first control action occurs at time 0.

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

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

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 1 with only one update action, and a unique starting state. However, Theorem 3.5 generalizes to PCHAs with multiple update actions as illustrated later in Section 5.

```
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 $next := now + \Delta_1$;
    $\langle loc, s \rangle := h(loc, s, z)$;
    $u := g(loc, s)$

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

Fig. 1. PHCA with parameters Δ_1, Δ_2, g, h, $\{f_l\}_{l \in \mathcal{L}}$. See, for example [Mitra 2007] for the description of the language.

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, $a_i \in A_{\mathcal{A}}$, $\tau \in trajs(\mathcal{S})$ 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 $\mathsf{Reach}_{\mathcal{A}} \subseteq \mathcal{I}$.

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 (see, for example [Bhatia and Szegö 1967]) provides a sufficient condition for proving invariance of a set $\mathcal{I}$ that is bounded by a closed surface. Theorem 3.5 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 $F_{lk} : \mathcal{X} \rightarrow \mathbb{R}$ is a differentiable function[2]. Formally,

[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.

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

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 3.2 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 3.2. *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 $\mathbf{x} \overset{\mathsf{control}}{\rightarrow} \mathbf{x}'$ and $\mathbf{x} \in \mathcal{I}$ then $\mathbf{x}' \in \mathcal{I}$.

(b) (control-free fragments) For each closed execution fragment $\beta = \tau_0\ \mathsf{update}(z_1)\ \tau_1\ \mathsf{update}(z_2)\ \ldots\ \tau_n$ 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 $\beta.\mathsf{ltime} \leq \Delta_1 + \Delta_2$, then $\beta.\mathsf{lstate} \in \mathcal{I}$.

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

Proof. Consider any reachable state $\mathbf{x}$ of $\mathcal{A}$ and any execution α such that $\alpha.\mathsf{lstate} = \mathbf{x}$. 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 $\beta_i.\mathsf{lstate} \in \mathcal{I}$, then $\beta_{i+1}.\mathsf{fstate} \in \mathcal{I}$.

Thus, it suffices to prove that for each $i \in \{0, \ldots, k\}$, if $\beta_i.\mathsf{fstate} \in \mathcal{I}$, then $\beta_i.\mathsf{lstate} \in \mathcal{I}$. We fix an $i \in \{0, \ldots, k\}$ and assume that $\beta_i.\mathsf{fstate} \in \mathcal{I}$. Let $\beta_i = \tau_0\ \mathsf{update}(z_1)\ \tau_1\ \mathsf{update}(z_2) \ldots \tau_n$, 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 $\beta_0.\mathsf{ltime} = 0$ and $\beta_0.\mathsf{lstate} \lceil \{loc, s\} = \beta_0.\mathsf{fstate} \lceil \{loc, s\}$ 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 control 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 [Owre et al. 1996]. 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, $\partial I_j \triangleq \{s \in I_l \mid F_{lj}(s) = 0\}$. 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 3.3. *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*.*

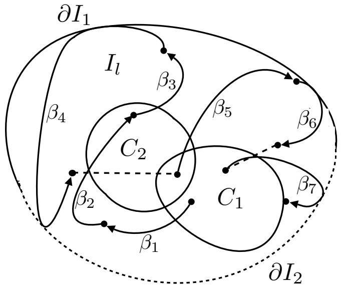


Fig. 2. A graphical explanation of Lemma 3.3 showing an invariant set I_l defined by two boundary functions. The boundary ∂I_1 is drawn in solid line whereas the boundary ∂I_2 is drawn in dotted line. The corresponding sets C_1 and C_2 are also shown.

In Figure 2, the control actions and the control-free execution fragments are shown by bullets and solid lines, respectively. The dashed lines represent the discrete transitions of the control actions. An execution 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.

(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 $\beta.\mathsf{fstate}$, $loc = l$, and $s = x_1$, where $l \in \mathcal{L}$ 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 $\gamma : [0, T] \rightarrow \mathcal{X}$ as $\forall\ t \in [0, T]$, $\gamma(t) \triangleq (\tau \downarrow s)(t)$. 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

[3] $\tau_1 \frown \tau_2$ is the trajectory obtained by concatenating τ_2 at the end of τ_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 3.3. The proof appears in the full version of the paper available from [Wongpiromsarn et al. 2008]

Lemma 3.4. *For a given $l \in \mathcal{L}$, 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 3.3) are satisfied if $C_j \subset I_l$ satisfies the following conditions: (a) C_j is closed, and (b) $C_j \cap \partial I_j = \emptyset$*

Theorem 3.5 combines the above lemmas and provides sufficient conditions for invariance of $\mathcal{I}$.

Theorem 3.5. *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 3.2, and conditions (a)–(d) of Lemma 3.3 for each $l \in \mathcal{L}_{\mathcal{A}}$. Then* $\mathsf{Reach}_{\mathcal{A}} \subseteq \mathcal{I}$.

Theorem 3.5 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. Although the PCHA of Figure 1 has one action of each type, Theorem 3.5 can be extended for periodically controlled hybrid systems with an arbitrary number of update actions. For PCHAs with polynomial vector fields, given semi-algebraic sets I_l and C_j, checking condition (a) and finding c_j and b_j that satisfy conditions (b) and (c) of Lemma 3.3 can be formulated as a sum of squares optimization problem (provided that I_l and C_j are basic semi-algebraic sets) or proving emptiness of certain semi-algebraic 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 [Prajna et al. 2002]. The quantifier elimination problem can also be formulated and allows the proof to be automated using, for example, QEPCAD [Brown 2003]. Alternatively, in Section 3.4, we show how an invariant set can be automatically computed using the constraint-based approach presented in [Gulwani and Tiwari 2008].

3.3 Sum of Squares Formulation for Checking the Invariant Conditions

Suppose the candidate invariant set I_l is a basic semi-algebraic 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 3.3 for a given basic semi-algebraic subset C_j, and (2) finding a subset C_j such that conditions (a)–(c) of Lemma 3.3 are satisfied. For the first case, the sum of squares problem is convex and can be solved using, for example, SOSTOOLS [Prajna et al. 2002]. For the second case, however, the problem is not convex but can still be automatically solved using an iterative scheme as presented in [Prajna and Jadbabaie 2004]. Roughly, we iterate between the following two steps: (i) fix the unknown multipliers and search for a subset C_j that satisfies conditions (a)–(c) of Lemma 3.3, and (ii) fix C_j and search for the unknown multipliers.

Checking the Invariant Condition for a Given Subset

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

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

where $G_{ji} : \mathcal{X} \to \mathbb{R}$ is a real polynomial for each $i \in \{1, \dots, p\}$. Using the generalized S-procedure (a special case of the Positivstellensatz) [Topcu et al. 2008], we obtain the following sufficient condition for condition (a) of Lemma 3.3.

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

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

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 above equation is non-negative. In other words, the above condition ensures that for each $k \in \{1, \dots, p\}$,

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

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, condition (b) of Lemma 3.3 can be formulated as the following optimization problem.

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

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

Finally, condition (c) of Lemma 3.3 can be formulated as the following optimization problem.

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

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

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

3.4 Example

In this section, we illustrate, on a simple example, how invariant verification of a PCHA can be partially automated. 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 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).

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 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 3.5 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 3.2, 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

```
signature                                                  1
  internal control
  input update($z' : Z$)                                   3

variables                                                  5
  internal $s : \mathbb{R} :=\ s_0 = 0$
  internal discrete $loc : \mathbb{R}$, $z : \mathbb{R}$, $u : \{a_1, a_2\}$      7
  internal $now : \mathbb{R}_{\geq 0} := 0$, $next : \mathbb{R}_{\geq 0} :=\ 0$
                                                           9
transitions
  input update($z'$)                                       11
  eff $z :=\ z'$

  internal control
  pre $now \geq next$                                      14
  eff $next :=\ now + \Delta$;
    $s := s - z + loc$;                                    16
    $loc := z$;
    if $s > 0$ then $u :=\ a_1$                            18
    else $u :=\ a_2$ fi
                                                           20
trajectories
  evolve $d(now) = 1$; $d(s) = u$                          22
  stop when $now = next$
```

Fig. 3. The state regulator system with parameters $a_1 \in \mathbb{R}_-$, $a_2 \in \mathbb{R}_+$ and $\Delta \in \mathbb{R}_+$.

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 3.2 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 3.3 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 3.3 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 3.2 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 3.3 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 3.3 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) $(F_{l1}(s_0) < 0) \vee (F_{l2}(s_0) < 0) \vee (G_1(s_0) \geq 0) \vee (F_{l1}(s) \neq 0) \vee (F_{l2}(s) < 0) \vee (s_0 \leq 0)$,

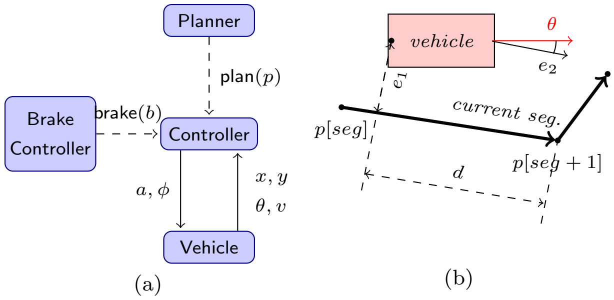
(b) $\kappa_1 \leq \eta_2$, $\kappa_1 > \eta_1$, and $\kappa_1 - \eta_1 \geq \Delta$.

Note that condition (a) is obtained from condition (a) of Lemma 3.3 while condition (b) is obtained from conditions (b) and (d) of Lemma 3.3. 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 $\eta_2 - \kappa_2 \geq \Delta$.

As described in [Gulwani and Tiwari 2008], 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



```
variables
output $x$:$\mathbb{R}$:= $x_0$; $y$:$\mathbb{R}$:= $y_0$;
  $\theta$:$\mathbb{R}$:= $\theta_0$; $v$:$\mathbb{R}$:= $v_0$
input $a$, $\phi$: $\mathbb{R}$

trajectories
evolve $d(x) = v\cos(\theta)$
  $d(y) = v\sin(\theta)$
  if $|u.\phi| \leq \phi_{max}$
    then $d(\theta) = \frac{v}{L}\tan(\phi)$
    else $d(\theta) = \frac{v}{L}\tan(\frac{\phi}{|\phi|}\phi_{max})$ fi
  if $v > 0 \vee a \geq 0$
    then $d(v) = a$
    else $d(v) = 0$ fi
```

(c)

Fig. 4. (a) Planner-Controller system. (b) Deviation and disorientation. (c) Vehicle.

$$\nu_1 F_{l1}(s_0) + \nu_2 F_{l2}(s_0) - \mu_1 G_1(s_0) + \lambda_1 F_{l1}(s) + \nu_3 F_{l2}(s) + \mu_2 s_0 + \mu_3 = 0 \qquad (2)$$

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

$$\nu_4 F_{l1}(s_0) + \nu_5 F_{l2}(s_0) - \mu_4 G_2(s_0) + \nu_6 F_{l1}(s) + \lambda_2 F_{l2}(s) - \nu_7 s_0 + \mu_5 = 0 \qquad (3)$$

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

Using the tool presented in [Gulwani and Tiwari 2008], the unknowns that satisfy (2), (3) 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 $I_l = [-0.1, 0.1]$).

4. AUTONOMOUS VEHICLE SYSTEM

In this section, we describe an autonomous ground vehicle (Alice) consisting of the physical vehicle and the controller (see Figure 4(a)). Vehicle captures the position, orientation, and velocity of Alice on the plane. Controller receives information about the state of Alice and a (possibly infinite) sequence of waypoints from a Planner, and periodically computes the steering (ϕ) and acceleration (a) such that Alice (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.

4.1 Vehicle

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. The Vehicle automaton (Figure 4) specifies the dynamics of the vehicle with acceleration (a) and steering

(ϕ) inputs, in terms of 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 output variables of Vehicle are (a) the x and y coordinates of the position with respect to a global coordinate system, (b) orientation θ with respect to the positive direction of the global x axis, and (c) 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.

4.2 Controller

Figure 5 shows the specification of the Controller automaton that reads the state of the Vehicle 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, 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 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 4(b)).

(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.

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 (not presented in this paper) 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:

```
signature
  input plan($p$:Seq[$\mathbb{R}^]$]); brake($b : On, Off$)                  2
  internal main
                                                                             4
variables
  input $x, y, \theta, v$: $\mathbb{R}$                                      6
  output $a, \phi$: $\mathbb{R}$ := (0, 0)
  internal $brake$: {$On$, $Off$} := $Off$                                   8
    $path$: Seq[$\mathbb{R}^2$] := $arbitrary$
    $new\_path$: Seq[$\mathbb{R}^2$] := $path$                               10
    $seg$: $\mathbb{N}$ := 1
    $e_1, e_2, d : \mathbb{R} := [e_{1,0},\ e_{2,0},\ d_0]$                  12
    $now$: $\mathbb{R}$ := 0; $next$:$\mathbb{R}_{\geq 0}$ := 0
                                                                             14
transitions
  input plan($p$)                                                            16
  eff $new\_path$ := $p$
                                                                             18
  input brake($b$)
  eff $brake$ := $b$                                                         20

  internal main                                                              22
  pre $now = next$
  eff $next$ := $now + \Delta$                                               24
  if $path \neq new\_path \vee d \leq 0$ then
    if $path \neq new\_path$                                                 26
      then $seg$ := 1; $path$ := $new\_path$
    elseif $d \leq 0$                                                        28
      then $seg$ := $seg + 1$ fi
        let $\vec{p} = \begin{bmatrix} path[seg+1].x - path[seg].x \\ path[seg+1].y - path[seg].y \end{bmatrix}$   30
        $\vec{q} = \begin{bmatrix} path[seg+1].y - path[seg].y \\ -(path[seg+1].x - path[seg].x) \end{bmatrix}$
        $\vec{r} = \begin{bmatrix} path[seg+1].x - x \\ path[seg+1].y - y) \end{bmatrix}$                          32
      $e_1 := \frac{1}{\|\vec{q}\|}\vec{q} \cdot \vec{r}$
      $e_2 := \theta - \angle\vec{p}$                                        34
      $d := \frac{1}{\|\vec{p}\|}\vec{p} \cdot \vec{r}$
  fi                                                                         36

  let $\phi_d = -k_1\ e_1 - k_2\ e_2$                                        38
  $\phi = \frac{\phi_d}{|\phi_d|} \min(\delta \times v, |\phi_d|)$
                                                                             40
  if $brake = On$ then $a$ := $a_{brake}$
  elseif $brake = Off \wedge v < v_T$                                        42
    then $a$ := $a_{max}$
  else $a$ := 0 fi                                                           44

trajectories                                                                 46
  evolve $d(now) = 1$
    $d(e_1) = v \sin(e_2)$                                                   48
    $d(e_2) = \frac{v}{L} \tan(\phi)$
    $d(d) = -v \cos(e_2)$                                                    50
  stop when $now = next$
```

Fig. 5. Controller with parameters $v_T, k_1, k_2 \in \mathbb{R}_{\geq 0}$, $\delta, \Delta \in \mathbb{R}_+$ and $a_{brake} < 0$.

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, temporary variables $\vec{p}$, $\vec{q}$, and $\vec{r}$ are computed to update e_1, e_2, d;

D. The steering ϕ is computed using a proportional control law saturated at $\delta \times v$. That is, the magnitude of the steering output ϕ is set to the minimum of $|-k_1 e_1 - k_2 e_2|$ and $\delta \times v$ (line 39).

E. The acceleration a is computed using 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.

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. The variables, actions, state transition functions of the corresponding PCHA can be found in the full version of the paper [Wongpiromsarn et al. 2008].

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 upper-bounded by e_{max}, where e_{max} is determined in terms of system parameters.

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

In Sections 5.1 and 5.2, we define a family $\{\mathcal{I}_k\}_{k \in \mathbb{N}}$ of subsets of $Q_{\mathcal{A}}$ and using Lemmas 3.3, 3.4, we conclude that they are invariant with respect to the control-free 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 $d \leq 0$. Hence, using Theorem 3.5, 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 5.3, 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 5.4, 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 National Qualifying Event of the 2007 DARPA Urban Challenge, Alice violated the safety property (A), leading to the stuttering behavior.

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

$$\mathcal{I}_k \triangleq \{\mathbf{x} \in Q \mid \forall i \in \{1, \ldots 6\}, F_{k,i}(\mathbf{x}.s) \geq 0\} \tag{4}$$

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

$$F_{k,1}(s) = \epsilon_k - s.e_1; \qquad F_{k,2}(s) = \epsilon_k + s.e_1; \tag{5}$$

$$F_{k,3}(s) = \phi_k + k_1 s.e_1 + k_2 s.e_2; \qquad F_{k,4}(s) = \phi_k - k_1 s.e_1 - k_2 s.e_2; \tag{6}$$

$$F_{k,5}(s) = v_{max} - s.v; \qquad F_{k,6}(s) = \delta s.v - \phi_b. \tag{7}$$

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

$$\theta_{k,1} = \frac{k_1}{k_2}\epsilon_k - \frac{1}{k_2}\phi_k \ \text{ and } \ \theta_{k,2} = \frac{k_1}{k_2}\epsilon_k + \frac{1}{k_2}\phi_k. \tag{8}$$

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 5.1 is shown Figure 6.

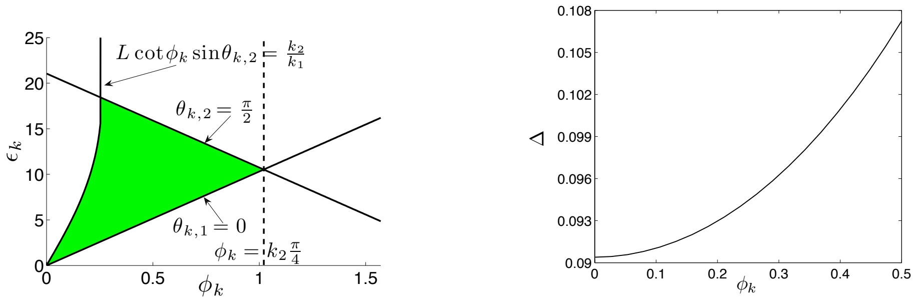
***Assumption* 5.1.** (Vehicle and controller design) (a) $\phi_k \leq \phi_b \leq \phi_{max}$ and $\phi_k < \frac{\pi}{2}$, (b) $0 \leq \theta_{k,1} \leq \theta_{k,2} < \frac{\pi}{2}$, (c) $L \cot\phi_k \sin\theta_{k,2} < \frac{k_2}{k_1}$, (d) $\Delta \leq \frac{c}{b}$ where $c = \frac{1}{\sqrt{k_1^2+k_2^2}}(\phi_k - \tilde{\phi})$, $b = v_{max}\sqrt{\sin^2\theta_{k,2} + \frac{1}{L^2}\tan^2(\tilde{\phi})}$ and $\tilde{\phi} = \cot^{-1}\left(\frac{k_2}{k_1 L \sin\theta_{k,2}}\right)$,[4] and (e) $\frac{\tan\phi_k}{2L} v_{max}\Delta \leq \frac{\pi}{2}$.

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* 5.2.** 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 in a *good* brake controller α with α.lstate $= \mathbf{x}$.

[4] Using Assumption 5.1(c), it can be shown that $\tilde{\phi} < \phi_k$ so $\frac{c}{b} > 0$.



(a) (b)

Fig. 6. (a) The set of (ϵ_k, ϕ_k) that satisfies Assumptions 5.1 (b) and (c) and are represented by the green region. (b) The relationship between the maximum bound on Δ and ϕ_k for $\epsilon_k = \frac{1}{k_1}\phi_k$.

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 3.3, 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 3.2 by applying Lemma 3.3.

First, we check that the conditions in Lemma 3.3 are satisfied. This analysis appears in [Wongpiromsarn et al. 2008]. It does not involve solving differential equations but relies on algebraic simplification of the expressions defining the vector fields and the boundaries of the invariant set.

The next lemma shows that conditions (a)-(c) of Lemma 3.3 are satisfied. The proof for $j = 5$ is presented here as an example.

Lemma 5.3. *For each $l \in \mathcal{L}$ and $j \in \{1, \ldots, 6\}$, the subtangential, bounded distance, and bounded speed conditions (of Lemma 3.3) are satisfied.*

PROOF. Define $C_5 \triangleq \{s \in I \mid s.v \leq v_T\}$. We apply Lemma 3.4 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, $C_5 \cap \partial I_5 = \emptyset$. Hence, from Lemma 3.4, 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, $v_T < s_0.v \leq v_{max}$. 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 $\frac{\partial F_5}{\partial s} \cdot f(s, g(l, s_0)) = -f_4(s, g(l, s_0)) \geq 0$. □

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 5.1(d), we prove the sampling rate condition.

Lemma 5.4. *For each $l \in \mathcal{L}$, the sampling rate condition is satisfied.*

Thus, conditions (a)–(d) of Lemma 3.3 are satisfied; from Theorem 3.5 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. Hence, any plan-free execution fragment, i.e., an execution fragment that does not contain a plan action, preserves invariance of $\mathcal{I}$ as stated in the following theorem.

Theorem 5.5. *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 $\mathbf{x}' \in \mathcal{I}$.*

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 5.7, 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. 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 proof appears in [Wongpiromsarn et al. 2008].

Lemma 5.6. *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 [Wongpiromsarn et al. 2008]. The plots showing the progress in the deviation and disoriantation are shown in Figure 7(a) and Figure 7(b), respectively.

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

Lemma 5.7. *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 Theorem 5.5, 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 5.6 and Lemma 5.7, 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 [Wongpiromsarn et al. 2008].

Lemma 5.8. *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 $\beta.\mathsf{lstate} \lceil d \geq \mathbf{x}.d - v_{max}\Delta$.*

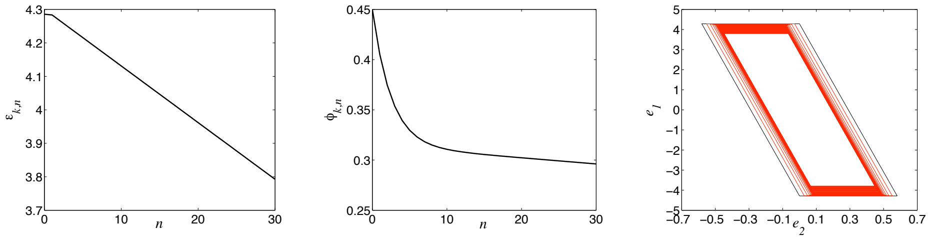
Finally, we conclude the section by establishing the segment progress property defined at the beginning of Section 5.

Theorem 5.9. *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 Theorem 5.5, 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 5.7, we see that at this state, $d \geq \mathbf{x}.d - v_{max}\Delta$. Applying Lemma 5.8 and using an invariance of $\mathcal{I}_k$ for any k proved in Theorem 5.5, 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 7(c).



(a) (b) (c)

Fig. 7. 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.

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 5.1 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 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 5.13 uses the segment progress property established in Section 5.3 to prove that this execution fragment preserves an invariance of $\mathcal{I}_0$. Then, in Lemma 5.14 and Lemma 5.15, 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 5.16 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 5.13 and Lemma 5.16 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.

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

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

$$\epsilon_n \leq \frac{1}{|\cos \sigma_i|} (\epsilon_0 - v_{max} \Delta |\sin \sigma_i|), \tag{9}$$

$$\phi_n \leq \phi_0 - k_1 v_{max} \Delta \sin |\sigma_i| - k_1 \epsilon_n (1 - \cos \sigma_i) - k_2 |\sigma_i|, \tag{10}$$

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 5.6.

The relationship between λ and the maximum value of σ which satisfies this assumption is shown in Figure 8.

***Remark* 5.11.** The choice of ϵ_0's and ϕ_0's affects both the requirements on a safe path (Assumption 5.10) and the definition of a *good* brake controller (Definition 5.2). 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 the introduction of the paper. Without having quantified the tradeoff, we inadvertently allowed a path to have sharp turns and also brakes at low speeds—thus violating safety.

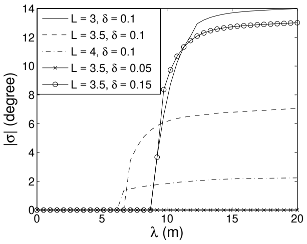


Fig. 8. Segment length vs. maximum difference between consecutive segment orientations, for different values of L and δ.

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 5.12.

***Assumption* 5.12.** (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 $\mathsf{Vehicle}$.

(b) Let v_p and θ_p be the speed and the orientation of the $\mathsf{Vehicle}$, 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 (8). 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,

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

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 5.3, the update rule of the variable *seg* and Lemma 5.7, 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 [Wongpiromsarn et al. 2008] for the complete proof.) Since we restrict the sharpness of the turn with respect to segment length (Assumption 5.10), we can then conclude that this execution fragment preserves an invariance of $\mathcal{I}_0$.

Lemma 5.13. *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 5.12 is sufficient to guarantee that if the path changes, then all the assumptions in the Lemma 5.13 are satisfied. All the proofs appear in [Wongpiromsarn et al. 2008].

Lemma 5.14. *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 5.15. *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* $\mathbf{x}' \in \mathcal{I}_0$*.*

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 5.16. *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 Theorem 5.5, $\beta_1.\mathsf{lstate} \in \mathcal{I}_0$. Applying Lemma 5.14 and Lemma 5.15, 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 5.13, $\beta.\mathsf{lstate} \in \mathcal{I}_0$. □

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

Theorem 5.17. *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 5.16, 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 5.13 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 9, which illustrate that the Vehicle is capable of making a sharp left turn, provided that the path satisfies Assumption 5.10. In addition, we are able to replicate the stuttering behavior described in the Introduction when Assumption 5.10 is violated.

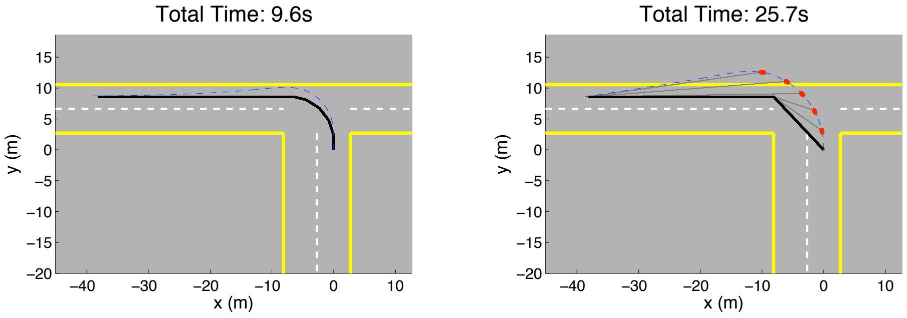


Fig. 9. The positions of Alice (dashed line) as it follows a path (solid line) to execute a sharp left turn. When *brake* is triggered a thick dashed (red) line is drawn on the position of Alice. *Left.* The path satisfies Assumption 5.10. *Right.* The path does not satisfy the assumption and the replan occurs due to excessive deviation.

6. CONCLUSIONS

Motivated by a design bug that caused an unsafe behavior of an autonomous vehicle (Alice) built at Caltech for the 2007 DARPA Urban Challenge, this paper introduced Periodically Controlled Hybrid Automata, 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 decomposition.

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.

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 3.2. Invariance of control steps can be partially automated using a theorem prover while invariance of control-free fragments can be automated using software tools for solving sum of squares problems (e.g. SOSTOOLS [Prajna et al. 2002]) or software tools for quantifier elimination (e.g. QEPCAD [Brown 2003], the constraint-based approach [Gulwani and Tiwari 2008]). 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 Sections 5.3 and 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 use ideas from [Chandy et al. 2008] for the progress proof.

7. ACKNOWLEDGMENTS

The authors gratefully acknowledge Sumit Gulwani and Ashish Tiwari for letting us use their nonlinear solver for solving ∃∀ problems.

REFERENCES

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138,* 1, 3–34.

Bhatia, N. P. and Szegö, G. P. 1967. *Dynamical Systems: Stability Theory and Applications.* Lecture notes in mathematics, vol. 35. Springer-Verlag, Berlin; New York.

Brown, C. W. 2003. QEPCAD b: a program for computing with semi-algebraic sets using cads. *SIGSAM Bull. 37,* 4, 97–108.

Burdick, J. W., DuToit, N., Howard, A., Looman, C., Ma, J., Murray, R. M., and Wongpiromsarn, T. 2007. Sensing, navigation and reasoning technologies for the DARPA Urban Challenge. Tech. rep., DARPA Urban Challenge Final Report.

Chandy, K. M., Mitra, S., and Pilotto, C. 2008. Convergence verification: From shared memory to partially synchronous systems. In *Proceedings of Formal Modeling and Analysis of Timed Systems (FORMATS'08).* Lecture Notes in Computer Science, vol. 5215. Springer Verlag, 217–231.

DuToit, N. E., Wongpiromsarn, T., Burdick, J. W., and Murray, R. M. 2008. Situational reasoning for road driving in an urban environment. In *International Workshop on Intelligent Vehicle Control Systems (IVCS).*

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

Gulwani, S. and Tiwari, A. 2008. Constraint-based approach for analysis of hybrid systems. In *Computer Aided Verification*, A. Gupta and S. Malik, Eds. Lecture Notes in Computer Science, vol. 5123. Springer, 190–203.

Henzinger, T. A., Kopke, P. W., Puri, A., and Varaiya, P. 1995. What's decidable about hybrid automata? In *ACM Symposium on Theory of Computing.* 373–382.

Kaynar, D. K., Lynch, N., Segala, R., and Vaandrager, F. 2005. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool. Also available as Technical Report MIT-LCS-TR-917.

Kloetzer, M. and Belta, C. 2006. A fully automated framework for control of linear systems from LTL specifications. In *Hybrid Systems: Computation and Control*, J. P. Hespanha and A. Tiwari, Eds. LNCS, vol. 3927. Springer, 333–347.

Lafferriere, G., Pappas, G. J., and Yovine, S. 1999. A new class of decidable hybrid systems. In *Hybrid Systems: Computation and Control*, F. W. Vaandrager and J. H. van Schuppen, Eds. LNCS, vol. 1569. Springer, 137–151.

Lynch, N., Segala, R., and Vaandrager, F. 2003. Hybrid I/O automata. *Information and Computation 185,* 1 (August), 105–157.

Mitra, S. 2007. A verification framework for hybrid systems. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA 02139.

Mitra, S., Wang, Y., Lynch, N., and Feron, E. 2003. Safety verification of model helicopter controller using hybrid Input/Output automata. In *Hybrid Systems: Computation and Control*, O. Maler and A. Pnueli, Eds. LNCS, vol. 2623. Springer, 343–358.

Owre, S., Rajan, S., Rushby, J., Shankar, N., and Srivas, M. 1996. PVS: Combining specification, proof checking, and model checking. In *Computer-Aided Verification, CAV '96*, R. Alur and T. A. Henzinger, Eds. Number 1102 in Lecture Notes in Computer Science. Springer-Verlag, New Brunswick, NJ, 411–414.

Platzer, A. and Clarke, E. M. 2008. Computing differential invariants of hybrid systems as fixedpoints. In *Computer-Aided Verification*, A. Gupta and S. Malik, Eds. Lecture Notes in Computer Science, vol. 5123. Springer, 176–189.

Prabhakar, P., Vladimerou, V., Viswanathan, M., and Dullerud, G. E. 2008. A decidable class of planar linear hybrid systems. In *Hybrid Systems: Computation and Control*, M. Egerstedt and B. Mishra, Eds. LNCS, vol. 4981. Springer, 401–414.

Prajna, S. and Jadbabaie, A. 2004. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, R. Alur and G. J. Pappas, Eds. LNCS, vol. 2993. Springer, 477–492.

Prajna, S., Papachristodoulou, A., and Parrilo, P. A. 2002. Introducing SOSTOOLS: A general purpose sum of squares programming solver. In *Proceedings of the 41st IEEE Conf. on Decision and Control.* 741–746.

Sankaranarayanan, S., Sipma, H. B., and Manna, Z. 2008. Constructing invariants for hybrid systems. *Formal Methods in System Design 32,* 1, 25–55.

Topcu, U., Packard, A., and Seiler, P. 2008. Local stability analysis using simulations and sum-of-squares programming. *Automatica 44*, 2669 – 2675.

Vladimerou, V., Prabhakar, P., Viswanathan, M., and Dullerud, G. E. 2008. STORMED hybrid systems. In *ICALP (2).* LNCS, vol. 5126. Springer, 136–147.

Wongpiromsarn, T., Mitra, S., Murray, R., and Lamperski, A. 2008. Periodically controlled hybird systems: Verifying a controller for an autonomous vehicle. Tech. Rep. CaltechCDSTR:2008.003, California Institute of Technology. Full version: `http://resolver.caltech.edu/CaltechCDSTR:2008.003`.

Wongpiromsarn, T., Mitra, S., Murray, R. M., and Lamperski, A. 2009. Periodically controlled hybrid systems: Verifying a controller for an autonomous vehicle. In *Hybrid Systems: Computation and Control*, R. Majumdar and P. Tabuada, Eds. LNCS, vol. 5469. Springer, 396–410.

Wongpiromsarn, T. and Murray, R. M. 2008. Distributed mission and contingency management for the DARPA urban challenge. In *International Workshop on Intelligent Vehicle Control Systems (IVCS).*

Received July 2009; January 2010; accepted June 2010