Safety Verification of Model Helicopter Controller Using Hybrid Input/Output Automata*

Sayan Mitra,[1] Yong Wang,[2] Nancy Lynch,[1] and Eric Feron[2]

[1] MIT Laboratory for Computer Science,
200 Technology Square, Cambridge, MA 02139
{mitras,lynch}@theory.lcs.mit.edu,
[2] MIT Laboratory for Information and Decision Systems,
77 Massachusetts Avenue, Cambridge, MA 02139
{y_wang,feron}@mit.edu

Abstract. This paper presents an application of the Hybrid I/O Automaton (HIOA) framework [12] in verifying a realistic hybrid system. A supervisory pitch controller for a model helicopter system is designed and then verified. The design of the supervisor is limited by the actuator bandwidth, the sensor inaccuracies, and the sampling rates. Verification is carried out by induction over the length of an execution of the composed system automaton. The HIOA model makes the inductive proofs tractable by decomposing them into independent discrete and continuous parts. The paper also presents a set of language constructs for specifying hybrid I/O automata.

1 Introduction

Formal verification of hybrid systems is a hard problem. It has been shown that checking reachability for even a simple class of hybrid automata is undecidable [7]. Algorithmic verification techniques have been developed for smaller subclasses of hybrid automata [1], but these subclasses are too weak to model realistic hybrid systems. Languages and tools [6] developed for algorithmic verification are also inadequate for describing general hybrid systems. More recently, algorithms for overapproximating the unsafe sets of general hybrid systems have been developed [3], but applying these algorithms to systems with high dimensionality remain a challenging problem.

An alternative to algorithmic verification is to derive the desired properties of an automaton by induction over the length of its executions. The Hybrid Input/Output Automaton (HIOA) model [13, 14, 12] model has been developed for this purpose; see [8, 19, 11] for related earlier works. Being more expressive, HIOA can model a larger class of hybrid systems. The inductive proofs are tractable in this model because they are decomposed into independent discrete

* Funding for this research has been provided by AFRL contract F33615-01-C-1850 and DARPA/AFOSR MURI Award F49620-02-1-0325

and continuous parts. Further, owing to the assertional style of proving the invariants, it will be possible to partially automate the proofs using mechanical theorem provers.

This paper presents the verification of a supervisory controller of a model helicopter system using the HIOA framework. The helicopter system (Figure 1), manufactured by Quanser [17], is driven by two rotors mounted at the two ends of its frame. The frame is suspended from an instrumented joint mounted at the end of a long arm. The arm is gimbaled on another instrumented joint and is free to pitch and yaw, giving the helicopter three degrees of freedom. The rotor inputs are either controlled by the user with a joystick, or by controllers designed by the user. Students of Aeronautics and Astronautics at MIT experiment with these systems by writing different controllers which often tend to damage the equipment by pitching the helicopter too high or too low. This is also a hazard for the users, and therefore the safety of the system is important.

The supervisory controller is designed to prevent the helicopter from reaching unsafe states. It periodically observes the position and the velocity of the helicopter and overrides the user controller by conservatively estimating the worst that might happen if the latter is allowed to remain in control. The design of the supervisor is limited by the actuator bandwidth, the sampling rate, and sensor inaccuracies. Safety of the supervisor is verified by modeling each component of the system as a hybrid I/O automaton, and proving a set of invariants for the composed system automaton.

This paper also describes a specification language for HIOA. In this language discrete transitions of hybrid I/O automata are specified in the usual precondition-effect style, and the continuous evolution is written in terms of constrained "state-space" models called activities. At present we have tool support for IOA [5], a formal language for distributed systems, which is similar to HIOA without the continuous part. We intend to extend the IOA Toolkit for checking HIOA code, by adding the language constructs for the continuous part. We are also working on building a theorem prover interface for HIOA.

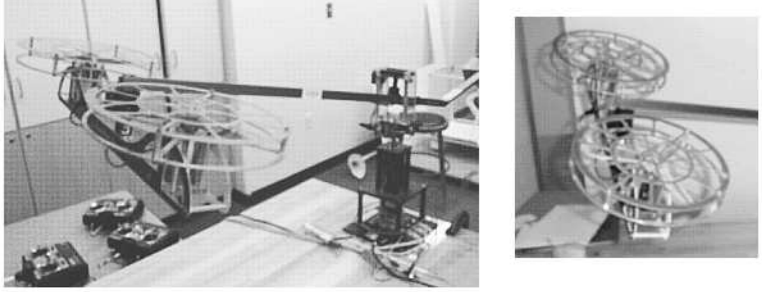

Fig. 1. Helicopter model with three degrees of freedom.

The contributions of this paper are: (1) demonstration of a realistic application of the hybrid I/O automata based verification methodology, (2) design of the supervisory controller which ensures safety of the Quanser helicopter system along the pitch axis, and (3) a set of language constructs for specifying hybrid I/O automata.

In Section 2 we review the hybrid I/O automata model and describe the specification language. We present the HIOA models of the system components and the supervisor in Sections 3 and 4 respectively. Due to limited space we present brief proof sketches for the important invariants required for proving safety of the system in Section 5. The full version of the paper with complete proofs appears as a technical report [16]. Concluding remarks and future directions for research are discussed in Section 6.

2 Hybrid I/O Automata

A brief review of the HIOA model is presented in this section. For a complete discussion refer to [12]. Earlier versions of the model appeared in [13] and [14].

We introduce some notations used in the rest of the paper. If f is a function and S is a set then we write $f\lceil S$ for the function g with $dom(g) = dom(f) \cap S$ such that for every $c \in dom(g)$, $g(c) = f(c)$. If also the range of f is a set of functions then we write $f \downarrow S$ for the function g with $dom(g) = dom(f)$ such that $g(c) = f(c)\lceil S$ for every $c \in dom(g)$.

2.1 The HIOA Model

A hybrid I/O automaton captures the hybrid behavior of a system in terms of discrete transitions and continuous evolution of its state variables. Let V be the set of variables of automaton $\mathcal{A}$. Each $v \in V$ is associated with a *(static) type* defining the set of values v can assume. A *valuation* $\mathbf{v}$ for V is a function that associates each variable $v \in V$ to a value in $type(v)$. A trajectory τ of V is defined as a mapping $\tau : J \rightarrow val(V)$ where J is a left closed interval of time. If J is right closed then τ is said to be *closed* and its *limit time* is the supremum of the domain of τ, also written as $\tau.ltime$. Each variable $v \in V$ is also associated with a *dynamic type* (or *dtype*) which is the set of trajectories that v may follow.

A hybrid I/O automaton $\mathcal{A}$ consists of : (1) a set V of variables, partitioned into *internal* X, *input* U, and *output variables* Y. The internal variables are also called state variables. $Z \triangleq X \cup Y$ is the set of *locally controlled or local variables.* (2) a set A of actions , partitioned into *internal* H, *input* I, and *output actions* O. (3) a set of states $Q \subseteq val(X)$, (4) a non-empty set of *start states* $\Theta \subseteq Q$, (5) a set of *discrete transitions* $\mathcal{D} \subseteq Q \times A \times Q$. A transition $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$ is written in short as $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$. (6) a set of *trajectories* $\mathcal{T}$ for V, such that for every trajectory τ in $\mathcal{T}$, and for every $t \in \tau.dom$, $\tau(t).X \in Q$. It is required that $\mathcal{T}$ is closed under prefix, suffix, and concatenation. The first state $\tau(0).X$ of trajectory is denoted by $\tau.fstate$. If $\tau.dom$ is finite then $\tau.lstate = \tau(\tau.ltime).X$.

In addition, a hybrid I/O automaton also satisfies: (1) the input action enabling property, which prevents it from blocking any input action and (2) the input trajectory enabling property, which ensures that it is able to accept any trajectory of the input variables either by allowing time to progress for the entire length of the trajectory or by reacting with some internal action before that.

An *execution* of $\mathcal{A}$ is a finite or infinite sequence of actions and trajectories $\zeta = \tau_0, a_1, \tau_1, a_2 \ldots$,where (1) each $\tau_i \in \mathcal{T}$, (2) $\tau_0.fstate \in \Theta$ and (3) if τ_i is not the last trajectory in ζ then τ_i is finite and $\tau_i.lstate \stackrel{a_{i+1}}{\rightarrow} \tau_{i+1}.fstate$. An execution is *closed* if the sequence is finite and the domain of the final trajectory is a finite closed interval. The length of an execution is the number of elements (actions and trajectories) in the sequence.

An invariant $\mathcal{I}$ of $\mathcal{A}$ is either derived from other invariants or proved by induction on the length of a closed execution of $\mathcal{A}$. The induction consists of a base case, and an induction step. The base case tests that $\mathcal{I}(s)$ is satisfied for all $s \in \Theta$. The induction step consists of : (1) A discrete part—which tests that for every discrete step $s \stackrel{\pi}{\rightarrow} s' \in \mathcal{D}$, $\mathcal{I}(s)$ implies $\mathcal{I}(s')$. (2) A continuous part—which tests that for any closed trajectory $\tau \in \mathcal{T}$, with $\tau.fstate = s$ and $\tau.lstate = s'$, $\mathcal{I}(s)$ implies $\mathcal{I}(s')$. We shall use s and s' to denote the pre and the post states of discrete transitions, and also the $fstate$ and the $lstate$ of closed trajectories, as will be clear from the context.

2.2 New Addition to HIOA Structure: *Activities*

In the earlier works [8, 19, 11] using the HIOA model, trajectories of automata were specified using an ad hoc mixture of integral, algebraic equations and English. This form of specification cannot be analyzed easily, and it does not enforce a consistent style in writing specifications. The specification language [15] used in this paper uses "state space" representation [9] of the trajectories. To make this representation work, the following extra structure has been introduced into the basic HIOA model of [12].

The time domain is assumed to be the set of reals R. A variable v is *discrete* if its dynamic type is the pasting closure of the set of constant functions from left closed intervals of time to $type(v)$. A variable is *continuous* if its dynamic type is the pasting closure of the set of continuous functions from left closed intervals of time to R. For any set S of variables, S_d and S_a refer to the discrete and continuous subsets of S respectively.

Let e be a real valued algebraic expression involving the variables in $X \cup U$. For a given trajectory τ, $\tau.e$ denotes the function with domain $\tau.dom$ that gives the value of the expression e at all times over τ. Given that v is a local continuous variable, a trajectory τ *satisfies* the algebraic equation $v = e$, if for every $t \in \tau.dom$, $(\tau \downarrow v)(t) = \tau.e(t)$. If an algebraic equation involves a nondeterministic choice such as $v \in [e_1, e_2]$, then τ satisfies the equation if for every $t \in \tau.dom$, $(\tau \downarrow v)(t) \in [\tau.e_1(t), \tau.e_2(t)]$. If the expression e is integrable when viewed as a function, then τ satisfies the differential equation $\dot{v} = e$, if for every $t \in \tau.dom$, $(\tau \downarrow v)(t) = (\tau \downarrow v)(0) + \int_0^t \tau.e(t')\ dt'$.

A *state model* of HIOA $\mathcal{A}$ consists of $|Z_a|$ number of independent algebraic and/or differential equations with exactly one equation having v or $d(v)$ as its left hand side. The right hand sides of the equations are algebraic expressions involving the variables in $X \cup U$. A state model specifies[3] the evolution of every variable v in Z_a from some initial valuation. A trajectory τ satisfies a state model E if at all times over τ, all the variables in Z_a satisfy the differential and algebraic equations in E with $\tau(0)$ defining the initial valuations.

An *activity* α of HIOA $\mathcal{A}$ consists of three components: (1) an *operating condition* $P \subseteq Q$, (2) a *stopping condition* $P^+ \subseteq Q$, and (3) a state model E. The set of trajectories defined by activity α is denoted by $[\alpha]$. A trajectory τ belongs to the set $[\alpha]$ if the following conditions hold:

1. τ satisfies the state model E.
2. For all $t \in \tau.dom$, $(\tau \downarrow X)(t) \in P$.
3. If $(\tau \downarrow X)(t) \in P^+$ for $t \in dom(\tau)$ then τ is closed and $t = \tau.ltime$.

We impose the following restrictions on hybrid I/O automata model in order to specify the trajectories of an automaton as the union of the sets of trajectories specified by its activities.

R1 Every variable is either discrete or continuous.
R2 Discrete variables are constant over trajectories, i.e.,
$\forall \tau \in \mathcal{T}, \tau.lval \lceil Z_d = \tau.fval \lceil Z_d$.
R3 Operating conditions are disjoint,i.e., $P_i \cap \mathcal{P}_j = \emptyset$ if $i \neq j$.

It is proved in [16] that a set of trajectories specified by a set of activities, satisfy the prefix, suffix, and concatenation closure properties.

2.3 Language Constructs

In the HIOA specification language variables are declared by their names and types. Varibales declared with the **analog** keyword are continuous, else they are discrete. Actions are declared by their names, types, and optional list of parameters. Algebraic expressions are written using the operators $+, -, *$, and $\backslash$. A nondeterministic assignment, such as $v \in [e_1, e_2]$, is written as $v :=$ **choose**$[e_1, e_2]$. The derivative of a continuous variable x is written as $d(x)$. The discrete transitions are written in the precondition—effect style of the IOA language [5]. An activity $\alpha : (P, P^+, E)$ is written as:

activity α **when** P **evolve** E **stop at** P^+.

For automata with a single activity, if either P or P^+ are not specified, then they are assumed to be equal to Q and $\emptyset$ respectively.

[3] By *specifies* we mean restricts rather than uniquely determines. Due to possible nondeterminism in the state model, unique determination might not be possible.

3 Specification of System Components

This section describes the HIOA models for the components of the helicopter system, except for the supervisory controller, which is in Section 4. Discrete and continuous communication among the components are shown in Figure 3. We consider the pitch dynamics of the helicopter, which are critical for safety. A complete dynamical model of the helicopter with three degrees of rotational freedom can be found in [18]. In practice the roll and yaw effects are eliminated by making the initial conditions along these axes to be zero and giving identical input to the two rotors. The pitch dynamics is described by $\ddot{\theta} + \Omega^2 \cos\theta = U(t)$, where Ω is the characteristic frequency of the system and U is the net input for the pitch axis ranging over U_{min} and U_{max}.

---

type RAD $=$ Real **suchthat** $(i :$ RAD, $|i| \leq \Theta)$ % Θ max abs val for angles
type RADPS $=$ Real **suchthat** $(i :$ RADPS, $|i| \leq \dot{\Theta})$ % $\dot{\Theta}$ max abs val for ang velocity
type UTYPE $=$ Real **suchthat** $(i :$ UTYPE $|\ U_{min} \leq i \leq U_{max})$
hybridautomaton Plant(Ω : Real)
variables
  input analog U : UTYPE,
  internal analog θ_p^0 : RAD, θ_p^1 : RADPS, *initially* $(\theta_p^0, \theta_p^1) \in \mathbf{U}$,
  output analog θ_e^0 : RAD, θ_e^1 : RADPS

trajectories
  activity *pitch_dynamics*
    evolve $d(\theta_p^0) = \theta_p^1$;
      $d(\theta_p^1) = -\Omega^2 \cos\theta_p^0 + U$;
      $\theta_e^0 = \theta_p^0$; $\theta_e^1 = \theta_p^1$

Fig. 2. HIOA specification of the plant

The Plant automaton (Figure 2) specifies the evolution of the pitch angle θ_p^0 and the velocity θ_p^1 with U as input. The global types RAD, RADPS, and UTYPE define the domains for variables representing angle, angular velocity and actuator output respectively. The state variables θ_p^0 and θ_p^1 are initialized to some value from the set $\mathbf{U}$, which is defined in equation (4). A Plant state s is *safe* if the pitch angle $s.\theta_p^0$ is within θ_{min} and θ_{max}, which are the lower and the upper safety bounds corresponding to the helicopter hitting the ground and a fragile mechanical stop. The set of safe states is defined as:

$$\mathbf{S} \triangleq \{s \mid \theta_{min} \leq s.\theta_p^0 \leq \theta_{max}\}. \tag{1}$$

The Sensor automaton (Figure 4) periodically conveys the state of Plant to the controllers as observed by the physical sensors. It is parameterized by the sampling period Δ, the sensor errors for pitch angle ϵ_0, and velocity ϵ_1. The variable *now* serves as a clock. The stopping condition of the *read* activity ensures that a *sample* action occurs after every Δ interval of time. The value of θ_d^0 (and θ_d^1) is nondeterministically chosen to be within $\pm\epsilon_0$ of θ_a^0 ($\pm\epsilon_1$ of θ_a^1 resp.) to model the noise or the uncertainties in the sensing devices.

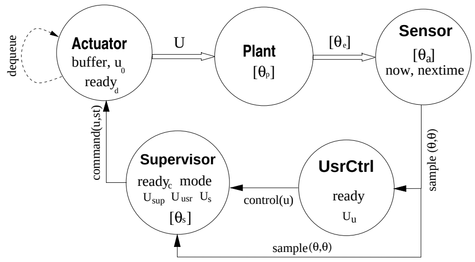

Fig. 3. Components of Helicopter system. Continuous and discrete communication among components are shown by wide and thin arrows respectively. The internal variables are marked inside the circles and internal actions are shown by a dashed self loop.

---

hybridautomaton `Sensor`(ϵ_0,ϵ_1, Δ : `Real`)

actions

  output *sample* (θ_d^0 : `RAD` , θ_d^1: `RADPS`)

discrete transitions

  output *sample* (θ_d^0 , θ_d^1)

  pre *now* = *next_time* $\wedge$

   $\theta_d^0 \in [\theta_a^0 - \epsilon_0, \theta_a^0 + \epsilon_0] \wedge$

   $\theta_d^1 \in [\theta_a^1 - \epsilon_1, \theta_a^1 + \epsilon_1]$

  eff *next_time* := *now* + Δ

trajectories

  activity *read*

   evolve d(*now*) =*1*; $\theta_a^0 = \theta_e^0$; $\theta_a^1 = \theta_e^1$;

   stop at *now* = *next_time*

variables

  input analog θ_e^0 : `RAD`, θ_e^1 : `RADPS`,

  internal analog θ_a^0 : `RAD` := 0, θ_a^1 : `RADPS` := 0,

   now: `Real` := 0,

  internal *next_time* : `Real` := Δ

  derived variable

   $s.time\_left \triangleq s.next\_time - s.now.$

Fig. 4. HIOA specification of the sensor and A/D conversion circuit

The `UsrCtrl` automaton (Figure 5) models an arbitrary user controller. It reads the *sample* action and triggers an output *control(u_d)* action, which communicates the user's output U_u to the supervisor. The output U_u is modeled as a nondeterministic choice over the entire range U_{min} to U_{max}. This captures our assumption that the user is capable of issuing arbitrarily bad control inputs to the plant. The design of a safe supervisor for `UsrCtrl` ensures that the system would be safe for *any* user controller because every controller must implement this specification of `UsrCtrl`.

The `Actuator` automaton (Figure 6) models the actuator and the D/A converter. The delay in the actuator response is modeled by a FIFO *buffer* of (u, st) pairs, where u is a command issued from `Supervisor`, and the scheduled time *st* is

hybridautomaton `UsrCtrl`

actions
  input *sample* (θ_d^0 : `RAD` , θ_d^1 : `RADPS`),
  output *control* (u_d : `UTYPE`)

variables
  internal θ_u^0: `RAD` := 0 , θ_u^1 : `RADPS` := 0,
    U_u : `UTYPE` := 0,
    ready : `Bool` := false

discrete transitions
  input *sample* (θ_d^0 , θ_d^1)
  eff $\theta_u^0 := \theta_d^0$; $\theta_u^1 = \theta_d^1$
    U_u := **choose** [U_{min}, U_{max}];
    ready := true

  output *control* (u_d)
  pre $(u_d = U_u) \wedge ready$
  eff *ready* := false

trajectories
  activity *void*
    evolve stop at *ready*

Fig. 5. Specification of User's Controller

the time at which u is to be delivered to the plant. A *command(u, m)* action appends $(u, timer + \tau_{act})$ to *buffer* and a *dequeue* action copies *buffer*`.head.u` to u_o and removes *buffer*`.head`. The following invariant for `Actuator` can be proved

type `MODES` = { `usr`, `sup` }
hybridautomaton `Actuator`(τ_{act})

actions
  input *command* (u : `UTYPE`)
  internal *dequeue*

variables
  internal u_o : `UTYPE` := 0, $ready_d$: `Bool` := false,
    buffer : `seq of` (`u:UTYPE`, `st:Real`, `m:MODE`) := {}
  output analog U : `UTYPE` := 0,
  input analog *now* : `Real`

discrete transitions
  input *command* (u)
  eff *buffer* + := $(u, now + \tau_{\text{act}})$;
    $ready_d$:= true

  internal *dequeue*
  pre *buffer*`.head.st` = *now* $\wedge$ $ready_d$
  eff u_o := *buffer*`.head.v`;
    buffer := *buffer*`.tail`;
    $ready_d$:= false

trajectories
  activity *d2a*
  evolve $U = u_o$
  stop at *buffer*`.head.st` = *now*

Fig. 6. Actuator and D/A conversion

by a simple induction.

Invariant 1 *In any reachable state s of* `Actuator`*,*
for all $0 \leq i < s.buffer.\mathtt{size} - 1$,
$s.now \leq s.buffer$`[i].st` $\leq s.buffer$`[i+1].st` $\leq s.now + \tau_{\text{act}}$.

4 Supervisory Controller

The goal of the supervisory controller is to ensure safety of the plant while interfering as little as possible with the user controller. There are well known algorithms [4, 2, 10] for synthesizing controllers for linear hybrid systems. Our design of the supervisory controller, however, is based on finding a *safe operating region* **U**, from where , if the supervisor takes over control then it is guaranteed to restore the plant to a safe state. In order to satisfy the minimal interference requirement it is also desirable to make **U** as large as possible.

4.1 Switching Regions

Consider a region $\mathbf{C} \subseteq \mathbf{S}$, from which all the reachable states are contained in **S**, provided that the input U to the plant is *correct*. By correct we mean that the input to the plant is $U = U_{min}$ (or U_{max}) if the pitch angle θ_p^0 is in the danger of reaching θ_{min} (θ_{max} resp.). As there is a $\tau_{\mathtt{act}}$ delay in `Actuator` *buffer*, the supervisor cannot change U instantaneously, and therefore the region **C** is not a safe operating region. We define another region $\mathbf{R} \subseteq \mathbf{C}$ as the set of states from which all reachable states over a period of $\tau_{\mathtt{act}}$ are within **C**. Even **R** is not a safe operating region because the supervisor *cannot* observe the plant state accurately, and relies on the periodic updates from the inaccurate sensors. Finally, we define the safe operating region **U** as follows: An *observed* state s' is in **U** if starting from any actual plant state s corresponding to s', all the reachable states over a Δ interval of time are in **R**.

Switching back to the user controller from the supervisor is performed at the boundary of an *inner* region $\mathbf{I} \subseteq \mathbf{U}$. This asymmetry in switching prevents high frequency chattering between the user and the supervisory controllers.

The regions **C**, **R**, **U**, and **I** are defined as follows. $U_{mag} = U_{max} - U_{min}$.

$$\mathbf{C} \triangleq \{s \mid s.\theta_p^0 \in [\theta_{min}, \theta_{max}] \wedge s.\theta_p^1 \in [\Gamma^-(s.\theta_p^0, 0), \Gamma^+(s.\theta_p^0, 0)]\}, \tag{2}$$

$$\mathbf{R} \triangleq \{s \mid \theta_{min} \leq s.\theta_p^0 \leq \theta_{max} \wedge \Gamma^-(s.\theta_p^0, \tau_{\mathtt{act}}) \leq s.\theta_p^1 \leq \Gamma^+(s.\theta_p^0, \tau_{\mathtt{act}})\}, \tag{3}$$

$$\mathbf{U} \triangleq \{s \mid \theta_{min} + \epsilon_0 \leq s.\theta_s^0 \leq \theta_{max} - \epsilon_0 \ \wedge U^-(s.\theta_s^0) \leq s.\theta_s^1 \leq U^+(s.\theta_s^0)\}, \tag{4}$$

$$\mathbf{I} \triangleq \{s \mid \theta_{min} + \epsilon_0 \leq s.\theta_s^0 \leq \theta_{max} - \epsilon_0 \ \wedge I^-(s.\theta_s^0) \leq s.\theta_s^1 \leq I^+(s.\theta_s^0)\}. \tag{5}$$

$$\Gamma^+(\theta, \mathcal{T}) = -U_{mag}\mathcal{T} + \sqrt{2(\Omega^2 \cos\theta_{max} - U_{min})(\theta_{max} - \theta + \frac{1}{2}U_{mag}\mathcal{T}^2)}, \tag{6}$$

$$\Gamma^-(\theta, \mathcal{T}) = U_{mag}\mathcal{T} - \sqrt{2(U_{max} - \Omega^2)(\theta - \theta_{min} + \frac{1}{2}U_{mag}\mathcal{T}^2)}, \tag{7}$$

$$U^+(\theta) = -\epsilon_1 + \Gamma^+(\theta + \epsilon_0, \tau_{\mathtt{act}} + \Delta) \qquad U^-(\theta) \quad = +\epsilon_1 + \Gamma^-(\theta - \epsilon_0, \tau_{\mathtt{act}} + \Delta)$$

$$I^+(\theta) = -2\epsilon_1 + \Gamma^+(\theta + 2\epsilon_0, \tau_{\mathtt{act}} + \Delta) \qquad I^-(\theta) = +2\epsilon_1 + \Gamma^-(\theta - 2\epsilon_0, \tau_{\mathtt{act}} + \Delta).$$

From the above definitions the following properties are derived.

Property 1 *Over the interval* $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$ *the following hold:*

1. $\Gamma^+(\theta, \mathcal{T})$ *and* $\Gamma^-(\theta, \mathcal{T})$ *are monotonically decreasing with respect to* θ.
2. $\Gamma^+(\theta, \mathcal{T})$ *is monotonically decreasing with respect to* $\mathcal{T}$. ($\mathcal{T} \geq 0$).
3. $\Gamma^-(\theta, \mathcal{T})$ *is monotonically increasing with respect to* $\mathcal{T}$. ($\mathcal{T} \geq 0$).
4. $\Gamma^+(\theta_{max}, \mathcal{T}) < 0$ *and* $\Gamma^-(\theta_{min}, \mathcal{T}) > 0$ *for* $\mathcal{T} > 0$.

Property 2 $\mathbf{I} \subseteq \mathbf{U} \subseteq \mathbf{R} \subseteq \mathbf{C} \subseteq \mathbf{S}$

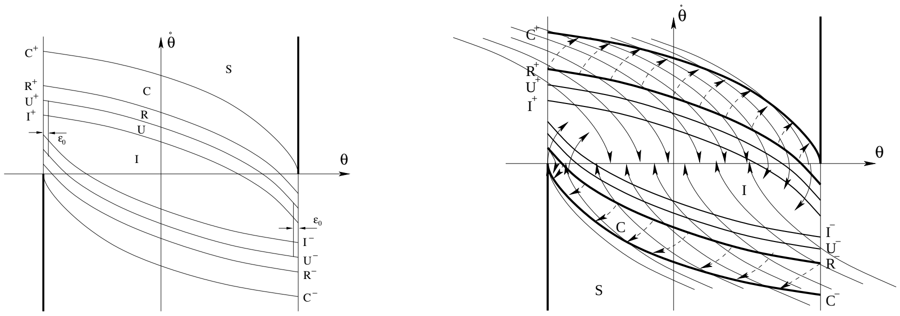

Fig. 7. (a) Regions in the statespace. (b) Trajectories in the settling (dashed lines) and recovery(solid lines) periods.

4.2 Supervisor Automaton

The `Supervisor` automaton (Figure 8) copies the observed plant state into internal variables θ_s^0 and θ_s^1 when the *sample* action occurs. Based on this state information the tentative output U_{sup} to the actuator is decided. When the *control* action occurs, the supervisor copies the user's command into another internal variable U_{usr}, and sets the values of U_s and *mode* for the next Δ interval based on (θ_s^0, θ_s^1) and the current value of *mode*. If *mode* is `usr` and the observed state is in $\mathbf{U}$ then *mode* remains unchanged and U_s is set to U_{usr}. If the present state is not in $\mathbf{U}$ then *mode* is changed to `sup` and the U_s is set to U_{sup}. If *mode* = `sup` then U_s is copied from U_{sup} and the mode changes only when (θ_s^0, θ_s^1) is in $\mathbf{I}$.

5 Analysis of Helicopter System

In this section we present the safety verification of the composed system. Let $\mathcal{A}$ denote the composition of the `Plant`, `Sensor`, `UsrCtrl`, `Actuator`, and the `Supervisor` automata. Safety is preserved if all the reachable states of $\mathcal{A}$ are contained within the region $\mathbf{S}$. It is assumed that: (1) $\theta_{min} < 0 < |\theta_{min}| < \theta_{max}$, (2) $U_{max} > \Omega^2$, $U_{min} \leq 0$, and (3) For any *sample* action $s \xrightarrow{\pi} s'$, if $s.\theta_s^1 > I^+(s.\theta_s^0)$ then, $s'.\theta_s^1 \geq I^-(s'.\theta_s^0)$, and if $s.\theta_s^1 < I^-(s.\theta_s^0)$ then, $s'.\theta_s^1 \leq I^+(s'.\theta_s^0)$. Assumptions (1) and (2) are derived from the dimensions of the physical system. Assumption (3) is a requirement which limits the speed of the plant and the sampling period so that it is not possible for the plant to jump across $\mathbf{I}$ without the sensors detecting it.

In the next section, first we present some preliminary properties of $\mathcal{A}$, then we state the invariants of $\mathcal{A}$, and prove some of the more important ones. The details of all the invariant proofs can be found in [16].

hybridautomaton `Supervisor`
actions
  input *sample* (θ_d^0: `RAD` θ_d^1: `RADPS`),
  input *control* (u_d : `UTYPE`),
  output *command* (u_d : `UTYPE`, m : `MODES`)

variables
  internal θ_s^0 : `RAD` $:= 0$, θ_s^1 : `RADPS` $:= 0$,
    U_{sup}, U_{usr}, U_s : `UTYPE` $:= 0$,
  internal $ready_c$: `Bool` := false, $mode$: `MODES` := usr
  internal analog rt : `Real` $:= 0$;

discrete transitions
  input *sample* (θ_d^0, θ_d^1)
  eff $\theta_s^0 := \theta_d^0$; $\theta_s^1 := \theta_d^1$;
    if $\theta_s^1 \geq I^+(\theta_s^0)$ **then** $U_{sup} := U_{min}$
    elseif $\theta_s^1 \leq I^-(\theta_s^0)$ **then** $U_{sup} := U_{max}$ **fi**

  output *command* (u_d, m)
  pre $ready_c \wedge (u_d = U_s) \wedge m = mode$
  eff $ready_c :=$ false

  input *control* (u_d)
  eff $U_{usr} := u_d$; $ready_c :=$ true
    if $mode =$ usr **then**
      if $(\theta_s^0, \theta_s^1) \in \mathbf{U}$ **then** $U_s := U_{usr}$
      else $U_s := U_{sup}$; $mode :=$ sup **fi**
    elseif $mode =$ sup **then**
      if $(\theta_s^0, \theta_s^1) \in \mathbf{I}$ **then** $U_s := U_{usr}$; $mode :=$ usr
      else $U_s := U_{sup}$ **fi fi**

trajectories
  activity *supervisor*
    when $mode =$ sup
    evolve $d(rt) = 1$ **stop at** $ready_c$

  activity *user*
    when $mode =$ usr
    evolve $rt = 0$ **stop at** $ready_c$

Fig. 8. HIOA specification of supervisor automaton

5.1 Some Preliminary Properties of A

The specification of the components of $\mathcal{A}$ satisfy restrictions **R2**, **R2** and **R3** and the plant state variables θ_p^0 and θ_p^1 are not modified by any discrete action. The next two properties follow from these facts:

Property 3 *Discrete variables of $\mathcal{A}$ are unaltered over all closed trajectories.*

Property 4 *For any discrete step $s \xrightarrow{\pi} s'$ of $\mathcal{A}$, $s'.\theta_p^0 = s.\theta_p^0$ and $s'.\theta_p^1 = s.\theta_p^1$.*

Invariant 2 follows from the code by a straightforward induction. Lemma 1 follows from Invariant 2 and indicates the times at which the different actions of $\mathcal{A}$ occur. Invariant 3 limits the size of the *buffer* and it is a consequence of Invariant 1 and Lemma 1.

Invariant 2 *In every reachable state s of $\mathcal{A}$, $0 \leq s.time\_left \leq \Delta$.*

Lemma 1 *In any execution of $\mathcal{A}$, sample, control, and command actions occur when $now = n\Delta$, and dequeue actions occur when $timer = \tau_{\mathsf{act}} + n\Delta$ for some integer $n > 0$.*

Invariant 3 *In every reachable state s, for all $0 \leq i < s.buffer.\mathtt{size} - 1$, $s.buffer$`[i+1].st`$= s.buffer$`[i].st` $+ \Delta$, and $s.buffer.\mathtt{size} \leq \lceil \frac{\tau_{\mathsf{act}}}{\Delta} \rceil$.*

5.2 User Mode

In this section we prove that $\mathcal{A}$ is safe in the user mode. We define a set of regions $\mathbf{A}_t$ for $0 \le t \le \Delta$, and Lemma 2 states the properties of the $\mathbf{A}_t$ regions. $\mathbf{A}_t \triangleq \{s \mid s.\theta_p^0 \in [\theta_{min}, \theta_{max}] \wedge s.\theta_p^1 \in [\Gamma^-(s.\theta_p^0, \tau_{\mathtt{eff}} + t), \Gamma^+(s.\theta_p^0, \tau_{\mathtt{eff}} + t)\}$.

Lemma 2 *The regions $\mathbf{A}_t$ satisfy:* 1. $\mathbf{A}_0 = \mathbf{R}$, 2. $\mathbf{U} \subseteq \mathbf{A}_\Delta$, *and* 3. *If $0 \le t \le t' \le \Delta$ then $A_{t'} \subseteq A_t$.*

The next lemma bounds the reachable sates over a singe trajectory and is used to prove safety when a tarjectory starts from the safe operating region $\mathbf{U}$. Invariant 4 makes use of Lemma 3. The safety of the system in the user mode is established by Invariant 5.

Lemma 3 *For any closed trajectory τ of $\mathcal{A}$, if $\tau.fstate \in \mathbf{A}_t$ then $\tau.lstate \in \mathbf{A}_{t-ltime(\tau)}$.*

Proof: Consider a closed trajectory τ. Assume that $s \in \mathbf{A}_t$. From the definition of $\mathbf{A}_t$ it follows that, $\theta_{min} \le s.\theta_p^0 \le \theta_{max}$ and $\Gamma^-(s.\theta_p^0, \tau_{\mathtt{eff}}+t) \le s.\theta_p^1 \le \Gamma^+(s.\theta_p^0, \tau_{\mathtt{eff}}+t)$. We conservatively estimate s' by considering the maximum and the minimum input U to `Plant`. First considering the maximum positive input, $U = U_{max}$, from the state model of `Plant` we get the upper bound on the acceleration at any state s'' in $\tau : d(s''.\theta_p^1) \le -\Omega^2 \cos\theta_{max} + U_{max}$. Integrating from t to t', it follows that $s'.\theta_p^1 \le (U_{max} - \Omega^2 \cos\theta_{max})t' + s.\theta_p^1$, and $s'.\theta_p^0 \le \frac{1}{2}(U_{max} - \Omega^2 \cos\theta_{max})t'^2 + s.\theta_p^1 t' + s.\theta_p^0$. Simplifying and using the definition of Γ^+ we get the following bounds on $s'.\theta_p^0$ and $s'.\theta_p^1$: $s'.\theta_p^0 \le \theta_{max}$, and $s'.\theta_p^1 \le \Gamma^+(s'.\theta_p^0, \tau_{\mathtt{eff}} + t - t')$. Similarly considering maximum negative output, $U = U_{min}$, we get the lower bounds on $s'.\theta_s^0$ and $s'.\theta_s^1$. $s'.\theta_p^0 \ge \theta_{min}$, and $s'.\theta_p^1 \ge \Gamma^-(s'.\theta_p^0, \tau_{\mathtt{eff}}+t-t')$. Combining equations all the above bounds on s' it follows that $s' \in \mathbf{A}_{t-t'}$. □

Invariant 4 *In any reachable state s, if $s.mode =$ `usr` and $\neg s.ready$ then $s \in \mathbf{A}_{s.time\_left}$.*

Invariant 5 *In any reachable state s, if $s.mode =$ `usr` then $s \in \mathbf{R}$.*

Proof: The base case holds because all initial states are in $\mathbf{U}$ and $\mathbf{U} \subseteq \mathbf{R}$. Consider any discrete transition $s \xrightarrow{\pi} s'$, with $s'.mode =$ `usr`. We split the proof into two cases: If $\neg s'.ready$, then using Invariant 4, $s' \in \mathbf{A}_{s'.time\_left} \subseteq \mathbf{R}$. On the other hand, if $s'.ready$, then $\pi \ne control$, and $s.mode =$ `usr` since only the *control* action can change *mode*. So from the inductive hypothesis $s \in \mathbf{R}$. It follows that $s' \in \mathbf{R}$ from the Property 4.

For the continuous part consider a closed trajectory τ with $\tau.fstate = s$, $\tau.lstate = s'$, and $s'.mode =$ `usr`. Once again there are two cases, if $\neg s'.ready$ then $s' \in \mathbf{R}$ by Invariant 4. Else if $s'.ready$, then $s.ready$ and $s.mode =$ `usr` because *ready* and *mode* does not change over the trajectories. Since s satisfies the stopping condition for activity *void* in `UsrCtrl`, therefore τ is a point trajectory, that is, $s' = s$. From the inductive hypothesis, $s \in \mathbf{R}$. Therefore $s' \in \mathbf{R}$. □

5.3 Supervisor Mode : Settling Phase

For proving safety in the supervisor mode, we first state some of the simple invariants. Invariant 6 states that, in all reachable with *ready* set to false, if the sensed plant state is within I^+ and I^-, then the system is in the user mode. Invariant 7 follows from the code of the *sample* action. And Invariant 8 is proved by a simple induction.

Invariant 6 *In any reachable state s,*
$I^-(s.\theta_s^0) \le s.\theta_s^1 \le I^+(s.\theta_s^0) \ \wedge\ \neg s.ready \Rightarrow s.mode = \mathtt{usr}.$

Invariant 7 *In any reachable state s,*
if $s.\theta_s^1 > I^+(s.\theta_s^0)$ *then* $s.U_{sup} = U_{min}$*, and*
if $s.\theta_s^1 < I^+(s.\theta_s^0)$ *then* $s.U_{sup} = U_{max}$.

Invariant 8 *In any reachable state s,*
$s.rt = n\Delta - s.time\_left$*, for some integer* $n \ge 1$.

We define two predicates $\mathcal{Q}_k^+$ and $\mathcal{Q}_k^-$ that capture the progress made by the system while the actuator delays the delivery of commands issued by the supervisor. A state s satisfies $\mathcal{Q}_k^+$ (or $\mathcal{Q}_k^-$), if the last k commands in $s.buffer$ are equal to U_{min} (or U_{max} respectively). More formally, for any $k \ge 0$,

$$\mathcal{Q}_k^+(s) \triangleq \forall i,\ max(0, s.buffer.\mathtt{size} - k) \le i < s.buffer.\mathtt{size},\ s.buffer[i].\mathtt{u} = U_{min},$$
$$\mathcal{Q}_k^-(s) \triangleq \forall i,\ max(0, s.buffer.\mathtt{size} - k) \le i < s.buffer.\mathtt{size},\ s.buffer[i].\mathtt{u} = U_{max}.$$

Clearly, for all $k > 0$, $\mathcal{Q}_k^+(s)$ implies $\mathcal{Q}_{k-1}^+(s)$, and therefore for any $k \ge s.buffer.\mathtt{size}$, $\mathcal{Q}_k^+(s)$ implies that $\mathcal{Q}_j^+(s)$ holds for all $j < s.buffer.\mathtt{size}$. Similar results hold for $\mathcal{Q}_k^-$. The next invariant states that every reachable state s in the supervisor mode, satisfies either $\mathcal{Q}^+_{\lceil \frac{s.rt}{\Delta} \rceil}(s)$ or $\mathcal{Q}^-_{\lceil \frac{s.rt}{\Delta} \rceil}(s)$, depending on whether s is above I^+ or below I^- respectively. In addition if $s.ready_d$ is true, that is, s is in between a *command* action and a *dequeue* action, then $\mathcal{Q}^+_{\lceil \frac{s.rt}{\Delta} \rceil+1}(s)$ or $\mathcal{Q}^-_{\lceil \frac{s.rt}{\Delta} \rceil+1}(s)$ holds, depending on the location of s with respect to I^+ and I^-.

Invariant 9 *In any reachable state s, such that* $s.mode = \mathtt{sup}$*:*
If $s.\theta_s^1 > I^+(s.\theta_s^0)$ *then (a)* $\mathcal{Q}^+_{\lceil \frac{s.rt}{\Delta} \rceil}(s)$*, (b) If* $ready_d$ *then* $\mathcal{Q}^+_{\lceil \frac{s.rt}{\Delta} \rceil+1}(s)$,
If $s.\theta_s^1 < I^-(s.\theta_s^0)$ *then (a)* $\mathcal{Q}^-_{\lceil \frac{s.rt}{\Delta} \rceil}(s)$*, (b) If* $ready_d$ *then* $\mathcal{Q}^-_{\lceil \frac{s.rt}{\Delta} \rceil+1}(s)$

The next invariant formalizes the notion that after a certain τ_{act} period of time in the supervisor mode the input to the plant is correct.

Invariant 10 *In any reachable state s, such that* $s.mode = \mathtt{sup}$ *and* $s.rt > \tau_{\mathsf{act}}$
1. If $s.\theta_s^1 > I^+(s.\theta_s^0)$ *then* $s.U = U_{min}$*, and 2. If* $s.\theta_s^1 < I^-(s.\theta_s^0)$ *then* $s.U = U_{max}$,

We split the execution of $\mathcal{A}$ in the supervisor mode (Figure 7(b)) into (a) a *settling phase* of length τ_{act} in which the input U to the plant is arbitrary, and (b) a variable length *recovery phase* during which $rt > \tau_{\mathsf{act}}$ and the input to the plant is correct, that is, in accordance with Invariant 10.

Next we define a set of regions $\mathbf{B}_t$, for $0 \le t \le \tau_{\mathsf{act}}$, which are analogous to the $\mathbf{A}_t$ regions: $\mathbf{B}_t \triangleq \{s \mid s.\theta_p^0 \in [\theta_{min}, \theta_{max}] \ \wedge\ s.\theta_p^1 \in [\Gamma^-(s.\theta_p^0,\ \tau_{\mathsf{act}} - t), \Gamma^+(s.\theta_p^0,\ \tau_{\mathsf{act}} - t)]\}$.

Lemma 4 states the relationship between the $\mathbf{B}_t$ regions. Invariant 11 bounds the location of a state s in terms of the $\mathbf{B}_t$ regions, when $s.rt \leq \tau_{\mathtt{act}}$. This implies the safety of the system in the settling phase.

Lemma 4 *The regions* $\mathbf{B}_t$ *satisfy:* 1. $\mathbf{B}_0 = \mathbf{R}$, 2. $\mathbf{B}_{\tau_{\mathtt{act}}} = \mathbf{C}$, 3. *If* $0 \leq t \leq t' \leq \tau_{\mathtt{act}}$ *then* $\mathbf{B}_t \subseteq \mathbf{B}_{t'}$.

Invariant 11 *For any reachable state s,*
if $s.mode = \mathtt{sup} \wedge s.rt \leq \tau_{\mathtt{act}}$ *then* $s \in \mathbf{B}_{s.rt}$.

5.4 Supervisor Mode: Recovery Phase

Invariant 12 states that $\mathbf{C}$ is an invariant set for the system in the recovery phase. A sketch of the proof is given here, the complete proof can be found in [16].

Invariant 12 *In any reachable states s,*
if $s.mode = \mathtt{sup}$ *and* $s.rt \geq \tau_{\mathtt{act}}$ *then* $s \in \mathbf{C}$.

proof sketch: The base case is trivially satisfied. The discrete part of the induction is also straightforward, the *control* action alters the *mode*. If $s.mode = \mathtt{sup}$ then using the inductive hypothesis, $s' \in \mathbf{C}$. Otherwise $s.mode = \mathtt{usr}$ and $s'.rt = 0$ and the invariant holds vacuously. For all other discrete actions the invariant is preserved. For the continuous part of the induction, consider closed trajectory τ with $s'.mode = \mathtt{sup}$ and $s'.rt \geq \tau_{\mathtt{act}}$. We claim that $s \in \mathbf{C}$. From Property 3 it is known that $s.mode = \mathtt{sup}$, (1) If $s.rt < \tau_{\mathtt{act}}$ then from Invariant 11 it follows that $s \in \mathbf{C}$. Otherwise (2) $s.rt \geq \tau_{\mathtt{act}}$ and from the inductive hypothesis it follows that $s \in \mathbf{C}$. If $s \in \mathbf{U}$, then from Lemma 3 it follows that $s' \in \mathbf{R} \subseteq \mathbf{C}$. So it remains to show that if $s \in \mathbf{C} \setminus \mathbf{U}$ then $s' \in \mathbf{C}$. This is proved by contradiction, suppose $s' \notin \mathbf{C}$, then there must exist $t' \in \tau.dom$ such that τ leaves the $\mathbf{C}$ at $\tau(t')$. Then it must be the case that the trajectory τ and the outer-normal of boundary of $\mathbf{C}$ should form an acute angle. It is known from Lemma 10 that at any intermediate state $\tau(t')$, the input U to the plant is correct. A contradiction is reached by showing that if $\tau(t')$ is on the boundary of $\mathbf{C}$, then the angle between the above-mentioned vectors is obtuse. Finally, combining the Invariants proved above the safety property of the composed system can be proved.

Theorem 1 *All reachable states of* $\mathcal{A}$ *are contained in* $\mathbf{C}$.

Proof: For any reachable state s, if $s.mode = \mathtt{usr}$ then $s \in \mathbf{R} \subseteq \mathbf{C}$ by Invariant 5. Otherwise $s.mode = \mathtt{sup}$, and there are two possibilities: if $s.rt < \tau_{\mathtt{act}}$ then, by Invariant 11, $s \in \mathbf{B}_{s.rt} \subseteq \mathbf{C}$. Else $s.rt \geq \tau_{\mathtt{act}}$ and it follows from Invariant 12 that $s \in \mathbf{C}$.

6 Conclusions

In this paper we have presented an advanced application of the HIOA framework for verifying hybrid systems. The safety of the designed supervisory controller was established by proving a set of invariants. The proof techniques demonstrate two properties that we believe are important for reasoning about complex hybrid systems: (1) the proofs are decomposed into discrete and continuous parts, which are independent of each other, and (2) the reasoning style is purely assertional, that is, based on the current state of the system, rather than complete executions.

The design of the supervisory controller uses a safe operating region of the plant, beyond which the supervisor overrides the user controller, performs appropriate recovery, and returns control to the user. The duration of the recovery period has not been discussed here, but it has been shown in [18] to be bounded. The size of the safe operating region, depends on the plant dynamics, sensor errors, sampling period, actuator bandwidth, and saturation. An implementation of the supervisory controller in the actual system is in progress. In the future we intend to design and verify a class of supervisory controllers that reduce unnecessary interferences by utilizing additional information about particular user controllers.

The specification language used is based on the hybrid I/O automaton model of [12] with the addition of certain extra structures to specify the trajectories using activities. We intend to incorporate the language extensions into a toolkit for automatically checking HIOA programs. At present we are also working on building a theorem prover interface for HIOA that will partially automate the verification process.

References

1. Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, P.-H. Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
2. Eugene Asarin, Olivier Bournez, Thao Dang, Amir Pnueli, and Oded Maler. Effective synthesis of switching controllers for linear systems. In *Proceedings of IEEE*, volume 88, pages 1011–1025, July 2000.
3. Alexandre M. Bayen, Eva Cruck, and Claire Tomlin. Guaranteed overapproximations of unsafe sets for continuous and hybrid systems: solving the hamilton-jacobi equation using viability techniques. In *Hybrid Systems: Computation and Control 2002*, volume 2289 of *LNCS*, pages 90–104. Springer, March 2002.
4. Enrique D. Ferreira and Bruce H. Krogh. Switching controllers based on neural network: Estimates of stability regions and controller performance. In *Hybrid Systems: Computation and Control 1998*, pages 126–142, 1998.
5. Stephen Garland, Nancy Lynch, and Mandana Vaziri. IOA: A language for specifying, programming and validating distributed systems. Technical report, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, October 1999.

6. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. In *Computer Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–483, 1997.
7. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *ACM Symposium on Theory of Computing*, pages 373–382, 1995.
8. Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of TCAS. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99),Phoenix, Arizona*, pages 115–125, December 1999.
9. David G. Luenberger. *Introduction to Dynamic Systems: Theory, Models, and Applications*. John Wiley and Sons, Inc., New York, 1979.
10. John Lygeros, Claire Tomlin, and Shankar Sastry. Controllers for reachability specifications for hybrid systems. In *Automatica*, volume 35, March 1999.
11. Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*, pages 1–22, Salt Lake City, Utah, March 1996. World Scientific Publishing Company.
12. Nancy Lynch, Roberto Segala, and Frits Vaandraager. Hybrid I/O automata. To appear in *Information and Computation*. Also, Technical Report MIT-LCS-TR-827d, MIT Laboratory for Computer Science Technical Report, Cambridge, MA 02139, January 13, 2003.
`theory.lcs.mit.edu/tds/papers/Lynch/HIOA-final.ps`.
13. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In T. Henzinger R. Alur and E. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, New Brunswick, New Jersey, October 1995. Springer-Verlag.
14. Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata revisited. In M.D. Di Benedetto and A.L. Sangiovanni-Vincentelli, editors, *Proceedings Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01),* Rome, Italy, volume 2034 of *lncs*. springer, March 2001.
15. Sayan Mitra. Language for Hybrid Input/Output Automata, 2002. Work in progress. http://theory.lcs.mit.edu/mitras/research/composing_activities.ps.
16. Sayan Mitra, Yong Wang, Nancy Lynch, and Eric Feron. Application of hybrid I/O automata in safety verification of pitch controller for model helicopter system. Technical Report MIT-LCS-TR-880, MIT Laboratory for Computer Science, Cambridge, MA 02139, January 2003. http://theory.lcs.mit.edu/~mitras/research/QuanTR02.ps
17. URL:. http://www.quanser.com/english/html/products/fs_product_challenge.asp?, lang_code=english&pcat_code=exp-spe&prod_code=S1-3dofheli.
18. Yong Wang, Masha Ishutkina, Sayan Mitra, Nancy A. Lynch, Eric Feron. Design of Supervisory Safety Control for 3DOF Helicopter using Hybrid I/O Automata, 2002. pre-print http://gewurtz.mit.edu/ishut/darpa_sec_mit/papers/quanser.ps.
19. H. B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. In T. Henzinger R. Alur and E. Sontag, editors, *Hybrid Systems III: Verification and Control (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems)*, volume 1066 of *Lecture Notes in Computer Science*, pages 101–113, New Brunswick, New Jersey, October 1995. Springer-Verlag.