Bounded Verification of Adaptive Flight Control Systems

Ashish Tiwari[*]

SRI International, Menlo Park, CA 94025

Abstract. **We formally verify a direct model-reference adaptive control (MRAC) method that is used to enable flight control in adverse conditions. We use the *bounded verification* approach and verify the system by introducing templates for both the assumptions and the guarantees, and using the tool `QEPCAD` to solve the resulting exists-forall formula. We also present results from an analysis performed on indirect and hybrid adaptive flight controllers.**

Nomenclature

x = 3×1 state vector consisting of roll, pitch and yaw rates

r = commanded value of x

$\vec{x}_m$ = desired value of x calculated using a reference model

$\vec{x}_e$ = error, or the difference between the desired and actual x

$\vec{intx}_e$ = integral of the error

L = weights of the Neural Network

β = fixed basis or kernel functions of the Neural Network

f = Difference between the dynamics of the undamaged and damaged aircraft

$\vec{u}_{PI}$ = proportional integral (PI) gain

$\vec{u}_{ad}$ = adaptive gain generated using the Neural Network

I. Introduction

Adaptive control is receiving a significant amount of attention lately. In particular, the goal of the "Integrated Resilient Aircraft Control" (IRAC) research project, which is part of the Aviation Safety Program under the NASA Aeronautics Research Mission Directorate, is to advance the art in adaptive control to enable flight control resiliency in adverse conditions. NASA has been conducting a flight test of a neural net intelligent flight control system on board a modified F-15 test aircraft.

While adaptive control can provide increased robustness in the presence of faults/damage in the aircraft, it can also potentially excite unmodeled dynamics and lead to instability and crash (X-15 crash in 1967). Verification and validation of adaptive flight control systems is a major challenge. In this context, formal verification can provide guarantees that are impossible to achieve using simulations and testing alone.

Formal verification techniques face several technical challenges when applied to adaptive flight control systems. First, completely automated verification approaches are not scalable. Hence, automated tools can only verify small designs, and even models with five variables are sometimes beyond the capabilities of current tools. Second, existing automated verification techniques have very limited support for nonlinearity. However, nonlinearity is a common feature of adaptive systems. Third, models of adaptive flight control systems contain unknown parameters with unknown dynamics. Several existing verification systems are not well-suited for handling unknown parameters. Finally, errors are also introduced when designed controllers are implemented in software. Software verification is a challenging problem in itself.

[*] Senior Computer Scientist, Computer Science Laboratory, 333 Ravenswood Ave, Menlo Park, CA 94025. AIAA Member.

In this paper, we propose a new formal verification technique – namely, bounded verification. Bounded verification is ideally suited for verification of continuous and hybrid dynamical systems, especially at the design phase. Moreover, it potentially eliminates several of the drawbacks of the existing formal verification approaches. Bounded verification has an added advantage. It explicitly produces "certificates" or "witnesses" for any property that is verified. These "certificates" of proof can be used in two distinctly useful ways:
(1) Certificates can be used to independently check that the result of the formal verification tool
(2) Certificates can be used to help in the verification of the implementation of the verified controller.[6]
Alternatively, certificates can be used to synthesize *monitors* that can be used to perform runtime verification of the control software. We describe a case study of applying bounded verification approach to formally analyzing a direct model-reference adaptive flight control that is based on neural networks. This paper presents details of the verification steps and the results obtained via the analysis. We also present results from analyzing an indirect and a hybrid adaptive flight controller.

This paper makes the following contributions:

- Introduce bounded verification as a general formal verification approach; especially for continuous and hybrid dynamical systems
- Present details of how bounded verification can be used to verify safety, reachability and bounded stability
- Present a formal model of an adaptive flight control system and results of formally verifying it using the bounded verification approach
- Propose the use of certificates generated by bounded verification to monitor software implementation of adaptive controllers
- Present results from an analysis of the recursive least squares learning procedure used inside an indirect and hybrid adaptive flight controller

This paper is organized as follows. In Section II, we introduce bounded verification. In Section III, we present a neural network adaptive controller and present a formal model of it that is used for verification. In Section IV, we present preliminary analysis results obtained using the bounded verification approach on the formal model of the adaptive control system.

II. Bounded Verification

The verification problem is to determine if a given formal model of a system satisfies a given property. Most of the prevailing verification approaches can be broadly classified into two classes:

- reachability-based methods
- abstraction-based methods

Reachability-based methods iteratively compute an over-approximation of the reachable set of states. A crucial choice in these methods is the representation they use for sets of states – convex polytopes,[3] hyper-rectangles,[5] zonotopes,[7,8] ellipsoids,[13] and level sets.[14] In contrast to reachability-based methods, abstraction-based methods explicitly compute an abstraction of the hybrid system and then model check the abstraction.[1,4,21] In this paper, we explore a third paradigm for verification of systems, especially continuous and hybrid systems, called the constraint-based approach; see Figure 1.

In the constraint-based approach, the verification problem is reduced to a constraint solving problem and then efficient constraint solving techniques – both symbolic and numerical – are used for solving the constraints.[10,16,18,20,22,23]

A crucial observation that enables constraint-based approach is that verification is the same as searching for a proof. What makes verification intractable is that, in general, the space of proofs is huge and it is impractical to search over it. However, we can make the search tractable by bounding it. A bounded search for proof of a property can be encoded as a constraint. Thus, constraint-based methods essentially perform *bounded verification*.

Bounded verification is the dual of bounded model checking. Whereas bounded model checking searches for a bounded counter example (for safety), bounded verification searches for a bounded proof (for safety).

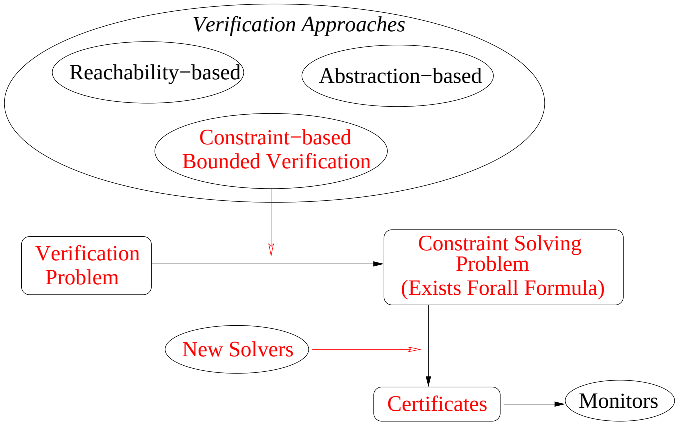


Figure 1. Overview of the technical approach: Formal verification techniques can be broadly classified into two categories: reachability-based and abstraction-based. In this paper, we consider a new class of techniques called constraint-based approaches. Bounded verification is an instance of this approach. Constraint-based verification approaches reduce the verification problem into a constraint-solving problem. A side-effect is that these approaches often produce a certificate for the verification performed. This certificate can be used to verify a refined implementation or monitor the final implementation.

Property	Verification *Witness/Certificate*
Stability	Lyapunov function
Safety	Inductive Invariant
Reachability	Ranking function

Table 1. Certificates produced during verification of different properties.

In both cases, the bounded search is encoded in the form of a constraint and the constraint is solved using modern constraint solvers, such as Satisfiability Modulo Theory (SMT) solvers.[19]

How to bound the search for proof of a property? The key observation used here is that proof of various important properties, such as safety, reachability and stability, is based on finding the appropriate witness or certificate; see Table 1. Safety can be proved by demonstrating the existence of an inductive invariant (also known as a barrier certificate).[16,20] Stability (and various variants of stability) can be proved by demonstrating the existence of (different variants of) a Lyapunov function.

In the bounded verification approach, verification is performed by doing a bounded search for a proof of the property – that is, a bounded search for the "right certificate".. Since the universe of possible certificates is often infinite, bounded verification achieves computability by limiting this search to certificates of a *specific form*. More specifically, bounded verification only searches for certificates that are instances of some fixed templates.

Continuous Dynamical Systems

Formal verification involves verifying a *model* of the system against a *property*. We use a continuous dynamical system to model the system.

Definition 1 (Continuous Dynamical System) *A* continuous dynamical system *$\texttt{CDS}$ is a tuple $(\texttt{X}, f)$ where $\texttt{X}$ is a finite set of variables interpreted over the reals $\mathbb{R}$, $\mathbf{X} = \mathbb{R}^{\texttt{X}}$ is the set of all valuations of the variables $\texttt{X}$, and $f : \mathbf{X} \mapsto \mathbf{X}$ is a vector field that specifies the continuous dynamics.*

Note that $\mathbb{R}^{\texttt{X}}$ is isomorphic to the n-dimensional real space $\mathbb{R}^n$ where $n = |\texttt{X}|$ is the number of variables in $\texttt{X}$. We assume that f is locally Lipschitz everywhere on the state space, which guarantees that the solution of the initial value problem $\frac{d\texttt{X}(t)}{dt} = f(\texttt{X}(t))$, $\texttt{X}(0) = \vec{\texttt{x}}_0$, always exists and is unique and will be denoted by $F(\vec{\texttt{x}}_0, t)$. Given a CDS $\texttt{CDS}$ and a set $\texttt{Init} \subseteq \mathbf{X}$ of initial states, the meaning of a continuous

$$\begin{array}{rrcl} (A1): & \texttt{Init} & \Rightarrow & V \geq 0 \\ (A2): & V > 0 & \Rightarrow & \frac{dV}{dt} < -\epsilon \\ (A3): & V \leq 0 & \Rightarrow & \phi \\ \hline & \texttt{Init} & \Rightarrow & \mathbb{F}(\phi) \end{array} \qquad \begin{array}{rrcl} (B1): & \psi \wedge \neg\phi & \Rightarrow & V > 0 \\ (B2): & \psi \wedge \neg\phi & \Rightarrow & \frac{dV}{dt} < -\epsilon \\ \hline & \mathbb{G}(\psi) & \Rightarrow & \mathbb{G}(\mathbb{F}(\phi)) \end{array}$$

Figure 2. On the left, an inference rule for verifying that a continuous system $\texttt{CDS} := (\texttt{X}, f)$ eventually reaches ϕ starting from any state in $\texttt{Init}$. On the right, an inference rule for verifying that a continuous system $\texttt{CDS} := (\texttt{X}, f)$ always eventually reaches ϕ under the assumption that always ψ holds.

dynamical system is simply the collection of all possible trajectories starting from an initial state. Formally, the semantics is the collection of all solutions $F(\vec{\mathtt{x}}_0, t)$ of the initial value problem $\frac{d\mathtt{X}(t)}{dt} = f(\mathtt{X}(t))$, $\mathtt{X}(0) = \vec{\mathtt{x}}_0$, where $\vec{\mathtt{x}}_0 \in \texttt{Init}$.

Stability Properties for CDS

Having defined the modeling formalism, we now move to the formalism for describing the *property.* In this paper, we are interested in proving *stability*-like properties of continuous dynamical systems; that is, starting from some initial states, does the system eventually (or always eventually) reach some desired set of states. We formally write such properties using temporal operators – eventually ($\mathbb{F}$) and always ($\mathbb{G}$).

Formally, a *property* is a set of states; in other words, we identify a property with the set of states in which it holds. The operators *eventually* ($\mathbb{F}$) and *always* ($\mathbb{G}$) are applied to a property to give a new (and a more complex temporal) property. Let ϕ denote some basic property; for example, ϕ could say that "the value of the variable y is zero". We use the notation $\mathbb{F}(\phi)$ to denote the property that holds at all those states starting from which the system can eventually reach a state in ϕ. We use the notation $\mathbb{G}(\phi)$ to denote the property that holds at all those states starting from which the system always within the set ϕ. Thus, if $\texttt{CDS} = (\texttt{X}, f)$ is a continuous dynamical systems and F denotes the trajectories of this $\texttt{CDS}$, and if $\phi \subseteq \mathbf{X}$ is a set of states, then $\mathbb{F}(\phi)$ and $\mathbb{G}(\phi)$ denote the following set of states:

$$\begin{array}{rcl} \mathbb{F}(\phi) & := & \{\vec{\mathtt{x}} \in \mathbf{X} \mid \text{for some } t \geq 0 : F(\vec{\mathtt{x}}, t) \in \phi\} \\ \mathbb{G}(\phi) & := & \{\vec{\mathtt{x}} \in \mathbf{X} \mid \text{for all } t \geq 0 : F(\vec{\mathtt{x}}, t) \in \phi\} \end{array}$$

For continuous systems, the following variants of the property that also considers "what happens in the limit" are also useful.

$$\begin{array}{rcl} \mathbb{F}^{cl}(\phi) & := & \{\vec{\mathtt{x}} \in \mathbf{X} \mid \texttt{Cl}(\{F(\vec{\mathtt{x}}, t) \mid t \geq 0\}) \cap \phi \neq \emptyset\} \\ \mathbb{G}^{cl}(\phi) & := & \{\vec{\mathtt{x}} \in \mathbf{X} \mid \texttt{Cl}(\{F(\vec{\mathtt{x}}, t) \mid t \geq 0\}) \subseteq \phi\} \end{array}$$

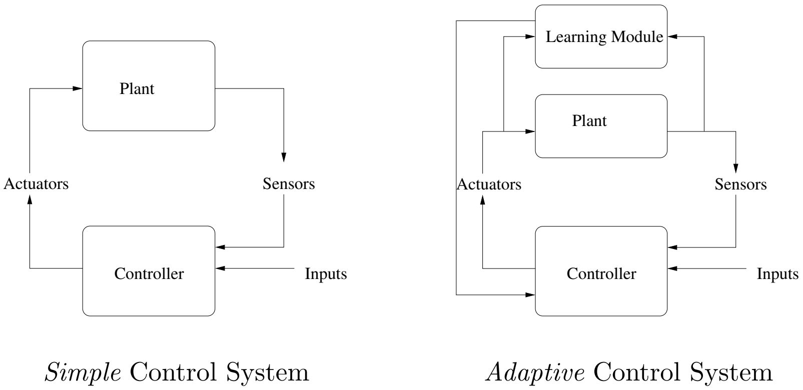
Here $\texttt{Cl}$ denotes the (topological) closure operator. In words, $\mathbb{F}^{cl}(\phi)$ denotes all states starting from which $\texttt{CDS}$ can eventually reach ϕ in either finite time or in the limit, and $\mathbb{G}^{cl}(\phi)$ denotes all states starting from which $\texttt{CDS}$ always stays within ϕ both in any finite time and in the limit.

Apart from the temporal operators, $\mathbb{F}$ and $\mathbb{G}$, we also use the usual Boolean connectives, such as implication $\Rightarrow$ and conjunction $\wedge$ to specify properties. Recall that we do not distinguish between a property (specified using a formula) and a set of states: a formula denotes all those states where it evaluates to *true*. Hence, logical implication $\Rightarrow$ is the same as the subset relation $\subseteq$ and logical conjunction $\wedge$ is the same as set intersection $\cap$. For example, if $\texttt{Init}$ is a set of (initial) states, and ϕ is a set of desired target states, then $\texttt{Init} \Rightarrow \mathbb{F}(\phi)$ means that all states in $\texttt{Init}$ eventually reach a state in ϕ. The reader can find more examples of properties that are stated using the temporal operators in Section IV.

Verification Approach

In the bounded verification approach, the verification task is reduced to a constraint solving task. This reduction depends on using sufficient conditions (given in the form of inference rules) for the property of interest.

Figure 2(left) presents an inference rule for verifying that all initial states eventually reach ϕ. The inference rule should be read as follows: if we can find a (Lyapunov-like) function $V : \mathbb{R}^n \mapsto \mathbb{R}$ and a positive



Simple Control System

Adaptive Control System

Figure 3. Simple and Adaptive Control System. Adaptive systems have an additional learning module that can be used by the controller.

constant $\epsilon > 0$ such that
(a) the value of V at any state in `Init` is non-negative, (Condition (A1)),
(b) the value of V decreases at a rate of atleast $-\epsilon$ whenever $V > 0$ (Condition (A2)), and
(c) at any state where V is non-positive, ϕ is true,
then we can conclude that every state in `Init` eventually reaches a state in ϕ. Note that the derivative of V with respect to time, $\frac{dV}{dt}$, is called the *Lie derivative*, $L_f(p)$, of p with respect to the vector field f. It can be computed using the chain rule as

$$L_f(p) \quad := \quad \sum_{x \in \mathtt{X}} \frac{\partial p}{\partial x}\frac{dx}{dt} \;:=\; \vec{\nabla} p \cdot f \;:=\; (\frac{\partial p}{\partial x_1}, \frac{\partial p}{\partial x_2}, \ldots) \cdot (\frac{dx_1}{dt}, \frac{dx_2}{dt}, \ldots) \tag{1}$$

Figure 2(right) presents an inference rule for verifying that a system *always eventually reaches* ϕ, that is, $\mathbb{G}(\mathbb{F}(\phi))$, under the assumption that the property ψ always holds. The proof for $\mathbb{G}(\psi) \Rightarrow \mathbb{G}(\mathbb{F}(\phi))$ is obtained by finding a (Lyapunov-like) function V and a positive constant $\epsilon > 0$ such that
(a) V is non-negative whenever ϕ is not true and ψ is true (Condition (B1)),
(b) V decreases (by rate atleast $-\epsilon$) whenever ϕ is not true and ψ is true (Condition (B2)).
These two conditions guarantee that, if ϕ does not hold, V decreases until it becomes less-than 0, at which point ϕ must necessarily hold.

The two inference rules given in Figure 2 for verifying `Init` $\Rightarrow \mathbb{F}(\phi)$ and $\mathbb{G}(\psi) \Rightarrow \mathbb{G}(\mathbb{F}(\phi))$ properties respectively, can be proved to be sound rules; that is, they provide sufficient checks for the inferred conclusions. However, before we can use them, we have to find the *witness* function V and the constant ϵ. In the bounded verification approach, we perform a bounded search for V. Specifically, we fix a template for V and search only for those V's that match the template and that also validate the conditions in the inference rule. Concrete application of this approach will be illustrated in Section IV.

Bounded verification has been used for safety verification by using templates for specifically searching for inductive invariants in the work of several authors.[12,16,18] More details on bounded verification, can be found in the work of Gulwani et al.[9] and Gulwani and Tiwari.[10]

III. Neural Net Direct Adaptive Control

A traditional *control system* consists of two components: a controller and a plant (see Figure 3). The controller receives as input the desired target state. The goal of the controller is to move the plant to this desired target state. It does so by sending a control signal to the plant. The controller can choose its control signal by taking into account the feedback it receives from the plant.

In an *adaptive control system*, there is an additional learning module that performs online learning based on the output of the plant and the control input generated by the controller. The controller can now also use the information generated by the learning module to generate the next control signal (see Figure 3).

Adaptive control methods can be broadly classified as direct, indirect, or hybrid. Indirect methods identify the unknown plant parameters and generate a control scheme using these parameter estimates. In contrast, direct methods directly adjust the control signal to account for plant uncertainties without explicitly identifying the plant parameters. Neural networks, and recursive least squares respectively, have been used to play the role of the learning module in direct, and indirect respectively, adaptive control methods. In particular, Rysdyk and Calise[17] described a neural net direct adaptive control method. This has subsequently formed the basis for the intelligent flight control system in the NASA F-15 test aircraft.[11]

A. Direct Neural Net Adaptive Flight Control

In this section, we describe the model of a direct neural net adaptive flight control that was formally verified. For further details on the model and its design principle, the reader is referred to the paper by Nguyen, Krishnakumar and Boskovic[15][a].

The high-level architecture of a direct adaptive controller is shown in Figure 4. Let $\vec{x}$ be the state vector consisting of the roll, pitch and yaw rates. As shown in Figure 4, the pilot (or a supervisory controller) gives a command $\vec{r}$, which is the desired target state. Using some known reference model, a trajectory to $\vec{r}$ is calculated. Let us assume the reference model is a linear system specified with matrices A_m and B_m. Thus, we calculate the trajectory to the desired $\vec{r}$ as follows:

$$\dot{\vec{x}}_m \quad = \quad A_m\vec{x}_m + B_m\vec{r} \tag{2}$$

where $\vec{r}$ is the target state (3×1 vector), and A_m, B_m are known 3×3 matrices. This yields two values: the desired state for the current time (denoted by $\vec{x}_m$) and the desired derivates for the current time (denoted by $\dot{\vec{x}}_m$).

If $\vec{x}$ denotes the actual (current) state, then $\vec{x}_e := \vec{x}_m - \vec{x}$ is the error. Next, a standard proportional-integral (PI) controller is used to ensure that position $\vec{x}_m$ is reached, and hence a PI controller calculates the desired rate of change, $\vec{u}_{PI}$:

$$\vec{u}_{PI} \quad = \quad K_p\vec{x}_e + K_i\vec{intx}_e \tag{3}$$

where $\vec{intx}_e$ is the integral of the error and K_p, K_i are constants.

$$\dot{\vec{intx}}_e \quad = \quad \vec{x}_m - \vec{x} \tag{4}$$

However, since we actually need to reach $\vec{r}$, the final desired velocity $\dot{\vec{x}}_d$ is a sum of $\dot{\vec{x}}_m$ and the output of the PI controller.

$$\dot{\vec{x}}_d \quad = \quad \dot{\vec{x}}_m + \vec{u}_{PI} \tag{5}$$

Using a known model of the (undamaged) aircraft, $\dot{\vec{x}}_d$ is inverted and the actual control input $\vec{u}$ is computed. Let us assume that the known model of the undamaged aircraft is given by matrices A, B and G. The dynamic inversion module uses these known dynamics (of the undamaged aircraft) to compute the required $\vec{u}$ for the given $\dot{\vec{x}}_d$,

$$\vec{u} \quad = \quad B^{-1}(\dot{\vec{x}}_d - A\vec{x} - G\vec{z}) \tag{6}$$

where the control vector $\vec{u}$ is a 3×1 vector consisting of the aileron, elevator, and rudder inputs. We are assuming that the vector $\vec{z}-$ the trim state vector of the angle of attack, angle of sideslip, and engine throttle, is fixed.

The dynamics of the actual aircraft can be described by

$$\dot{\vec{x}} \quad = \quad A\vec{x} + B\vec{u} + G\vec{z} + f(\vec{x}, \vec{u}, \vec{z}) \tag{7}$$

where f represents the unknown term modeling the damage and/or uncertainty. The value $\vec{x}$ is used to calculate the error $\vec{x}_e$ and it thus closes the loop in Figure 4.

The direct adaptive modification adds an additional loop, shown in red, to the above architecture (see Figure 4). This additional loop goes through a neural net module. The neural net module takes as input the current position, $\vec{x}$, and the current control vector, $\vec{u}$, and computes an adaptive gain term, $\vec{u}_{ad}$. This

[a] The Matlab scripts for the model were provided to us by Dr. Steven A. Jacklin (NASA Ames Research Center).

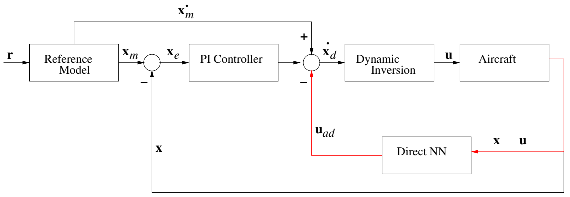


Figure 4. Neural Net Direct Model-Reference Adaptive Control.

adaptive gain term is *directly* subtracted from $\dot{\vec{x}}_d$, and thus, the old Equation 5 is replaced by the following new equation:

$$\dot{\vec{x}}_d \quad = \quad \dot{\vec{x}}_m + \vec{u}_{PI} - \vec{u}_{ad} \tag{8}$$

The rationale for introducing the new term $\vec{u}_{ad}$ is that this additional term will compensate for the difference between the dynamics of the actual (possibly damaged) aircraft and the dynamics of the known model of the (undamaged) aircraft. In this way, adaptive control can potentially help keep the system stable even under adverse/damage conditions.

The idea behind analyzing an adaptive system is to represent the weight update equations of the neural network using continuous differential equations. Since all the other components are modeled using differential equations, the entire system can then be modeled as a continuous dynamical system. Specifically, the neural network block takes the current state error $\vec{x}_e$ and the current integral of the error $\vec{intx}_e$ as input, updates its weights and then outputs the adaptation term $\vec{u}_{ad}$, which is computed as follows:

$$\vec{u}_{ad} \quad = \quad L^T\beta \tag{9}$$

where L is a $n \times 3$ matrix consisting of the weights in the neural network and β is a $n \times 1$ vector of kernel (or basis) functions. The notation L^T denotes the transpose of the matrix L. Thus, $\vec{u}_{ad}$ is a weighted sum of the kernel functions. The kernel functions can consist of linear functions – $\vec{x}, \vec{u}$; quadratic functions – $x_i x_j$ for $x_i, x_j \in \vec{x}$; and more complex nonlinear functions, such as trigonometric functions, applied to the variables $\vec{x}, \vec{u}$.

The dynamics of the neural net weight update law is given by

$$\dot{L} \quad = \quad -\Gamma\beta(\vec{intx}_e^T K_i^{-1} + (\vec{x}_m - \vec{x})^T K_p^{-1}(I + K_i^{-1})) \tag{10}$$

where Γ is a fixed constant.

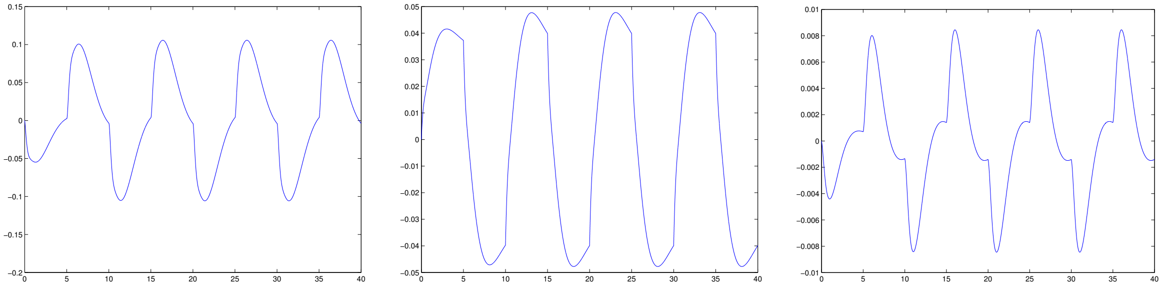
Formally, we model the direct adaptive flight controller as a continuous dynamical system. The state variables are

$$\mathtt{X} \quad := \quad \{\vec{x}, f, \vec{x}_m, \vec{intx}_e, \vec{x}_d, L, \beta, \vec{r}\}$$

and the dynamics of these state variables are given by Equations 2, 3, 4, 6, 7, 8, 9, and 10 above. We can rewrite the dynamics, by eliminating definitions and using only the state variables $\mathtt{X}$, as follows:

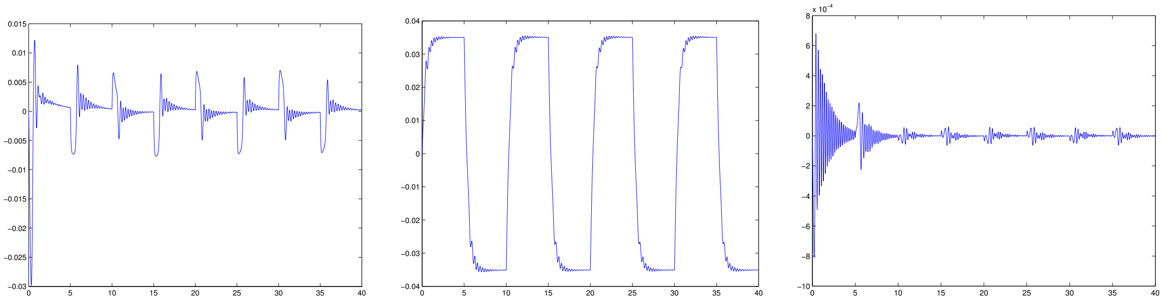
$$\begin{aligned}
\dot{\vec{x}} \quad &= \quad A\vec{x} + BB^{-1}(\dot{\vec{x}}_d - A\vec{x} - G\vec{z}) + G\vec{z} + f(\vec{x}, \vec{u}, \vec{z}) \\
\dot{\vec{x}}_m \quad &= \quad A_m\vec{x}_m + B_m\vec{r} \\
\dot{\vec{intx}}_e \quad &= \quad \vec{x}_m - \vec{x} \\
\dot{\vec{x}}_d \quad &= \quad A_m\vec{x}_m + B_m\vec{r} + K_p(\vec{x}_m - \vec{x}) + K_i\vec{intx}_e - L^T\beta \\
\dot{L} \quad &= \quad -\Gamma\beta(\vec{intx}_e^T K_i^{-1} + (\vec{x}_m - \vec{x})^T K_p^{-1}(I + K_i^{-1}))
\end{aligned}$$

Note that we have not provided the dynamics of three state variables, namely f, β and $\vec{r}$ above. The variables f and $\vec{r}$ are treated as inputs – that is, their dynamics are unknown – and the dynamics of the variable β depends on the choice of the kernel function. Since we are not assuming any fixed set of kernel functions, $\vec{\beta}$ and its dynamics are also unknown. The variable $\vec{u}$ has been eliminated by substitution. Note also that all symbols other than the state variables that are used above represent known constants. We wish to analyze the system for stability under suitable assumptions on $\vec{\beta}$ and $\vec{f}$.



Roll rate Pitch rate Yaw rate

Figure 5. Simulating a damaged aircraft that is controlled using a standard PI controller *without* adaptation. A fixed wing damage scenario is assumed here.



Roll rate Pitch rate Yaw rate

Figure 6. Simulating the model after including a direct NN model reference adaptive controller. A fixed wing damage scenario is assumed here. The learning rate for the standard MRAC controller is $\Gamma = 10^4$.

IV. Analysis Results

Before we formally verify the controller design, we present a few simulation results. In Figure 5, we present the simulation of the PI controller *without* any adaptation, but in the presence of fixed wing damage scenarios. In Figure 6, we present the simulation of the same PI controller, but now enhanced with an adaptation, and under the same wing damage scenario. The adaptation is using a direct model reference neural network based adaptive controller, whose learning rate is $\Gamma = 10^4$. Qualitatively, it is clear from the plots that adaptation helps in this particular damage scenario.

The goal of formal verification is to provide guarantees, not for a specific damage scenario, but across a whole range of possible damage scenarios. This can be achieved using formal verification technology.

We are interested in proving that, for the model of the direct neural net adaptive flight controller, the error $\vec{x}_e := \vec{x}_m - \vec{x}$ and the integral of this error, namely $\vec{intx}_e$, remain bounded. Since our property of interest relates only to $\vec{x}_e$ and $\vec{intx}_e$, we can eliminate some of the other variables by substitution and obtain the following reduced model:

$$\begin{aligned}
\dot{\vec{x}}_e &= -K_p\vec{x}_e - K_i\vec{intx}_e + L^T\beta - f \\
\dot{\vec{intx}}_e &= \vec{x}_e \\
\dot{L} &= -\Gamma\beta(\vec{intx}_e^{\,T} K_i^{-1} + \vec{x}_e^T K_p^{-1}(I + K_i^{-1})) \\
\dot{\beta} &= f_1 \\
\dot{f} &= f_2
\end{aligned} \qquad (11)$$

where the new state variables are $\vec{x}_e, \vec{intx}_e, L, \beta, f$ and f_1, f_2 are unknown parameters, whereas Γ, K_p, K_i are known fixed parameters.

The matrices K_p and K_i are diagonal matrices. Specifically, $K_p := \sqrt{2}K$ and $K_i := K^2$, where $K :=$ $\mathtt{diag}(3.5, 2.5, 2)$. Consequently, the three dimensions can be separately analyzed and the results can be subsequently put together.

We first verify that always eventually the error goes below some fixed bound. We prove this under the assumption that the neural net approximates, upto some bound, the change in dynamics caused by aircraft damage. Thus, our goal is to prove the following property:

$$\mathbb{G}((L^T\beta - f)^2 < a) \quad \Rightarrow \quad \mathbb{G}(\mathbb{F}(\vec{x}_e^2 < c)) \tag{12}$$

for some constants a and c. We use the inference rule in Figure 2(right) for this purpose.

To apply the inference rule in Figure 2(right), we use templates for the assumption ψ and for the witness V. These templates are shown below.

Assumption ψ	$(L^T\beta - f)$ is bounded	Template: $\lvert\lvert L^T\beta - f\rvert\rvert^2 \leq a$
Guarantee ϕ	$\lvert\lvert x_e\rvert\rvert$ is bounded	Template: $\lvert\lvert x_e\rvert\rvert^2 \leq c$
Witness V	A Lyapunov function	Template: $\lvert\lvert x_e\rvert\rvert^2 + b\lvert\lvert intx_e\rvert\rvert^2$

Note that the templates introduce three new unknown variables: a, b and c. With the above choices of ψ, ϕ and V, the rule in Figure 2(right) reduces the verification of Property 12 to provability of the following formula:

$$\begin{aligned}
&\exists a, b, c, \epsilon > 0 : \forall x_e, intx_e, L, \beta, f : \\
&\quad (||L^T\beta - f||^2 \leq a \wedge ||x_e||^2 > c \quad \Rightarrow \quad ||x_e||^2 + b||intx_e||^2 \geq 0) \;\wedge \\
&\quad (||L^T\beta - f||^2 \leq a \wedge ||x_e||^2 > c \quad \Rightarrow \quad \frac{d}{dt}(||x_e||^2 + b||intx_e||^2) < -\epsilon)
\end{aligned}$$

which says that there exist ($\exists$) values for a, b, c, ϵ such that for any ($\forall$) value of $x_e, intx_e, L, \beta, f$, the two implications hold. The two implications above correspond to the two conditions, (B1) and (B2), in Figure 2(right). Clearly, the first implication holds whenever $b \geq 0$. Hence, we can replace the first implication by the simpler constraint $b \geq 0$. Thus, we are left with only the second implication. Recall that we deal with the 3 dimensions separately, and hence the variables $x_e, intx_e, f$ in the formula above are real-valued variables (and not vectors) and $L^T\beta$ is also real valued. We introduce a new variable for the expression $L^T\beta - f$. The resulting $\exists\forall$ formula is solved using the tool `QEPCAD` to obtain the following values for the parameters a, b, c, ϵ:

$$b = 10, a = 1, c = 1, \epsilon = 1$$

This completes the proof of the following property:

$$\mathbb{G}((L^T\beta - f)^2 \leq 1) \;\Rightarrow\; \mathbb{G}(\mathbb{F}(x_e^2 \leq 1)). \tag{13}$$

In other words, we have proved that, assuming $L^T\beta - f$ is bounded, the error x_e eventually falls below some bound – irrespective of the initial weights of the neural net L, the kernel functions of the neural net β, or the type of damage f.

We next try to prove that the error x_e always eventually drops below a constant factor of the neural net approximation error – again irrespective of β, f and L. This property is formally written as:

$$\mathbb{G}(\mathbb{F}(||x_e||^2 \leq c||L^T\beta - f||^2))$$

We again use the inference rule in Figure 2(right). Specifically, we use the following templates:

Guarantee ϕ	$\frac{\lvert\lvert x_e\rvert\rvert}{\lvert\lvert L^T\beta - f\rvert\rvert}$ is bounded	$\lvert\lvert x_e\rvert\rvert^2 \leq c\lvert\lvert L^T\beta - f\rvert\rvert^2$
Witness V	Exists a Lyapunov function	$\lvert\lvert x_e\rvert\rvert^2 + b\lvert\lvert intx_e\rvert\rvert^2$

Note that we make no assumptions, and hence the assumption ψ is just *true*. Inference rule in Figure 2(right), when applied using the templates above, generates the following $\exists\forall$ formula:

$$\begin{aligned}
&\exists b, c, \epsilon > 0 : \forall x_e, intx_e, L, \beta, f : \\
&\quad (||x_e||^2 > c||L^T\beta - f||^2 \quad \Rightarrow \quad ||x_e||^2 + b||intx_e||^2 \geq 0) \;\wedge \\
&\quad (||x_e||^2 > c||L^T\beta - f||^2 \quad \Rightarrow \quad \frac{d}{dt}(||x_e||^2 + b||intx_e||^2) < -\epsilon)
\end{aligned}$$

The above formula is found to be equivalent to *false* by QEPCAD. In other words, there is no choice of values for b, c, ϵ that "works". The reason is that the derivative of V can be arbitrarily close to 0 and there is no nonzero value for ϵ that works. Hence, we weaken the property and try to prove that x_e eventually, *in the limit*, drops below a constant factor of the neural net approximation error. This property is formally written as:

$$\mathbb{G}(\mathbb{F}^{cl}(||x_e||^2 \leq c||L^T\beta - f||^2))$$

To prove this weaker property, we use a variant of the rule in Figure 2(right). Essentially, rather than requiring that the derivative of V be less than some nonzero $-\epsilon$, we now test if the derivative of V is less than $-\epsilon * (||we||^2 - c||L^T\beta - f||^2)$. The values for b, c returned by QEPCAD are: $b = 10, 25c > 1$. This shows that the following property holds of the model:

$$\mathbb{G}(\mathbb{F}^{cl}(x_e^2 \leq 0.05(L^T\beta - f)^2)).$$

We have thus proved two versions of the property related to showing that the error $\vec{x_e}$ is eventually bounded. The first version shows that the error is always eventually bounded *assuming* the neural network closely approximates the change in the dynamics caused by damage (f). The second one shows that the ratio of the error $\vec{x_e}$ to the approximation error ($L^T\beta - f$) is always eventually bounded *without* requiring any further assumptions.

In the third analysis, we also consider the weight update law of the neural network and prove a stability property of the combined system and neural network. Recall the model of the system given in Equation 11. The variable β represent the basis functions used in the neural network and we do not wish to assume the form of these functions. Hence, the dynamics of β, specified by f_1 in Equation 11, are unknown. Also unknown is f_2, which specifies the dynamics of the damage. Let us analyze the system under the assumption that f_1, f_2 are both zero, that is, β is an unknown, but fixed constant, and the damage f is unknown but fixed. Under these assumptions, the system in Equation 11 becomes a parametric linear system – parameterized by the value of β and f. We can, however, prove that for *all* choices of the parameters, the resulting linear system is asymptotically stable. We do this as follows: we write the characteristic polynomial of the parametric linear system, which is a degree 3 polynomial whose coefficients are polynomials in the parameter β. We then prove two formulas using QEPCAD: the first formula states that, for all β, all real solutions of the characteristic polynomial are negative and the second formula states that, for all β, the real part of all complex solutions of the characteristic polynomial are negative. Both formulas are $\forall$ formulas. The second formula is obtained by introducing variables a, b representing a complex $a + \imath b$ solution of the characteristic polynomial.

The third analysis shows that the weights of the neural network remain bounded and hence it partly proves the correctness of the neural network weight update laws. However, note that we have made two assumptions: the first is that the damage term f is not changing and second is that the neural network basis functions are not changing. While the former is a reasonable assumption, the latter is a strong assumption. However, it remains a challenge to perform analysis under any weaker assumption on the dynamics of β.

Analyzing Indirect and Hybrid Adaptive Controllers

We briefly describe some preliminary analysis results from our effort on analyzing indirect and hybrid adaptive controllers. Unlike direct adaptive controllers, indirect methods do not directly change the value of the control input, but instead actively learn the model of the damaged plant (aircraft). The learned model is then used in the dynamic inversion module (see Figure 4) to compute the actual control input that is sent to the aircraft. Hybrid adaptive controllers combine direct and indirect methods to better adapt to the changing plant model.

A recursive least squares (RLS) algorithm is often used to actively learn the plant model. A critical step in formally analyzing indirect and hybrid adaptive controllers is, therefore, establishing the correctness of the RLS procedure. However, this is a tricky task since even formulating the formal statement of correctness is challenging. Further, identifying the invariants of the RLS procedure is also a challenge.

We briefly describe the RLS procedure, and then describe its analysis. The RLS procedure estimates some unknown parameters given some measurements. Specifically, it assumes that some measurements

$$\{(\vec{\theta}_1, \epsilon_1), (\vec{\theta}_2, \epsilon_2), (\vec{\theta}_3, \epsilon_3), \ldots\}$$

```
[phi, R] = RLS(phi, theta, R, epsilon)
  err = epsilon - phi' * theta;
  tmp = 1 + theta' * R * theta;
  phi = phi + (R * theta * err') / tmp;
  A = (R * theta * theta') / tmp;
  R = (I - A) * R ;
return([phi, R]);
```

```
n = 20; N = 5; Phi = rand(N,1);
phi = zeros(N,1); R = 1e4*eye(N);
for i = 1:n
  theta = 10 * rand(N, 1);
  epsilon = theta' * Phi;
  theta(1,1) = theta(1,1) - epsilon/Phi(1,1);
  [phi,R] = RLS(phi,theta,R,0);
end
```

Figure 7. The Basic RLS Procedure (in Matlab syntax) in the left. On the right, a counter example for convergence of RLS in Matlab syntax.

are given, where $\vec{\theta_i}$ is a vector representing the sensor readings of the state of the system (at some timepoint i) and ϵ_i is the output of the system (at timepoint i). We assume that the output is a linear combination of the state variables, that is, for all i,

$$\epsilon_i := \vec{\theta_1}^T \vec{\phi}$$

where $\vec{\phi}$ is an unknown vector of parameters. The problem is to estimate $\vec{\phi}$ from the available data, assuming that the data may be noisy. The recursive-least-squares (RLS) procedure is an algorithm for solving this problem.

The RLS procedure, described in Figure 7(left), works by maintaining two values: current best estimate $\vec{\phi}$ and a square matrix R. As a new data point $(\vec{\theta}, \epsilon)$ arrives, the procedure updates these two values based on the observed data (as shown above). Initially, R is chosen to be equal to a diagonal matrix with very large positive values on the diagonal and ϕ is chosen to be the zero vector.

We now state a critical correctness claim that we would like to make about the RLS procedure. It is crucial for proving stability of the indirect (and hybrid) adaptive controllers that use the RLS procedure for learning plant parameters. The claim can be stated as:
"If the inputs to the RLS procedure are a sequence of values $\{(\vec{\theta_i}, \epsilon_i) \mid i = 0, 1, \ldots\}$ such that $\epsilon_i = \vec{\theta_i}^T \vec{\Phi}$ for some unknown, but fixed, $\vec{\Phi}$, and if the values $\vec{\theta_i}$ are sufficiently distributed, then the estimate $\vec{\phi}$ computed by RLS *converges* to $\vec{\Phi}$."
The key aspect in the above claim is the requirement that the values $\vec{\theta_i}$ be sufficiently distributed. If $\vec{\theta_i}$'s are chosen in some highly constrained scenarios, then the rate of convergence could become very slow or there may not no convergence at all. Example 1 below provides such a scenario.

Example 1 (Counter Example for Convergence of RLS) *Suppose $\vec{\Phi}$ is the desired set of parameters and $\vec{\phi}_0 = \vec{0}$ is our initial estimate. Suppose that all the inputs to the RLS procedures, $\vec{\theta_i}$'s, are perpendicular to $\vec{\phi}_0 - \vec{\Phi}$. Then we can* prove *that the RLS procedure* will not converge. *The code in Figure 7(right) generates such a scenario: it first picks a random $\vec{\Phi}$, and then it generates all inputs $\vec{\theta_i}$ such that $\vec{\theta_i}^T \vec{\Phi}$ is zero. Running the code shows that the value of $\vec{\phi}$ does not converge to $\vec{\Phi}$. In fact, $\vec{\phi}$ remains very close to its initial value, as one would expect.*

One can argue that (a) it is unlikely that the vectors $\vec{\theta_i}$ will be orthogonal to $\vec{\Phi}$ and (b) there will be disturbance in the system and errors in the sensor readings so that even if the ideal vector $\vec{\theta_i}$ is orthogonal to $\vec{\Phi}$, the vector obtained via sensor readings will not be exactly orthogonal. However, both these arguments are fallible. First, very often the vectors $\vec{\theta_i}$ are orthogonal to $\vec{\Phi}$ – in fact, this was the case in our model of hybrid adaptive flight controller. Second, while disturbances indeed help in convergence, but convergence can still be very slow. For example, let us modify the counter example generation code in Figure 7(right) to introduce 0.1% fluctation in the $\vec{\theta}$ values. Figure 8(left) shows the plot of the error in estimation (as a percentage) against the number of iterations. Note the slow rate of convergence. Even after 10000 iterations, there is still a 10% error in the estimate. (Without the 0.1% fluctation, there is always a 100% error in the estimate for any number of iterations.)

One way to speed up convergence when the data points are relatively close to the worst case, is to scale the data by multiplying by a large factor. We multiply $\vec{\theta}, \epsilon$ by a constant factor, MM, before calling the RLS procedure. Figure 8(right) shows the estimation error plotted against the iteration number for four different choices of MM, namely, $1, 2, 5, 10$.

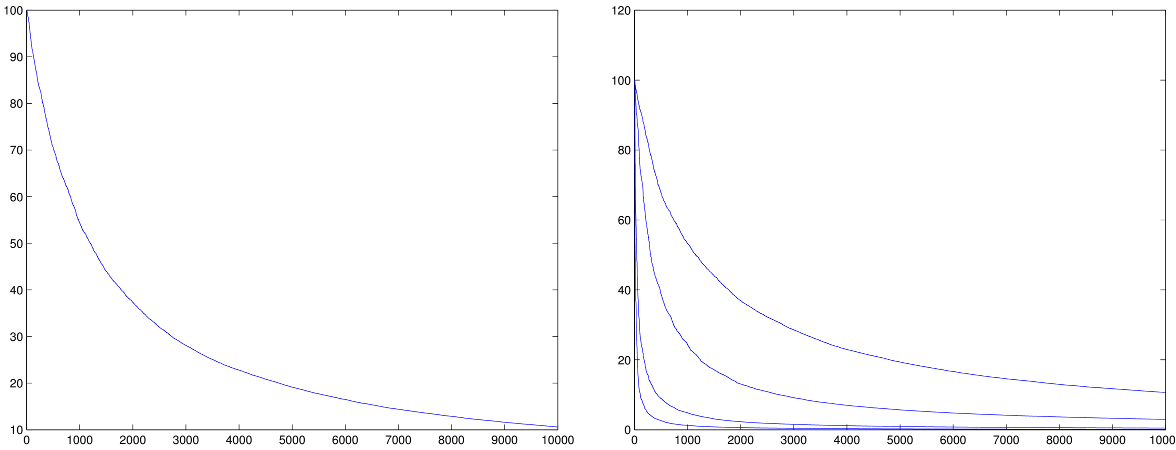


Figure 8. Left: Rate of convergence of RLS when there is 0.1% random fluctuation in the values of $\vec{\theta}$ picked by the code in Figure 7(right). Right: The same plot, but now also including the plots when a scaling factor of $2, 5$ and 10 are used. There is a clear improvement in the rate of convergence of RLS as we scale up data using a higher scaling factor.

As mentioned above, in our model for indirect and hybrid adaptive control, we found that the RLS procedure was being used exactly in the scenario where the input data, $\vec{\theta}$'s, was (almost) orthogonal to the parameters to be learnt $\vec{\Phi}$, and hence it was a bad case for the RLS procedure. Simulations of the indirect and hybrid adaptive controllers showed a poor convergence rate. We performed additional simulations using a higher scaling factor. The plots of the roll and yaw rates versus time for indirect control are shown in Figure 9 and the same plots, for hybrid control are shown in Figure 10. As expected, the use of a higher scaling factor improved convergence and performance. Note that we simulated a scenario whereby a portion of the left wing is damaged in the aircraft and the pilot commands the aircraft to attain a target pitch; the target is changed every 5 time units to show the effect of learning (adaptation). The same scenario was also used in the plots shown in Figure 5 and Figure 6. Note also that the model of the damaged aircraft used for simulations is a linear model – and hence the indirect adaptive controller by itself is sufficient to learn the model of the damaged aircraft. Hence, the indirect controller performs better than the hybrid controller in this case (as is also evident from the plots).

V. Conclusion

We presented a novel approach for verification of continuous and hybrid systems called bounded verification. We presented preliminary results obtained from formally analyzing a model of direct neural net adaptive flight control. The model was analyzed for bounded stability using the bounded verification approach. We also analyzed indirect and hybrid adaptive controllers by studying the convergence of the recursive least squares algorithm. While formal verification provides increased assurance in the safety of the system, it can also potentially help by identifying ways to improve the system design or performance.

If a verification task is successfully completed using the bounded verification approach, then we additionally obtain a witness, or certificate, for the proof of the property. This certificate can be used to statically verify a software implementation,[6] or monitor it at runtime.

There are several different avenues for future work. While, in theory, `QEPCAD` can decide the validity of arbitrary first-order formulas (in particular, of $\exists\forall$ formulas) over the reals, in practice it is unreliable. One direction for future work involves developing scalable and robust tools for checking validity of $\exists\forall$ formulas over the reals. A different direction for future work consists of building and verifying more refined models of adaptive control systems. Specifically, it will be interesting to model uncertainties in the model of the (damaged) aircraft and concrete details of the kernel functions used in the neural net. Finally, another potentially useful future work consists of developing heuristics – specific for the domain of adaptive flight control – for automatically generating templates and efficiently solving the $\exists\forall$ constraints.

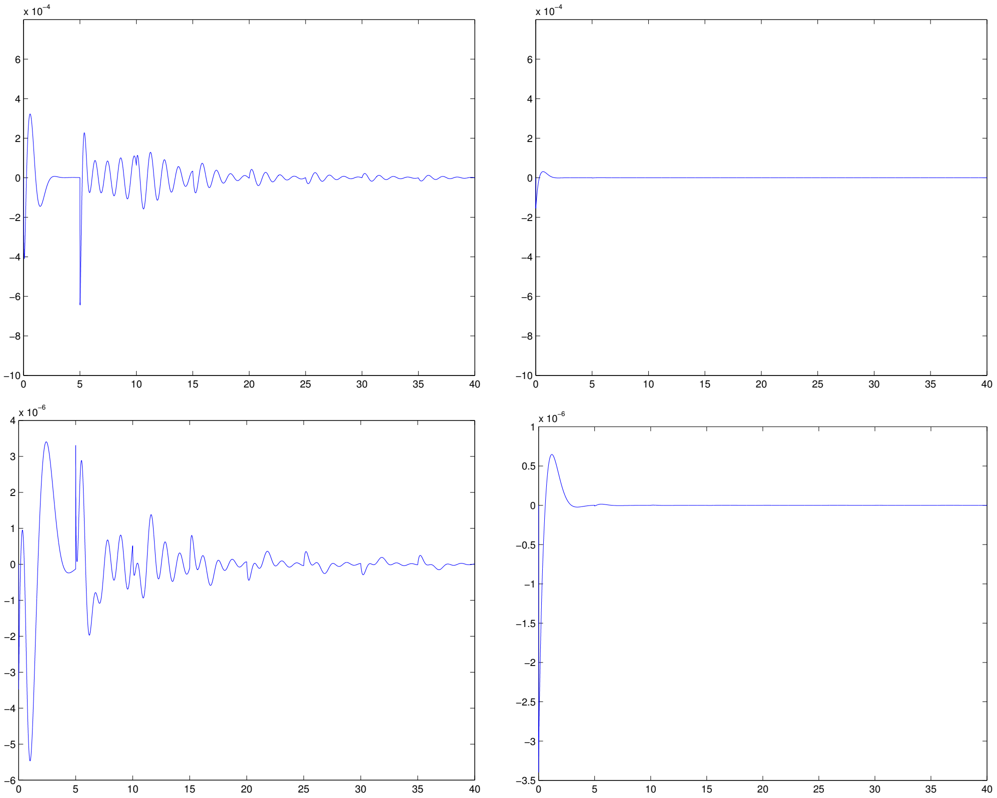


Figure 9. Indirect adaptive control: Roll (top) and yaw (bottom) rates plotted against time. The RLS procedure used $MM = 1$ in the left plots and used $MM = 10^5$ in the right plots. We have used different scaling on the y-axis in the bottom left and bottom right plots.

Acknowledgments

The author thanks Dr. Steven A. Jacklin (NASA Ames) for providing the models of adaptive flight controllers used in this research. Research supported in part by the National Science Foundation under grants CNS-0720721 and CSR-0917398 and by NASA under Grant NNX08AB95A.

References

[1] R. Alur, T. Dang, and F. Ivancic. Progress on reachability analysis of hybrid systems using predicate abstraction. In *Hybrid Systems: Computation and Control, HSCC 2003*, volume 2623 of *LNCS*, pages 4–19. Springer, 2003.

[2] NASA Ames Research Center. Matlab scripts for simulating neural net adaptive flight control. Personal Communication.

[3] A. Chutinan and B. H. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *37th IEEE Conference on Decision and Control*, 1998.

[4] E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *9th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2003*, volume 2619 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2003.

[5] T. Dang and O. Maler. Reachability analysis via face lifting. In T. A. Henzinger and S. Sastry, editors, *HSCC*, volume 1386 of *LNCS*, pages 96–109. Springer, 1998.

[6] E. Feron and M. Roozbehani. Certifying controls and systems software. *CoRR*, abs/cs/0701132, 2007.

[7] A. Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, volume 3414 of *LNCS*, pages 291–305, 2005.

[8] A. Girard and C. Le Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis. In *HSCC*, volume 4981 of *LNCS*, pages 215–228, 2008.

[9] S. Gulwani, S. Srivastava, and R. Venkatesan. Program analysis as constraint solving. In *Proc. ACM Conf. on Prgm. Lang. Desgn. and Impl. PLDI*, pages 281–292, 2008.

[10] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Proc. 20th Intl. Conf. on Computer Aided Verification, CAV 2008*, volume 5123 of *LNCS*, pages 190–203. Springer, 2008. July 7-14, 2008, Princeton, NJ.

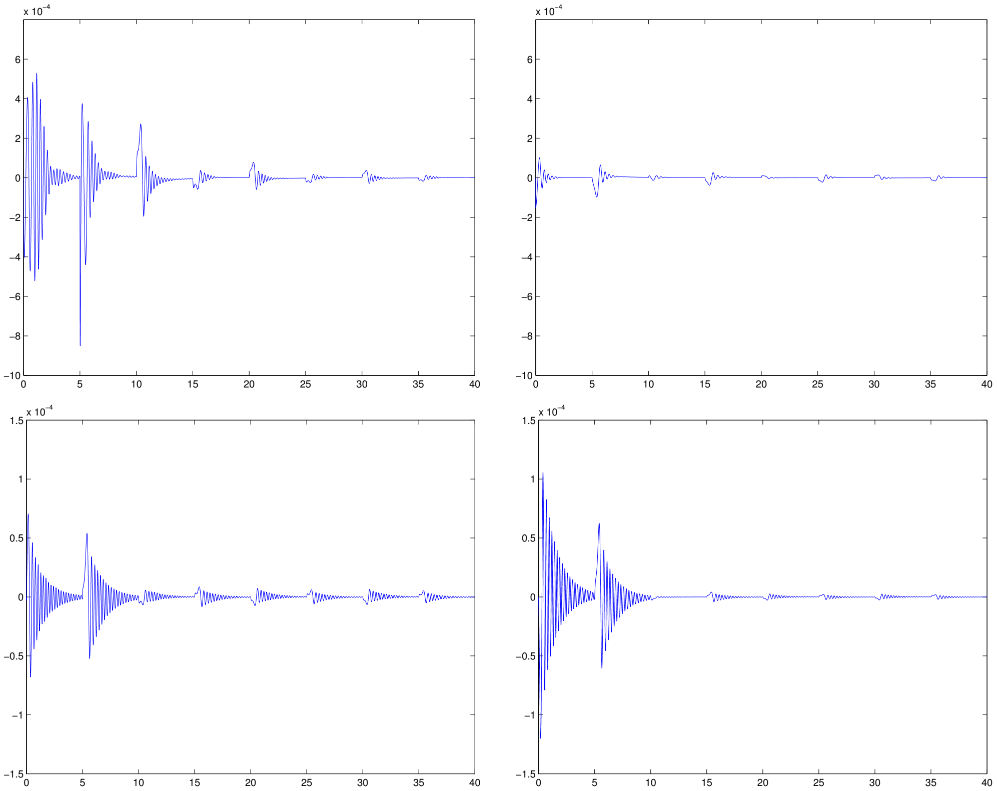


Figure 10. Hybrid adaptive control: Roll (top) and yaw (bottom) rates plotted against time. The RLS procedure used $MM = 1$ in the left plots and used $MM = 10^6$ in the right plots.

[11] J. Kaneshige and J. Burken. Enhancements to a neural adaptive flight control system for a modified f-15 aircraft. In *AIAA Guidance, Navigation, and Control Conference*, 2008. AIAA-2008-6986.

[12] Deepak Kapur. Automatically generating loop invariants using quantifier elimination. In *Deduction and Applications*, 2005.

[13] A. B. Kurzhanski and P. Varaiya. On ellipsoidal techniques for reachability analysis. *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications and Algorithms*, 9:347–367, 2002.

[14] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *HSCC*, volume 1790 of *LNCS*, pages 310–323. Springer, 2000.

[15] N. Nguyen, K. Krishnakumar, and J. Boskovic. An optimal control modification to model-reference adaptive control for fast adaptation. In *AIAA Guidance, Navigation, and Control Conference*, 2008. AIAA-2008-7283.

[16] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Proc. 7th Intl. Workshop on Hybrid Systems: Computation and Control, HSCC 2004*, volume 2993 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2004.

[17] R. T. Rysdyk and A. J. Calise. Fault tolerant flight control via adaptive neural network augmentation. In *AIAA Guidance, Navigation, and Control Conference*, 1998. AIAA-1998-4483.

[18] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In R. Alur and G. J. Pappas, editors, *Hybrid Systems: Computation and Control, HSCC 2004*, volume 2993 of *Lecture Notes in Computer Science*, pages 539–554. Springer, 2004.

[19] SRI International. *Yices: An SMT solver.* `http://yices.csl.sri.com/`.

[20] A. Tiwari. Approximate reachability for linear systems. In *Proc. 6th Intl. Workshop on Hybrid Systems: Computation and Control, HSCC 2003*, volume 2623 of *Lecture Notes in Computer Science*, pages 514–525. Springer, 2003.

[21] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In *HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, 2002.

[22] A. Tiwari and G. Khanna. Nonlinear Systems: Approximating reach sets. In *Proc. 7th Intl. Workshop on Hybrid Systems: Computation and Control, HSCC 2004*, volume 2993 of *LNCS*, pages 600–614. Springer, March 2004.

[23] U. Topku, A. Packard, P. Seiler, and T. J. Wheeler. Stability region analysis using simulation and sum-of-squares programming. In *Proc. American Control Conference*, 2007.