Formally Analyzing Adaptive Flight Control[⋆]

Ashish Tiwari

SRI International
Menlo Park, CA 94025
tiwari@csl.sri.com

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 QEPCAD to solve the resulting $\exists\forall$ formula.

1 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.

This short paper describes a case study of formally analyzing a direct model-reference adaptive flight control that is based on neural networks. The paper presents some preliminary analysis results obtained using the *bounded verification* approach.

2 Neural Net Direct Adaptive Control

A traditional *control system* consists of two components: a controller and a plant (see Figure 1). 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.

[⋆] Research supported in part by the National Science Foundation under grant CNS-0720721 and by NASA under Grant NNX08AB95A.

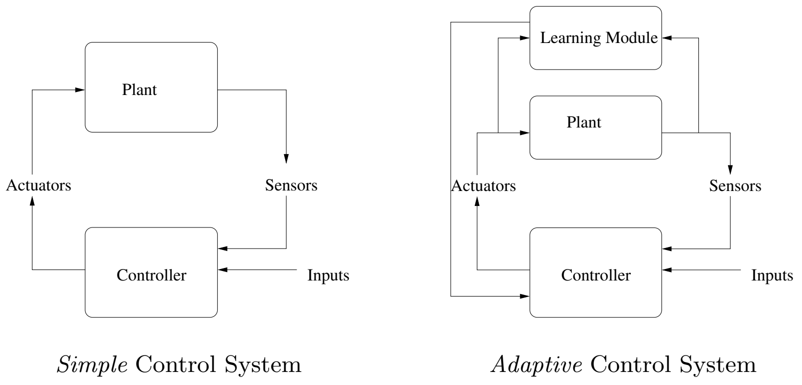

Fig. 1. Simple and Adaptive Control System. Adaptive systems have an additional learning module that can be used by the controller.

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 1).

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 have been used to play the role of the learning module in both direct and indirect adaptive control methods. In particular, Rysdyk and Calise [10] 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 [6].

2.1 Direct Neural Net Adaptive Flight Control

In this paper, we focus on direct neural net adaptive flight control. The main source of our model is [8] and [1]. The high-level architecture of such a controller is shown in Figure 2. In the figure, the pilot (or a supervisory controller) gives a command $\boldsymbol{r}$, which is the desired target position. Using some known reference model, a trajectory to $\boldsymbol{r}$ is calculated. Thus, $\boldsymbol{x_m}$ is the current desired position and $\dot{\boldsymbol{x_m}}$ is the current desired velocity computed using the reference model. A standard PI controller (proportional integral) is used to ensure that position $\boldsymbol{x_m}$ is reached. However, since we actually need to reach $\boldsymbol{r}$, the final desired velocity $\dot{\boldsymbol{x_d}}$ is a sum of $\dot{\boldsymbol{x_m}}$ and the output of the PI controller. Using a known model of the (undamaged) aircraft, $\dot{\boldsymbol{x_d}}$ is inverted and the actual control input $\boldsymbol{u}$ is computed.

The direct adaptive modification adds an additional loop to the above architecture (see Figure 2). This additional loop goes through a neural net module.

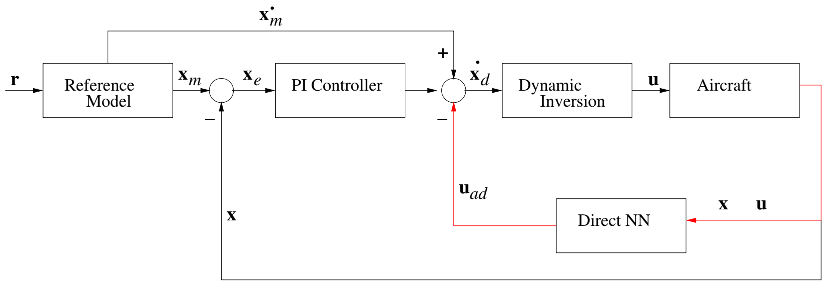

Fig. 2. Neural Net Direct Model-Reference Adaptive Control.

r	commanded value of x
x_m	desired value of x, calculated using reference model
x	actual value of x, determined by the damaged aircraft
x_e	error, $x_m - x$
$intx_e$	integral of the error, $\int x_e$
L	weights of the NN
β	fixed functions, $L^T\beta =$ adaptive control term
f	Damaged dynamics, $f = \dot{x} - \dot{x}_u$
u_e	PI gain, $K_p x_e + K_i intx_e$
$\dot{x_d}$	$\dot{x_m} + u_e - u_{ad}$
$\dot{L}$	weight update / neural net learning
Γ, K_p, K_i, A_m	Known constants
r, f, $\dot{f}$	Unknown/Symbolic Parameters

Table 1. Meaning of the variables used in the paper and in Figure 2

The direct neural net takes as input the current position, $\boldsymbol{x}$, and the current control vector, $\boldsymbol{u}$, and computes an adaptive gain term, $\boldsymbol{u}_{ad}$, that is *directly* subtracted from $\dot{\boldsymbol{x_d}}$. The idea 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 the above 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 be modeled as a continuous dynamical system.

2.2 Model

Let $\boldsymbol{x}$ be a 3×1 vector consisting of the roll, pitch, and yaw rates of the aircraft. The control vector $\boldsymbol{u}$ is also a 3×1 vector consisting of the aileron, elevator, and rudder inputs. We are assuming that the vector $\boldsymbol{z}$ – the trim state vector of the angle of attack, angle of sideslip, and engine throttle, is fixed. The dynamics of the aircraft are given by

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + G\boldsymbol{z} + f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{z}) \tag{1}$$

where A, B, G are known matrices in $\Re^{3\times3}$ and f represent the unknown term (caused by uncertainty or damage to the aircraft).

The desired trajectory is computed using the reference model as

$$\dot{\boldsymbol{x_m}} = A_m\boldsymbol{x_m} + B_m\boldsymbol{r} \tag{2}$$

where $\boldsymbol{r}$ is the pilot command (3×1 vector), and A_m, B_m are known matrices in $\Re^{3\times3}$.

The desired acceleration, $\dot{\boldsymbol{x_d}}$, is given by

$$\dot{\boldsymbol{x_d}} = A_m\boldsymbol{x_m} + B_m\boldsymbol{r} + K_p(\boldsymbol{x_m} - \boldsymbol{x}) + K_i\boldsymbol{intx_e} - \boldsymbol{u_{ad}} \tag{3}$$

$$\dot{\boldsymbol{intx_e}} = \boldsymbol{x_m} - \boldsymbol{x} \tag{4}$$

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

The dynamic inversion module uses the known dynamics (of the undamaged aircraft) to compute the required $\boldsymbol{u}$ for the given $\dot{\boldsymbol{x_d}}$,

$$\boldsymbol{u} = B^{-1}(\dot{\boldsymbol{x_d}} - A\boldsymbol{x} - G\boldsymbol{z}) \tag{5}$$

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

$$\dot{L} = -\varGamma\beta(\boldsymbol{intx_e}^T K_i^{-1} + (\boldsymbol{x_m} - \boldsymbol{x})^T K_p^{-1}(I + K_i^{-1})) \tag{6}$$

where $\varGamma$ is a fixed constant. Equations 1, 2, 3, 4, 5, and 6 together completely describe the model.

Note that the state space of the model is described by $\boldsymbol{x}, \boldsymbol{x_m}, \boldsymbol{intx_e}, L, \boldsymbol{\beta}, \boldsymbol{f}$. The variables $\dot{\boldsymbol{x_d}}$ and $\boldsymbol{u}$ are internal variables that can be eliminated by substitution. Note that $\boldsymbol{f}$ and its dynamics are unknown. Note also that since we are not assuming any fixed set of kernel functions, $\boldsymbol{\beta}$ and its dynamics are also unknown. We wish to analyze the system for stability under suitable assumptions on $\boldsymbol{\beta}$ and $\boldsymbol{f}$.

3 Bounded Verification

The verification problem is to determine if a given model satisfies a given property. Bounded verification approach is based on performing a bounded search for a proof of the fact that the given model satisfies the given property. The main observation behind this approach is that the verification problem, in most cases, reduces to the search for the "right witness". For example, we can prove safety by searching for the "right" inductive invariant and we can prove termination, respectively stability, by searching for the "right" ranking function, respectively Lyapunov function. Since the universe of these witnesses is often infinite, bounded verification achieves computability by limiting this search to witnesses of a *specific form*. More specifically, witnesses that are instances of some fixed templates are searched.

Formally, we first define a continuous dynamical system.

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

Note that $\mathbb{R}^{\mathtt{X}}$ is isomorphic to the n-dimensional real space $\mathbb{R}^n$ where $n = |\mathtt{X}|$ is the number of variables in $\mathtt{X}$. We assume that f is Lipshitz, which guarantees that the solution of the initial value problem $\frac{d\mathtt{X}(t)}{dt} = f(\mathtt{X}(t))$, $\mathtt{X}(0) = \mathtt{x}_0$, always exists and is unique and will be denoted by $F(\mathtt{x}_0, t)$. Given a CDS $\mathtt{CDS}$ and a set $\mathtt{Init} \subseteq \mathbf{X}$ of initial states, the meaning of a continuous dynamical system is simply the collection of all possible trajectories starting from an initial state. Formally, the semantics is the collection of all solutions $F(\mathtt{x}_0, t)$ of the initial value problem $\frac{d\mathtt{X}(t)}{dt} = f(\mathtt{X}(t))$, $\mathtt{X}(0) = \mathtt{x}_0$, where $\mathtt{x}_0 \in \mathtt{Init}$. The above semantics using flow functions is broadly referred to as the *flow semantics* [12]. One can also give a *transition semantics* using discrete state transition systems [5], but the distinction [2] is not relevant here.

We reuse the temporal operators $\mathbb{G}$ and $\mathbb{F}$ and give them the following natural definitions. If $\mathtt{CDS} = (\mathtt{X}, f)$ is a continuous dynamical systems and $\phi \subseteq \mathbf{X}$ is a set of states, then $\mathbb{F}(\phi)$ and $\mathbb{G}(\phi)$ denote the following set of all states:

$$\mathbb{F}(\phi) := \{\mathtt{x} \in \mathbf{X} \mid \mathtt{Cl}(\{F(\mathtt{x}, t) \mid t \geq 0\}) \cap \phi \neq \emptyset\}$$
$$\mathbb{G}(\phi) := \{\mathtt{x} \in \mathbf{X} \mid \mathtt{Cl}(\{F(\mathtt{x}, t) \mid t \geq 0\}) \subseteq \phi\}$$

that is, $\mathbb{F}(\phi)$ denotes all states starting from which $\mathtt{CDS}$ can eventually reach ϕ and $\mathbb{G}(\phi)$ denotes all states starting from which $\mathtt{CDS}$ always stays within ϕ. Note that $\mathtt{Cl}$ denotes the (topological) closure operator.

In this paper, we are interested in proving stability properties of continuous dynamical systems; that is, starting from some initial states, the system eventually (or always eventually) reaches some desired set of states.

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. We treat formulas and sets of states interchangeably with the understanding that a formula denotes all those states where it evaluates to *true*. Hence, logical implication $\Rightarrow$ is the same as the subset relation $\subseteq$.

Figure 3(left) presents an inference rule for verifying $\mathbb{F}(\phi)$. The proof for $\mathbb{F}(\phi)$ is achieved by finding a (Lyapunov-like) function $V : \mathbb{R}^n \mapsto \mathbb{R}$ that is initially non-negative (Condition (A1)), but which decreases whenever $V > 0$

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

Fig. 3. 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 `Init`. On the right, an inference rule for verifying that a continuous system $\texttt{CDS} := (\texttt{X}, f)$ always eventually reaches ϕ.

(Condition (A2)). These two conditions guarantee, under some mild technical assumptions, that eventually $V \leq 0$, and hence Condition (A3) then guarantees that eventually ϕ holds. 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

$$\begin{aligned} L_f(p) := \sum_{x \in \texttt{X}} \frac{\partial p}{\partial x} \frac{dx}{dt} \ &:= \ \boldsymbol{\nabla} p \cdot f \\ &:= (\frac{\partial p}{\partial x_1}, \frac{\partial p}{\partial x_2}, \ldots) \cdot (\frac{dx_1}{dt}, \frac{dx_2}{dt}, \ldots) \end{aligned} \qquad (7)$$

Figure 3(right) presents an inference rule for verifying that a system always eventually reaches ϕ, that is, $\mathbb{G}(\mathbb{F}(\phi))$, irrespective of its starting state. The proof for $\mathbb{G}(\mathbb{F}(\phi))$ is obtained by finding a (Lyapunov-like) function V that is non-negative whenever ϕ is not true (Condition (B1)), but which decreases whenever ϕ is not true (Condition (B2)). Again, under some mild technical assumptions, these two conditions guarantee that always eventually $V \leq 0$ and always eventually ϕ holds.

The two inference rules given in Figure 3 for verifying $\mathbb{F}(\phi)$ and $\mathbb{G}(\mathbb{F}(\phi))$ properties respectively, are sound (under some reasonable technical assumptions that are not detailed here). However, before we can use them, we have to find the "witness" function V. 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 4.

Bounded verification is the dual of bounded model checking. Whereas bounded model checking searches for a bounded counterexample (for safety), bounded verification searches for a bounded proof (for safety). Bounded verification has been used for safety verification by using templates for specifically searching for inductive invariants in the work of several authors [11, 9, 7]. More details on bounded verification, can be found in the work of Gulwani et al. [3] and Gulwani and Tiwari [4].

$$\frac{\begin{array}{ll}(C1): & \psi \wedge \neg\phi \Rightarrow V > 0 \\ (C2): & \psi \wedge \neg\phi \Rightarrow \frac{dV}{dt} < 0\end{array}}{\mathbb{G}(\psi) \Rightarrow \mathbb{G}(\mathbb{F}(\phi))}$$

Fig. 4. Inference rule for verifying that a continuous system $\texttt{CDS} := (\texttt{X}, f)$ always eventually reaches ϕ under the assumption that always ψ holds.

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

Table 2. Verifying that $\boldsymbol{x_e}$ is always eventually bounded under the assumption that $L^T\beta - f$ is bounded.

4 Analysis Results

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

$$\begin{aligned}\dot{\boldsymbol{x_e}} &= -K_p\boldsymbol{x_e} - K_i\boldsymbol{intx_e} + L^T\beta - f \\ \dot{\boldsymbol{intx_e}} &= \boldsymbol{x_e} \\ \dot{L} &= -\Gamma\beta(\boldsymbol{intx_e}^T K_i^{-1} + \boldsymbol{x_e}^T K_p^{-1}(I + K_i^{-1})) \\ \dot{\beta} &= f_1 \\ \dot{f} &= f_2\end{aligned}$$

where the new state variables are $\boldsymbol{x_e}, \boldsymbol{intx_e}, L, \beta, f$ 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 $K := \texttt{diag}(3.5, 2.5, 2)$. 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, that is, $\mathbb{G}(\mathbb{F}(x_e^2 \leq c))$, for some constant c. We prove this under the assumption that the neural net approximates, upto some bound, the change in dynamics caused by aircraft damage. We use the inference rule in Figure 3(right), but modified to include an assumption ψ, as shown in Figure 4.

To apply the inference rule in Figure 4, we use templates for the assumption ψ, the guarantee ϕ and the witness V. These templates are shown in Table 2.

Note that the choice of a sum of squares as a template for V guarantees that the Condition (C1) in Figure 4 always holds (there is an implicit assumption that $b \geq 0$). Hence we do not write Condition (C1) below. The $\exists\forall$ formula generated

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

Table 3. Verifying that $\frac{||x_e||}{||L^T\beta-f||}$ is always eventually bounded.

by Condition (C2) using the templates from Table 2 is:

$$\exists a, b, c : \forall x_e, intx_e, L, \beta, f : (L^T\beta - f)^2 \leq a \ \wedge \ (x_e^2 > c) \ \Rightarrow \ \frac{d}{dt}(x_e^2 + b \ intx_e^2) < 0$$

Since we deal with the 3 dimensions separately, the variables in the formula above are real-valued variables (and not vectors). Using QEPCAD, we get values for the parameters a, b, c as: $b = 10, 25c > a > 0$. This proves, for example, that

$$\mathbb{G}((L^T\beta - f)^2 \leq 0.01) \ \Rightarrow \ \mathbb{G}(\mathbb{F}(x_e^2 \leq 1)). \tag{8}$$

In other words, we prove 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 can also show 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 time we do not need any assumption on the neural net approximation error and we can use the inference rule in Figure 3(right). Specifically, we use the templates shown in Table 3. Condition (B2) from Figure 3(right), when applied using the templates in Table 3, generates the following $\exists\forall$ formula:

$$\exists b, c : \forall x_e, intx_e, L, \beta, f : ||x_e||^2 > c||L^T\beta - f||^2 \ \Rightarrow \ \frac{d}{dt}(||x_e||^2 + b||intx_e||^2) < 0$$

The values for b, c returned by QEPCAD are: $b = 10, 25c > 1$. This shows that the following property holds of the model:

$$\mathbb{G}(\mathbb{F}(x_e^2 \leq 0.05(L^T\beta - f)^2)). \tag{9}$$

We have proved above two versions of the property related to showing that the error $\boldsymbol{x_e}$ is eventually bounded. Formula 8 shows that the error is always eventually bounded *assuming* the neural network closely approximates the change in the dynamics caused by damage (f). Formula 9 shows that the ratio of the error $\boldsymbol{x_e}$ to the approximation error ($L^T\beta - f$) is always eventually bounded *without* requiring any further assumptions. In the final analysis, we prove another variant of the same property by showing that the error is eventually bounded *unconditionally.* In the process, we will also show that the weights of the neural network remain bounded and hence this partly proves the correctness of the neural network weight update laws. We only need to assume that the there is an optimal value for the weights, L^*; that is, $f = L^*\beta$. In other words, we assume that the neural network basis functions β are expressible enough to precisely

Guarantee ϕ	$\|\|x_e\|\|$ is bounded	$\|\|x_e\|\|^2 \leq a$
Witness V	A Lyapunov-like function	$\|\|x_e\|\|^2 + b\|\|intx_e\|\|^2 + c\|\|L - L^*\|\|^2 - d$
Assumption ψ	$\|\|\beta\|\|$ is bounded	$\|\|\beta\|\|^2 \leq e$

Table 4. Verifying that $||x_e||$ is always eventually bounded.

capture the difference in the aircraft dynamics (f) caused by the damage. Our analysis will show that the computed weights, L, are not too far off from the optimal weights L^*.

We again use the inference rule in Figure 4, but this time on the templates outlined in Table 4. Using `QEPCAD` to solve the $\exists\forall$ constraint generated by Condition (C2) of Figure 4, we get the following values for a, b, c, d, e:

$$b = 10, \quad c = \frac{1}{2200}, \quad 20(d-a)^2 e < 11a^2.$$

Note that `QEPCAD` is used to eliminate the internal $\forall$ quantifiers and hence it generates a formula on a, b, c, d, e (shown above). Since V in Table 4 is not a sum of squares, we also need to satisfy Condition (C1). It is easily verified that this can be achieved by setting $d = a$. If we set $d = a$, then the condition $20(d-a)^2 e < 11a^2$ holds for all values of e and any nonzero value of a. Hence, the assumption $||\beta||^2 \leq e$ is trivially satisfied. Thus, we prove the property

$$\mathbb{G}(\mathbb{F}(x_e^2 \leq a)) \tag{10}$$

for any positive a. Note also that the proof of the above property implicitly shows that the weights L of the neural network are eventually bounded, whenever the error x_e exceeds a certain bound.

5 Conclusion

We presented preliminary results obtained from formally analyzing a model of direct neural net adaptive flight control. The model was analyzed for bounded stability. The bounded verification approach was used which is based on using templates for the witness functions and the assumptions. The $\exists\forall$ constraints resulting from the bounded verification approach were solved using `QEPCAD`.

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. A third direction for future work consists of analyzing indirect and hybrid adaptive flight control systems. 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.

Acknowledgments. We thank the anonymous reviewers for their comments.

References

1. NASA Ames Research Center. Matlab scripts for simulating neural net adaptive flight control. Personal Communication.
2. P. Cuijpers and M. Reniers. Lost in translation: Hybrid-time flows vs real-time transitions. In *Proc. 11th HSCC*, volume 4981 of *LNCS*, pages 116–129. Springer, 2008.
3. 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.
4. 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.
5. T. A. Henzinger. A theory of hyrid automata. In *Proc. 11th IEEE Logic in Comp. Sci. LICS*, pages 278–292, 1996.
6. 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.
7. Deepak Kapur. Automatically generating loop invariants using quantifier elimination. In *Deduction and Applications*, 2005.
8. 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.
9. 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.
10. 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.
11. 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.
12. A. J. van der Schaft and J. M. Schumacher, editors. *An introduction to hybrid dynamical systems*, volume 251 of *Lecture Notes in Control and Information Science*. Springer, 2000.