Formally Analyzing Adaptive Flight Control

Ashish Tiwari
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025

Supported in part by NASA IRAC NRA grant number: NNX08AB95A

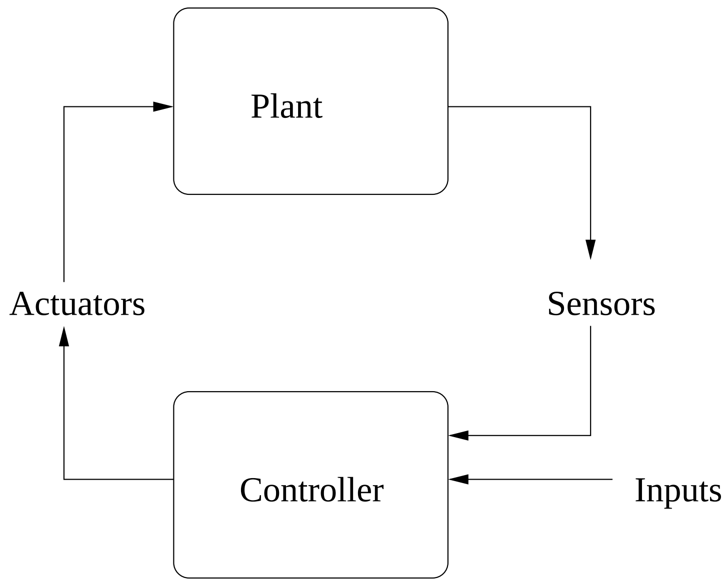
System Development

$$
\begin{array}{ccc}
\text{Design} & — & \text{Verify} \\
\downarrow & & | \\
\downarrow & & | \\
\downarrow & & | \\
\text{Implementation} & — & \text{Verify}
\end{array}
$$

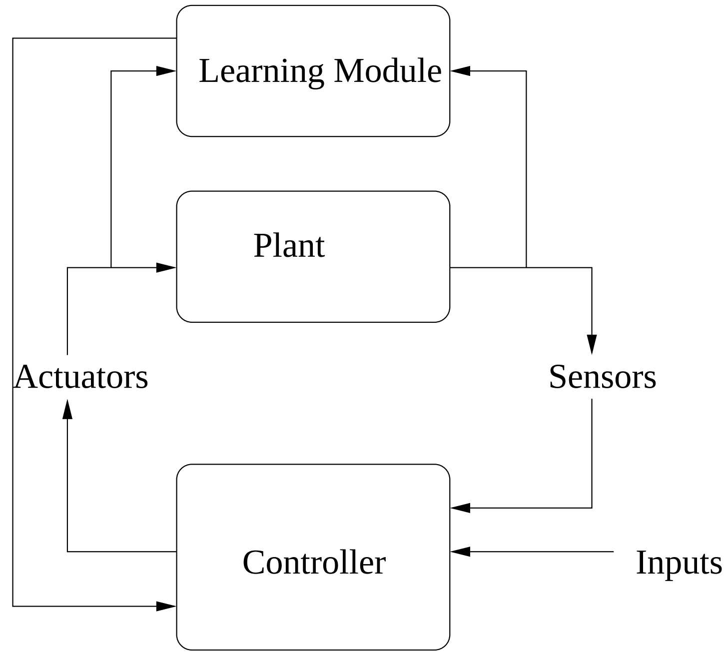
Focus here is on verification at the design phase of

Adaptive flight control systems

Adaptive Control Systems

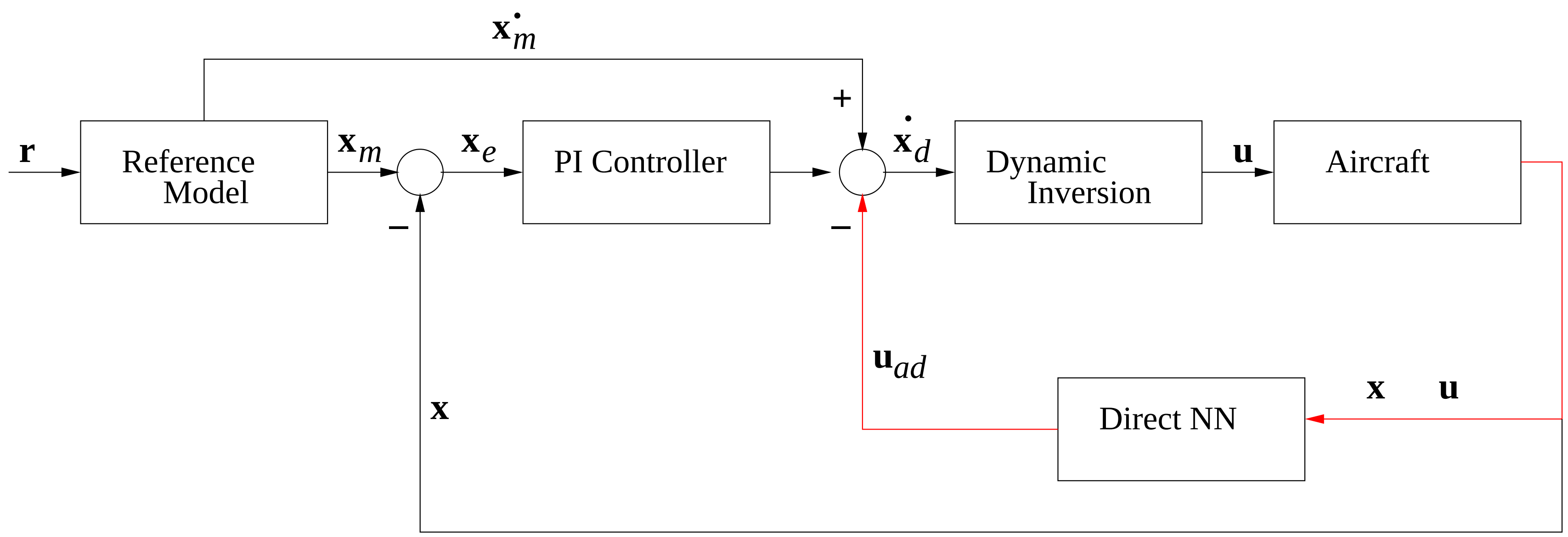


Simple Control System



Adaptive Control System

Direct NN Adaptive Flight Control



Adaptive: Additional red loop

To compensate for the unknown dynamics arising from aircraft damage

Verifying Adaptive System

Challenges:

- Unknown plant (aircraft) model
- Nonlinear functions (kernel functions)
- Unknown initial weights of the neural net
- Unknown assumptions
- Complexity of model: mixed discrete and continuous, dimension

Formal Verification

Formal verification gives correctness guarantees – for all possible behaviors

1. Build a model of the system
   (a) Model each component – controller, aircraft, NN
   (b) Model disturbances – nondeterminism, symbolic parameters
   (c) Specify the property
2. Formally verify the system

You verify what you model

Why Formal Verification?

Why use formal verification?

1. Alternative to doing simulation and testing
2. Equivalent to doing an analytic proof
3. Do a new proof, or machine check/validate a hand proof
4. Verify different safety and stability properties
5. Redo proofs if design is changed
6. Applies to both design and implementation
7. Helps in certification

Bounded Verification

Typical verification approaches–

- iterative over-approximation of the reachable set
- abstraction
- smart simulations

Bounded Verification is a different technique for Safety and Stability verification of Continuous and Hybrid dynamical systems

- Reduce verification problem to constraint solving
- Use modern constraint solvers to solve the constraint

Outline/Summary

1. Bounded Verification: Verification $\mapsto \exists\forall$ solving
2. Solving $\exists\forall$ formulas
3. Analyzing adaptive flight control
   - 3.1 Modeling Neural Network Direct MRAC
   - 3.2 Verifying stability and invariance properties of the model using the bounded verification technique

Sources for the Model:

- N. Nguyen and K. Krishnakumar, "*An optimal control modification to model-reference adaptive control for fast adaptation*", AIAA GNC 2008.
- Matlab scripts for simulating direct, indirect, and hybrid adaptive flight control (source: Stephen A. Jacklin, NASA Ames)

Part I: Bounded Verification

Bounded Verification

A generic approach for analysis of continuous and hybrid dynamical systems based on symbolic constraint solving

Key Observation: Verification = searching for right witness

Property	Witness
Stability	Lyapunov function
Safety	Inductive Invariant
Liveness	Ranking function
Controllability	Controlled Invariant

How to find the right witness?

Finding the Witness

Key idea: Bounded search for witnesses of a specific form

High-level outline of the procedure:

1. Fix a form (template) for the witness function
   Quadratic template: $ax^2 + by^2$
2. Existence of a witness (of the chosen form) is encoded as a constraint

$$\exists a, b : \forall x, y : ax^2 + by^2 \geq c \Rightarrow \frac{d}{dt}(ax^2 + by^2) < 0$$

3. Solve the constraint

Quick Introduction to Logic

Let $V(a,b,x,y) := ax^2 + by^2$

There exist values for a, b, c such that for all values of x, y, if $V(a,b,x,y) \geq c$, then $\dot{V} < 0$

$$\exists a, b, c : \forall x, y : V(a,b,x,y) \geq c \Rightarrow \frac{dV}{dt} < 0$$

Add requirement that a, b, c are positive

$$\exists a, b, c : a > 0 \wedge b > 0 \wedge c > 0 \wedge (\forall x, y : V(a,b,x,y) \geq c \Rightarrow \frac{dV}{dt} < 0)$$

Tarski's Result: These formulas can be solved

Safety Verification using Inductive Invariants

A discrete-time system always remains inside the set $Safe(\vec{x})$ of good states if there is an inductive invariant $Inv(\vec{x})$ such that

$$
\begin{array}{lrcl}
\texttt{Init}: & \forall\vec{x}: Init(\vec{x}) & \Rightarrow & Inv(\vec{x}) \\
\texttt{Ind}: & \forall\vec{x},\vec{x'}: Inv(\vec{x}) \wedge t(\vec{x},\vec{x'}) & \Rightarrow & Inv(\vec{x'}) \\
\texttt{Safe}: & \forall\vec{x}: Inv(\vec{x}) & \Rightarrow & Safe(\vec{x})
\end{array}
$$

Template: $\mathcal{I}nv(\vec{a},\vec{x})$

Generated Constraint:

$$
\begin{array}{ll}
\exists\vec{a}:\forall\vec{x},\vec{x'}: & (Init(\vec{x}) \Rightarrow \mathcal{I}nv(\vec{a},\vec{x})) \ \wedge \\
& (\mathcal{I}nv(\vec{a},\vec{x}) \wedge t(\vec{x},\vec{x'}) \Rightarrow \mathcal{I}nv(\vec{a},\vec{x'})) \ \wedge \\
& (\mathcal{I}nv(\vec{a},\vec{x}) \Rightarrow Safe(\vec{x}))
\end{array}
$$

Safety Verification: Continuous-Time

A continuous-time system $\dot{\vec{x}} = f(\vec{x})$ always remains inside the set $Safe(\vec{x})$ of good states if
there is an inductive invariant $\mathcal{I}nv(\vec{a}, \vec{x})$ such that

$$\begin{aligned} \exists \vec{a} : \forall \vec{x} : \qquad & (Init(\vec{x}) \Rightarrow \mathcal{I}nv(\vec{a}, \vec{x})) \ \wedge \\ & (\vec{x} \in \partial \mathcal{I}nv(\vec{a}, \vec{x}) \Rightarrow f(\vec{x}) \in \mathbf{T}\mathcal{I}nv(\vec{a}, \vec{x})) \ \wedge \\ & (\mathcal{I}nv(\vec{a}, \vec{x}) \Rightarrow Safe(\vec{x})) \end{aligned}$$

The middle condition can be formulated for polynomial systems as: $p \geq 0$ is inductive if

$$\forall (\vec{x}) : p(\vec{x}) = 0 \Rightarrow \vec{\nabla} p(\vec{x}) \cdot f(\vec{x}) \geq 0$$

Digression

Unsound, but sound variant and even relatively complete variants exist

$$
\begin{array}{ll}
(A1) & \texttt{Init} \Rightarrow p \geq 0 \\
(A2) & p = 0 \Rightarrow L_f(p) \geq 0 \\
(A3) & p \geq 0 \Rightarrow \texttt{Safe} \\
(A4) & p = 0 \Rightarrow \vec{\nabla} p \neq 0 \\
\hline
 & \texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}
\end{array}
$$

Figure 1: Sound, but incomplete, rule for safety verification of polynomial CDS $\texttt{CDS} := (\mathsf{X}, \texttt{Init}, f)$ and safety property $\texttt{Safe} \subseteq \mathbf{X}$.

Relatively complete

Bounded Stability Verification

$$\begin{array}{rrcl}
(S1): & \texttt{Init} & \Rightarrow & V \geq 0 \\
(S2): & V > 0 & \Rightarrow & \frac{dV}{dt} < 0 \\
(S3): & V \leq 0 & \Rightarrow & \phi \\
\hline
 & \texttt{Init} & \Rightarrow & \mathbb{F}(\phi)
\end{array}
\qquad
\begin{array}{rrcl}
(T1): & \neg\phi & \Rightarrow & V > 0 \\
(T2): & \neg\phi & \Rightarrow & \frac{dV}{dt} < 0 \\
\hline
 & true & \Rightarrow & \mathbb{G}(\mathbb{F}(\phi))
\end{array}$$

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

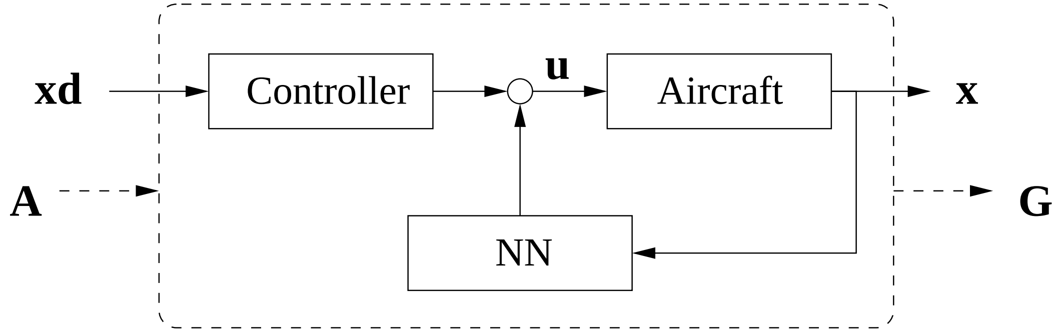
Proving Bounded Stability

Constraints can also encode that some function is a Lyapunov function.

Some systems may not be globally stable

We can also generate assumptions on the inputs (subset of the global state space) that will guarantee stability or safety

Idea: Use a template for the assumption



Pick Template for G: $\mathbf{V(x) = x^T x - k}$

Pick Template for A: $\mathbf{xd < a\,x}$

Exist(a,k): Forall(x): $\mathbf{x^T x - k > 0}$ **and** $\mathbf{xd < ax}$ **implies** $\mathbf{d/dt(x \quad x - k) < 0}$

Eliminate Forall(x)

Exist(a,k): Exist(λ): (...)

Solve for all variables

k = 60, a = 5, ...

(This proves bounded stability of the system)

Controllability Verification

Our approach can be used to synthesize controllers that preserve safety and/or stability

A continuous-time system $\dot{\vec{x}} = f(\vec{x}, \vec{u})$ can be made to remain inside the set $Safe(\vec{x})$ of good states if
there is an controlled inductive invariant $\mathcal{C}Inv(\vec{a}, \vec{x})$ such that

$$\begin{array}{ll} \exists \vec{a} : \forall \vec{x} : & (Init(\vec{x}) \Rightarrow \mathcal{C}Inv(\vec{a}, \vec{x})) \ \wedge \\ & (\vec{x} \in \partial \mathcal{C}Inv(\vec{a}, \vec{x}) \Rightarrow \exists \vec{u} : f(\vec{x}, \vec{u}) \in \mathbf{T}\mathcal{C}Inv(\vec{a}, \vec{x})) \ \wedge \\ & (\mathcal{C}Inv(\vec{a}, \vec{x}) \Rightarrow Safe(\vec{x})) \end{array}$$

Similarly for controlled Lyapunov function

Overview of Bounded Verification

Given continuous dynamical system, and optionally property $Safe$:

- Guess a template $\mathcal{I}nv(\vec{a}, \vec{x})$
  - For stability, this will be a Lyapunov function
  - For safety, this will be an inductive invariant
- Guess a template for the assumption $\mathcal{A}(\vec{b}, \vec{x})$ (if any)
- Generate the $\exists\forall$ verification condition: $\exists \vec{a}, \vec{b} : \forall \vec{x} : \mathcal{A}(\vec{b}, \vec{x}) \wedge \cdots \Rightarrow \phi$
  - Formula ϕ states that $\mathcal{I}nv$ is a Lyapunov fn/inductive invariant
- Solve the formula to get values for $\vec{a}$ and $\vec{b}$

Related Work

The bounded verification approach encompasses

- Template-based invariant generation (Sankaranarayanan et al., Kapur)
- Barrier certificates (Prajna et al.)
- Constraint-based approach for verification (Gulwani et al.)

Bounded verification is the dual of bounded falsification
(aka bounded model checking)

The real problem is
deciding $\exists\forall$ formulas over the reals

Part II: Solving $\exists\forall$ formulas

Solving $\exists\forall$ formulas

Bounded verification: verification of hybrid systems $\mapsto$ checking validity of $\exists\vec{u} : \forall\vec{x} : \phi$

When ϕ is over polynomials, this is decidable (e.g. QEPCAD)

More practically, use heuristics to decide $\exists\vec{u} : \forall\vec{x} : \phi$

1. Eliminate $\forall$: $\exists\vec{u} : \forall\vec{x} : \phi \mapsto \exists\vec{u} : \exists\vec{\lambda} : \phi'$
2. Search for $\vec{u}$ and $\vec{\lambda}$ over a finite domain using SMT (bit vector) solver

Step 1: $\exists\forall$ to $\exists$

For linear arithmetic, Farkas' Lemma eliminates $\forall$

$\forall \vec{x} : p_1 \geq 0 \wedge p_2 \geq 0 \Rightarrow p_3 \geq 0$, iff
$\exists \vec{\lambda} : p_3 = \lambda_1 p_1 + \lambda_2 p_2 \;\wedge\; \lambda_1 \geq 0 \;\wedge\; \lambda_2 \geq 0$

For nonlinear, we can still use this and be sound, but incomplete

We can partially regain completeness by using Positivstellensatz

Step 2: $\exists$ to Bit-Vectors

Farkas Lemma/Posit. : $\exists\forall \mapsto \exists$

Solving the $\exists$ formula

One approach: Search for solutions in a finite range using bit-vector decision procedures

$$
\begin{aligned}
& \exists u \in \mathbb{R} : (u^2 - 2u = 3 \wedge u > 0) \\
\Leftarrow \quad & \exists u \in \mathbb{Z} : (u^2 - 2u = 3 \wedge u > 0) \\
\Leftarrow \quad & \exists u \in \mathbb{Z} : (-32 \leq u < 32 \ \wedge \ u^2 - 2u = 3 \wedge u > 0) \\
\Leftarrow \quad & \exists \vec{b} \in \mathbb{B}^6 : (u * u - 2 * u = 3 \wedge u > 0)
\end{aligned}
$$

We use Yices to search for finite bit length solutions for the original nonlinear constraint

$$\vec{b} = 000011$$

Overall Approach

Given hybrid system HS and optionally property Safe:

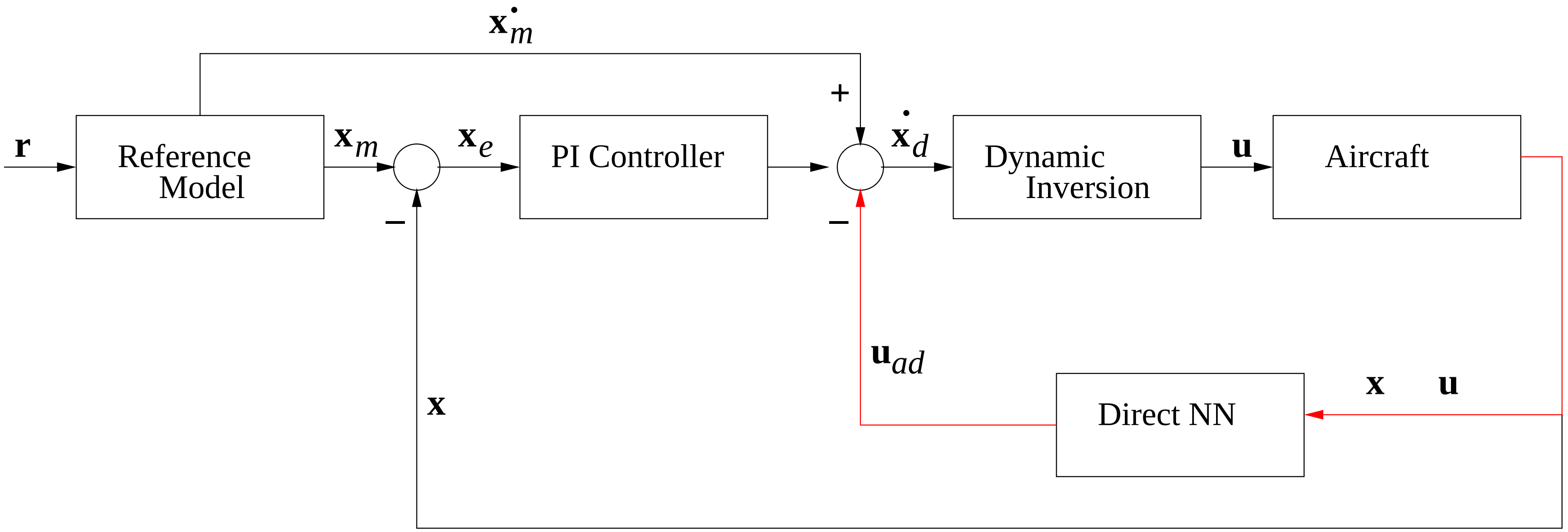
- Guess a template for witness $\mathcal{I}nv(\vec{u}, \vec{x})$
- Generate the verification condition: $\exists \vec{u} : \forall \vec{x} : \phi$
- Solve using either QEPCAD or
  - Eliminate $\forall$ using Farkas' Lemma: $\exists \vec{u} : \exists \vec{\lambda} : \psi$
  - Guess sizes for $\vec{u}, \vec{\lambda}$: $\exists \vec{bv_u} : \exists \vec{bv_\lambda} : \psi'$
  - Ask Yices to search for solutions
- If a satisfying assignment is found, system proved safe

Part III.I

Modeling NN Direct

Model Reference Adaptive Control

NN Direct Model Reference Adaptive Control



Sources:

- N. Nguyen and K. Krishnakumar, "*An optimal control modification to model-reference adaptive control for fast adaptation*", AIAA GNC 2008.
- Matlab scripts for simulating direct, indirect, and hybrid adaptive flight control (source: Stephen A. Jacklin, NASA Ames)

Step 1: Modeling Direct MRAC

$\vec{x}$: 3×1 vector of roll, pitch, and yaw rates of the aircraft.

$\vec{u}$: 3×1 vector of aileron, elevator, and rudder inputs.

$\vec{z}$: 3×1 trim state vector of angle of attack, angle of sideslip, and engine throttle.

The dynamics of the aircraft are given by

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

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

Step 1: Modeling Direct MRAC

We tried to build a continuous dynamical system model

State space: $x_m, intx_e, x, L, \beta, f$

$$
\begin{array}{rcl}
\dot{x_m} & = & A_m(x_m - r) \\
int\dot{x}_e & = & x_m - x \\
\dot{x} & = & A_m(x_m - r) + K_p(x_m - x) + K_i intx_e - L'\beta + f \\
\dot{L} & = & -\Gamma\beta(intx_e^T K_i^{-1} + (x_m - x)^T K_p^{-1}(I + K_i^{-1})) \\
\dot{\beta} & = & \ldots \\
\dot{f} & = & \ldots
\end{array}
$$

Constants	:	Γ, K_p, K_i, A_m,
Unknown/Symbolic Parameters	:	r, f, $\dot{f}$

Step 1: Modeling Direct MRAC

r	commanded value for x
x_m	desired value for x, calculated using reference model
x	actual value for 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, $L'\beta$ = adaptive control term
f	Damaged dynamics, $f = \dot{x} - \dot{x}_u$
u_e	$K_p x_e + K_i intx_e$
$\dot{x}_d$	$\dot{x_m} + u_e - uad$
$\dot{L}$	weight update / neural net learning

Step 1: Modeling Direct MRAC: Issues

Dynamics for β: $\dot{\beta} = \ldots$

- There are two options here:

  Option 1. Use β from the NASA Matlab scripts

  Option 2. Leave β as unknown symbolic parameters

- If we use Option 1

  There is an algebraic loop on u: $u(t)$ depends on $u(t)$

  Leads to complications – not pursued further.

- If we use Option 2

  Analysis independent of β

  Need assumption on β (to capture damaged dynamics f)

  Used in [NguyenKrishnakumar08]

Step 1: Modeling Direct MRAC: Issues

Dynamics of f: $\dot{f} = \ldots$

- Dynamics of damaged aircraft:

$$\dot{x} \quad = \quad A_u\vec{x} + B_u\vec{\sigma} + F_u\vec{u} + f(\vec{x}, \vec{\sigma}, \vec{u})$$

  f is unknown

- $\dot{f}$ is also unknown
- We leave f and $\dot{f}$ as unknown symbolic parameters
- We wish to prove properties of the system for any f, $\dot{f}$
- Which is not possible, hence need assumptions

We will verify . . . assuming that . . .

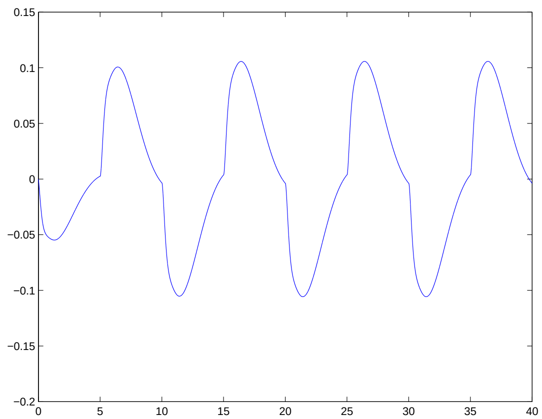
Step 1: Final Model

$$
\begin{aligned}
\dot{x_e} &= -K_p x_e - K_i intx_e + L'\beta - f \\
in\dot{t}x_e &= x_e \\
\dot{L} &= -\Gamma\beta(intx_e^T K_i^{-1} + (x_m - x)^T K_p^{-1}(I + K_i^{-1})) \\
\dot{\beta} &= f_1 \\
\dot{f} &= f_2
\end{aligned}
$$

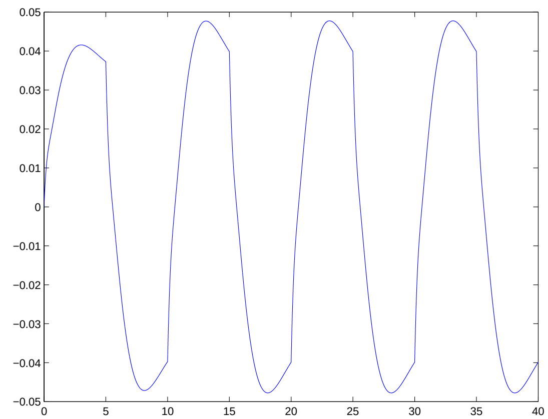
state variables	$x_e, intx_e, L, \beta, f$
unknown parameters	f_1, f_2
fixed parameters	Γ, K_p, K_i

Step 1: Simulating the Original Model

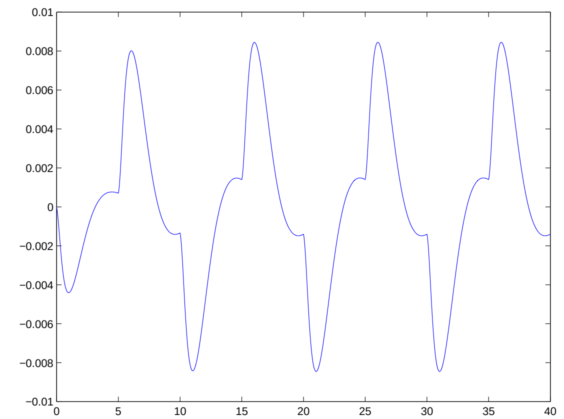
Standard PI Controller without adaptation:



Roll rate



Pitch rate



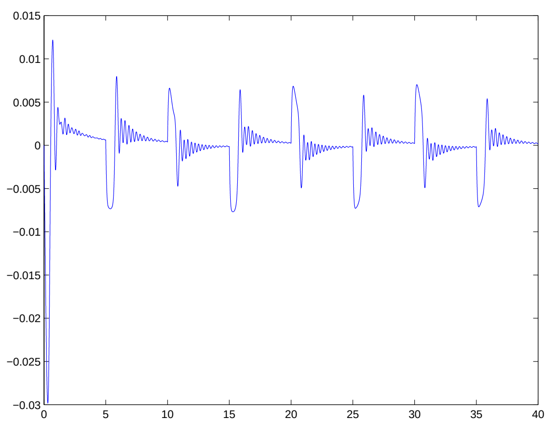
Yaw rate

Pitch command : Roll and Yaw respond bcos of aymmetric damage

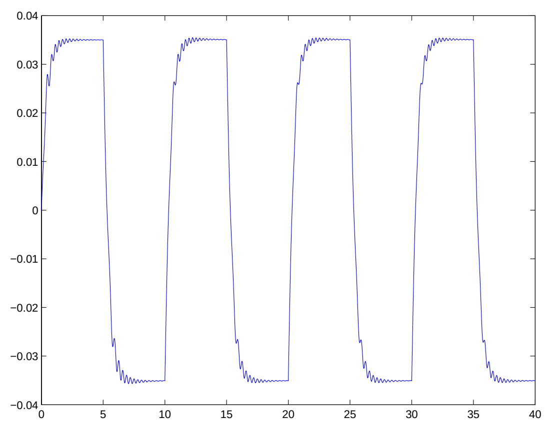
Response unacceptable due to excessive roll and yaw rates

Step 1: Simulating the Model with MRAC

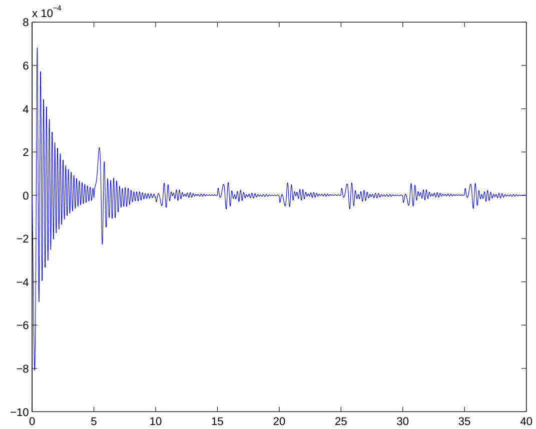
Standard MRAC Controller using learning rate $\Gamma = 10^4$:



Roll rate



Pitch rate



Yaw rate

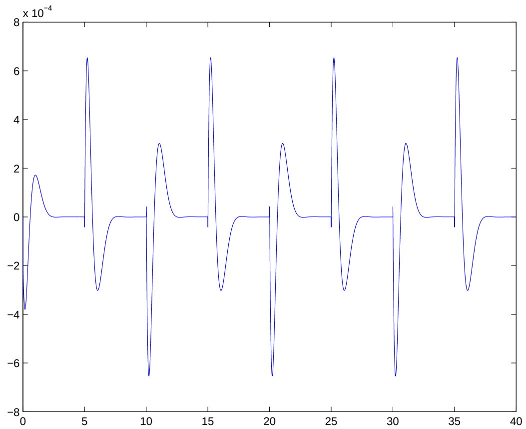
Pitch command : Roll and Yaw respond bcos of aymmetric damage

Tracking performance improves drastically

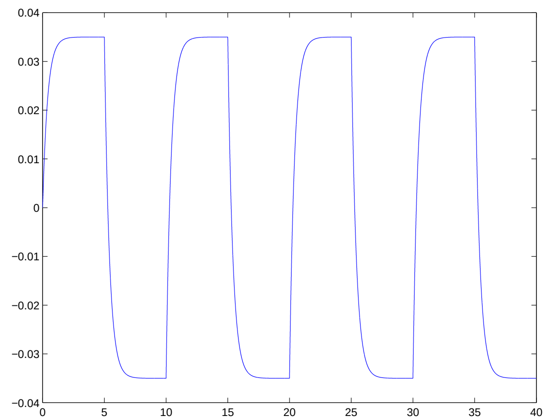
High-frequency oscillations in yaw, lesser in pitch, roll channel

Step 1.5: Simulating the Original Model

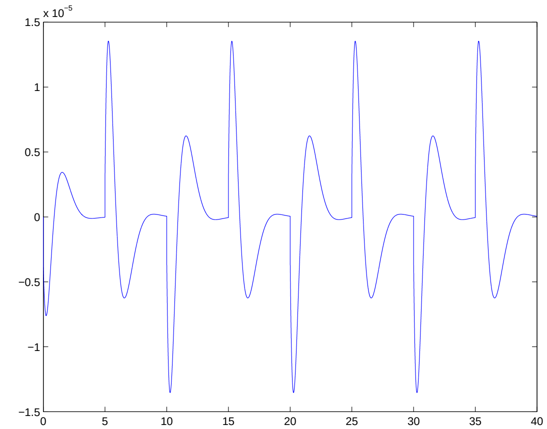
Adaptation based on estimating f:



Roll rate



Pitch rate



Yaw rate

Pitch command : Roll and Yaw respond bcos of aymmetric damage

Tracking performance improves drastically

Any High-frequency oscillations?

Part III.I

Verifying NN Direct

Model Reference Adaptive Control

Step 2: Verifying the Model

We first verify that error remains bounded assuming that the NN works properly

Assumption	$(u_{ad} - f)$ is bounded	Template: $\|\|L'\beta - f\|\|^2 \leq a$
Assumption	$\|\|x_e\|\|$ exceeds bound	Template: $\|\|x_e\|\|^2 > c$
Guarantee	Exists a Lyapunov function	Template: $\|\|x_e\|\|^2 + b\|\|intx_e\|\|^2$

Generated formula: $\exists a, b, c : \forall x_e, intx_e, L, \beta, f : \ldots$

Values computed by the constraint solver: $b = 10, 25c > a > 0$

Assuming $L'\beta - f$ is bounded, the error x_e eventually remains bounded – irrespective of $\beta, f, L, \dot{f}, \ldots$

Step 2: Verifying the Model

The above property holds even under a different assumption.

Assumption	$\frac{\|\|x_e\|\|}{\|\|u_{ad}-f\|\|}$ exceeds bound	$\|\|x_e\|\|^2 > c\|\|u_{ad} - f\|\|^2$
Guarantee	Exists a Lyapunov function	$\|\|x_e\|\|^2 + b\|\|intx_e\|\|^2$

Generated formula: $\exists b, c : \forall x_e, intx_e, L, \beta, f : \ldots$

Values computed by the constraint solver: $b = 10, 25c > 1$

The error x_e always eventually drops below a constant factor of the NN approximation error – irrespective of $\beta, f, L, \dot{f}, \ldots$

Step 2: Verifying the Model

Can we show that the weights L also eventually remain bounded ?

Assume $f = L^{*\prime}\beta$

Assume	β is bounded	$\|\|\beta\|\|^2 \leq e$
Assume	$\|\|x_e\|\|$ exceeds bound	$\|\|x_e\|\|^2 > a$
Prove	Exists an invariant	$\|\|x_e\|\|^2 + b\|\|intx_e\|\|^2 + c\|\|L - L^*\|\|^2 \leq d$

Generated formula: $\exists a, b, c, d, e : \forall x_e, intx_e, L, \beta, f : \ldots$

Values computed by the constraint solver:
$b = 10, c = \frac{1}{2200}, 20(d-a)^2 e < 11a^2$

When $||x_e||^2 > a$, then the set $||x_e||^2 + b||intx_e||^2 + c||L - L^*||^2 < d$ is an invariant – assuming β^2 is bounded by e.

Step 2: Verifying the Model

Can we show that the weights L also eventually remain bounded ?

Assume $f = L^{*\prime}\beta$

Assume	$(u_{ad} - f)$ is bounded	$\|\|L'\beta - f\|\|^2 \leq e$
Assume	$\|\|x_e\|\|$ exceeds bound	$\|\|x_e\|\|^2 > a$
Prove	Exists an invariant	$\|\|x_e\|\|^2 + b\|\|intx_e\|\|^2 + c\|\|L - L^*\|\|^2 \leq d$

Generated formula: $\exists a, b, c, d, e : \forall x_e, intx_e, L, \beta, f : \ldots$

Values computed by the constraint solver:
$b = 10, c = \frac{1}{2200}, (d - a)e < 1210a^2$

When $||x_e|| > a$, then the set $||x_e||^2 + b||intx_e||^2 + c||L - L^*||^2 < d$ is an invariant – assuming $(L - L^*)'\beta$ is bounded.

Step 2: Verifying the Model: Issues

- Constraint solver: $\exists\forall$ formulas over the reals
  - Our implementation: fast, but incomplete
    - ⋆ Poor in handling squares
    - ⋆ Can not solve all the constraints
  - QEPCAD: slow and unreliable, but complete
- Automation of template generation
  - difficult in general
  - possible for NN adaptive flight control systems
- Automating model extraction

Other Case Studies

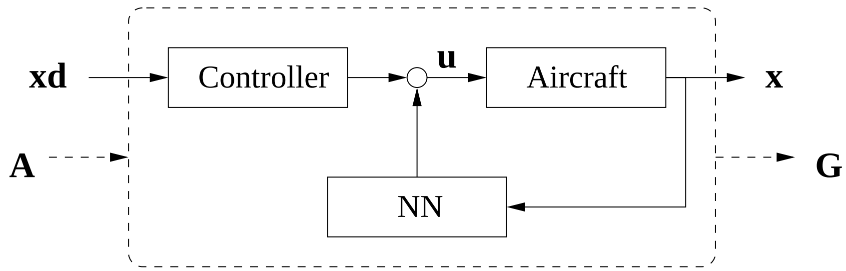
The same approach used to verify bounded stability of a flight controller from:

T. Lee and Y. Kim, " Nonlinear adaptive flight control using backstepping and neural networks controller", J. of Guidance, Control, and Dynamics:24(4), 2001.

The method has also been used to verify traditional control systems and other hybrid dynamical systems

- adaptive cruise control in automobiles
- models from systems biology
- human blood glucose metabolism model

Recap: Overall Approach



Pick Template for G: $\mathbf{V(x) = x^T x - k}$

Pick Template for A: $\mathbf{xd < a\,x}$

Exist(a,k): Forall(x): $\mathbf{x^T x - k > 0}$ and xd < ax implies d/dt(x   x − k) < 0

Eliminate Forall(x)

Exist(a,k): Exist(λ): (...)

Solve for all variables

k = 60, a = 5, ...

(This proves bounded stability of the system)

Part IV

Discussion and Conclusion

What is novel in the technique?

Computer Science

- The template+constraint-solving approach is different from the usual verification approaches
  - reachability
  - abstraction
- Bounded Falsification (BMC) vs. Bounded Verification

Control

- The approach is standard, but the novelty is in generating more precise constraints and using symbolic solvers for testing their feasibility

Why is the technique so effective?

- This is the classical approach – only slightly modified to
  - generate more precise constraints
  - that can be non-convex
  - solved using modern solvers such as
    - ⋆ fast constraint solvers called SMT solvers
    - ⋆ complete symbolic solver like QEPCAD

  replacing optimization by feasibility or satisfiability
- Systems have several invariants/Lyapunov functions – that can be searched using few templates
- Correct systems have simple witnesses
- Robust technique does not require any careful tuning or a smart user
  Handles unknown parameters

Future Work

- Modeling and Analysis
  - Complete analysis of NN direct MRAC
  - Analyze other variants of direct MRAC
  - Analyze indirect and hybrid NN adaptive flight control
- Add automation for template generation for this specific domain
- Improve automation for constraint solving

Tool

We have generic prototype implementations for:

- Generating constraint from continuous dynamical model: Given a CDS and templates, generates an $\exists\forall$ constraint
- Eliminating $\forall$ quantifier: Given an $\exists\forall$ constraint, eliminates the $\forall$ and return an $\exists$ formula
- Solver for $\exists$ formulas
- Off-the-shelf tool QEPCAD

Tool Development: Issues

- Constraint generation only for safety verification
  - Need constraint generation for stability verification
  - May need a careful study of the underlying proof rule
- Extracting CDS model from a more intuitive front-end description ?
- Solver for $\exists\forall$ constraints
  - Need to balance completeness and efficiency
  - Domain-specific heuristics

Conclusion

- We are verifying designs of NN adaptive flight control systems
- The bounded verification approach
  - reduces verification to $\exists\forall$ constraint solving