Advanced Verification and Validation Procedures and Tools for the Certification of Learning Systems in Aerospace Applications

Stephen Jacklin
NASA Ames Research Center

Aerospace Control and Guidance Systems Committee Meeting No. 98
October 11-13, 2006
Williamsburg, VA
Presentation Outline

- Adaptive control system architecture
- Learning systems and V&V problems
- V&V methodologies, tools, and procedures for (partial) solution
  - Emphasis on methods that may be helpful toward meeting future certification requirements
Why Adaptive Flight Control?

- Adaptive flight control systems are critically needed
  - To retain vehicle handling qualities in the event of damage
  - To maintain robust control in unknown or changing conditions
  - To enable autonomous flight control of aircraft and spacecraft
  - To adapt to changing mission requirements

- Many compelling applications
  - Neural adaptive flight control
  - Fault tolerant flight control
  - Damage adaptive control
Verification and Validation of Adaptive Controllers

- Adaptive controllers won’t be part of the future unless they can be proven to be highly safe and reliable

- Rigorous methods for adaptive system V&V are needed
  - To ensure control system failures do not occur
  - To ensure control systems perform their intended function
  - To ensure against unintended or undesired functionality

- FAA certification regulations do not yet address adaptive systems
  - RTCA DO-178C still under development

- V&V of adaptive control systems for aerospace applications complicated by hybrid systems
Outer-loop controller (Real-Time Operating System) controls lower-level computers and controllers
Inner-Loop Controller Architecture

- These are continuous adaptive control systems

\[
\frac{dx}{dt} = Ax + Bu, \quad y = Cx + Du
\]

\[
u = K_P \varepsilon + K_I \int \varepsilon dt + K_D \left( \frac{d\varepsilon}{dt} \right)
\]

Diagram:
- Desired State, X
- Measured State, X'
- Error, \( \varepsilon \)
- PID Controller
- Feedback Gains
- Control Surfaces
- Sensors
- Y
- U
Many Ways to Introduce Learning

Control augmentation and system identification are two

Desired State, $X$

Measured State, $X'$

Error $\varepsilon$

LQG Controller

$U$

$L_{\text{Aug}}$

Learning algorithm for Adaptive Control

Transfer Matrix Elements

Learning algorithm for System Identification

Feedback Gains

Control Surfaces

Sensors

$X$, $\varepsilon$, $U$

$Y$

$Y$

$U$

$Y$
V&V Problems for Learning Systems

- Learning systems face some special problems for certification

- **Continuous System Problems**
  - Non-determinism
  - Learning accuracy
  - Learning stability

- **Finite State System Problems**
  - Shear size of the programs ( > 20 million SLOC)
  - Coverage testing
  - Regression testing of modified software
Non-Determinism

- An important certification concern is that fielded software duplicate its behavior as tested

- A reasonable requirement for adaptive systems comprised of mathematical algorithms
  - Should behave in a predictable manner if started from the same initial conditions and then given a specified input history
  - But, a valid concern is the need to restart controller in flight

- Only partially reasonable to account for non-determinism of the environment
  - Adaptive systems intentionally designed to maintain controller performance in unforeseen situations
    - Vehicle damage
    - Unexpected operating conditions
Most Controllers Are Deterministic

The learning environment is non-deterministic

Damage or Failure

Pilot Inputs

Control Law

Inverse Plant Model

Neural Network

Control Augmentation

Measurements
Health Monitoring Can Help

Executive Computer (RTOS) → Real-time Displays → Mission Planning → Health Management

Flight Control

Pilot Inputs → Control Law → Inverse Plant Model

Neural Network → Control Augmentation

Measurements

Make Controllability Assessment
Learning Accuracy Issues

- Most fundamental problem in adaptive control is “Can the inner-loop systems learn?”
- Many simple gradient-based search methods abound (steepest descent, back propagation)
- Methods that use 2nd derivative (Hessian) are faster (Conjugate Gradient, Levenberg-Marquardt)
- Currently, no analytical or formal method exists to guarantee learning convergence to the globally optimal values within a given time
Local Convergence Problem

- Learning process may converge to a local optimum rather than the global optimum

![Diagram showing local minima and a global minimum.](image)
Subtle Convergence Problems

- Suppose we have an inverse controller whereby we desire to adaptively identify the inverse transfer matrix \( T \) at each step

\[
\Delta u = [T] \Delta y
\]

\[
T_{k+1} = T_k + \text{Update}
\]

\[
\text{Update} \approx K_{\text{Learn}} (\Delta u_k - [T_k] \Delta y_k)
\]

- What happens as

\[
(\Delta u_k - [T_k] \Delta y_k) \rightarrow \text{Zero}\
\]
Sensor Noise Corrupts Learning

- **Answer:** although initially good for control, measurement noise soon corrupts the system identification process if left on-going
  - Small changes in the output, when dominated by noise, obscure the effect of small control changes

\[
(\Delta u_k - [T_k](\Delta y_k + \text{noise}))
\]

- **Persistent excitation or disabling of learning when not required, or both, are two possible solutions**
Inner-loop Controller Stability

- Stability of adaptive inner-loop controllers is well understood
  - Bode, Root Locus, Nyquist Chart, Nichols, etc.

- Stability of controllers with learning is a more difficult problem
  - Liapunov theory can show learning errors are bounded, but applying Lyapunov theory is difficult
  - Liapunov theory can’t explicitly calculate the maximum learning error bounds from theory without incorporation of the real, detailed system dynamics
Simulation Remains a Vital Step for Convergence and Stability Testing

- Traditional V&V starts with analysis of convergence and stability, but then usually rapidly transitions to simulation

<table>
<thead>
<tr>
<th>Model Fidelity</th>
<th>Simulation Type &amp; Test Bed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Desk Top Computer (Matlab-Simulink)</td>
</tr>
<tr>
<td>Low-Medium</td>
<td>Work Station (nonlinear models)</td>
</tr>
<tr>
<td>Medium</td>
<td>Simulation with target flight computer</td>
</tr>
<tr>
<td>Medium-High</td>
<td>Hardware-in-the-loop (cockpit + FC)</td>
</tr>
<tr>
<td>Medium-High</td>
<td>Aircraft-in-the-loop simulator (ground)</td>
</tr>
<tr>
<td>High</td>
<td>Motion-based simulation</td>
</tr>
</tbody>
</table>
Simulation is a Useful Tool

- Explore initial concept feasibility studies
- Evaluate and compare learning algorithms
- Find stable learning rule update gains
- Evaluate overall controller stability
- Determine how much learning happens at each step of the simulation
- Evaluate learning convergence tendencies and convergence speed
- Evaluate ad hoc fixes that are difficult to analyze
- Evaluate human-machine interactions
Integrated Simulation Labs

NASA Langley SAFETI Laboratory
Problems with Simulation and Testing

- Testing can never prove the absence of errors
  - Totally relies on the engineer to identify all test scenarios

- Providing adequate test case coverage is time-consuming and hard to prove
  - DO-178B requires complete MC/DC testing to evaluate all possible outcomes given all possible inputs

- High-fidelity simulation gets progressively more expensive as fidelity increases
  - Cost and lack of time may cause some desired code modifications to be rejected, and code function limited instead

What else can be done?
Look at the Software Development Process

System Requirements → System Architectural Design → Software Requirements Analysis

Software Architectural Design → Software Detailed Design → Software Coding

Software Requirements Analysis → Software Qualification Testing → Software Integration

Software Integration → Software Unit Testing

System Qualification Testing → System Integration

Testing of Software Code Against the Requirements

Progressive Transformation Of Requirements Into Code

Lifecycle Phases
What Goes Wrong?

- Over Cost & Behind Schedule
- Late Discovery of Requirements Issues
- System tests
- Integration test
- Component test
- Unit test
- Complex & Detailed Requirements for Implementation and Test

- Requirements Creep & Bloat
- Architecture Loses Coherence
- Requirements Development
- Architecture and Design
- Implementation

- $$$$$
- $$$
- $$
- $$
- $$$

ACGSC Meeting No. 98 in Williamsburg, VA
Automated V&V Processes

- System Requirements
- System Architectural Design
- Software Requirements Analysis
- Software Architectural Design
- Software Detailed Design
- Software Coding
- Software Unit Testing
- System Integration
- Software Qualification Testing
- System Qualification Testing
- Program Synthesis
- Model Checking & Comp. Verification
- Advanced Testing
- Formal Methods
What is Model Checking?

- Programs represented as individual state machines with V&V properties specified in temporal logic
  - exhaustively explores all executions in a systematic way
  - handles millions of combinations – hard to perform by humans
  - reports errors as traces and simulates them on system models
Model Checking

- Model checking programs can explore all reachable (finite) states (JPF2, NuSMV, SPIN)
  - Good tool for mode logic V&V of outer-loop controller
  - Can analyze up to about $10^{120}$ states (paths)
  - Very useful for analyzing multi-threaded programs

- Symbolic analysis of code allows analysis of all feasible execution paths for any valid input data
  - Inputs are symbolic and hence covers all concrete inputs

- Inputs are constrained by the code structure rather than by the tester’s imagination

- A model allows rapid V&V of software patches and revisions

- Model checking may be extended to do V&V continuous domain of adaptive systems
Compositional Verification

- Model checking by itself does not scale well due to the large computer memory requirements

- Compositional verification allows large systems to be logically divided at their interfaces
  - System modeled using language such as UML 2.0
  - Each subsystem can be checked individually
  - All interfaces are verified in the process

- A large benefit is that integration errors are detected early - at the design phase
  - Hundreds of pages of requirements can be analyzed for early detection of defects, prior to coding
  - Allows many problems to be identified up front as a result of just doing the modeling
  - Model facilitates quick analysis of software upgrades
Program Synthesis: From V to Y

- Code generation can be used throughout the software process
  - Model analysis
  - Coding (multiple platforms)
  - Testing (continuous domain)
- Generate efficient and documented code
- Automated support for safety certification
- Reduce schedule, costs, errors
Automatic Generation of Certified Code

- These tools can generate adaptive software programs that are easier to certify
- AutoFilter for Kalman Filter generation
  - Ewen Denney & Bernd Fischer – NASA Ames
- Safety Critical Application Development Environment (SCADE) - Esterol
  - Used for auto-pilot code generation from precise formal rules and specifications
  - Also generates readable and traceable code
- Boeing Zbra compiler
  - V. Santhanam, Boeing Wichita
  - Subset of Ada language for safety-critical applications
  - Built-in anomaly checker to look for real-time code problems
Automated Code Review

- Static analysis is a formal method used to look for programming errors in an automated way
- Coverity, Polyspace, Parasoft, Clint, UNO, CGS, …..
- Can catch a lot of relatively simple mistakes
  - Buffer over-runs -- Over-flows
  - Un-initialized variables -- Unreachable code
  - Out-of-bounds array ref -- Mixed mode comp
  - Wrong use of formal parameters
- Very fast, but can generate many false-positives
Automated Methods Should Pay for Themselves

- Automated Test Generation
- Early Modeling and Validation
- Requirements Development
- Design Verification
- Architecture and Design
- Verifiable Program Synthesis
- Implementation (by contractor)
- Technology Insertion Here
- System tests
- Integration test
- Component test
- Unit test
- Cost/Schedule Savings Here

ACGSC Meeting No. 98 in Williamsburg, VA
Tools for In-Flight Software Assurance

- Health monitoring likely to be important
- Rule Extraction methods (NNRules, RuleX)
  - A type of safety wrapper
  - English readable rules about the trained neural network (IF \{cond 1 AND cond 2 \ldots\} THEN Result)
  - Shown useful for pre-trained neural networks
- NASA Ames Confidence Tool is an example
  - Based on Bayesian probability theory
  - Looks at variance of neural network parameters to judge functionality of network \ldots without knowing correct value of network weights
The Confidence Tool, based on a Bayesian approach, provides a measure of how well the neural network is performing at the moment.
Confidence Tool Results

Pitch Axis Confidence Metric

HIGH CONFIDENCE

2 Degree Left Stabilator Failure

SMALL CONTROL ERRORS
Development of Process Guides

- Necessary to ensure the methodologies developed for learning systems can become a well-defined process
  - Critical in shaping new certification requirements, e.g., DO-178C, that may address adaptive systems
  - Document a body of knowledge with application examples and lessons learned

- Should be comprehensive
  - Outer-loop finite state executive programs
  - Inner-loop learning controllers
  - Clear guidance on tool utilization and options / benefits
Summary Points

- V&V tools and procedures are being developed that may be used to address critical certification issues
  - Simulation remains the backbone of verification, but …
  - Automated tools for finite state executives and continuous systems are being developed, and
  - Tools to prove learning convergence and stability are available
  - Health management critically needed for learning systems

- More work remains to be done
  - Application of model checking to outer-loop controllers
  - Extend model checking to continuous systems
  - Pursue development of synthesis tools for certification
  - Development tools to prove on-line software assurance
  - Develop certification process guides for learning systems
Contact Information

Stephen Jacklin
Intelligent Systems Division

NASA Ames Research Center
Moffett Field, CA
Mail Stop: 269-2

Phone: (650) 604-4567

Email: stephen.a.jacklin@nasa.gov