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

Stephen A. Jacklin*, Johann M. Schumann†, and Pramod P. Gupta‡
NASA Ames Research Center, Moffett Field, CA, 94035

and

Michael Richard§, Kurt Guenther**, and Fola Soares††
NASA Dryden Flight Research Center, Edwards, CA, 93523

Adaptive control technologies that incorporate learning algorithms have been proposed to enable automatic flight control and vehicle recovery, autonomous flight, and to maintain vehicle performance in the face of unknown, changing, or poorly defined operating environments. In order for adaptive control systems to be used in safety-critical aerospace applications, they must be proven to be highly safe and reliable. Rigorous methods for adaptive software verification and validation must be developed to ensure that control system software failures will not occur. Of central importance in this regard is the need to establish reliable methods that guarantee convergent learning, rapid convergence (learning) rate, and algorithm stability. This paper presents the major problems of adaptive control systems that use learning to improve performance. The paper then presents the major procedures and tools presently developed or currently being developed to enable the verification, validation, and ultimate certification of these adaptive control systems. These technologies include the application of automated program analysis methods, techniques to improve the learning process, analytical methods to verify stability, methods to automatically synthesize code, simulation and test methods, and tools to provide on-line software assurance.

I. Introduction

Highly advanced adaptive control systems are needed to fulfill the present and future aerospace needs of the nation. Adaptive control technologies that incorporate learning algorithms have been proposed to enable automatic flight control and vehicle recovery, autonomous flight, and to maintain vehicle performance in the face of unknown, changing, or poorly defined operating environments. For civil aviation, adaptive control systems have been proposed that use learning to recover loss of vehicle control due to sudden aircraft damage or component failure.[1-3] For robotic applications, the ability to learn gives adaptive control systems greater capability to adapt to changing mission requirements after deployment. Adaptive control systems have virtually unlimited applications for NASA space exploration applications, including mated flight vehicle coordination, docking, and control of autonomous robots, flyers, and satellites.[4,5]

Because most of these applications are in safety-critical areas, it is obvious that adaptive control systems with learning systems will never become part of the future unless it can be proven that this software is highly safe and reliable. Rigorous methods for adaptive software verification and validation must be developed by NASA and others to ensure that control system software failures will not occur, to ensure the control system functions as

* Computer Scientist, Computational Sciences Division, M/S 269-2, Senior Member AIAA.
† Computer Scientist, RIACS/NASA Ames, M/S 269-3.
‡ Senior Scientist, QSS/NASA Ames, M/S 269-3.
§ Aerospace Engineer, Controls and Dynamics Branch, M/S 4840D, Member AIAA.
** Research Engineer, AS&M, M/S 4830D, Member AIAA.
†† Research Engineer, Contek Research, M/S 4830D, Senior Member AIAA.

required, to eliminate unintended functionality, and to demonstrate that certification requirements can be satisfied. To help bridge this gap, NASA and others are conducting software reliability research aimed at developing usable procedures and methods to verify the reliability of adaptive control system software employing learning algorithms.

The organization of the remainder of this paper is as follows: section 2 provides a cursory look at adaptive control system architecture to identify the general structure. Section 3 will present some of the major problems that arise in the use of learning algorithms for adaptive systems. In section 4, the major procedures and tools presently developed or currently being developed to enable the verification, validation, and ultimately certification of these adaptive control systems will be presented. These technologies include the application of automated program analysis methods, techniques to improve the learning process, analytical methods to verify stability, methods to automatically synthesize code, simulation and test methods, and tools to provide on-line software assurance.

II. Adaptive Control System Architecture

Even though this paper is primarily aimed at addressing the problems of adaptive control systems with learning algorithms, it is important to recognize that adaptive algorithms are in reality only one part of the total control system. In nearly all cases, the adaptive controller itself must be controlled by a non-adaptive, supervisory controller that interfaces to a human being and other systems. This larger program, usually implemented using finite state automata, is called the outer-loop controller.

A. Outer-loop Control Architecture

Control systems for large aircraft and spacecraft are usually comprised of hybrid systems involving both inner-loop and outer-loop control architectures (Fig. 1). The outer-loop controller governs the conditional execution of the inner loop controllers. From a systems perspective, the outer-loop controller generally represents the top-level executive program that coordinates a myriad of mission management actions including providing a human-machine interface, health monitoring, tracking of vehicle performance, navigation, guidance, and flight control laws to implement a variety of functions, perhaps even implementation of autonomous flight control. Outer-loop control software is generally comprised of finite state automata in which the sequencing of control tasks and multiple program threads is done conditionally based on finite state logic.

For example, the outer-loop controller of a jetliner might be programmed to issue pilot warnings when certain aircraft flight conditions (altitude, speed) are inconsistent with pilot actions (e.g., flaps or landing gear in wrong position).[6] This logic is called finite state logic because it is implemented using only a discrete logic to represent the conditional status of the aircraft (e.g., "speed is equal to or below maximum" is either true or false). The states are either achieved or they are not; there is no in-between. Another good example of the outer-loop control logic is the kernel of an autonomous agent used to operate a space craft, such as the NASA Deep Space 1 spacecraft[7]. In this case, the outer loop controller uses a planner and a scheduler to decide when to sequence different program tasks. Resources (e.g., motors, camera) are either free to use or they are not (locked). The onboard logic must ensure that the multiple program threads do not conflict with each other in the use of shared resources and that the logic of the outer-loop controller does not create a situation from which there is no recovery.

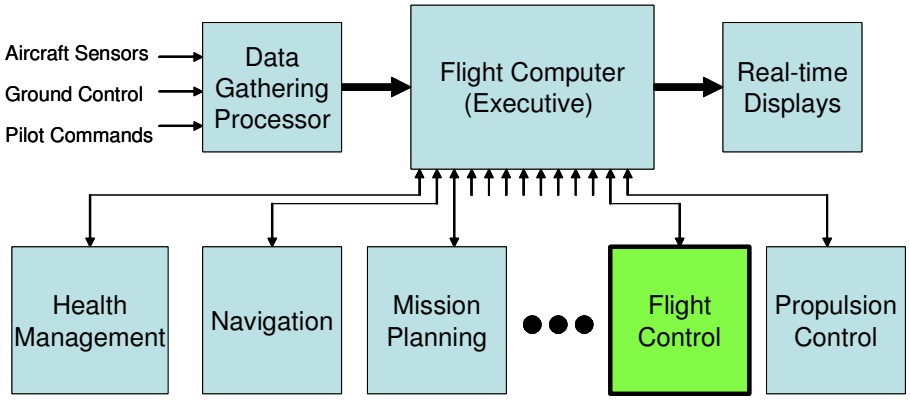

Figure 1. Outer-loop controller architecture.

B. Non-adaptive Inner-loop Control Architecture

Inner-loop controllers can provide very specific functions, such as providing vehicle flight control, stability augmentation, or active vibration control. The term inner-loop control can be used to refer to feedback loops controlling servo-actuators themselves. However, in the context of the present discussion, the inner-loop controller is meant to refer to the system indicated in Fig. 2.

Most modern control systems model the dynamics of this inner loop as a set of first-order, differential equations of the form

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{1}$$

where x represents a vector of vehicle "states" to be controlled, u is a vector of control inputs, and y is a vector of measurements[8]. For non-adaptive control, the A, B, C, and D coefficient matrices are fixed parameters whose values are identified prior to operation. In this scheme, an error signal formed as the difference between the desired state and measured state is fed to a controller that, if designed correctly, generates inputs to minimize the error signal. The most widely used controller of this type is the Proportional Integral Derivative (PID) controller, due to its simplicity, performance and robustness. A PID controller forms a control signal that is proportional to the error, proportional to the integral of the error, and/or proportional to the derivative of the error:

$$u = K_P(\varepsilon) + K_I\left(\int(\varepsilon)dt\right) + K_D\left(\frac{d\varepsilon}{dt}\right) \tag{2}$$

The proportionality constants (K_P, K_I and K_D) are called the gains of the controller and are normally vectors for multi-input, multi-output systems. Tuning the controller is a matter of finding the right gain settings. If the gains are selected too large, the system will exhibit instability; yet if selected too low, the system response may be too sluggish, or even unstable. Analytical methods for calculating stable gain values (e.g., root-locus, Nyquist, Bode, Nichols) are well understood and have been widely used.[8]

Although fairly simple to implement, PID and other types of conventional controllers unfortunately have the limitation that once the controller is put into operation, the gains and system parameters do not change. If the performance of the controller degrades after start-up, the only remedy is to stop the system and re-tune the gains prior to resuming operation. Adaptive control systems have the advantage of being able to adjust controller gains while in operation to avoid this problem.

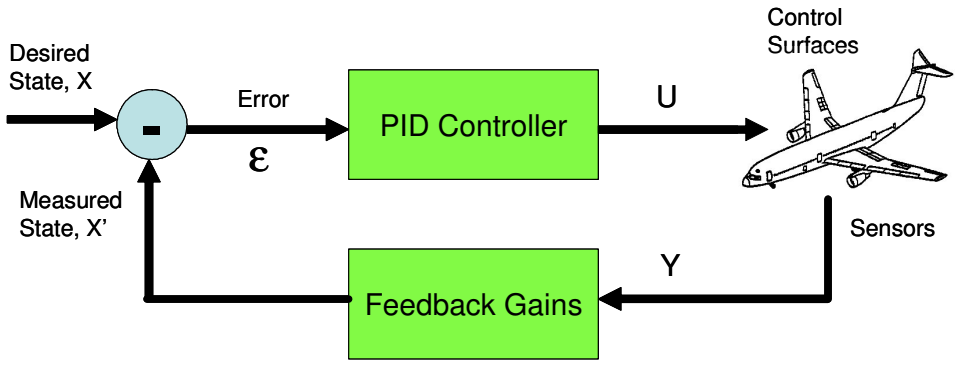

Figure 2. Conventional (non-adaptive) inner-loop controller.

C. Adaptive Control Architecture Using Learning Algorithms

Adaptive control systems with learning algorithms have been proposed to help aircraft maintain consistent controller performance in the face of unforeseen events, such as sudden loss of a control surface or gradual deterioration of control system components.[2,9] For example, if an airplane aileron ceases to function, the aircraft may yet be controllable using the remaining working control surfaces and propulsion sources. To do this automatically, an adaptive controller is required that can learn how the working control surfaces can be used to fly the plane.

Figure 3 provides a notional diagram of an adaptive control system to illustrate two possible roles in which learning can be used. First, a learning algorithm may be used to adaptively identify a transfer matrix (stability derivative matrix) that relates the control inputs to the system outputs. The identified transfer matrix can then be used in a minimum variance control law or linear quadratic Gaussian (LQG) controller[8] to achieve better system performance in a changing flight environment. In this way, the learning algorithm effectively changes the controller gain matrix. The system identification may be done in real-time (on-line) by a number of learning algorithms (e.g., Kalman filter[10], LMS[11] algorithms). Jacklin[12] provides substantial detail of the use of learning algorithms to reduce helicopter noise and vibration with this type of inner-loop controller. Another way in which learning algorithms can be used is to directly compute control commands to augment a non-adaptive (PID) controller. An example of this type of controller for an F-15 aircraft is presented in Ref. 13. The controller uses a neural network to generate supplementary control inputs to help the pilot fly the aircraft in the event of control surface failure or aircraft damage.

In order to be used on-line, all of the learning algorithms update a previous value of the identified parameters based on some learning error as follows

$$\begin{aligned} T_i &= T_{i-1} + K_{Learn} * \left(Learning\ \ error \right) \\ T_i &= T_{i-1} + K_{Learn} * \left(y_{Measured} - y_{\Pr edicted} \right) \end{aligned} \tag{3}$$

This is the learning update rule. Here, T_i are the parameter values (e.g., transfer matrix row or neural network weights) at step i and the "learning error" is typically computed as the difference between the measured state (y) and the state predicted using the model parameters. K_{Learn} is the adaptation learning gain and it has a large effect on algorithm stability and learning convergence. All learning algorithms, except the non-recursive ordinary least squares method, share this type of recursive learning equation in one form or another.

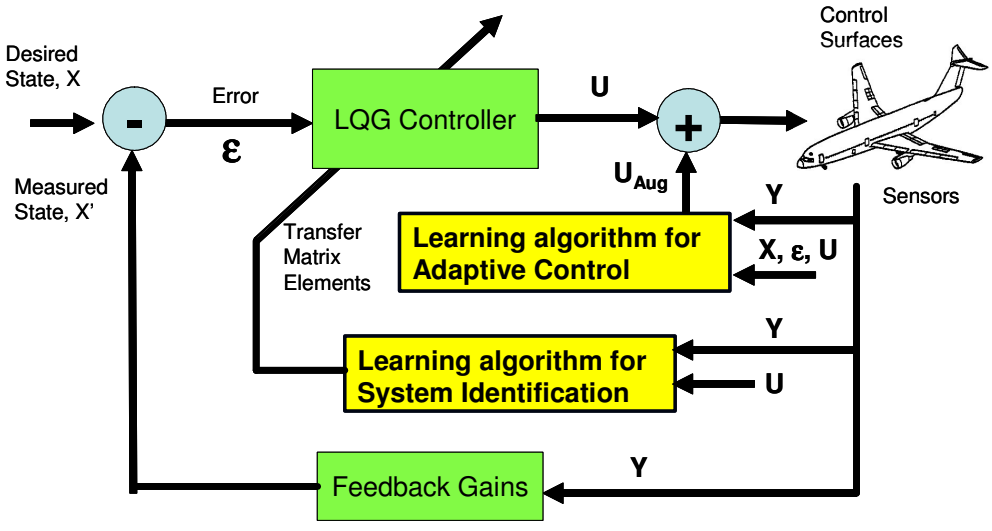

Figure 3. Generic adaptive control system.

Over the years, many learning algorithms have been proposed, including the gradient descent, back-propagation, Newton, Quasi-Newton, conjugate gradient, and Levenberg-Marquardt methods.[10,14] The algorithms differ in their search methods and the step length. For example, the gradient descent algorithm uses a fixed step-length and makes parameter changes proportional to the steepest derivative of change. Some algorithms (e.g., conjugate gradient) use the second derivative of the error with respect to the weights (properties of the Hessian) to help optimize speed and convergence stability.[15]

III. Learning System Verification and Validation Problems

The process of analyzing and checking the correctness of software is termed verification and validation (V&V).[16] This checking ensures the software functions as designed and meets the user requirements. In order for software to be certified by the FAA under the standards presented in RTCA DO-178B[17], the V&V process must be carefully documented and the behavior of the fielded software must be the same as that evidenced during V&V testing. For this reason, and because there is no explicit guidance in DO-178B for adaptive systems, it is generally felt that certification of learning system presents a major difficulty. However, as has been recently pointed out by Santhanam[18], adaptive software will behave deterministically if given the same inputs and started with the same initial conditions. The real verification and validation problems faced by learning systems is proving that the learning process is convergent and repeatable, that the convergence rate is acceptably fast, and that the learning process is stable.

A. Learning Convergence Problems

It is in general difficult to analytically evaluate the convergence of learning algorithms because the adaptation process is stochastic and can change in an unpredictable manner over time. One reason for this difficulty is that convergence is heavily influenced by the choice of initial conditions (of parameters and tuning factors), by the degree of measurement noise, and by the actions of the controller.

For example, for the simple case of a single-parameter system, Fig. 4 illustrates what could go wrong when trying to search for the value of x that minimizes an error function. In this case, the learning process converges to a local minimum because the initial condition on x (point 1) happens to be to left of the local minimum. Using steepest descent learning, corrections are made proportional to the local gradient. In this example, the search ultimately ends up at point 5, the local minimum. But, had the initial guess for x been to the right of the global minimum, the global minimum (i.e., the best answer) would have been found using the same process.

This dependency on initial condition poses a considerable problem for leaning systems, especially considering that realistic system transfer matrices and neural networks used in modern control applications typically have dozens of parameters. In fact, is not uncommon that these transfer matrices and neural networks have more parameters than the actual number of degrees of freedom in the system being controlled. In that case, many local optimum learning solutions usually exist. Unless the learning system is initialized in exactly the same way each time, a formal guarantee that the system will converge to the same solution each time may not exist.

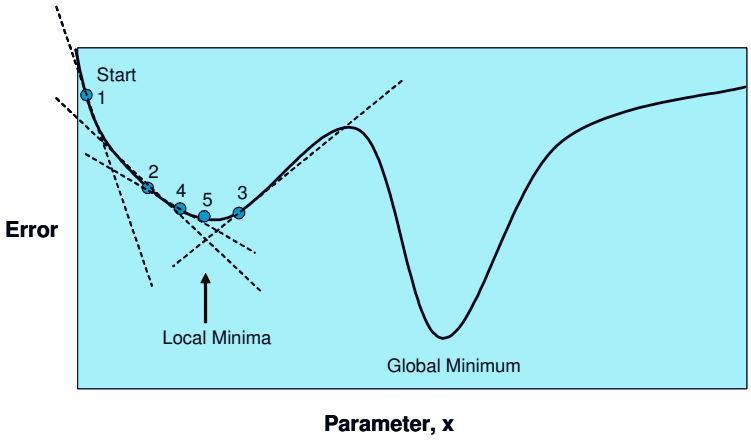

Figure 4. Convergence to local optimum instead of global optimum.

Unfortunately, starting the learning process from the same exact initial conditions each time is not a feasible solution. Adaptive controllers with learning algorithms will need to be restarted while the aircraft or spacecraft is in flight, either during a system re-boot or when a pilot or an outer-loop controller decides to enable adaptive control.

Another reason for lack of convergence is that the learning process may stop if the adaptive controller successfully performs its intended function.[12] Long before the learning process completes, the learning algorithm may be able to provide the controller with a transfer matrix that, although not strictly correct, is good enough to allow the controller to produce a control input that yields very good control performance. If this happens, subsequent control inputs to the system will be almost identical, since there is no reason to change the control if performance is good. However, the presence of measurement noise on the sensor inputs makes the measured vehicle (feedback) state appear to be changing. As a result, the learning algorithm may seek to learn a system transfer matrix that relates the very small changes in control to the measurement noises. This tends to produce a null matrix of identified system parameters. Ad hoc methods to fix this problem include shutting off the learning process at intermediate times, or adding persistent excitation[12] to the control commands for the sake of improving the learning process. Such ad hoc fixes, however, are very difficult to analytically verify and validate.

B. Speed of Learning Convergence

Lack of sufficient learning convergence rate is another problem faced by adaptive systems, and is closely coupled with the problem of stability discussed below in section C. Even if learning algorithm convergence can be assured, the learning process must happen in a sufficiently short amount of time in order to be useful. A learning system used to fly an aircraft must yield productive learning in as little as a few seconds.

Convergence time is a function of both how long it takes the learning algorithm to perform the numerical computations of one iteration cycle, and the number of iterations required for convergence. In the not too distant past (80's), only computationally efficient algorithms like the Kalman filter or steepest descent gradient search could be entertained for application to real time systems. The advent of high-speed computational capability allows a wider variety of learning algorithms to be considered, some of which may produce more learning per iteration.

In general, there are no analytical or formal methods available to guarantee that a learning algorithm will converge to a solution within a given amount of time, unless various assumptions are made about the initial conditions and system dynamics. These will be addressed in section IV below.

C. Learning Algorithm Stability Problems

One of the most important problems of adaptive control systems is finding an update method that provides stable, yet sufficiently fast learning. Conventional (non-adaptive) controller designs of the type shown previously in Fig. 1 become unstable when the control gains are made too high. In that case, the controller commands serve to over-correct the system repeatedly, so that the control commands oscillate wildly.

Over the last half of the twentieth century, a variety of analytical methods were developed to aid control system designers in the selection of stable controller gains for non-adaptive systems. These methods include the root locus method, Bode method, Nyquist plots, and Nichols chart techniques. Given a mathematical representation of the system, these methods find the region where the controller gains produce stable performance and also indicate the approximate highest gains allowed for fast, yet well-damped, dynamic response.

The stability of the learning (system identification) algorithms are equally important. Incorrect learning of the system model parameters can lead to control system instability. If the identified parameters are used to generate corrective control actions, a failure of the learning algorithm can lead to large fluctuations in the control outputs, and possible loss of control.

D. Knowing When Not to Use a Learning Control System

As a last thought toward the problems of learning systems, it should be mentioned that sometimes learning systems are proposed for application where less exotic control system technology will work better. For example, gain scheduling[19] is a classical control technique often employed to offer partial adaptive capability, while avoiding many of the problems associated with learning systems. To implement this scheme, a number of controller gain sets are determined (off-line) for a finite number of operating conditions or aircraft configurations. The flight control computer is then programmed to select the correct gains based on the current flight condition (airspeed, altitude, etc.). Interpolation between gain sets offers some additional adaptivity, but may not work well in highly nonlinear control situations or for unanticipated operating conditions. This method has been successfully utilized in many aerospace applications. The success of the gain scheduling method depends on the degree to which the system operates in discrete, well-defined operating regimes.

A problem is knowing when and when not to use a learning system. Learning is most useful for applications for which it is impossible to predict in advance the exact behavior of the system dynamics to be controlled. Some examples are robotic manipulator applications that require identification of changing mass moments of inertia, spacecraft control applications for landing or mating with other spacecraft in uncertain environments, or aircraft control systems designed to automatically help a pilot fly a damaged aircraft for which the exact nature of the damage is not known in advance.

IV. Verification and Validation Procedures and Tools for Learning Systems

Satisfying the certification requirements for adaptive control systems will likely necessitate the application of several tools and verification methods. In this section, some of the methods and tools for the verification and validation of adaptive control systems are presented. It should be mentioned at the outset that this body of knowledge is in no way felt to comprehensively cover all means available. The verification and validation of adaptive control systems and learning algorithms is still very much an active areas of intense research, so much so that it is in no way possible to reference all existing tools and methods rapidly being developed. These tools and techniques include the application of automated program analysis methods, techniques to improve the learning process, analytical methods to verify stability, methods to automatically synthesize code, simulation and test methods, and tools to provide on-line software assurance.

A. Tools for Automated Static Analysis

Although not limited to learning systems, automated static analysis tools have been developed to automatically inspect code for a number of bad programming practices and simple programming errors. Examples of such mistakes include use of undeclared variables, use of un-initialized variables, out-of-bound array referencing, wrong input of formal parameters in subroutine calls, use of inconsistent data type, and performing mixed-mode computations. A rich body of checklists, coding standards, and literature about this topic has been published.[20] Code review also examines the code to verify that comments are well used throughout and that the program contains no dead or unreachable code.

Static analysis tools analyze every instruction in the source code to determine if the operations performed in that instruction can create a problem at runtime. These tools can efficiently detect a wide range of problems even before unit testing including: buffer overruns, un-initialized variables, arithmetic overflows and underflows, and unreachable code. The advantage of static analysis methods is their ease of use. Static analysis programs are used in much the same way an ordinary compiler is used to compile a program. These methods can save many hours of human code review and can therefore provide substantial cost savings. However, a continuing challenge is the development of static analysis methods that detect only real problems (not false alarms).

There are several static analysis tools available, for example Coverity[21], PolySpace[22], Parasoft[23], Clint[24], UNO[24], and CGS[25]. All of these tools analyze the source code, without actually executing it. The source code does not need to be modified or instrumented. Results of the analyses are usually displayed in a user-friendly way and the results are usually quickly available. For example, CGS (used for Deep Space 1, Mars Pathfinder, and Mars Exploration Rover (C) code) was able to check hundreds of thousands of lines of spacecraft computer code in less than a half hour.

B. Analytical Methods to Improve Learning Convergence

Perhaps the most difficult verification task is to prove that the learning algorithm of an adaptive control system converges to the global minimum under all operating conditions. Even more difficult than this is the problem of knowing what methodology can be used to improve learning performance in the event the learning convergence is discovered to be unsatisfactory. In this section, some methods to improve learning convergence are discussed, primarily in the context of neural network weight identification. This section is not meant to comprehensively summarize all relevant work, but it does highlight some of the more recent advances in finding good methods to accelerate the learning process and to improve convergence.

Attempts to improve the convergence of learning algorithms have concentrated on the selection of better energy functions and selection of variable learning rate and momentum.[26-30] Some benchmarking fast learning algorithms have also been derived in the literature.[31-33] The most commonly used method is to effectively reduce the weights on past inputs and outputs.[33] This mechanism is called weight decay by adding a term to the optimizing function that is the sum of the squares of the weights such that during training, this term penalizes older weights. This method has been applied to the back-propagation (BP) learning algorithm to effectively speed up the convergence rate by solving the flat-spot problem[14]. By placing more weight on the most recent measurements, this method of

weight adjustment improves the convergence of the learning process by preventing the error signal from becoming too small relative to the average signal.

Since the back-propagation (BP) algorithm[34] based on an iterative gradient algorithm to minimize the mean square error (MSE) between the desired outputs and the actual outputs for the particular inputs to neural networks was proposed many years ago, it has been successfully applied to many areas of science and engineering. In addition to the BP algorithm, various learning rules have been proposed for training of various types of neural networks.[35-39] In developing the training algorithm for neural networks, optimization algorithms have played an important role.

In neural network applications, the learning process (training) of multilayer feed-forward neural networks (MLFNNs) has been realized by of a plethora of first- and second-order algorithms in the literature.[36] The majority of these learning algorithms are based on the steepest descent method in the form of the back-propagation algorithm. Although the steepest descent (SD) method is a good learning algorithm for simple models, even for those cases, the method takes an extremely long time to achieve convergence. Faster convergence speed can be obtained using the Gauss-Newton (GN) and Levenberg-Marquardt (LM) methods. These methods are generally cited as second-order techniques and have quadratic convergence learning patterns due to their use of the Hessian matrix.

However, even these algorithms often fail to converge unless the initial estimation is sufficiently close to the solution. Furthermore, it is difficult to guess a good initial point to avoid sticking at a local minimum even if the neural network size is of only several neurons. To overcome this convergence problem, homotopy methods have been proposed to efficiently find multiple solutions for the nonlinear algebraic equations.[40-42] Homotopy methods reach the globally optimum by defining mappings or "paths" to the desired solution.[43,44] In other words, the homotopy methods trace the solution path for a problem and find multiple solutions lying on the paths. The homotopies are classified in two classes such as "Fixed-point homotopy" and "Newton homotopy". Fixed-point homotopy methods have wider convergence regions for the initial guess compared with the Newton homotopy methods.[42]

In Ref. 45, the authors have proposed a new fast algorithm based on a modified form of the conventional BP algorithm. In order to increase the convergence speed of the BP method, the learning performance index was modified to include both linear and nonlinear errors of the output neuron. This achieved a three-fold increase in convergence speed.

The convergence of the new Mixed Least Square and Least Fourth (MLSLF) algorithm requires less iterations than the SBP and provides better generalization. The MLSLF uses an additional tuning parameter to modify the learning speed, sometimes beneficially, yet can cause the divergence in case of bad choice.

Other fast algorithms include the Modified Back propagation algorithm[45], Recursive Least Square algorithm[49] and the Marquardt algorithm[48]. A sparse gradient algorithm has been proposed to modify the cost function as the sum of the mean square error and a penalty term of the weight vector with an adaptive regularization parameter, which shows the asymptotic convergence of the penalty sequence to zero.[50-53] The Back Propagation Through Time (BPTT)[46] algorithm[55,56] has been proposed to train neural networks in fewer iterations, but may be highly prone to stopping at local minima in the error surface. Mastorocostas[57] has proposed a means of training feed-forward networks at a faster rate using constrained optimization methods. Oyan[58] proposes a relaxed model of variable kernel density as a means to make the convergence rate of the mean square error approach order O(n-1), regardless of the dimension of the data set.

C. Analytical Methods to Verify Learning Algorithm Stability

As mentioned above, the learning gains have a marked influence on learning performance. Although higher learning gains tend to increase the speed of learning, high gains also tend to promote instability of the learning algorithm. Certainly, it is possible to verify the suitability of a particular learning gain through simulation at a given operating condition (see section F). The problem with this approach is that defining the stability boundaries of a multiple-input, multiple-output adaptive control systems can require many test points at each of many possible operating conditions. For this reason, analytical methods that can determine learning system stability are needed.

The concept of adaptive control system stability applies to both the outer-loop and inner-loop control software. Verification of outer-loop adaptive controller stability can be done by a variety of methods. Although the outer-loop control behavior of most adaptive control systems is nonlinear, one approach is to approximate the system with a linear representation and then apply the stability methods used to analyze linear system stability. Of course, the modeling assumptions made to perform this step introduce a level of error into the system. Moreover, the learning process of the inner-loop controller must be frozen or fixed when this is attempted, because the coupled behavior of the inner and outer loop controllers is always nonlinear.[8] Given these significant limitations, the methods of Bode,

Nyquist, Root Locus, Ruth Hurowitz, or Nichols can be used to determine the stability of the adaptive system as a function of the controller gains.

Alternatively, the outer-loop controller stability can be analyzed as a nonlinear system if the control system is relatively simple. Two popular techniques are the phase plane and describing function approaches. The phase plane method provides a graphical representation of the system state by computing state trajectories in a two dimensional plane called the phase plane (typically, velocity versus displacement plots).[59-62] The phase plane approach has been used to analyze stability in biomedical, orbital mechanics, vacuum tube circuits, and dynamic vehicle control applications. The major disadvantage of the phase plane analysis method is that the two-dimensional, graphical approach is cumbersome to apply to multiple-input, multiple-output systems. The other method, the describing function (DF) method, can be used to determine the necessary and sufficient conditions for the nonlinear feedback system stability. The describing function utilizes Fourier input-output analysis to find an "equivalent gain" of the nonlinear system in the frequency domain. Describing functions are most generally used for single-input, single-output feedback systems, although there have been generalizations to multiple-input multiple-output systems.[59,63] The phase plane and describing function methods have been used to analyze the outer-loop stability of the shuttle flight control system during orbiter repair maneuver and to examine Space Shuttle control system stability using a flexible robotic arm.[4,64]

Analytical verification of the inner-loop (learning or system identification) stability is very important for hybrid systems because instability of the inner loop dynamics can cause the outer loop control to become unstable as well. Generally, stability analysis of learning systems falls into either an analysis of the inner-loop learning algorithm, or for the hybrid system as a function of the learning update rule with the outer-loop controller gains held fixed. Each method will be briefly described.

One approach of analytically verifying the stability of the inner-loop controller is to directly analyze the learning rule update equation to mathematical determine the sufficient conditions for stability. In Ref. 12, the stability of the Kalman filter, generalized Kalman filter, LMS filter, and generalized LMS filters are analyzed in this manner. Although the analysis presented therein provides insight into the factors governing stability, these factors are a function of the input covariance matrix, which is generally an unknown quantity in most on-line applications.

Some progress has been made finding stable learning update rules for adaptive systems using Lyapunov's second method[65,67] Lyapunov analysis is difficult to apply to realistic aerospace applications, but it is attractive because it can be used to find regions of learning system stability without knowledge of the exact solution. Application of Lyapunov's second method to a neural network learning algorithm of an adaptive flight control system is described in Ref. 65.

The Lyapunov method is based on an energy analysis of the system. Basically, if it can be shown that the total system energy as a function of time is either constant or decreasing, the system is stable. For this to be true, sufficient care must be taken when defining the system under study. If stable, the results of Lyapunov analysis show that the errors between the desired and actual outputs of the learning systems are ultimately bounded. However, the ultimate bounds on the error as well as the time required to achieve these bounds can't be explicitly calculated from the theory.

D. Testing and Simulation Tools for Learning System Verification

Since analytical determination of stability, convergence, and performance is often difficult to apply, computer-based simulation plays a major role in the verification of learning systems. Many aspects of adaptive systems learning, in particular convergence and stability, can only be analyzed with simulation runs that provide enough detail and fidelity. For example, a stuck rudder on an aircraft cannot be expressed as a linear model since the failure introduces a bias. Simulation provides a fairly rapid way to:

- evaluate and compare different learning algorithms
- find good control and learning rule update gains
- determine how much learning is actually accomplished at each step
- evaluate the effect of process and measurement noise on learning convergence rate
- determine stability boundaries
- test algorithm execution speed on actual flight computer hardware
- conduct piloted evaluation of the learning system – while in the simulator
- simulate ad hoc techniques of improving the learning process, such as adding persistent excitation to improve identification, or stopping the learning process after error is less than a specified error or after a specified number of iterations.

Simulators differ primarily in the fidelity with which the plant is modeled. Higher fidelity simulations require more complicated mathematical models of the adaptive system and also greater use of actual (and expensive) controller hardware. In order to be cost-effective, the lowest fidelity test bed should be used as much as possible. The behavior of simple linear models should be compared to that of higher fidelity nonlinear models when they are available to ensure that analysis performed using the linear model still applies. Table 1 presents one representation of the simulation hierarchy from lowest to highest fidelity.

The lowest fidelity simulations are usually run on a desktop computer using user-friendly simulation tools in the Matlab/Simulink[67] environment. This simulator typically includes the control laws and a linear plant which accounts for the aircraft aerodynamics, mass properties, and thrust from the engines. Models may include some uncertainties or perturbations. The linear simulator is most often used in early control law design and analysis or to calculate linear gain and phase margins. It is important to note that nonlinear adaptive controllers can be represented linearly, but the linear model may not provide results with the required accuracy. Changes to the plant model can be simulated by changing the system transfer function from one matrix to another with varying frequency. By varying the amount of change, the stability boundaries of the system can be determined. Concomitant with this process is an evaluation of the system tuning parameters that are used in the learning algorithm. The desktop simulation environment provides a quick way to compare different learning algorithms and controller architectures. Only the most promising designs need be simulated using higher fidelity simulations.

The next level of simulation provides additional fidelity by incorporating important model nonlinearities. Adaptive controller nonlinearities are common, and therefore must be evaluated in nonlinear simulation. This simulation is typically run on a workstation computer to handle the more intensive numerical operations. The additional complexity of the nonlinear simulation allows detailed modeling of sensor input uncertainties, actuator dynamics, and assessment of adaptive controller performance and stability. Inputs to the controller can come from a "pilot" in the loop, but more commonly from a predetermined set of inputs, typically impulses, steps, doublets, or frequency sweeps. Obviously, the time spent to develop realistic aeroelastic and aerodynamic models is only justified after low-level simulation verifies that the basic learning system is sound.

Higher fidelity test beds, like software-in-the-loop, hardware-in-the-loop, and aircraft-in-the-loop simulations, contain a combination of software models and hardware. These software simulations typically run in dedicated computing environments with a cockpit and out-the-window graphics (e.g., see Refs. 68 and 69). Typically they contain software models of nonlinear aerodynamics, engine dynamics, actuator models, and sensor models. The most common elements of these test beds are some of the flight processors, communication buses and a cockpit. The aircraft-in-the-loop simulator maximizes the use of flight hardware components and minimizes the number of software models. Using the actual aircraft/spacecraft flight computer is a particularly important advantage of this simulation, since all computers tend to handle exceptions differently and may have differences in their numerical routines. Either the actual aircraft may be tied into the nonlinear simulation, or an iron-bird aircraft may be used to provide actuators, sensor noise, actual flight wiring, and some structural interactions. These test beds allow for a complete check out of all interfaces to the flight hardware, timing tests, and various failure modes and effects analysis (FMEA) testing, which is not possible in a simpler configuration.

Simulators that include a cockpit to interface with the pilot can either be fixed-based or motion-based. Fixed-based simulators have non-moving cockpits and usually include out-the-window graphics. The moving cockpit of motion-based simulators additionally provide the pilot with some of the physical cues of actual flight. Piloted simulation is an important part of verification and validation testing of flight control software.

Table 1. Types of Simulation

Test Bed	Pilot Interface Fidelity	Model Fidelity	Test Environment
Desktop Computer	Low	Low	Linear simulation using Matlab or Simulink
Work Station	Low	Low-Medium	Models that include nonlinearities
Simulator	Low	Medium	Dedicated aircraft model and hardware
Sim w/ Hardware –in-the Loop	Medium to High	Medium-High	Actual aircraft target flight computer and cockpit
Sim Aircraft-In-the-Loop	High	Medium-High	Simulator with flight computer and aircraft
Motion-Based Simulator	High	High	Nonlinear simulation with moving cockpit

E. Tools to Improve Simulation Coverage

A problem encountered in performing simulation is providing adequate test coverage. Coverage concerns the percentage to which 1) every decision in a program has been executed at least once, 2) every decision in the computer program has been taken with all possible outcomes at least once, 3) every condition in a decision in the program has taken all possible outcomes at least once, 4) every condition in a decision has been shown to independently affect that decision's outcome, and 5) all entry and exit points of the program have been tested.[70] Merely running a collection of simulation test cases may discover some problems, but can't guarantee the absence of problems. In order to help simulation achieve greater coverage, various tools and methods are being developed to implement simulation in a more systematic manner.

In one such effort, the Automated Neural Flight Controller Test (ANCT)[71] tool is being developed in the MATLAB environment. ANCT has been designed to help test engineers evaluate different flight conditions, quantify performance, and determine regions of stability. It is equipped with a graphical user interface that allows the engineer to specify input parameters, minimum and maximum values, step increments, and success/failure requirements. ANCT uses MySQL for storage and management of the test cases, input data, and output data. ANCT is designed to analyze a MATLAB/Simulink model by simulating the model using all possible combinations of the model inputs parameters. By introducing random numbers into the test inputs and parameters, the user can perform Monte Carlo simulation to estimate the sets of model parameters and inputs that correspond to the system responses that are of interest. Based upon user input, ANCT creates the test cases and evaluates the time-series outputs during a specified time or condition window by using a user-specified output evaluation function. This process yields a performance score that represents the degree to which an output violates user-defined criteria or failure criteria, and includes a Pass/Fail status vector of the outputs. ANCT also provides a genetic algorithm to explore the ranges over which parameters and inputs are allowed to vary.

A simulation based robustness analysis tool, RASCLE (Robustness Analysis for Control Law Evaluation) has also been developed to help explore combinations of learning system parameters and operating conditions.[72] The RASCLE simulation tool is used to interface with existing nonlinear simulations and incorporates search algorithms to uncover regions of instability with as few runs as possible. RASCLE uses a gradient algorithm to identify the direction in the uncertainty space along which the stability of the system is most rapidly decreasing. RASCLE provides an intelligent simulation-based search capability that can be used in Monte Carlo simulation evaluations.[73]

F. Model Checking Methods for Learning Systems

Over the last decade, the formal method of model checking has become an important tool for the verification of finite state automata. Many types of model checkers have been proposed, for example explicit state model checkers (e.g., SPIN, JPF, JPF2), symbolic (e.g., SMV, NuSMV), and hybrid model checkers[74-76]. Most of these techniques require that a model of the actual control program be written in a special language of the model checker, although some model checkers do not have this restriction.[77] Once a finite state model of the program has been developed by using a logical abstraction method, the model checker can test all possible executions paths of the program and report any execution that leads to a violation of a user-defined property. Model checkers can find errors that are almost impossible to find by human code review.

Model checkers have found considerable application for outer-loop adaptive control system verification. They have been useful for verification of autonomous systems such as NASA's Remote Agent and K9 Mars Rover[78]. The outer-loop controller of these programs use planners and schedulers to coordinate the actions of multiple program threads that execute in parallel. As pointed out in Ref. 79, the program threads may sometimes interact in unexpected ways or conflict with each other in the use of shared resources. Traditional, scenario-based testing may never discover some bad thread interaction sequences if they occur infrequently or only under circumstances not envisioned by the test team. The use of these formal methods allow complete verification of every possible program execution path and can verify the programs are free of the most severe problems in multi-threaded programs, including thread deadlocks and data races. As an aviation example, the NuSMV model checker has been used by Rockwell Collins to provide verification of the mode logic of the FCS 5000 flight guidance system being developed for use in business and regional jet aircraft.[80] In this testing, NuSMV was used to check models (consisting solely of Boolean and enumerated types) with over 10^{20} reachable states in less than an hour.

For the most part, model checking does not lend itself well to verification of inner-loop control and learning or adaptation. The reason is that these processes are generally modeled as continuous systems, rather than as finite state automata. Nevertheless, some recent progress has been made attempting to apply the technique of hybrid model checking to continuous systems. Reference 81 describes an application of Java PathFinder to the control of a robotic vehicle. The vehicle dynamics are modeled in the time domain as a set of first order differential equations (Eq. 1 applies here.) The execution of the inner-loop controller is controlled by an outer-loop autonomous agent

planner and scheduler. Although the continuous variables (x, u, y) could assume an infinite number of values, and thereby presenting a state explosion problem for the model checker, the use of Java PathFinder is made possible through representing theses values as discrete quantities. The use of an approximation function converts the continuous variables into discrete values. The idea is similar to rounding a decimal number to the nearest integer, only in this case, the truncation is considerably coarser. With this abstraction of the continuous space, the variables can be made to take on relatively few values. This allows for the recognition of previous "states" in the model checking sense of the word, and hence an exploration of the continuous model checking space becomes possible. Of course, this search is exhaustive only to the extent the approximation function is valid. If the approximation function is too coarse, important states will likely be missed.

G. Program Synthesis Methods for Certifiable Code Generation

With very few exceptions, the software development cycle transforms software design specifications into code by a manual process that is then verified by code review, unit testing, functional integration testing, and validation.[16] Ultimately, if the code is for a civil aviation application, certification approval of the software may be sought. Proving adequate coverage of the learning system can be a particularly difficult aspect of the certification process.

As an alternative to this development approach, a number of software vendors are now proposing a number of tools that can help to produce certifiable code, including code for learning systems. It is important to understand at the outset that none of these tools can produce certified code, because airborne software is certified only as part of the verified and validated avionics package for a particular aircraft. For this reason, software produced by these tools must still undergo a certification process to meet RTCA DO-178B standards. Nevertheless, these tools produce code that has a much higher chance of passing certification requirements.

AutoFilter is a tool being developed at NASA to automatically generate certifiable Kalman Filter code from high-level declarative specifications of state estimation problems.[82] Although Kalman filters are widely used for state estimation in safety-critical systems, the complex mathematics and choice of many tuning parameters make implementation a difficult task. The AutoFilter tool not only generates Kalman filter code automatically from high level specifications, but also generates various human-readable documents containing both design and safety related information required by certification standards such as DO-178B. Program synthesis is accomplished through repeated application of schemas, or parameterized code fragment templates and a set of constraints formalizing the template's applicability to a given task. Schemas represent the different types of learning algorithms. AutoFilter applies rules of the logic backwards and computes, statement by statement, logical formulae or *safety obligations* which are then processed further by an automatic theorem prover. To perform this step automatically, however, auxiliary annotations are required throughout the code. AutoFilter thus simultaneously synthesizes the code and all required annotations. The annotations thereby allow automatic verification and produces machine-readable certificates showing that the generated code does not violate the required safety properties.

Eurocopter has used the Safety Critical Application Development Environment (SCADE) tool to automatically generate certifiable auto-pilot software for the EC135 and EC155 helicopters.[83] To use the SCADE tool, the user must describe the auto-pilot using precise formal specifications and rules for naming and structuring (Lustre language). The tool incorporates a model checker (Design Verifier) to provide extensive code coverage according to the formal specifications. SCADE generates readable and traceable C or Ada code for the auto-pilot and also automatically generates documentation useful for certification purposes.

Another type of code synthesis is performed by auto-coders of such simulation programs as the Mathworks Matlab and Simulink[67]. As stated above, these desktop simulation programs allow rapid development of learning system simulations by providing the use with a convenient visual programming interface. Mathworks offers an autocoder tool in its Real-Time Workshop to translate the symbolic programming language seen on the desktop to actual real-time application code, usually in C or Ada. To help verify the code generated by this program, Mathworks has incorporated runtime memory-checking tools, such as Rational's Purify; coverage tools, such as Rational's PureCoverage; and static analysis tools, such as Lint. In addition, a code coverage analysis capability built into the Target Language Compiler™ (TLC) helps verify that the TLC files responsible for converting the model to generated code are thoroughly tested. To help improve code traceability to requirements, automatically hyperlinks between the generated code and the blocks from which it was generated are produced. Automatically generated code from MathWorks tools has been certified for use on safety-critical projects such as Honeywell's Commercial Aviation Systems Primus series avionics systems.

In a desire to develop reliable software tools for safety-critical applications, Boeing has elected to use a subset of the Ada programming language, called Zbra[84]. Boeing defined this subset based on its experience with safety critical systems, and language and tool construction issues that make commercial compilers too complex to

certify. One of the most significant features of the Zbra compiler is its built in anomaly checker that looks for a number of error-prone coding patterns, such a use-before-set, no-use-after-set, static conditions that render parts of code unreachable, and loops whose exit conditions are loop-invariant. Aside from eliminating many real-time problems found in most C code (e.g., pointers), Zbra was also developed to produce code that is cleaner and more traceable. This was done to make the certification process easier.

H. Tools for On-line Software Assurance

As mentioned previously, learning systems may be used to identify transfer matrices and neural networks whose number of adjustable parameters greatly exceeds the true system degrees of freedom. Although this may sound like a strength, it is actually a weakness because it makes it difficult to know whether the learning process has locked one of many locally optimal point solutions, or the actual globally optimum solution. Although the techniques of model order reduction or pruning[47] could be used to reduce the modeling degrees of freedom, and hence the number of possible solutions, the understanding of the physical system is rarely available to do this. Transfer matrix elements or neural network weights found to be near zero at one test condition or scenario, may be highly important at another. For this reason, research is being performed to develop tools to assess the on-line performance of the learning algorithms.

As one approach to this problem, NASA Ames Research Center has developed a tool called the Confidence Tool to analyze the probability distribution of the neural network output using a Bayesian approach[85]. This approach combines mathematical analysis with dynamic monitoring to compute the probability density function of neural network outputs while the learning process is on-going. The Confidence Tool produces a real-time estimate of the variance of the neural network outputs. A small variance indicates the network is likely producing a good, reliable estimate, and therefore, good performance of the neural network software can be expected. The confidence tool can be used for pre-deployment verification as well as a software harness to monitor quality of the neural network during flight. The outputs of the Confidence Tool can be used as a signal to stop and start neural network adaptation and also be used (with modification) to provide a guarantee of the maximum network error for certification purposes. A strength of the Confidence Tool is that it can be used to verify convergent learning and yet only requires the specification of two tuning parameters that are easily computed based on some preliminary information about the system process and measurement noises. A weakness of the tool is that it cannot distinguish between local and global learning solutions.

Rule extraction schemes have been proposed as a means to determine if non-adaptive neural networks (trained before operation) provide a globally optimum solution. Such methods include NNRules, M-of-N Rules, KT, Rulex and others.[86-88] To apply the rule extraction method, the data set used to train neural network is also used to develop a set of mathematical "if-then" rules to describe the functioning of the neural network. These rules are basically of the type

IF condition 1 AND condition 2 AND condition 3 THEN *RESULT*

where the conditions are queries that produce a yes-no, true-false result (e.g., airspeed less than 100 kts?) and the result relates something about the neural network (e.g., 0.5 < network weight(23) < 0.7). These rules, in essence, define the globally optimum solution based on the current training data set. The idea is that once the fixed neural network is placed into operation, data collection can continue to see if the rule set is still adequate to describe the network. If it is, assurance of a globally optimum solution can be generated. The collection of such data may be useful for certification purposes. The weakness of the rule extraction approach is that it is not useful to judge the performance of neural networks that adaptively learn while in operation to respond to new environments and system uncertainties.

V. Concluding Remarks

The FAA certification requirement to show that learning software programs meet their intended function, do not negatively impact other systems or functions on the aircraft, and are safe for operation, as pointed out in RTCA DO-178B[17], involves more than just running a set of test cases. The complete verification and validation of learning systems should not be viewed as running test cases and comparing expected results to actual results because such testing can never reveal the absence of errors. The verification and validation objectives must be satisfied by a combination of reviews, analyses, the development of test cases and procedures, and the subsequent execution of those test procedures.

Future progress towards the certification of learning software requires continued development of new tools for automated static analysis, model-checking for hybrid systems, methods for certifiable code generation, and tools to

provide on-line software assurance. Simulation and methods to automate simulation will continue to remain very important tools, because at present only they can really test and explore the most nagging problems of adaptive system verification, like algorithm stability and convergent learning. And yet the fact that testing can never reveal the absence of errors is a major shortcoming of simulation.

In all likelihood, a combination of analysis, tools, and simulation will be needed to fully address the full aspect of the certification problem for learning systems. Nevertheless, a relevant cautionary remark of Heimdahl[89] made for the concept of N-version programming applies equally well here too: the application of a combination of tools and analyses doesn't mean success, because we don't know how much confidence we can put in N-tools and methodologies as opposed to one.

References

[1]Kaneshige, J., Bull, J., and Totah, J., "Generic Neural Flight Control and Autopilot System," AIAA-2000-4281.

[2]Rysdyk, R. T., and Calise, A.J., " Fault tolerant flight control via Adaptive Neural Network Augmentation," AIAA 98-4483, August 1998.

[3]Nguyen, N.T., Bright, M.M., and Culley, D.E., "Feedback Adaptive Flow Control of Air Injection in Compressors", AIAA Guidance, Navigation, and Control Conference, San Francisco, CA, Aug. 15-18, 2005.

[4]Hall, R., Barrington, R., Kirchwey, K. and Alaniz, A., "Shuttle Stability and Control during the Orbiter Repair Manuever, AIAA Guidance, Navigation an dControl Conference and Exhibit, 2005, AIAA 2005-5852.

[5]KrishnaKumar, K., Kaneshige, J., Waterman, R., Pires, C., and Ippoloito, C., "A Plug and Play GNC Architecture Using FPGA Components," Proceedings of Infotech@aerospace Conference, Arlington, VA, Sept. 26-29, 2005.

[6]Degani, A., Taming HAL, Designing Interfaces Beyond 2001, Published by Palgrave-Macmillan, 2004

[7]Deep Space 1 program NASA's Jet Propulsion Laboratory. http://www.jpl.nasa.gov/news/fact_sheets/ds1.pdf

[8]Franklin G, Powell J, Emami-Naeini A. *Feedback Control of Dynamic Systems*. Addison-Wesley Publishing Company: MA, 1994

[9]Tomayko, J.E., edited by Gelzer, Christian, The Story of Self-Repairing Flight Control Systems, NASA Dryden Flight Research Center, 2003

[10]Ljung, L., *System Identification: Theory for the user*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[11]Widrow, B. and Stearns, S.D., *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[12]Jacklin, S. A., "Comparison of Five System Identification Algorithms for Rotorcraft Higher Harmonic Control," NASA TP 1998-207687, May 1998.

[13]Kim, B. S. and Calise, A. J., "Nonlinear Flight Control using Neural Networks," AIAA J. Guidance, Control, and Dynamics, Vol. 20, No. 1, 1997

[14]Ng, S., Cheung, C., Leung, S., and Luk, A., "Fast Convergence for Back-propagation Network with Magnified Gradient Function," *Proc. of IEEE Inter. Joint Conf. on Neural Networks (IJCNN'03)*, Portland, USA, 20-24 June 2003, vol.3, pp.1903-1908.

[15]Malki, H., Canelon, J., Shieh, L., and Jacklin, S. "Neural Network-Based Frequency-Domain Vibration Modeling for Black Hawk Helicopter" AIAA-2004-1977 45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Palm Springs, California, Apr. 19-22, 2004

[16]Jacklin, S.A., Lowry, M.R., Schumann, J.M., Gupta, P.P., Bosworth, J.T., Zavala, E., Kelly, J.W., Hayhurst, K.J., Belcastro, C.M., and Belcastro, C.M., "Verification, Validation, and Certification Challenges for Adaptive Flight-Critical Control System Software," AIAA Guidance Navigation and Control Conference and Exhibit, 2004. *Automated Reasoning*, *Lect. Notes Comp.*

[17]*Software Considerations in Airborne Systems and Equipment Certification,* Document No RTCA (Requirements and Technical Concepts for Aviation) /DO-178B, December 1, 1992.

[18]Santhanam, V. "Can Adaptive Flight Control Software be Certified to DO-178B Leve A?", NASA and FAA Software and CEH Conference, Norfolk, VA, July 26-28, 2005.

[19]Fromion, V., and Scorletti, G., "A Theoretical Framework for Gain Scheduling," International Journal of Robust and Nonlinear Control, 13(6):951–982, 2003.

[20]Nelson, S. D., and Schumann, J. "What Makes a Code Review Trustworthy?", Proceedings of 37th Hawaii International Conference on System Sciences (HICSS-37 2004), 5-8 January 2004

[21]http://www.coverity.com

[22]http://www.polyspace.com

[23]http://www.parasoft.com

[24]Holzmann, G.J., "Static Source Code Checking for User-Defined Properties," Integrated Design and Process Technology, IDPT-2002.

[25]Brat, G., and Venet, A., "Precise and Scalable Static Program Analysis of NASA Flight Software," Proceedings of the 2005 IEEE Aerospace Conference, Big Sky, MT, March 5 - 12, 2005.

[26]Jacobs, R.A., "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295-307, 1988.

[27]Weir, M.K., "A method for self-determination of adaptive learning rates in back propagation," *Neural Networks*, vol. 4, pp. 371-379, 1991.

[28]Yu, X.H., Chen, G.A., and Cheng, S.X., "Acceleration of backpropagation learning using optimised learning rate and momentum," *Electronics Letters*, Vol. 29, No. 14, pp. 1288-1289, 8th July 1993.
[29]van Ooyen, A. and Nienhuis, B., "Improving the convergence of the back-propagation algorithm," *Neural Networks*, vol. 5, pp. 465-471, 1992.
[30]Ahmad, M. and Salam, F.M.A., "Supervised Learning Using the Cauchy Energy Function," *International Conference on Fuzzy Logic and Neural Networks*, 1992.
[31]Fahlman, S.E., "An empirical study of learning speed in backpropagation networks," *Technical report*, CMU-CS-88-162, 1988.
[32]Riedmiller, M., and Braun, H., "A direct adaptive method for faster back-propagation learning: The RPROP Algorithm," *Proc. of Inter. Conf. on Neural Networks*, vol 1, pp. 586-591, 1993.
[33]Treadgold, N.K. and Gedeon, T.D., "Simulated Annealing and Weight Decay in Adaptive Learning: The SARPROP Algorithm," *IEEE Trans. Neural Networks*, vol. 9, no. 4, July 1998, pp.662-668.
[34]Rumelhart, D.E., Hinto, G.E., and Williams, R.J, "Learning representations by back-propagation errors", *NATURE*, **vol.323**, 9, pp.533-536, Oct., 1986.
[35]Hagan, M.T. and Menhaj, M., "Training feedforward networks with the Marquardt algorithm", IEEE Trans. Neural Networks, vol.5, pp.989-993, Nov., 1994.
[36]Battiti, R. "First and second-order methods for learning: Between steepest descent and Newton's method", Neural Computation, vol.4, no.2, pp.141-166, 1992
[37]Ninomiya, H., Tomita, C., and Asai, H., "An Efficient Algorithm with Second-Order Convergence for Myltilayer Neural Networks", Proc.IEEE&INNS/IJCNN'03, July, 2003.
[38]Ninomiya, H. and Kinoshita, N., "A New learning Algorithm without Explicit Error Back-Propagation", Proceedings of IEEE&INNS/IJCNN'99, July, 1999.
[39]Ninomiya, H. and Sasaki, A., "3-Layer Recurrent Neural Networks and their Supervised Learning Algorithm", Proc.IEEE&INNS/IJCNN'01, July, 2001.
[40]Coetzee, F.M., "Homotopy Approaches for the Analysis and Solution of Neural Network and Other Nonlinear Systems of Equations", Doctoral Thesis, Carnegie Mellon University, May, 1995.
[41]Coetzee, F.M., and Stonick, V.L., "On a Natural Homotopy Between Linear and Nonlinear Single-Layer Networks'', IEEE Trans. On Neural Networks, vol. 7, no. 2, pp. 307-317, March 1996.
[42]Tomita, C., Yashida, M., Ninomiya, H., and Asai, H., "Note on learning algorithm based on Newton homotopy method for feedforward neural networks", Proc. 2004 RISP International Workshop on Nonlinear Circuit and Signal Processing, pp.128-132, March, 2004.
[43]Yamamura, K., Sekiguchi, T., and Inoue, Y., "A Fixed-Point Homotopy Method for Solving Modifyed Nodal Equations", IEEE Trans. Circuit & Systems -I, vol.46, no.6 pp.654-665, June, 1999.
[44]Ushida, A., Yamagami, Y., and Nishio, Y., "An Efficient Algorithm for Finding Multiple DC Solutions Based on the SPICE-Oriented Newton Homotopy Method", IEEE Trans. CAD, vol.21, no.3 pp.337-347, March, 2002.
[45]Abid, S., Fnaiech, F., and Najim, M., "A Fast Feed-Forward Training algorithm using A Modified Form of the standard Back Propagation Algorithm" IEEE Trans. Neural Network Vol.12, No 2, March 2001.
[46]Abid, S., Fnaiech, F., and Najim, M., "Evaluation of the feedforward neural network covariance matrix error", IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP' 2000) June 5-9, Istanbul, Turkey.
[47]Fnaiech, N., Sayadi, M., Fnaiech, F., Jervis, B.W. and Cheriet, M., "A Pruned Multilayer Neural Network For Brain Diseases Classification", Proc. of NNESMED/CIMED 03, July 2003 Sheffield, UK.
[48]Hagon, M.T. and Menhaj, M.B., "Training feedforward networks with the Marquardt algorithm" IEEE Transactions on Neural Networks, Vol. 5, pp. 989, November 1994.
[49]Scalero, R. S. and Tepedelenglioglu, N., "A Fast New Algorithm for Training Feedforward Neural Networks" IEEE Transactions on signal processing. Vol 40. No, 1, January 1992.
[50]Abid, S., Fnaiech, F., Jervis, B.W., and Cheriet, M., " Fast Training of Multilayer Perceptrons with a Mixed Norm Algorithm", IJCNN 2005 Conference Proceedings, Montreal, Canada.
[51]Rao, Y.N., Kim, S.P., Sanchez, J.C., Erdogmus, D., Principe, J.C., Carmena, J.M., Lebedev, M.A., and Nicolelis, M.A., "Learning Mappings in Brain Machine Interfaces with Echo State Networks", *accepted to ICASSP*, 2005.
[52]Wang, Y., Kim, S., and Principe, J.C., "Comparison of TDNN Training Algorithms in Brain Machine Interfaces", IJCNN 2005 Conference Proceedings, Montreal, Canada.
[53]Yang, X., Song, Q., and Liu, S., "Pre-selection of Working Set for SVM Decomposition Algorithm", IJCNN 2005 Conference Proceedings, Montreal, Canada.
[54]Piche, S., "Steepest Descent Algorithms for Neural Network Controllers and Filters," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 198-221, March 1994.
[55]Frasconi, P., Gori, M., and Soda, G., "Local Feedback Multilayered Networks," *Neural Computation*, vol. 4, pp. 120-130, 1992.
[56]Tsoi, A.C. and Back, A.D., "Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 229-239, March 1994.
[57]Mastorocostas, P.A., "A Constrained Optimization Algorithm for Training Locally Recurrent Globally Feedforward Neural Networks", IJCNN 2005 Conference Proceedings, Montreal, Canada.

[58]Oyan, Y., Ou, Y., Hwang, S., Chen, C., and Chang, D.T., "Data Classification with a Relaxed Model of Variable Kernel Density Estimation", IJCNN 2005 Conference Proceedings, Montreal, Canada.
[59]Atherton, D.P., Nonlinear Control Engineering, Describing Function Analysis and Design, Van Nostrand Reinhold, London 1975.
[60]Blaquiere, A., Nonlinear Control Systems, Academic Press, New York, 1966.
[61]Gibson, J.E., Nonlinear Automatic Control, McGraw-Hill, New York, 1963.
[62]Inagaki S., Kushiro I., and Yamamoto M, "Analysis on Vehicle Stability in Critical Cornering Using Phase-Plane Method," JSAE Review, Vol 16, 1995.
[63]Gelb, A. and van der Velde, W.E., Multiple input Describing Functions and Nonlinear Systems Design, McGraw-Hill, New York, 1968
[64]Penchuck A., Hattis P., Kubiak E., A frequency domain stability analysis for a phase plane control system, Journal of Guidnace, Control and Dynamics, vol. 8, 1985 pp. 50-55.
[65]Yerramalla, S., Cukic, B., and Fuller, E., "Lyapunov Analysis of Neural Network Stability in an Adaptive Flight Control System,", Sixth Symposium on Self-Stabilbizing Systems, 2003.
[66]Yerramalla, S., Cukic, B., and Fuller, E., "Lyapunov Stability Analysis of Quantization Error for DCS Neural Networks," International Joint Conference on Neural Networks, Oregon, July 2003.
[67]MATLAB, software simulation tool from The MathWorks Inc. http://www.mathworks.com/products.
[68]Belcastro, Celeste, and Belcastro, Christine, "On the Validation of Safety Critical Aircraft Systems, Part II: Analytical & Simulation Methods", Proceedings of AIAA Guidance Navigation and Control Conference, Austin TX, August 2003.
[69]Duke, E.L., Brumbaugh, R.W., and Disbrow, D., "A Rapid Prototyping Facility for Flight Research in Advanced Systems Concepts," IEEE Computer, May 1989.
[70]Hayhurst, K.J., Veerhusan, D.S., Chilenski, J.J., and Rierson, L.K., "A Practical Tutorial on Modified Condition/Decision Coverage," NASA/TM-2001-210876, May 2001.
[71]Soares, F., Loparo, K.A., Burken, J., Jacklin, S.A., and Gupta, P.P., "Verification and Validation of Real-time Adaptive Neural Networks using ANCT Tools and Methodologies," Proceedings of Infotech@aerospace Conference, Arlington, VA, Sept. 26-29, 2005.
[72]Bird, R.: RASCLE Version 2.0: Design Specification, Programmer's Guide, and
User's Guide. Baron Associates, Inc., February, 2002.
[73]Belcastro, Christine, and Belcastro, Celeste, "On the Validation of Safety Critical Aircraft Systems, Part I: Analytical & Simulation Methods", Proceedings of AIAA Guidance Navigation and Control Conference, Austin TX, August 2003.
[74]Holzmann, G. J., *The Spin Model Checker Primer and Reference Manual*, Addison-Wesley, Boston, MA, 2004.
[75]McMillan, K., *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 2003.
[76]Visser, W., Havelund, K., Brat, G., Park, S., and Lerda, F., "Model Checking Programs", Kluwer Academic Publisher, 2002.
[77]Havelund, K., "Using Runtime Analysis to Guide Model Checking of Java Programs," *SPIN Model Checking and Software Verification*, Vol. 1885 of Lecture Notes in Computer Science. pp. 245–264, Springer, 2000.
[78]Giannakopoulou, D., Pasareanu, C., and Cobleigh, J., "Assume-Guarantee Verification of Source Code with Design-Level Assumptions", Proceedings of the 26th International Conference on Software Engineering (ICSE'2004), Edinburgh, Scotland, May 2004.
[79]Artho, C., Havelund, K., and Biere, A., "High-Level Data Races", Proceedings 1st International Workshop on Verification and Validation of Enterprise Information Systems (VVEIS'03), Angers, France, 2003.
[80]Miller, S., Anderson, E., Wagner, L., Whalen, M., and Heimdahl, M., "Formal Verification of Flight Critical Software,", AIAA Guidance, Navigation, and Control Conference, San Francisco, CA, Aug 15-18, 2005. Paper AIAA-2005-6431.
[81]Scherer, S., Lerda, F., Clarke, E., "Model Checking of Robotic Control Systems," Proceedings of ISAIRAS 2005 Conference, Munich, Germany, Sept. 5-8, 2005
[82]Denney, E., Fischer, B., Schumann, J., Richardson, J., "Automatic Certification of Kalman Filters
for Reliable Code Generation", IEEE Paper No. 1207: 0-7803-8870-4/05.
[83]Dormoy, F.X., "SCADE* The Cost and Time Effective solution for Safety Critical Software Development", Esterel Technologies © 2001 — www.esterel-technologies.com
[84]Santhanam, V., "Crafting an FAA - Qualifiable Compiler," Software Tools Forum, Embry-Riddle Aeronautical University, College of Engineering, Daytona Beach, FL, May 18-19, 2004.
[85]Gupta, P. and Schumann, J., "A Tool for Verification and Validation of Neural Network Based Adaptive Controllers for High Assurance Systems", In Proceedings High Assurance Software Engineering (HASE). IEEE, 2004.
[86]Marjorie, D., Taylor, B., and Skias, S., "Rule Extraction From Dynamic Cell Structure Neural Network Used in a Safety Critical Application," Proceeding of Florida Artificial Intelligence Research Society Conference, Miami FL, May 17-19, 2004.
[87]Andrews, R. and Geva, S., "*RULEX & CEBP Networks As the Basis for a Rule Refinement System*," *Hybrid Problems, Hybrid Solutions*, John Hallam (Ed), IOS Press, 1995, pp1-12.
[88]Andrews, R., J. Diederich, and A. B. Tickle. 1995. A survey and critique of techniques for extracting rules from trained artificial neural networks. Knowledge-Based Systems, 8(6):373-389.
[89]Heimdahl, M.P.E, "Tool Intensive Software Development: New Challenges for Verification, Validation, and Certification," Proceedings of FAA and Embry Riddle Aeronautical University Software Tools Forum,May, 2004.