Toward V&V of Neural Network Based Controllers

Johann Schumann
RIACS / NASA Ames
schumann@email.arc.nasa.gov

Stacy Nelson
Nelson Consulting Company
NelsonConsult@aol.com

ABSTRACT

Online adaptation is a powerful means to handle unexpected slow or catastrophic changes of the system's behavior (e.g., a stuck or broken rudder of an aircraft). Therefore, adaptation is one way for realizing a self-healing system. Substantial research and development has been made to use neural networks (NN) for such tasks (e.g., integrated in various unmanned helicopters and test-flown on a modified F-15 aircraft). Despite the advantages of adaptive neural network based systems, the lack of methods to perform certification, verification, and validation (V&V) of such systems severely restricts their applicability.

In this paper, we report on ongoing work to develop V&V techniques and processes for NN-based safety-critical control systems, in our case an aircraft flight control system. Although the project ultimately aims at V&V of online adaptive systems, this paper focuses on the first part of this project dealing with so-called pre-trained neural networks (PTNN). V&V techniques developed here are important pre-requisites for handling the online adaptive case. In particular, we describe highlights of a process guide which has been developed within this project and discuss important V&V issues which need to be addressed during certification.

1. INTRODUCTION

Modern mission profiles and safety-critical applications (e.g., in civil or military aviation) require that the system (hardware and software) works reliably and without failures. It is also desired that the system can cope with unforeseen catastrophic changes or slow degradation. Such events lead to different handling characteristics which ultimately can lead to mission failure or even loss of life.

In this paper, we focus on a specific application, an Intelligent Flight Control System (IFCS) for an aircraft. The IFCS is an excellent example of an adaptive system because the adaptive control system learns about the changes in the aerodynamics from sensors on or connected to aircraft surfaces (flaps, elevators, rudders, ailerons, etc.) and provides vital updates to flight control system to compensate in the case of a failure. The importance of this application becomes evident by the fact that during the last 30 years, at least 10 aircraft have experienced major flight control system failures claiming more than 1100 lives. Therefore, the National Transportation Safety Board (NTSB) recommended *"research and development of backup flight control systems for newly certified wide-body airplanes that utilize an alternate source of motive power separate from that source used for the conventional control system"*.

Based upon these requirements, a variety of approaches for *adaptive control* (cf., e.g., [16]) has been developed. There, all changes in the plant dynamics are handled by the controller with the aim that the original response to control input is kept (or at least re-gained). A relatively novel, but very common approach is to use neural networks for adaptive control. Here, a variety of different architectures has been developed [15, 5, 2, 7, 20, 19, 10]. They differ substantially in the method of control (e.g., direct, inverse [2], predictive [15, 19]), the neural network architecture (e.g., feed-forward networks[15], dynamic cell structures (DCS) [9, 20], or SigmaPi [10]), as well as implementation details.

In safety-critical applications areas, such controllers have been successfully flown in unmanned aircraft (e.g., [2]), and in a modified F-15 aircraft at the NASA Dryden Flight Research Center.

Although these approaches are very promising with respect to their performance and their capability to adapt and self-heal the system, the question how to *ensure* the correct behavior of such a system has not been addressed in a satisfying way. In general, each piece of software which is used in a safety-critical application has to go through a rigorous certification process. Here, the certification authorities (e.g., the FAA for civil avionics) need to be convinced that the system including the software cannot fail. Current best practices (e.g., as defined in DO-178B [4]) basically rely on extensive testing—an approach principally not suited for adaptive systems.

In this paper, we describe preliminary research results on development of V&V (verification and validation techniques) for a neural network based Intelligent Flight Control System (IFCS). The IFCS is being constructed as part of the Intelligent Flight Control Project (IFC), a collaborative effort among the NASA Dryden Flight Research Center, the NASA Ames Research Center, the Boeing Phantom Works, the Institute of Software Research, Inc., and the West Virginia University. The goal is to develop and demonstrate a

WOSS '02, Nov 18–19, 2002, Charleston, SC, USA

flight control system which can effectively identify aircraft control characteristics using NNs and utilize this information to optimize aircraft performance in nominal and failure conditions. Within this project, the authors are developing advanced V&V techniques and processes [12] for the various "generations" of IFCSs. In this paper, we will report on results of the first part of this project, focusing on pre-trained Neural Networks (PTNN). In such a configuration, the NN is trained off-line using data from simulations, wind-tunnel experiments, and test-flights. Then the weights are frozen, before the actual control system is deployed. Although such a system cannot adapt to unforeseen changes, it is considered to be an important precursor for design, implementation, and deployment of truly adaptive systems (as currently being developed within the IFC project).

The rest of the paper is structured as follows: Section 2 introduces the basic architecture of the IFCS. In Section 3, we discuss the general question of performing V&V on a NN based system, before we will focus on a proposed V&V process for PTNNs (Section 4). In Section 5, we discuss some numerical V&V issues which are of importance for our application. These are based on the (often ignored) fact that NN algorithms are in essence numerical optimization algorithms. In Section 6, we summarize and sketch out future research directions.

2. INTELLIGENT FLIGHT CONTROL

The goal of IFCS is to develop and demonstrate a flight control system that can efficiently identify aircraft stability and control characteristics using neural networks and utilize this information to optimize aircraft performance in both normal and (simulated) failure conditions. The aircraft used to test IFCS has been highly modified from a standard F-15 configuration to include canard control surfaces, thrust vectoring nozzles, and a digital fly-by-wire flight control system.

Figure 1 shows a detailed view of the IFCS as used within the project. Sensor data coming from the aircraft (e.g., altitude, airspeed) flow to the Pre-Trained Neural Network (PTNN), Parameter Identification (PID), and Online Learning Neural Network (OLNN). The PTNN contains baseline derivatives computed from wind tunnel data. For the first part of the project, only this part of the system is used to control the aircraft. The output of the PTNN is then passed to the SOFFT (Stochastic Optimal Feed-Forward & Feedback Technology) controller, which takes into account the pilot inputs and provides control commands to the aircraft. SOFFT is a flight control architecture that is based on an explicit model-following concept. It uses the neural network outputs (stability and control derivatives data) to establish a plant model that controls the aircraft so it can achieve the desired handling qualities, and to continually optimize the feedback controller by integrating the neural network data in a real-time.

For the on-line adaptive configuration, both switches in Fig. 1 are closed. Thus actual sensor data from the PID are compared to the baseline derivatives. Deviations indicate a change of the aircraft's aerodynamic behavior which the triggers the adaptation. These derivative errors are used to train the on-line neural network which actually learns to *correct* the data, produced by the PTNN For details on the architecture see [9].

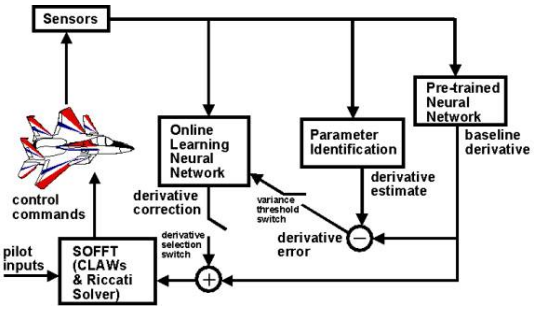

Figure 1: Overview of the IFCS architecture

3. V&V OF ADAPTIVE NN SYSTEMS

Any attempt toward verification of an (NN-based) adaptive system has to address substantially different issues compared to traditional software. One on hand, a NN-based system is a computer implementation of an analog process which usually results in an iterative numerical algorithm. Here, classical, formal verification is almost impossible, and validation by testing very difficult and time-consuming (but well understood, cf. e.g., [3, 6]). On the other hand, NN based adaptive systems pose a much more fundamental problem with respect to verification: verification usually is defined as the process to make sure that specification and implementation are equivalent. If, however, the adaptive system is supposed to adapt toward *unforeseen* changes in the system, then, of course there exists no specification of this event, and thus traditional verification is meaningless.

We therefore propose a different, multi-layered approach to V&V of NN based systems (Figure 2). A major prerequisite to developing high-quality software is to use an appropriate *Software process* which covers all stages from initial requirements to deployment and which is specifically tailored toward adaptive NN based systems (see Section 4).

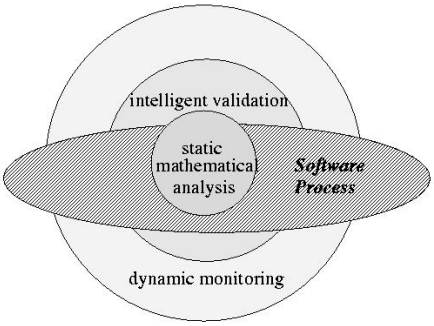

Figure 2: Layers of NN V&V methods

For the individual techniques, we identify three layers. The core-layer contains rigorous, mathematically sound results concerning robustness, stability, and convergence. Current state-of-the-art, however, only can provide relatively weak results, often in form of asymptotic guarantees. Typical examples here include Lyapunov proofs of (asymptotic) stability, or Vapnik-Cherenovis-dimension arguments to rea-

son about the NN's generalization abilities.

However, for our safety-critical applications, these results don't suffice, as, for example, they cannot provide convergence guarantees in the required short period of time (usually on the order of a few seconds). Thus, methods on the second layer need to address these issues by *intelligent testing*. Here, techniques to reduce the number of required test cases in the nominal and pre-analyzed failure case can be applied. Analysis of numerical issues, as well as tests for convergence and robustness of the system belongs to that layer.

For truly adaptive systems, however, we still don't have a guarantee for performance (will there ever exist one?). Here, the third layer comes into play: methods in this layer will *dynamically monitor* the NN and its behavior. Although it ultimately cannot provide the guarantee, it at least can return dynamic information on how the NN currently behaves and if the current state of the system is recoverable (healable) at all. Research on these methods are currently performed [17].

4. A NEURAL NET V&V PROCESS

In order for a neural network to fly on spacecraft or aircraft, it must be certified. In this context, certification is the process of obtaining a certificate from NASA and/or the Federal Aviation Authority (FAA) to indicate conformance with airborne software standards. One of the major prerequisites for certification is that all activities during the entire software development and software life cycle are performed according to a detailed *software process*. A process guidebook defines all steps of the process and provides guidance on the required documents and activities (e.g., which kinds of tests have to be carried out). In the project, a process guide for the IFCS is being developed. It is based upon applicable existing standards and best practices and has been augmented to specifically deal with the Neural Networks aspects of the system. In this paper, we describe the major elements of the process guide for PTNNs [12]. A process guide for on-line adaptation is currently in work and closely follows the line of this guidebook. There are several applicable NASA and industry V&V Standards for IFCS:

- NASA Guidebook for Safety Critical Software, NASA-GB-1740.13-96 [13]
- Trial-Use Standard for Information Technology Software Life Cycle Processes - Software Development, J-STD-016-1995
- IEEE Standard for Software Test Documentation, IEEE Std 829-1998 (Revision of IEEE Std 829-1983)
- NASA Procedures and Guidelines (NPG) 2820.DRAFT and NASA Software Guidelines and Requirements [14]. This document references IEEE/EIA Standards 12207.0, 12207.1, and 12207.2 which in turn reference standards published in 1995 as ISO/IEC 12207 [8].
- NASA Procedures and Guidelines (NPG) 8730.DRAFT 2, Software Independent Verification and Validation (IV&V) Management. This standard discusses the requirements for independent verification and validation.

  In a nutshell, a manned mission and any mission or program costing more than $100M will require IV&V[1].

In addition to the NASA standards, RTCA DO-178B [4] contains guidance for determining that software aspects of airborne systems and equipment comply with airworthiness certification requirements.

In order to ensure that verification techniques for neural networks meet these V&V standards, the following guidance is provided for integrating V&V of PTNN into the Software Life Cycle.

First, a well-defined process with discrete Software Life Cycle phases (Figure 3) including documented work products (called deliverables) must be defined for the overall project. This process must include analysis procedures established to ensure correctness of deliverables and scheduled reviews of major product releases. V&V of neural networks need to be applied to all phases of this Software Life Cycle process. They may be used to enhance rather than replace traditional testing.

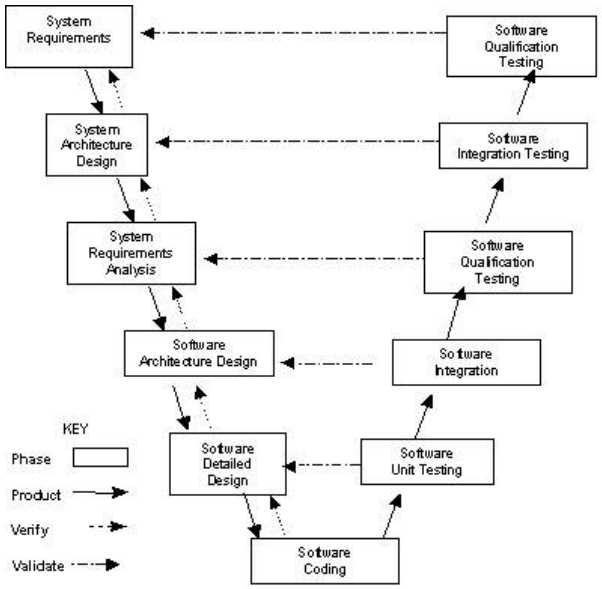

Figure 3: Lifecycle

In the following we list phases of the Life Cycle which have been substantially augmented for V&V of neural networks:

- *Systems Requirements*, *System Architectural Design*, and *Software Requirements Analysis*: Documentation for these phases must include the specification for the NN and its architecture. This description needs to contain the type of NN (feed-forward, Self Organizing Map, etc), the learning algorithm (gradient descent, Least Means Squared, Levenberg-Marquardt, Newton's method, etc.), and all parameters of the NN architecture (e.g., number of layers and hidden nodes, activation functions, initialization). Furthermore, a concise description of the inputs and outputs (including units and the expected and maximal range of values) and acceptable errors and training set(s) for PTNN

[1]Report review sent via email from Ken Costello, project manager, NASA IV&V Facility, dated October 13, 2001.

must be provided. These design and requirements descriptions must be provided in detail, but not necessarily in a formal language. Software models (e.g., Simulink) are also acceptable; however all parameters which might be hidden in the simulator must be made explicit.

- *Software Detailed Design* must include a description of precise code constructs required to implement the NN, including all data structures and algorithms (e.g., libraries for matrix operations).

- *Unit Testing* must include both black and white box testing for modularized NN code.

- *Software Integration* should verify that the NN interfaces with other software including proper inputs and outputs for the NN.

- *Software Qualification Testing* should ensure that the requirements are sufficiently detailed to adequately and accurately describe the NN.

- *System Integration Testing* should verify that the architectural design is detailed enough so, when implemented, the NN can interface with system hardware and software in various fidelity testbeds.

- *System Qualification Testing* should verify that the system requirements are sufficient enough to ensure that, when implemented, the NN will interface properly with the system in production.

5. NUMERICAL ASPECTS OF NN V&V

When it comes to actually perform V&V on a NN-based system, the primary focus of the analysis is laid on the neural network specific tasks of training, learnability, convergence, and generalization. Quite often, however, an important aspect is overlooked: a neural networks is in essence a numerical method to approximate a function; its training method is usually an algorithm to solve an unconstrained quadratic optimization problem (minimization of the error $E = \sum_i (o_i - t_i)^2$ where o_i is the actual NN output and t_i the expected training result). Therefore, a process for V&V of neural networks also needs to investigate the numerical issues underlying the NN and its implementation. Within this project, [18] discusses a list of important issues which need to be taken into account when V&V is attempted. Although the issues discussed in the following are tailored toward PTNN, principally the same issues occur when the neural network is trained online.

There are three distinct classes of problems (although problems in one class can also cause problems in another class): *(i)* general numeric properties, like scaling, conditioning, or sensitivity analysis, *(ii)* properties/issues specifically related to the training algorithm (e.g., convergence, termination), and, finally, *(iii)* issues with respect to the actual implementation on a digital computer (e.g., round-off errors, accuracy of library functions). In the following, we will discuss some of these properties and will give practical examples.

5.1 Scaling and Condition Number

It is obvious that the data ranges of inputs (and also of the outputs) of the neural network should be of the same order of magnitude in order to obtain best-possible accuracy. The reason for this is that the inputs are usually combined as a weighted sum to produce the activation levels of the artificial neurons on the second layer. Thus, extremely large input values would either dominate the sum or would require very small weights, leading to low accuracy. A typical example for unfavorable ranges of data is when one input of the NN represents the current altitude in feet (range: $0 \ldots 50{,}000$ft) and the other a small angle, e.g., $\alpha \in \{0, \ldots, 0.09\text{rad}\}$. In these cases, *scaling* of the inputs is important. This is usually accomplished by multiplying each input with a constant factor. Not so obvious, however, is the fact that scaling can affect the behavior of *some* training algorithms. For example, the well-known gradient descent algorithm is influenced by scaling, whereas a Newton method is not (for a detailed discussion see [3, 1, 6]).

In order to assess the NN's behavior with respect to changes in the input values or the weights, *sensitivity analysis* can be performed. For a sensitivity analysis of the neural network during training, it is often helpful to consider, how small changes in the *weights* affect the error function (and thus how fast and good the network can be trained). We start from the the quadratic form of the error function E. Let Φ be a quadratic representation (or approximation) of the error function E in the neighborhood of the (local) minimum $\mathbf{x}_0$. Φ is defined as

$$\Phi(\mathbf{x}) = \mathbf{g}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \boldsymbol{G} \mathbf{x}$$

where $\mathbf{g} \in \mathbb{R}^n$ is the gradient $\nabla E|_{\mathbf{x}_0}$, and $\boldsymbol{G}$ is the Hessian matrix. When we plot Φ in the vicinity of the minimum, we obtain surfaces shaped like a bowl or an elongated valley; the contour lines are ellipses. Fig. 4 shows the contours for two examples. A training algorithm tries to find the minimum by moving small straight segments along this error surface. It is easy to see that such a search algorithm can find the minimum much easier in a round bowl than in a long and narrow valley where small deviations can cause the algorithm to get stuck or "thrown" out of the valley.

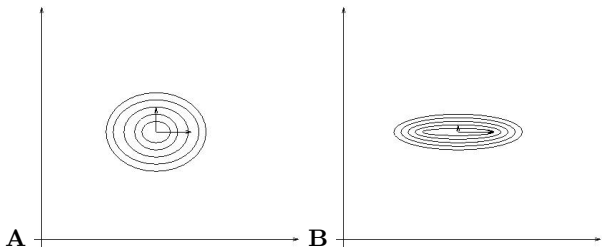

Figure 4: Contours of a well conditioned (A) and bad conditioned problem. The minimum is at the center of the ellipses, arrows mark the semi-axes of the ellipses.

More formally, the behavior of Φ (and thus also of E) in the vicinity of $\mathbf{x}_0$ is determined by the eigensystem of $\mathbf{G}$ The ratio of the largest and the smallest eigenvalue of $\mathbf{G}$ is defined as the condition number $\mathcal{C}$. $\mathcal{C}$ also determines the

eccentricity of the ellipses in Figure 4: a small condition number corresponds to an almost circular error surface (A); a large $\mathcal{C}$ results in a steep and long "valley" (B) which can lead to serious problems. Thus, the condition number of the problem is an important metric for the behavior of the NN with respect to training and convergence. Thus, all tests which are performed during NN V&V should, in addition to demonstrating convergence, calculate and assess the condition number.

5.2 Properties of the Training Algorithm

Training of a NN is an iterative numerical optimization algorithm, minimizing the error. Depending on the algorithm (e.g., gradient descent, Newton, Levenberg-Marqhart) and the training data, the algorithm must be stopped when optimal training of the NN has been reached. More specifically, the training algorithm needs to be stopped when: (i) the minimum has been reached (i.e., the problem has been solved), (ii) we have ground to a halt (i.e., there is no more progress), and (iii) we have exhausted memory or time resources.

In *all* cases, a careful analysis of the trained network and the reason why the training algorithm has stopped is important. [3, 6] elaborates on a number of possibilities why the training algorithm might have located a "wrong" local minimum or why it is oscillating or diverging, ultimately leading to an incorrect behavior of the NN-based system.

5.3 Hidden Library Problems

When implementing a numerical algorithm, the built-in library functions (e.g., exp, sin, cos) are mostly used "as given" without further consideration. In a safety-critical application, however, it is worthwhile to also analyze and test these functions carefully.

Many mathematical library functions are implemented using elaborate iterative numerical algorithms which are often directly written in assembly code for maximum performance. In practice, however, such implementations can have serious flaws and show other unexpected behavior.

Figure 5 shows the difference between a Java implementation of an exponential function (using Taylor-series expansions) and a standard UNIX library function. For small values of x, the accuracy is good, larger values result in unacceptable errors. This deviation is due to the very inaccurate calculation of the higher-order Taylor terms $x^i/i!$. In practice this means that the Java subroutine returns an acceptable value as long as its argument is always within a range of $-6 < x < 6$. For a safety-critical application, however, V&V has to guarantee that this range-restriction always holds or the subroutine should not be used.

Usually, the runtime of a numerical library function is considered to be small and constant. However, in practice, function evaluation for different parameter values can take different amounts of time. These effects, as shown in Figure 6 can be substantially large (here, approximately 14%). In particular in time-critical applications, these effects can have disastrous effects, because they can cause the entire system to miss hard timing deadlines. Since such timing errors are extremely difficult to detect, every effort needs to be made during design and V&V to avoid such situations upfront.

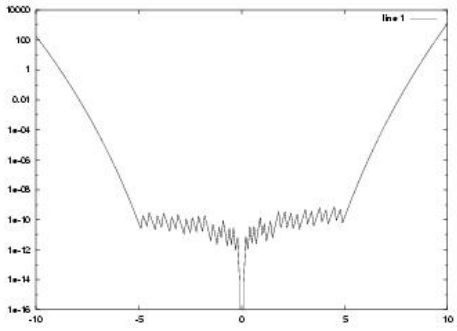

Figure 5: Square error between two different library implementations of the exponential function $\exp(x)$ (logarithmic y-scale).

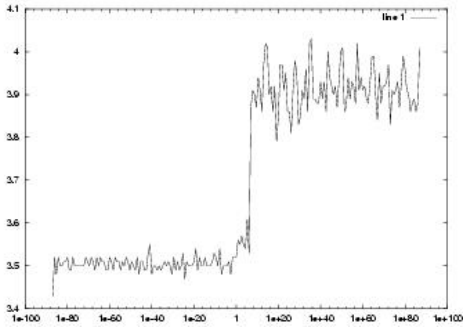

Figure 6: Run-time of library function $\sin(x)$ over a wide range of input values (log-scale). Run-time in seconds for 50,000 evaluations.

6. DISCUSSION AND FUTURE WORK

In this paper, we have reported ongoing work on the development of a V&V process and specific techniques for performing V&V of pre-trained neural networks. Although this process has to deviate in important aspects from one for standard software, it is expected that V&V of a system with a PTNN can be performed with reasonable effort. This, however, is only a first step. V&V of a system with on-line adaptation requires a combination of different and novel techniques to be effective. We are confident that our layered approach as described in Section 3 is helpful to address this topic. More specifically, we are currently working on the following aspects:

Classical techniques and techniques discussed so far only address the individual components (neural network and the controller), but not the entire system. It will be necessary to perform research on combining the different properties of the different components to obtain guarantees for the entire system. For example, stability and convergence properties of the neural network (e.g., expressed using VC-dimension arguments) need to be combined with stability proofs for the controller (e.g., with Lyapunov functions) to demonstrate stability and robustness of the NN-controller-plant system. First theoretical results on this have been presented in, e.g., [11].

As mentioned in Section 3, these mathematical results are often of asymptotic nature and cannot provide guarantees on issues like timing. Here, it is expected that testing and dynamic monitoring techniques are vital to demonstrate the adaptive behavior of the system in the nominal and analyzed failure case. However, the analysis of test results can be quite different. For example, a NN based system might adapt differently toward the same failure scenario, based upon different history and random initial values. This kind of non-determinism (e.g., to use different actuators to compensate a failure) is perfectly in alignment with the requirements and needs to be taken into account during testing.

The ultimate goal to provide guarantees for all unexpected cases can probably never be reached. So, for example, there always exists a point where the physical system is damaged/changed to such an extend that adaptation toward controllable behavior is simply not possible. In such cases, any attempt at adaptation would be futile. However, an adaptive system in a safety-critical application needs to quickly recognize this fact and should provide a warning (e.g., to the pilot who then can decide to eject). We are currently developing a *monitoring harness* around the neural network which, using Bayesian techniques, is capable of producing a measure on how the NN behaves [17].

With a combination of these techniques which are to be applied during a traditional V&V-phase and also during deployment, more and more guarantees about the NN's behavior can be given, thus facilitating the application of adaptive systems in safety-critical domains.

7. REFERENCES

[1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon-Press, Oxford, 1995.

[2] A. Calise and R. Rysdyk. Adaptive model inversion flight control for tiltrotor aircraft. In *AIAA Guidance, Navigation and Control Conference 1997*, number AIAA-97-3758. AIAA, 1997.

[3] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, volume 16 of *Classics in Applied Mathematics*. SIAM, 1996.

[4] DO-178B: Software considerations in airborne systems and equipment certification. URL: `http://www.rtca.org`, 1992.

[5] S. S. Ge, T. Lee, and C. J. Harris. *Adaptive Neural Network Control of Robotic Manipulators*, volume 19 of *World Scientific Series in Robotics and Intelligent Systems*. World Scientific, 1998.

[6] P. Gill, W. Murray, and M. Wright. *Practical Optimization*. Academic Press, 1981.

[7] M. Idan, M. Johnson, and A. Calise. A hierarchical approach to adaptive control for improved flight safety. In *AIAA Guidance, Navigation and Control Conference 2001*, number AIAA-2001-4209. AIAA, 2001.

[8] IEEE standards 12207.0, 12207.1, 12207.2. URL: `http://ieeexplore.ieee.org`, 1997.

[9] C. Jorgensen. Direct adaptive aircraft control using neural networks. Technical Report TM-47136, NASA, 1997.

[10] J. Kaneshige and K. Gundy-Burlet. Integrated neural flight and propulsion control system. In *AIAA Guidance, Navigation and Control Conference 2001*, number AIAA-2001-4386. AIAA, 2001.

[11] A. Kelkar. Neural networks for modeling and control of dynamic systems. Presentation at NASA Ames, Code IC, 2001.

[12] D. Mackall, S. Nelson, and J. Schumann. Verification and Validation of Neural Networks of Aerospace Applications. Technical Report CR-211409, NASA, 2002.

[13] NASA guidebook for safety critical software. Technical Report NASA-GB-1740.13-96, NASA, 1996.

[14] NASA procedures and guidelines NPG: 2820.draft, NASA software guidelines and requirements as of 3/19/01. NASA Ames Research Center, Moffett Field, California, USA, 2001. (Responsible Office: Code AE/Office of the Chief Engineer).

[15] M. Norgaard, O. Ravn, N. Poulsen, and L. K. Hansen. *Neural Networks for Modeling and Control of Dynamic Systems*. Springer, 2002.

[16] A. Sastry and M. Bodson. *Adaptive Control: Stability, Convergence and Robustness*. Prentice Hall, 1994. `http://www.ece.utah.edu/~bodson/acscr`.

[17] J. Schumann. Vericonn: Verification of controllers based on adaptive neural networks — white paper—. Technical report, NASA Ames, Automated Software Engineering, 2001.

[18] J. Schumann. V&V issues for neural networks. Technical Report RIACS-TR-XX-02, RIACS, 2002.

[19] D. Soloway and P. Haley. Reconfigurable flight control using neural generalized predictive control. In *AIAA Space 2000 Conference*, number AIAA-2000-5328. AIAA, 2000.

[20] J. Totah. Adaptive flight control and on-line learning. In *AIAA Guidance, Navigation and Control Conference 1997*, number AIAA-97-3537. AIAA, 1997.