Cite as:
Thompson, S., Davies, M., and Gundy-Burlet, K., "Hybrid Decompositional Verification for Discovering Failures in Adaptive Flight Control Systems." In *AIAA Infotech,* Atlanta, Ga., April 20-22, 2010.

Hybrid Decompositional Verification for Discovering Failures in Adaptive Flight Control Systems

Sarah Thompson*
SGT, Inc, Moffett Field, CA, 94035

Misty D. Davies† and Karen Gundy-Burlet‡
NASA Ames Research Center, Moffett Field, CA, 94035

Adaptive flight control systems hold tremendous promise for maintaining the safety of a damaged aircraft and its passengers. However, most currently proposed adaptive control methodologies rely on online learning neural networks (OLNNs), which necessarily have the property that the controller is changing during the flight. These changes tend to be highly nonlinear, and difficult or impossible to analyze using standard techniques. In this paper, we approach the problem with a variant of compositional verification. The overall system is broken into components. Undesirable behavior is fed backwards through the system. Components which can be solved using formal methods techniques explicitly for the ranges of safe and unsafe input bounds are treated as white box components. The remaining black box components are analyzed with heuristic techniques that try to predict a range of component inputs that may lead to unsafe behavior. The composition of these component inputs throughout the system leads to overall system test vectors that may elucidate the undesirable behavior.

I. Introduction

Adaptive flight control systems that utilize online learning neural networks (OLNNs) can theoretically allow an aircraft to maintain controllability after catastrophic failure. A prototype adaptive control system was successfully flown on the NASA F-15 ACTIVE aircraft using technology developed by the NASA Intelligent Flight Control (IFCS) project. However, the usefulness of these control systems is limited by their impermeability to standard validation and verification (V&V) techniques. In real systems, unmodeled dynamics are rife. Online learning neural networks will necessarily adapt based on these unmodeled dynamics, and it is difficult to bound the worst-case performance of these changing and highly nonlinear systems. Even in the

* Staff Scientist, Intelligent Systems Division, Mail Stop 269-1, AIAA Senior Member.
† Research Computer Engineer, Intelligent Systems Division, Mail Stop 269-1, AIAA Member.
‡ Research Scientist, Intelligent Systems Division, Mail Stop 269-3, AIAA Associate Fellow.

cases where the performance of the ideal control system can be bounded, this boundedness is usually achieved by making assumptions that limit the ability of the control system to adapt or by unrealistically simplifying the dynamics of the aircraft. Traditional verification and validation methods based on process (like those used to satisfy DO-178B requirements) are unlikely to provide an acceptable guarantee of safety for learning systems, and new validation and verification techniques are necessary[1-3]. Significant efforts towards the validation and verification of OLNNs have been made, and a recent survey highlights promising efforts ranging from the verification of the neural net design process to theoretical Lyapunov stability proofs that take into account uncertainties in the system[4]. Recent formal methods advances include symbolic bounded model checking using mathematical models of the system[5,6]. There has also been recent progress made on optimization techniques that can bound the behavior of the system given uncertainties[7-9]. In general, verification efforts have focused on verification of theoretical models; with the exception of runtime monitoring[10-14] the actual performance of the control system as implemented in code remains a rarely tackled problem.

The Robust Software Engineering (RSE) group within the Intelligent Systems Division at NASA Ames Research Center has developed a suite of techniques that, when used in combination, may speed the discovery of adaptive control system failures as implemented in flight software[15-18]. Compositional verification allows the breakdown of the overall system into multiple components. Each component is tackled separately, and the behaviors of the components are composed to describe the behavior of the entire system. Each component can be analyzed using any of a plethora of formal methods techniques, including model-checking, abstract interpretation, and symbolic execution, with the overall goal of finding component inputs that would produce some undesirable output. The range of possible behaviors for the entire system is not currently tractable to explicit techniques. At the system level, we use a combination of unsupervised[19] and supervised[20,21] machine learning techniques in a directed Monte Carlo global sensitivity analysis[22] to model the behavioral structure of the component and to predict the component-level input test vectors. Simulation-based validation depends on heuristics and cannot guarantee that the system is safe; however, validation testing is likely to uncover unsafe behaviors not discovered by using formal methods on simplified systems[23-25].

II. Methodology

As a prototype for this methodology we are using one of the implementations of the IFCS direct adaptive flight control system[26-28]. The control system we are using was implemented as a Mathworks Simulink model and was used as a research-level tool to understand neural networks in 2001. The model was not seeded with any known errors for the current paper; all errors existed in the model at the beginning of our validation and verification effort. A high-level flow graph is shown in Fig. 1. The output from the standard proportional-integral-derivative (PID) controller for the aircraft is sent to the OLNNs. The OLNNs compare the PID controller output with the output from the linearized plane reference model. The OLNNs attempt to drive the error to zero by augmenting the output from the PID controller before it is fed into the nonlinear dynamic inverse. The actuator model allows the control surfaces of the plane to be individually failed at any configuration. This kind of model-based design allows for ease of decomposing the overall system into modules, and then autocoding individually modeled blocks into C/C++ code.

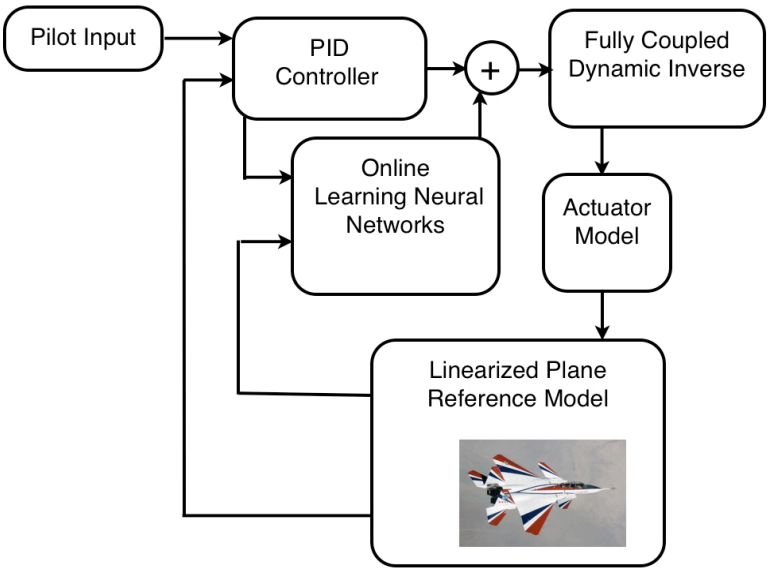

Figure 1. The IFCS adaptive control system. *This is the working version of the IFCS adaptive control system as used in this paper. The original version is modeled in Simulink. The Monte Carlo Filtering techniques described in this paper utilize the Simulink model directly. The other verification techniques described in this paper are performed on the autocoded C/C++ from MATLAB's Real Time Workshop.*

Decomposing the system into modules is highly desirable from the point of view of automated verification – many techniques tend toward execution time that is exponential in the size of the code under test, so the benefits are frequently substantially better than linear. Secondly, the techniques that can be applied to a particular subsystem are dependent upon the code itself, with linear algorithms being far more amenable to analysis than their nonlinear counterparts. In order to maximize fidelity, the formal methods analysis is conducted on the actual C/C++ flight code rather than on the Simulink model from which it was generated. Decomposition was carried out at the level of the Simulink model, with subsystems rendered in embeddable C++ source code separately by Real Time Workshop.

An initial, naïve attempt was made to find failures in the system using a Monte Carlo Filtering directed technique alone. Monte Carlo Filtering is a type of global sensitivity analysis in which we choose the inputs and ranges most likely to lead to some output[22]. Most analyses of this type are computationally expensive, tend to be limited to relatively small numbers of theoretically independent inputs, and also tend to assume that relationships between the inputs and outputs are smooth[35,36]. The types of problems being solved here involve failures—hence they can be non-smooth and of high dimensionality. To overcome the complications involved in finding the correlation coefficients for this sort of problem, we choose in practice to ignore the correlation coefficients altogether and use machine learning techniques that sample the space and solve the original question directly.

For this testing, we dispersed 11 controller parameters: 8 parameters for the PID controller and three learning gains within the neural controller. Each of these continuous controller parameters was used in an 3-factorial Monte Carlo[Barrett] experiment with 5 discretized bins for each parameter. The 3-factorial Monte Carlo approach assures that, for discrete variables, every possible test vector containing three values is exercised. Choosing a random value from within each bin discretizes continuous values. This process created 487 test vectors for the initial Monte Carlo simulation.

In general, we use the system-level requirements to create penalty functions for the Monte Carlo Filtering analysis. Each time a requirement is shown not to hold, the run is marked as a 'failure' for that requirement. It is possible for an individual run to 'fail' more than one requirement. We did not have access to the initial requirements document for the Simulink and

Stateflow example used here, and we were blind as to what failures (if any) might exist in the code. The initial test suite was scanned for

- changes in the sideslip angle, β, greater than 5 degrees between time steps
- changes in the angle of attack, α, greater than 30 degrees between time steps
- changes in the absolute height, *h*, greater than 1000 feet between time steps
- a roll rate, *p*, greater than 200 degrees/sec at any time
- a pitch rate, *q*, greater than 110 degrees/sec at any time
- a yaw rate, *r*, greater than 30 degrees/sec at any time
- any run in which an output value became infinity or NAN

during a 10 second simulation with a time step of 0.005 seconds. The pilot input consisted of one set of doublets simultaneously performed in all three axes. During the initial analysis of the results we discovered that slightly over half of the runs had failed, and that these failure cases produced NAN values.

For the sensitivity analysis step of the Monte Carlo Filtering analysis, we chose to correlate the NAN runs with their associated inputs. All of the runs with NAN values were sorted into their own class, and we used a supervised machine learning algorithm known as TAR3, a treatment learner[Menzies], to find rules involving a combination of up to 4 variables that increased the probability of getting a NAN on a run. The treatment learner found only one input in isolation—a PID roll controller gain, significantly correlated with the increase in probability for a NAN run. The nominal value for this parameter was 0.5, and the variable was dispersed between 0.1 and 0.6. As you can see in Fig. 2, values for this gain between 0.54 and 0.6 practically guaranteed a failure, with less than 10% of the runs in this range designated as successes. However, NAN failures were distributed outside this range in almost equal proportion to successes, and there appears to be no 'safe' range throughout the dispersion. This is the best information we can glean from the Monte Carlo Filtering validation testing. Some other technique must be used to determine the actual cause of the NANs.

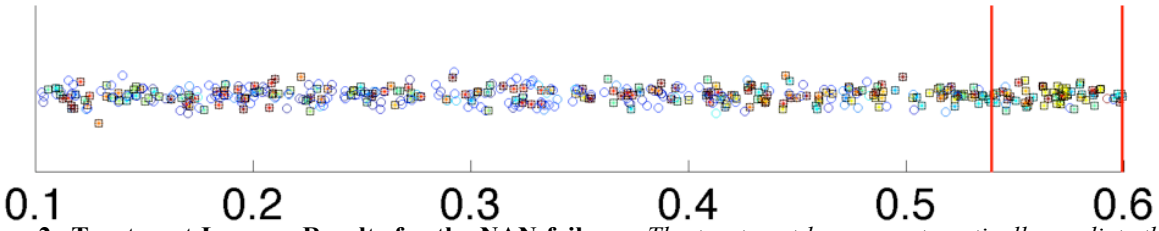

Figure 2. Treatment Learner Results for the NAN failures. *The treatment learner automatically predicts that a PID roll controller gain between 0.54 and 0.60 will increase the probability of a NAN failure. Each data point on this plot is an individual run. Blue circles are successes. Other colors are outlined by black boxes, and represent runs in which output values became NAN. Failures are colored by a blue-red gradient, with red data points corresponding to the runs that failed earliest. The red lines on the plot illustrate the range selected by the treatment learner.*

We also naïvely attempted to run through an entire simulation using the MCP explicit-state model checker[16]. To generate the C/C++ code for the simulation, we used MATLAB's Real-Time Workshop using the settings for embedded code and making sure that each block within the Simulink model was written as its own reusable function. MCP is capable of directly checking C and C++ code without prior translation or model extraction.

Model checkers are often not recommended for general use in control system verification because of the state-explosion problem—each state must be held in memory and, for large, continuous systems, the model checker is likely to run out of memory before the program has finished executing. When explicit model checkers have been used for control system verification, they are usually limited to standard PID controllers that are not in a loop with the plant and have been most successful when the system is abstracted[30]. The MCP model checker ran out of memory the first time through the simulation loop, during the convergence iterations for the neural network learning. As a sanity check, the simulation was repeated with only the PID controllers in the loop—the neural networks were removed. The Monte Carlo testing was repeated with the same test vectors and there were no NAN failures. During this attempt, MCP managed to execute several times around the loop before running out of memory.

Our next attempt used the compositional structure of the code imposed by the Simulink model to divide the work between the Monte Carlo Filtering validation testing and the explicit-state model checker. Given bounded inputs from other techniques, model checking is an effective way to efficiently generate explicit counterexamples that make it very clear why a particular piece of code has failed. The treatment learner's choice of the PID roll controller gain as the strongest correlation with the NANs, along with the fact that there were no NAN failures when the neural networks were removed from the simulation, suggested that the problem lay within the neural network component augmenting the roll command. We instrumented the inputs and the outputs to the roll neural network controller (shown in Fig. 3) and monitored the Simulink simulation for the outputs becoming NAN. We quickly found that there was a time step in which Uad1 became NAN even when all of the input variables were real values. This input test vector became the input values to the Roll_nn C++ function during MCP execution.

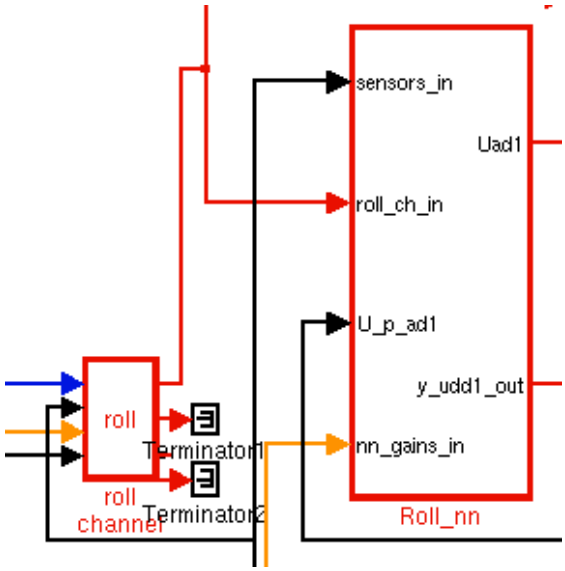

Figure 3. A Close-up of the Roll Controller Portion of the Simulink Monitor. *The input variables to Roll_nn—sensors_in (a bus variable), roll_ch_in (a bus variable), U_p_ad1 (a continuous value), nn_gains_in (a bus variable)—along with the two continuous output variables, were all monitored during Simulink execution. An input test vector of all real numbers producing a output Uad1 value of NAN was used as an input to the model checker.*

III. Results

The autogenerated code from the roll neural network was fed into the model checker MCP. The neural network code in isolation was small enough that MCP could analyze it to failure without running out of memory. MCP has been modified to be able to detect NANs in the

```
---
** starting the model **
--- Step #0
--- rtu_U_p_ad1 = 5.204576e+05 (size 8)
--- mf1 = -7.478910e+01
--- mf1 = -5.205324e+05
--- mf1 = 5.205324e+05
--- mf1 = inf
--- mf1 = inf
--- mf1 = 0.000000e+00
--- mf1 = 0.000000e+00, lb->mf1 = 0.000000e+00
--- u1dd = nan
### GEN2_w_test2_Roll_nn.c(348): Assertion failure: 0
```

Figure 4. Selected Results from the MCP model checker. *Using the input test vector of all real numbers from the SIMULINK instrumentation, the MCP model checker was able to provide a trace for the NAN result.*

floating point values and report the trace of variables that led to the NAN. A snippet of the results from the model checker are shown in Fig. 4.

The autogenerated SIMULINK code is well-annotated, as shown in Fig. 5. It is easy, using the comments, to trace the autogenerated code back to the original SIMULINK model. In this case, a value was not checked for its size before being fed to an exponential function. Placing a limit on the size of the value before the exponential eliminated the NAN results.

```
/* Product: '<S57>/Product4' incorporates:
 *  Constant: '<S54>/cp10'
 *  Constant: '<S55>/cp20'
 */
localB->Product4_a[0] = 1.0000000000000002E-02;
localB->Product4_a[1] = localB->cp11 * 0.1;
localB->Product4_a[2] = localB->cp12 * 0.1;
localB->Product4_a[3] = localB->cp13 * 0.1;

/* Sum: '<S55>/Sum2' */
localB->MathFunction1 = localB->MathFunction1 - rtu_U_p_ad1;

/* Gain: '<S55>/Gain' */
localB->MathFunction1 = -localB->MathFunction1;

/* Math: '<S55>/Math Function' */
```

Figure 5. Selected Autogenerated Code from Real Time Workshop. *The code is well-annotated, with the comments directing the reader back to the blocks in the original SIMULINK model.*

.

It is important to note that this work in this paper made no attempt to discover whether the flaw was in the original design of the controller or in the implementation of the control design. The authors had no knowledge of the original requirements the designers had in mind. Much more could have been done if those requirements had existed. However, a NAN result in a controller could have been a fatal flaw had it been implemented on an aircraft. The process described here is a highly automated way to discover these flaws.

IV. Conclusion

Model-based design is gaining traction in the space and aviation industries. Among its many advantages it gives the validation and verification practitioner information about the ways that the programmers decomposed the design. This information can be used in divide-and-conquer compositional verification techniques where many tools can be used in concert to provide automatic verification.

The test example used here was research-level code. With very little information about the original intended requirements for the code, we used a series of automatic techniques in order to uncover a serious bug. In particular, we used a heuristic directed testing technique in order to achieve the necessary scalability for the code, and then a formal methods technique in order to

order to get an explicit trace of the error. If we had received more information about the intended requirements of the control system design (MATLAB allows you to specify requirements with the model which will be autocoded as comments into the RTW code for code traceability purposes) there are two more formal methods techniques which would have been promising to use:

- *Partial evaluation/Symbolic Execution*. This approach allows certain functions (with bounded loops, limited use of pointers and strictly linear arithmetic operations) to be transformed into satisfiability modulo theorem (SMT) problems, which allow solvers such as Yices[34] to be used to generate test cases for arbitrarily chosen outputs.
- *Abstract Interpretation*. This approach[31-33] allows some nonlinear functions to be safely approximated, making it possible to derive useful information about their behavior. Though this approach can generate false positives, when performed correctly it can never yield false negatives – consequentially, if code is shown to be correct by this method, this is a mathematically sound result.

As formal methods techniques become easier for the everyday practitioner to use, it is important to counteract the impression that formal verification for control systems is too expensive and difficult.

Acknowledgments

This research was conducted at NASA Ames Research Center. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

[1]Jacklin, S., and Schumann, J. Gupta, P., Richard, M., Guenther, K., and Soares, F. "Development of Advanced Verification and Validation Procedures and Tools for the Certification of Learning Systems in Aerospace Applications." In *AIAA Infotech* 2005.

[2]Santhanam, V. "Can Adaptive Flight Control Software Be Certified to DO-178B Level A?" In *NASA and FAA Software and CEH Conference*, 2005.

[3]Jacklin, S., Lowry, M., Schumann, J. Bupta, P., Bosworth, J., Zavala, E., Kelly, J., Hayhurst, K., Belcastro, C. and Belcastro, C., "Verification, Validation, and Certification Challenges for Adaptive Flight Critical Control System Software." In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004.

[4]Schumann, J. and Liu, Y. (ed.), *Applications of Neural Networks in High Assurance Systems*, Studies in Computational Intelligence, Springer, Berlin, 2010, pp. 1-19.

[5]Gulwani, S. and Tiwari, A. "Constraint-based Approach for Analysis of Hybrid Systems," In *CAV* 2008.

[6]Tiwari, A. "Formally Analyzing Adaptive Flight Control," In NSV II (co-located with CPSWeek 2009).

[7]Crespo, L., Giesy, D., and Kenny, S., "Robustness Analysis and Robust Design of Uncertain Systems," *AIAA Journal*, Vol. 46, No. 2, 2008, pp.388-396.

[8]Menon, P., Bates, D. and Postelthwaite, I., "Nonlinear Robustness Analysis of Flight Control Laws for Highly Augemented Aircraft," *Control Engineering Practice*, Vol. 15, 2007, pp.665-662.

[9]Menon, P., Bates, D. and Postelthwaite, I., "Computation of Worst-Case Pilot Inputs for Nonlinear Flight Control System Analysis," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 1, 2006, pp.195-199.

[10]Yerramalla, S., Cukic, B., and Fuller, E. "Lyapunov Stability Analysis of Quantization Error for DCS Neural Networks," In *The Proceedings of the International Joint Conference on Neural Networks, IJCNN* 2003.

[11]Schumann, J. and Gupta, P. "Monitoring the Performance of a Neuro-adaptive Controller," In *Proceedings of the 24th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, 2004.

[12]Liu, Y., Yerramalla, S., Fuller, E., Cukic, B., and Gururajan, S. "Adaptive Control Software: Can We Guarantee Safety?" In *Proceedings of the 28h International Computer Software and Applications Conference; Workshop on Software Cybernetics*, 2004.

[13]Liu, Y., Cukic, B., Jiang, M., and Xu, Z. "Predicting with Confidence—An Improved Dynamic Cell Structure," *Advances in Neural Computation, Vol 1*. Springer, Heidelberg, 2005, pp 750-759.

[14]Schumann, J., Liu, Y., "Tools and Methods for the Verification and Validation of Adaptive Aircraft Control Systems," In *IEEE Aerospace Conference,* 2007.

[15]Giannakopoulou, D., Kramer, J. and Cheung, S.C. "Analysing the Behaviour of Distributed Systems Using Tracta,"

Journal of Automated Software Engineering, special issue on Automated Analysis of Software, Vol. 6(1), Kluwer Academic Publishers, January 1999, pp. 7-35.

[16]Brat, G. and Thompson, S. "Verification of C++ Flight Software with the MCP Model Checker," *IEEE Aerospace Conference*, March 2008.

[17]Schumann, J., Gundy-Burlet, K., Pasareanu, C., Menzies, T., and Barrett, T. "Tool Support for Parametric Analysis of Large Software Systems", *Proceedings of Automated Software Engineering, 23rd IEEE/ACM International Conference,* 2008. Gundy-Burlet, K., Schumann, J., Barrett, T., and Menzies, T., "Parametric Analysis of a Hover Test Vehicle Using Advanced Test Generation and Data Analysis," *AIAA Aerospace*, 2009.

[18]Gundy-Burlet, K., Schumann, J., Barrett, T., and Menzies, T., "Parametric Analysis of ANTARES Re-entry Guidance Algorithms Using Advanced Test Generation and Data Analysis," *9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2007.

[19]Fischer, B., and Schumann, J. "Autobayes: A System for Generating Data Analysis Programs From Statistical Models," *Journal of Functional Programming*, Vol. 13, 2003, pp. 483-508.

[20]Hu, Y., "Treatment Learning: Implementation and Application," Masters Thesis, Department of Electrical Engineering, University of British Columbia, 2003.

[21]Hu, Y., and Menzies, T. "Data Mining for Very Busy People," *IEEE Computer*, Vol. 36, No. 11, 2003, pp. 22-29.

[22]Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S., *Global Sensitivity Analysis: The Primer*, Wiley, Chichester, 2008, Chaps. 1, 5.

[23]Bateman, A., Ward, D. and Balas, G. "Robust/Worst-Case Analysis and Simulation Tools." In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2005.

[24]Pacheco, C., Lahiri, S., Ernst, M., and Ball, T., "Feedback-directed Random Test Generation." In *ICSE'07, Proceedings of the 29th International Conference on Software Engineering*, 2007.

[25]Satpathy, M., Yeolekar, A. and Ramesh, S. "Randomized Directed Testing (REDIRECT) for Simulink/Stateflow Models." In *EMSOFT'08*, 2008.

[26]Kaneshige, J., Bull, J., and Totah, J., "Generic Neural Flight Control and Autopilot System." In *AIAA Guidance, Navigation, and Control Conference*, 2000.

[27]Rysdyk, R. and Calise, A. "Fault-tolerant Flight Control via Adaptive Neural Network Augmentation," *AIAA American Institute of Aeronautics and Astronautics*, Vol. AIAA-98-4483, 1998, pp. 1722-1728.

[28]Calise, A. and Rysdyk, R. "Nonlinear Adaptive Flight Control Using Neural Networks," *IEEE Control Systems Magazine*, Vol. 21, No. 6, pp. 14-26.

[29]Barrett, A., "A Combinatorial Test Suite Generator for Gray-Box Testing", *Third IEEE International Conference on Space Mission Challenges for Information Technology*, 2009.

[30] Scherer, S., Lerda, F., and Clarke, E., "Model Checking of Robotic Control Systems." In *Proceedings of ISAIRAS 2005 Conference*, 2005.

[31]Futamara, Y. "Partial Evaluation of Computation Process—an Approach to a Compiler-Compiler," *Systems, Computers Control*, Vol. 2, Issue 5, 1971, pp. 45-50.

[32]Jones, N., Gomard, C. and Sestoft, P., *Partial Evaluation and Automatic Program Generation*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[33]Cousot, P. and Cousot, R. "Abstract Intepretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages,* 1977, pp. 238-252.

[34]Dutertre, B. and de Moura, L. "The Yices SMT Solver," Tool paper found at URL: http://yices.csl.sri.com/tool-paper.pdf [cited 4 November 2009].

[35]Rose, K., Smith, E., Gardner, R., Brenkert, A. and Bartell, S. "Parameter Sensitivities, Monte Carlo Filtering, and Model Forecasting Under Uncertainty," *Journal of Forecasting*, Vol. 10, 1991: pp. 117-133.

[36]Oakley, J. and O'Hagan, A. "Probabilistic Sensitivity Analysis of Complex Models: A Bayesian Approach." *Journal of the Royal Statistical Society B*, Vol. 66, 2004, pp. 751-769.