Y. He, H.K.H. Lee, and M. Davies. "Towards Validation of an Adaptive Flight Control Simulation Using Statistical Emulation." AIAA Infotech@Aerospace Conference, Garden Grove, CA, 19-21 June, 2012.

Towards Validation of an Adaptive Flight Control Simulation Using Statistical Emulation

Yuning He* and Herbert K.H. Lee†

University of California Santa Cruz, Santa Cruz, CA, 95064, USA

Misty D. Davies‡

NASA Ames Research Center, Moffett Field, CA, 94035, USA

Traditional validation of flight control systems is based primarily upon empirical testing. Empirical testing is sufficient for simple systems in which a.) the behavior is approximately linear and b.) humans are in-the-loop and responsible for off-nominal flight regimes. A different possible concept of operation is to use adaptive flight control systems with online learning neural networks (OLNNs) in combination with a human pilot for off-nominal flight behavior (such as when a plane has been damaged). Validating these systems is difficult because the controller is changing during the flight in a nonlinear way, and because the pilot and the control system have the potential to co-adapt in adverse ways—traditional empirical methods are unlikely to provide any guarantees in this case. Additionally, the time it takes to find unsafe regions within the flight envelope using empirical testing means that the time between adaptive controller design iterations is large. This paper describes a new concept for validating adaptive control systems using methods based on Bayesian statistics. This validation framework allows the analyst to build nonlinear models with modal behavior, and to have an uncertainty estimate for the difference between the behaviors of the model and system under test.

Nomenclature

$\sim$	=	'Distributed as', referring to data that corresponds with a probability distribution
m	=	The mean of a Gaussian distribution
σ	=	The standard deviation of a Gaussian distribution
$\mathcal{X}$	=	A set of inputs, the set might consist of scalars, vectors, and/or categorical variables
$\mathcal{Y}$	=	A set of outputs, the set might consist of scalars, vectors, and/or categorical variables
$\vec{\beta}$	=	A vector of regression coefficients
$R = \text{diag}(\vec{r})$	=	A diagonal matrix of p positive roughness parameters $\vec{r} = (r_1, \ldots, r_p)$
c	=	A correlation function between input variables
K_{lat}	=	lateral stick gain
K_{lon}	=	longitudinal stick gain
K_{pedal}	=	rudder gain
ζ	=	damping coefficient
$w_{p,q,r}$	=	proportional gain (all axes)
K_3	=	controller gain (yaw axis)
$\lambda_{p,q,r}$	=	NN learning rates (all axes)

*Ph.D. Candidate, Department of Applied Statistics, UCSC School of Engineering, 1156 High Street.
†Professor, Department of Applied Statistics, UCSC School of Engineering, 1156 High Street.
‡Research Computer Engineer, Intelligent Systems Division, Mail Stop 269-1, AIAA Member.

$\vec{err}^{1,2}_{p,q,r}$	=	error wrt. reference model (proportional, integral), all axes [degrees, degrees*sec]
$\vec{h}$	=	altitude [feet]
$\vec{p}, \vec{q}, \vec{r}$	=	roll, pitch, and yaw rates [degrees/sec]
$\vec{\alpha}$	=	angle of attack [degrees]
$\vec{\beta}$	=	sideslip angle [degrees]
F	=	a Boolean variable: TRUE for cases which fail and FALSE otherwise
T	=	time to failure [number of time steps]

I. Introduction

Engineers design aircraft control systems using a host of assumptions. These control systems are then proved to be stable under these assumptions using Lyapunov or other well-studied and trusted methods. In reality, however, the assumptions used to design the original control system do not hold exactly. As a result, the original design must be modified to remove instabilities that come from modeling errors in the plant and in the environment.

Discovering these errors and, after the errors are removed, collecting evidence that the control system is behaving as expected, is done through validation. The meaning of the word *validation* is not consistent across communities: here we mean that we are attempting to answer the question "Did I build the right thing?" as opposed to the verification question "Did I build the thing right?" A more distinct way of saying this is that we are *determining the degree to which a model (and its associated data) is an accurate representation of the real world from the perspective of the intended uses of the model.*[1, 2]

The validation of aerospace software, including flight control systems, is primarily done through empirical testing. Inputs to the systems are selected—often randomly, sometimes through elicitation of test cases from domain experts—and the outputs between the lower-fidelity and higher-fidelity systems are compared. Occasionally, this is no more sophisticated than graphing the outputs on top of each other and asking an engineer to determine whether the data match. This approach is more likely to be sufficient if the behavior of the system is linear (or approximately linear), if there is relatively little noise in the measurements, if the dimensionality of the system is low, and if each component of the system is designed so that it is sensitive to relatively few of the other components in the system. The one truly adaptive and well-connected component left in the system is the human being, and the human being is usually modeled either as a perfect controller, as an adversary to the rest of the system[3] or as an automaton who commands doublets despite the behavior of the plane and the environment.

It is arguable whether current airspace systems are simple enough to be validated using empirical testing alone. Future airspace systems will need more capabilities in order to respond to a more congested and heterogenous environment. Designs that enable these capabilities tend to be distributed, flexible, and information-dependent, and may include adaptive elements other than human beings. In short, future capabilities are certain to be bought at a cost of increased complexity. In this regime, empirical testing is woefully inadequate as a validation method. In order to enable these future capabilities we must develop new validation techniques.[4–7]

For an aerospace system of any appreciable size, the space of inputs and outputs is infinite. If the system is complex enough, isolated test points chosen from this infinite space do a poor job of allowing an analyst to infer the behavior of the system between test points. Increasing the number of test points randomly in order to generate the necessary confidence in the model requires an intractable number of tests.[8] What we need is a way to compare low-fidelity and high-fidelity data in a way that suggests the behavior of the system at untested values, and also gives a reliable level of confidence in that suggestion.

While aerospace systems are complex, they do have the advantage that they are based in physics. This means that, for large regions of the space, a small change in the inputs will lead to a small change in the outputs. In the parts of the space where this is true, we should be able to approximate the space between our test points in a reliable way. However, aerospace systems also have physical discontinuities (shock waves, for example), and modal behaviors (e.g. autopilot modes) that must be discovered and explored in a more combinatorial way.

As a first step towards improving current validation methods, we propose an approach based on Bayesian statistics. This approach has several benefits:

Uncertainty Quantification Approximations for untested points have a mathematically-based estimate of certainty given for the approximation.

Representation of Mixed Discrete & Continuous Relationships The use of Treed Gaussian Processes (discussed in Section §II.B) allows for a different mathematical model to be used on either side of a discontinuity.

Explicit Consideration of Time Series For adaptive and autonomous systems, the behavior of the system is dependent on the events that have happened previously.

Automation Our process requires less domain expertise than current validation techniques. More automation work remains to be done, as we will discuss in Section §V.

We have applied our approach to the analysis of a NASA Gen2 Intelligent Flight Control System (IFCS) simulator.[9] The IFCS is an adaptive control system designed to give fault tolerance in the event of sensor or actuator failures. There is significant work on new methods for verifying and validating adaptive control methods,[10] but most work has focused on verification of theoretical models[11] or on runtime monitoring.[12–16] There has also been recent progress made on optimization techniques that can bound the behavior of the system given uncertainties,[3,17,18] but this uses mathematical models of the system instead of collected data. We will highlight the differences between our proposed, novel approach and these other methods in Section §V.

II. Methodology

As mentioned in the introduction, our key goal is *determining the degree to which a model (and its associated data) is an accurate representation of the real world from the perspective of the intended uses of the model.*[1,2] For our purposes, Figure 1 illustrates our problem. The dark circle to the left of the figure represents reality, and our goal is to use the light circle to the far right of the figure (in this case, we are assuming that we are dealing with software-intensive systems) to predict or to affect reality. However, between reality and our final computational model, we usually have several other models, and each model increases abstraction at the cost of new errors. We have different artifacts from each differing-fidelity model, and our validation technique must compare these artifacts to make a prediction about a.) whether or not we are likely to achieve our goal, and hopefully, b.) with what confidence we are making our prediction.

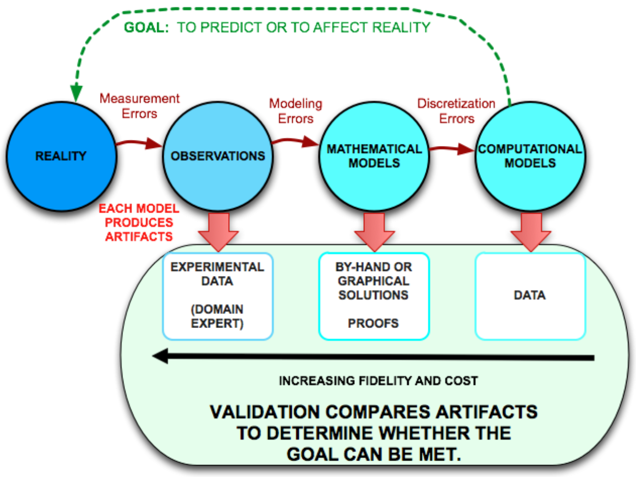
GOAL: TO PREDICT OR TO AFFECT REALITY
Measurement Errors
Modeling Errors
Discretization Errors
REALITY
OBSERVATIONS
MATHEMATICAL MODELS
COMPUTATIONAL MODELS
EACH MODEL PRODUCES ARTIFACTS
EXPERIMENTAL DATA (DOMAIN EXPERT)
BY-HAND OR GRAPHICAL SOLUTIONS PROOFS
DATA
INCREASING FIDELITY AND COST
VALIDATION COMPARES ARTIFACTS TO DETERMINE WHETHER THE GOAL CAN BE MET.

Figure 1. Validation Between Several Differing Fidelity Models of Reality

As a very simple, one-dimensional validation example, examine Figure 2, in which we compare experimental data for two different airfoils against the linear theory most commonly used as an approximation to reality. In particular, we compare a Cessna 172 airfoil (wind tunnel data shown as X's) and an X31 airfoil (wind tunnel data shown as circles) against thin airfoil theory (the solid line). We are assuming that the angle-of-attack α is a given, and that an engineer would like to predict the lift coefficient C_L. A person with domain knowledge examining this figure might make the following determinations:

Thin Airfoil Theory is a Good Predictor of the Cessna 172 Airfoil (In Some Range) In particular, thin airfoil theory can predict the behavior of the Cessna 172 up to about $\alpha = 10$ degrees very well. Between about 10 and about 17 degrees, the thin airfoil theory works less well. At some point between 15 and 20 degrees you appear to have a non-linear behavior (a stall), at which point thin airfoil theory ceases to be a good predictor.

Thin Airfoil Theory is a Poor Predictor of the X31 Airfoil The slope of the lift curve for the X31 is much less steep than thin airfoil theory predicts.

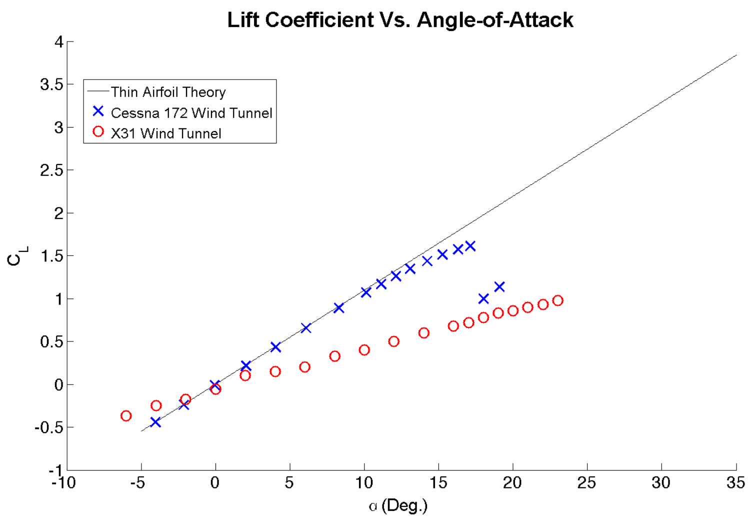

Figure 2. A Comparison of Two Different Airfoils Against Thin Airfoil Theory

This sort of straight comparison between theoretical or computational data and experimental data (or expert opinion) is traditionally the way that validation is done for aerospace systems. It works well when the quantities of interest are dependent on only a few variables, and the relationships are more-or-less linear. It also tends to assume that people are the only adaptive elements of the system, and those people can be trusted to do the right thing. At this point, few of these assumptions hold true for the National Airspace,[19] and, arguably, NextGen will be even more difficult to reliably validate. The NextGen airspace has been optimized to allow for maximum traffic, and optimized systems are inherently more non-linear. NextGen airspace is also more likely to be complex and have interdependent components—the heterogeneity of the airspace is expected to increase for both aircraft performance and equipage. Also, the autonomy and adaption of elements in the system other than people will increase. In the current and near-term we are already seeing tools that suggest traffic solutions both for air traffic controllers and for pilots, and in the far-term we are likely to see tools that replace functions that are currently delegated to people.

We are advocating for a new validation process; one in which the assumptions listed in the previous paragraph do not need to hold. Such a process would need to be able to predict nonlinear behavior with discontinuities (shocks, stalls, and modes), and to be able to reason over time series behaviors (like trajectories). To that end, we have begun research on a framework utilizing *statistical emulation* (Section §II.A) for validation. We have extended statistical emulation using *Treed Gaussian Process* (Section §II.B) for the prediction of time series curves, and we have applied the technique to prediction within an adaptive flight control system. We are in the process of extending the technique further to find explicit boundaries between successful and unsuccessful behavior in these non-linear, high-dimensional spaces, and also to choosing test points within the system that most quickly decrease the remaining statistical uncertainty for the behavior of airspace systems.

II.A. Statistical Emulation

In many fields of science, sophisticated mathematical models are designed and implemented with large computer simulators, in order to predict complex real-world phenomena. A computer simulator is used to run computer experiments: a number of runs of the simulator with various inputs. Standard techniques for sensitivity and uncertainty analysis of these systems always demand a very large number of model runs to thoroughly analyze the system, and even when a single model run takes several minutes or seconds, these methods become impractical. For example, the Monte Carlo method common in practice for analyzing the system is simple to implement but becomes impractical when the code is costly to run and/or a high degree of certainty is required for the system, because of the large number of runs required.[8]

Statistical emulation can overcome these limitations. We can describe the simulator as a function $f(.)$ that maps inputs $\mathcal{X}$ into an output $\mathcal{Y} = f(\mathcal{X})$. In general, $\mathcal{X}$ and $\mathcal{Y}$ are sets that include scalars, time series, and categorical variables. For our initial adaptive control simulation work, $\mathcal{X}_i$ is a vector $\vec{x_i}$ for simulation run i in which every element of $\vec{x_i}$ is a scalar input value, as shown in Table 1. Similarly, $\mathcal{Y}$ is a set containing time-series vectors $\vec{y_i^j}$ and scalar values. An approximation $f^*(.)$ of $f(.)$ can be used for the analysis, where $f^*(.)$ has been chosen for ease of prediction and for the ability to approximate functions of the type we see in the simulator to arbitrary closeness. If the approximation is good enough, then the uncertainty and sensitivity measures produced by the analysis will be sufficiently close to those obtained using the original simulator $f(.)$. And if the approximation is also simpler than the original function, (more importantly, if can be computed much faster than $f(\mathcal{X})$), then it can help with the complex analysis tremendously.

A statistical emulator is a statistical approximation to the simulator. It provides an entire probability distribution for the simulator $f(\mathcal{X})$. We can regard the mean of that distribution as the approximation $f^*(\mathcal{X})$ to $f(\mathcal{X})$. The statistical emulator provides a distribution around that mean; this means that it describes how close the approximation is likely to be to the true $f(\mathcal{X})$. In essence, a statistical emulator is a probability distribution for the entire function $f(.)$. The key benefit for using statistical emulation in aerospace is that we can *quantify uncertainty.* Since the emulator gives a probability distribution for the true function $f(.)$, this induces a probability distribution for any uncertainty and sensitivity analysis or other measures that might be derived from $f(.)$. Because of the incredibly complex nature of the engineering and scientific problems, using statistical emulation also allows the analyst to benefit from a very well-established theory of stochastic processes in the statistical literature—this literature describes how to define the uncertainty in the approximation, and also describes the best ways to analyze different configurations of the system.

The criteria that a good emulator should satisfy are:

1. At a design point $\mathcal{X}_i$, the emulator should reflect the fact that we know the true value of the simulator output, so it should return $f^*(\mathcal{X}_\rangle) = \mathcal{Y}_\rangle$ with no uncertainty.

2. At other points, the distribution for $f(\mathcal{X})$ should give a mean value $f^*(\mathcal{X})$ that represents a good interpolation or extrapolation of the training data, and the probability distribution around this mean should be a realistic expression of uncertainty about the values the simulator might produce.

It is important to contrast the types of statistical emulation we are proposing with other, commonly-used types of emulation. Some common approaches to emulation include fitting regression models to the data, including using neural network for regression. However, it is also very easy to see that commonly-used regression models do not satisfy the above two criteria. For instance, a high order polynomial may satisfy the first criterion but would then fail the second. In practice, neural network interpolations typically fail criterion 2 by not allowing for enough uncertainty about how the true simulator will behave.

The Gaussian process, which is used in our paper, is an analytically very tractable form of stochastic process. A properly fitted Gaussian process emulator will satisfy the two fundamental criteria. The model includes a description of the correlation or covariance of the outputs, which enables the model to encompass the idea that differences in the output will be small if there are only small differences in the inputs. A Treed Gaussian Process (TGP) is a further extension from GP which has the benefit of a non-stationary covariance structure (as discussed in detail in the next section), and thus more flexible modeling.

II.B. Gaussian Processes, Treed Gaussian Processes, and Classification Treed Gaussian Processes

A Gaussian Process (GP) is a widely used statistical model. It can be viewed as the generalisation of a Gaussian distribution over a finite vector space, to a function space of infinite dimension. GPs are used to infer or predict function values at a finite set of prediction points from the observed data. A Gaussian distribution is fully specified by its mean and covariance matrix; similarly, a Gaussion Process is fully described by its mean and covariance function. For some real-valued p-dimensional inputs $\vec{x}$, a GP is defined on the space of functions such that the function values $y(\vec{x})$ at any finite set of input points $\vec{x}$ have a joint Gaussian distribution.

Let $h : \mathbb{R} \to \mathbb{R}^m$ be a vector of m regression functions, $\vec{\beta}$ be a vector of regression coefficients, and $R = \text{diag}(\vec{r})$ be a diagonal matrix of p positive roughness parameters $\vec{r} = (r_1, \ldots, r_p)$. An example of a

1-dimensional output GP with exponential correlation function can be given by:

$$y = f(\cdot)|\vec{\beta}, \sigma^2, \vec{r} \sim N(m(\cdot), c(\cdot,\cdot)\sigma^2), \quad (1)$$
$$m(\vec{x}) = h^T(\vec{x})\vec{\beta}, \quad (2)$$
$$c(\vec{x}, \vec{x}') = \exp\{-(\vec{x}-\vec{x}')^T R(\vec{x}-\vec{x}')\}, \quad (3)$$

The core of the GP model is the correlation function c which defines how the output values at different inputs $y(\vec{x})$ and $y(\vec{x}')$ are correlated. A correlation function must be positive definite and satisfy $c(\vec{x}, \vec{x}) = 1\ \forall \vec{x}$. The correlation function given in the above example specifies the exponentially decreasing correlation with the normalized distance between $\vec{x}$ and $\vec{x}'$, scaled according to parameters $r_1, \ldots, r_p$ to account for different magnitudes and different importances of the various input variables in $\vec{x}$. Other classes of correlation functions are possible. As one example, the Matern family contains a smoothness parameter that specifies how many times differentiable the resulting process should be. Figure 3 illustrates the kinds of curves that are possible with GPs. Each subplot shows GPs from a different correlation function. Within each subplot, the different trajectories result from different initial states in the space

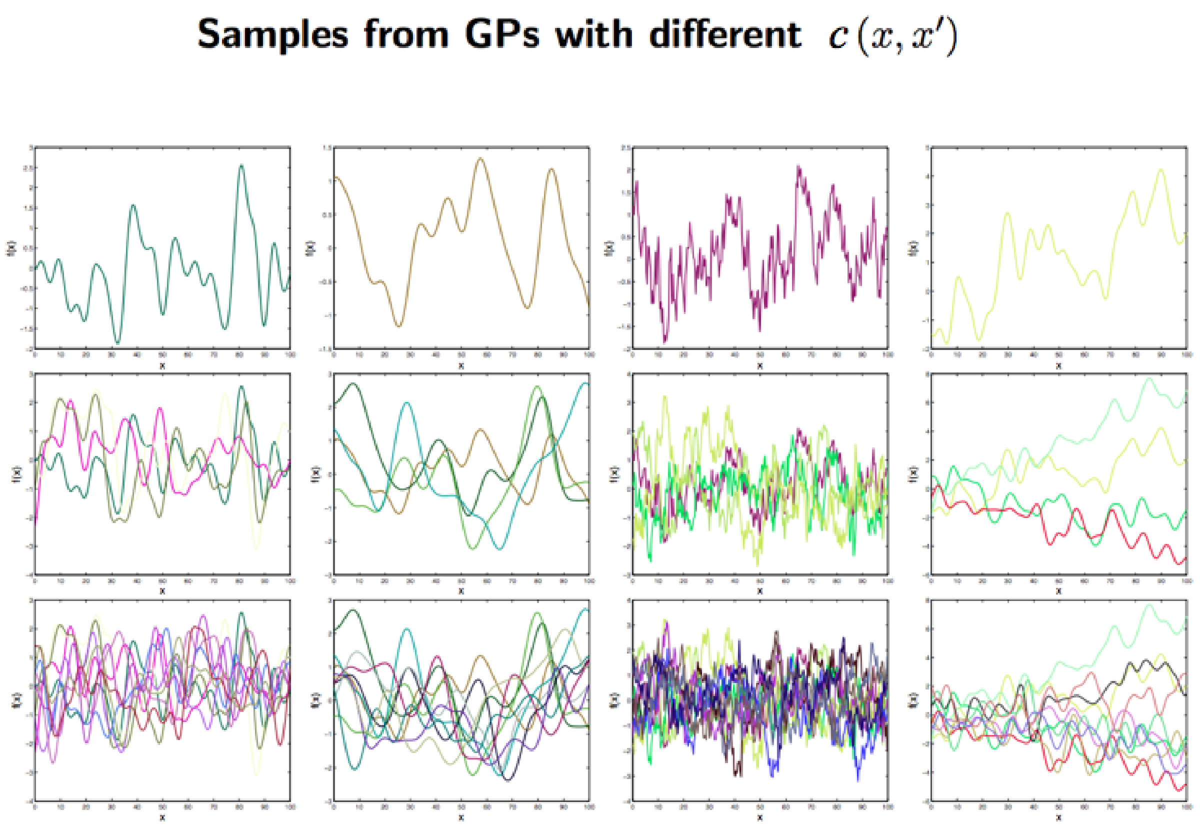

Figure 3. 1D GP sample path example with different correlation functions.

Typical GPs often confine the functions to *stationarity*, which means GP parameters such as the mean and covariance function don't change when shifted in time or space. While stationarity is a reasonable assumption for many data sets, still many exhibit only local stationarity.

The Treed Gaussian Process (TGP) model is more flexible and overcomes this limitation by subdividing the input space and modeling each region r_ν of the input space using a different GP. Fitting different, independent models to the data in separate regions of the input space naturally implements a globally nonstationary model. Moreover, dividing up the space results in smaller local covariance matrices, which are more quickly inverted thus provides better computational efficiency. And, partitions can often offer a natural and data-inspired blocking strategy for latent variable sampling in classification.

The TGP subdivision process is hierarchical and done by recursively partitioning the input space. This recursive partitioning is represented by a binary tree in which nodes correspond to regions of the input space, and children are the resulting regions from making a binary split on the value of a single variable so that region boundaries are parallel to coordinate axes. The parameters defining the subdivision process are part of the TGP statistical model and ultimately determine the number of regions in the subdivision. Figure 4 gives an example of a function which can be fit very well using TGP. The vertical dashed lines show the three different regions used to fit the function; these regions are found automatically. The predicted value of the function that predicts the data points is given by the solid line in this figure, and the outer dashed curves give the confidence interval. For a detailed discussion of TGP, see .[20]

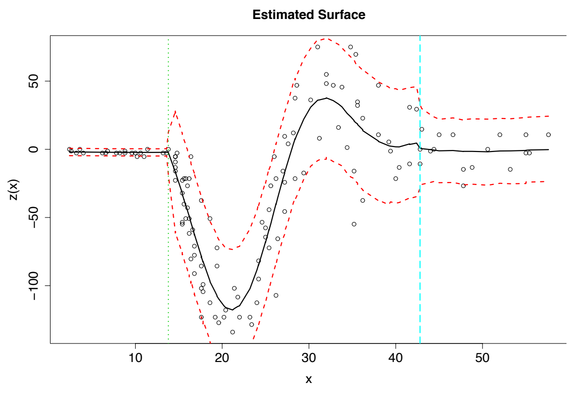

Figure 4. TGP example. The input space was divided into three regions with a distinct GP inferred in each region.

For the work on validation of the IFCS adaptive control system, we use an even higher fidelity Gaussian Process model, in particular a *Classification Treed Gaussian Process* model. Before fitting a mathematical relationship to an input value, the input is first sorted into one of M possible classes. This is done by introducing M latent variables $Z_m(\vec{x})$, $m = 1, \ldots, M$, to define class probabilities via a softmax function

$$p(C(\vec{x}) = m) = \frac{\exp(-Z_m(\vec{x}))}{\sum_{i=1}^{M} \exp(-Z_i(\vec{x}))} \tag{4}$$

In CTGP, each class function $Z_m(\vec{x})$ is modeled using a TGP and each class may use a different tree:

$$Z_m(\vec{x}) \sim \text{TGP}_m, \quad m = 1, \ldots, M. \tag{5}$$

Here we have used a shorthand notation $Z_m(\vec{x}) \sim \text{TGP}_m$ to indicate that the scalar-valued function $Z_m(\vec{x})$ is modeled using a Treed Gaussian Process (TGP). See[21,22] for CTGP algorithms.

III. Application: An IFCS Adaptive Flight Control Simulator

For our initial foray into the validation of complex aerospace systems, we used an early prototype simulator for the NASA Generation 2 (Gen-2) Intelligent Flight Control System (IFCS). This control system uses neural networks to stabilize an aircraft under a failure condition; the goal of the system is to have consistent controller performance in the event of any unforeseen event. The neural networks learn how remaining, functional control surfaces can be used to drive the error between what the pilot commands and the behavior of the plane to zero. For a detailed description of the IFCS, see .[9,23–26]

Our simulator is an early prototype implemented in Mathworks' Simulink,[27] and has MATLAB wrappers to control the values of simulation. This early prototype version is not always capable of stabilizing the aircraft after a failure. It is possible for a trial to become unstable; this results in a set of time series that is shorter than the maximum possible length of the simulation (20 seconds). One time step within the simulation is 0.01 seconds.

Each step within the simulation is *deterministic*—a set of the same inputs will always lead to the same time series outputs. However, the overall behavior of the adaptive control system is hard to predict.[3,5,7,14,18,28]

This is true for many reasons, some of the limitations we have already overcome include the fact that small changes in the input space can lead to very different behavior in the time series. We have managed this limitation by using Classification Treed Gaussian Process (Section §II.B), as highlighted in Section §IV. We have extended Gaussian Process techniques in order to mathematically predict time series of different lengths.

For this set of experiments, we chose to keep the initial altitude, the pilot input, and the initial velocity constant for every trial. We model the pilot input as a doublet done simulataneously in all three axes. A doublet is a series of step inputs. The aircraft starts in a trim state with no command for each of the three axes. At one second after the beginning of the simulation, there is a maximum positive command in each axis, that is held until two seconds. At two seconds after the beginning of the simulation, there is a step change to the maximum negative command, held until three seconds. At three seconds after the beginning of the simulation, the commanded pilot input for all three axes returns to zero.

Table 1 gives the input parameters that we varied for our experiments. Each of the input parameters is a scalar value. We will extend our toolchain to explicitly handle time-series inputs (like pilot inputs) in future work. The table also gives the outputs that we predict in our experiments. The parameters F and T are scalar values; all of the other output parameters are time series of varying lengths.

Name	Description
Input parameters	
K_{lat}	lateral stick gain
K_{lon}	longitudinal stick gain
K_{pedal}	rudder gain
ζ	damping coefficient
$w_{p,q,r}$	proportional gain (all axes)
K_3	controller gain (yaw axis)
$\lambda_{p,q,r}$	NN learning rates (all axes)
Outputs	
$e\vec{r}r^{1,2}_{p,q,r}$	error wrt. reference model (proportional, integral), all axes [degrees, degrees*sec]
$\vec{h}$	altitude [feet]
$\vec{p}, \vec{q}, \vec{r}$	roll, pitch, and yaw rates [degrees/sec]
$\vec{\alpha}$	angle of attack [degrees]
$\vec{\beta}$	sideslip angle [degrees]
F	a Boolean variable: TRUE for cases which fail and FALSE otherwise
T	time to failure [number of time steps]

Table 1. Inputs and outputs. Note that the pilot inputs are kept constant in this experiment and are not considered to be a input.

IV. Results

Our goal in this paper is to present a toolchain which might be used for the validation of complex, non-linear, aerospace systems. In order to progress the toolchain to this end, we have had to solve several significant technical challenges. These challenges are discussed in detail in several pre-publication papers. [29, 30] Here we give an overview of the work, highlighting only the most significant results, and then discussing gaps which still remain to be solved in Section §V.

Our approach for emulating the curves for a single output variable is to represent the curves in an orthogonal basis which capture curve characteristics, and then to predict the output curve for a new input by predicting the coefficients for the desired output curve's basis representation. We allow for the possibility of output curves whose length varies with input, which may occur when a simulator fails to run to completion for some configurations of inputs, and the time to failure depends on the input configuration. The use of a reduced basis representation allows for a fixed length representation of output curves and computational tractability while retaining a high degree of fidelity.

When output curves can be grouped into a small set of clusters of similar curve length, shape, and frequency content, we find that fitting distinct models for different classes of output curves improves the prediction. Therefore, we add a class parameter to our statistical model. Our complete model is built on top of statistical models for individual output functions from input to class, input to output curve length, and input to each of the coefficients in a reduced basis curve representation. The class function has a categorical output, while the length and coefficient functions have real-valued outputs. We propose modeling using non-stationary Treed Gaussian Process (TGP) models for both modeling the class membership as well as the mapping from inputs to outputs within a class

Some of the challenges in predicting the flight outputs include very different output lengths with a wide variety of frequencies. The basic shapes of output curves can vary quite a lot. We observe that the output curves can be grouped into clusters of similar curve length, shape, and frequency content. General appearance and frequency content of success and failure curves are typically quite different. When the time to failure is about 250 time steps, the curves for different runs look similar to each other. When time to failure is between 350 to 500 time steps, the curves look similar to each other as well. First we will present results from classification strategies of the output curves, then we will show the results of time to failure prediction, then finally results of actual output curve prediction.

We have examined classification strategies using not just two classes (success and failure) but also four classes, with classes defined by the ranges of the output length. Our experimental data set consists of a total of 967 runs obtained from the simulation. Table 2 summarizes the results for two different classification strategies: dividing the data into two classes (*Two Class Problem*) and into four classes (*Four Class Problem*). The *Two Class Problem* strategy separates the data into *failure* and *success* categories, where runs in *success* complete at 1901 time steps (the length of the simulation) and runs in the *failure* category end earlier. The *Four Class Problem* strategy separates the data into four different classes: three different failure classes (*failure1* , *failure2* , and *failure3* respectively), and one class for all of the successes. Each of the three failure classes has a characteristic time series output shape, and we correctly hypothesized that using a different mathematical model for each of these shapes would improve performance.

Two Class Problem			Four Class Problem		
failure	$0 < T \leq 1900$	569 runs	*failure1*	$0 < T \leq 180$	257 runs
			failure2	$180 < T \leq 280$	241 runs
			failure3	$280 < T \leq 1900$	71 runs
success	$T = 1901$	398 runs	*success*	$T = 1901$	398 runs
total		967 runs	*total*		967 runs

Table 2. A summary of the data classification strategies we used in our experiments.

In addition to choosing the number of classes, we also performed experiments to determine the best classifer for the data. We split the 967 runs into 637 training examples and 300 test examples to evaluate classifier performance. We implemented five different classification methods; as a whole, the TGP-based classification methods gave the lowest classification error rates.

For curve prediction, we randomly split the 967 runs into 867 training examples and 100 test examples. The CTGP 2-class and 4-class classification results for the prediction training and test sets are shown in Figure 5(a) and (b), respectively.

	failure	*success*	total
training	507	360	867
test	62	38	100
# errors	3	8	11
error rate	4.5%	21.1%	11.8%

(a)

	failure 1	*failure* 2	*failure* 3	*success*	total
training	228	213	66	360	867
test	29	28	5	38	100
# errors	5	9	5	8	27
error rate	17.2%	32.1%	100%	21.1%	27%

(b)

Figure 5. CTGP classification performance on the curve prediction training and test sets. (a) 2-class. (b) 4-class. See the text for details.

In addition to choosing the number of classes, we also performed experiments to determine the best classifer for the data. We split the 967 runs into 637 training examples and 300 test examples to evaluate classifier performance. We implemented five different classification methods, as a whole, the TGP-based classification methods gave the lowest classification error rates of 11.8%.

For the 2-class case, the 867 training examples have 507 examples in the *failure* class and 360 in the *success* class. The CTGP overall 2-class error rate is very low at 11%, making 11 incorrect classifications out of 100 tests.

In Table 3 we have a further breakdown for the types of errors made by the CTGP classification algorithm. For the 2-class problem, the error rate was just 4.5% for classifying *failure* cases, incorrectly classifying only 3 of the 62 true *failure* tests. CTGP incorrectly classified 8 of the 38 test *success* runs as *failure* , for a *success* error rate of 21.1% — this type of error would correspond to a 'false alarm' if the algorithm was used in real time to try to predict failures. We have plans for how to bring down the overall false alarm rate.

The number of examples in the training and test sets for the 4-class case, as well as the classification error rates per class and overall are reported in Figure 5(b). The overall CTGP classification error rate is 21%, making 21 incorrect classifications out of the 100 tests. Although the overall error rate is higher for the 4-class problem than the 2-class problem, the error rate for the *success* class is lower. In the 4-class case, CTGP misclassified only 2 of the 38 *success* tests, for a *success* error rate of just 5.3%. The *success* class had more training examples than the other 3 classes. The *failure* 1 and *failure* 2 error rates were 17.3% and 32.1%, respectively. All 5 test cases in class *failure* 3 were classified incorrectly. Because the number of training data for this class is substantially lower than the other classes, this result is not surprising and should be disregarded. It is expected that the error rate will improve with more training samples.

	Two Class Problem	Four Class Problem
false alarm rate	21.1%	21.1%
missed failure rate	4.5%	30.6%

Table 3. False Positives and False Negatives Percentage

We computed the mean absolute prediction error from both TGP and Support Vector Machine (SVM) models. SVM is a commonly used technique for the prediction of nonlinear systems. We measured the average of the absolute difference $\bar{E}$ of the predicted time to failure T_{est} and the corresponding ground truth T over the 100 test runs as $\bar{E} = \frac{1}{N}\sum_{i=1}^{N} |T - T_{est}|$.

The first row of Table 4 gives the mean absolute prediction error from TGP and SVM without classification. Even TGP's prediction error of 368.7 is large given the range of time to failure. All of the other methods that we tried gave even worse results. For example, the average absolute prediction error for the next-best model after TGP and SVM is 706.47. Since none of the methods give a good result when we fit a single model to the data globally, we clearly need another approach.

	TGP	SVM
all	368.7	481.8
failure only	33.9	42.5

Table 4. Mean absolute prediction error for time to failure using TGP and SVM

Our new method utilizes the classification strategy we described in the above sections. And we have concluded that the two class strategy is our preferred strategy, using CTGP we can accurately predict whether an input will result in a failure or success, which also provides useful information for improving the flight control system. We would like to go further to predict the actual time to failure. If CTGP predicts success for a given input $\vec{x}$, then we predict T as 1901 since all successful runs have 1901 time steps. If CTGP predicts failure for a given input $\vec{x}$, then we use another method to predict T. Our strategy in this case is to learn the mapping from $\vec{x}$ to $T(\vec{x})$ for failure runs only. In other words, we fit a model using only training runs that result in simulator failure. Among the 867 training inputs that we selected, there are 505 on which the simulator failed. We fit TGP and SVM models on these 505 failure training runs. Of the 100 test inputs, 62 failed. We tested the prediction of TGP and SVM trained on the failure runs to predict the time to failure for these 62 tests. In the second row of Table 4, we present the mean absolute prediction error for both TGP and SVM over the 62 failure test cases. For both models, the prediction error improved an order of magnitude using our classification strategy over training and predicting without classification (first row). Once again TGP outperformed SVM. Furthermore, now we have a good prediction method using classification strategy and TGP model. The average absolute prediction error of TGP on the failure test runs is small compared to the range of time to failure.

Figure 6 gives a summary of the 100 test results. The results for TGP are shown on the left, and the results for SVM are shown on the right. The 62 tests which failed are on the left of each plot; the independent axis is the test number and test numbers 1-62 are failure cases. Similarly, the 38 cases which succeed are on the right of each plot. Each data point in the plot is a prediction of the time to failure. Means for the failure and success times are given as dashed lines. TGP results in predictions that are much closer to the true mean failure times. What's more: TGP has an overall lower misclassification rate, and only misclassifies one true failure.

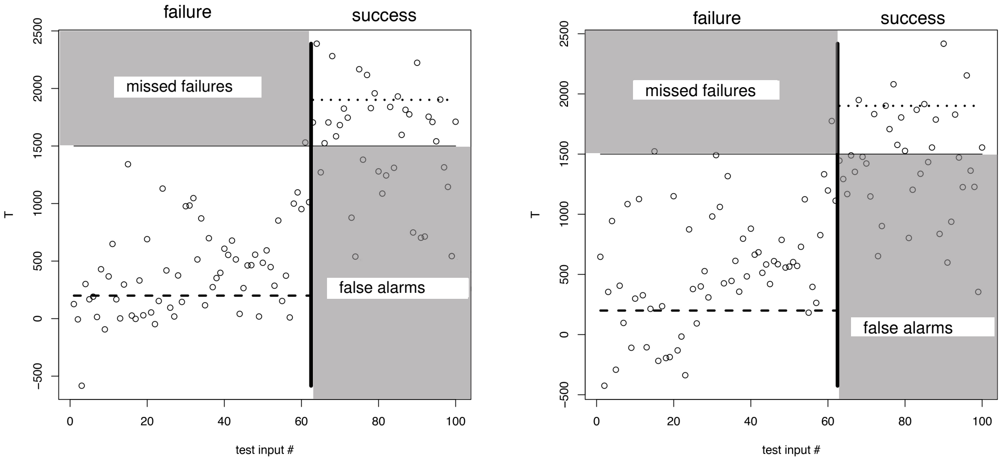

Figure 6. Prediction of time to failure with TGP (left) and SVM (right).

Now we will present results from the actual output curve prediction. The true and predicted output curves of a selected output variable for the 3 failure classes and the success class are shown in Figure 7. Note that the 4 different classes have differing amounts of training data. Of the 867 training runs, 228 are *failure* 1 cases, 213 are *failure* 2 cases, 64 are *failure* 3 cases, and 360 are *success* cases. The very small amount of training data for *failure* 3 makes this class the most difficult to predict.

V. Conclusion

In this paper, we highlight the first steps in a framework for the statistical validation of complex, nonlinear aerospace systems. We have made an argument that traditional methods for validation really only work well when the quantities of interest are dependent on only a few variables, and that the relationships between the variables and the quantities of interest should be more-or-less linear. We have argued that the National Airspace has already reached a point in which the assumptions traditional validation is based on no longer hold.

In this paper, we explore the possibility that we can use *statistical emulation* (Section §II.A) to overcome the limitations of traditional validation techniques. Statistical emulation has a key benefit in that it allows for *uncertainty quantification*—a necessity for the validation of safety-critical systems. Statistical emulation, particularly statical emulation based on Classification Treed Gaussian Processes, has been used to reliably predict the behavior of complex, non-linear, high-dimensional systems that are locally smooth. We have demonstrated that statistical emulation can be used to reliably predict time-to-failure for an adaptive flight control simulator. We have also shown that, given an input, we can predict the time series curve outputs of the simulator with high accuracy. As a byproduct of this prediction, the behavior of the simulator becomes quantified, and can be compared against experimental data for validation. We have extended statistical emulation to explicitly handle time series outputs of varying lengths. With some work, we will be able to

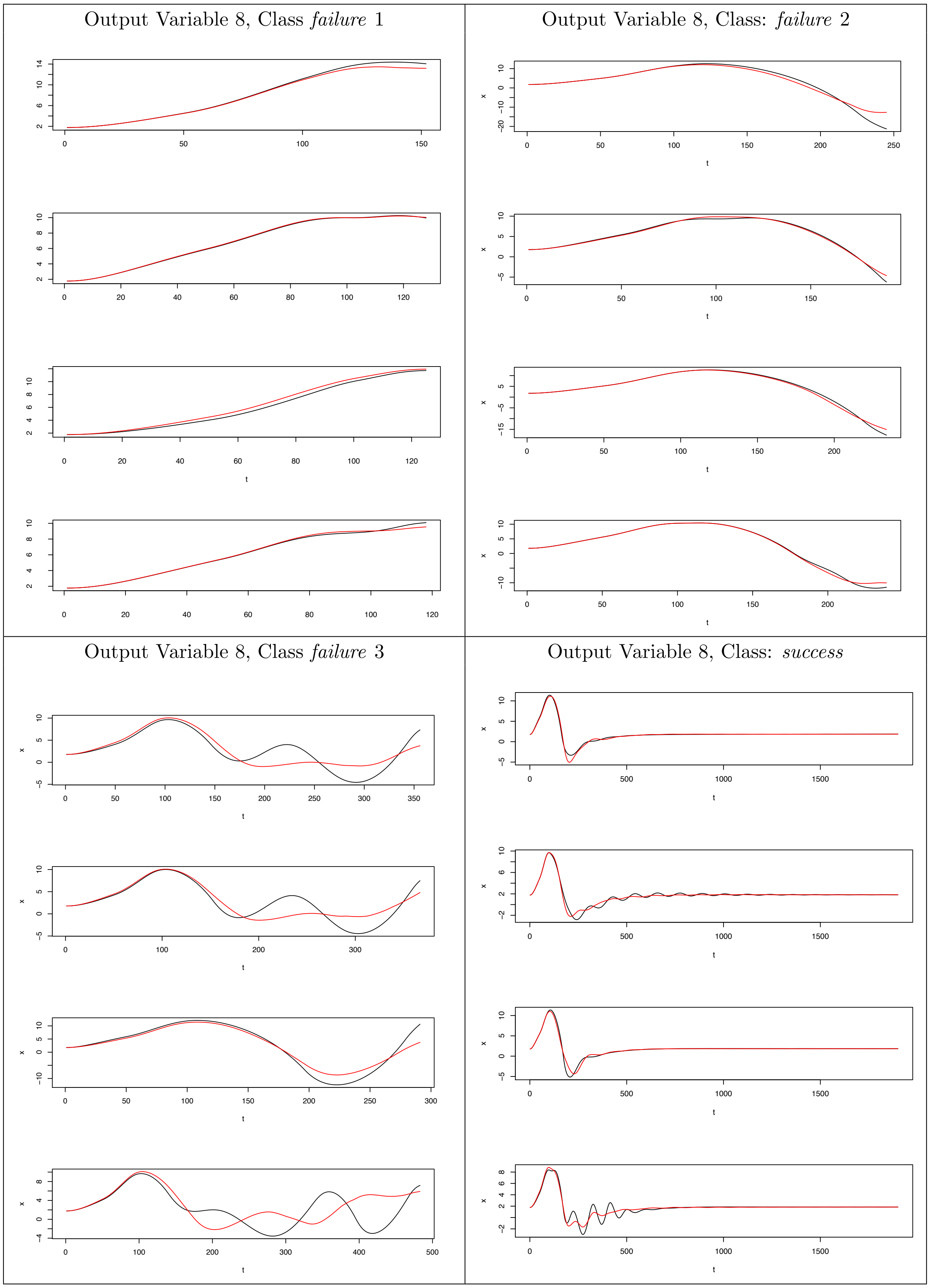

Figure 7. $D = 25$ **fourier-based predictions: Output variable 8: (upper left) class** ***failure*** **1, (upper right) class** ***failure*** **2, (lower left) class** ***failure*** **3, (lower right) class** ***success*** **.**

explicitly handle time series inputs, as well.

However, statistical emulators still need to be advanced for use in flight-critical system validation. We have shown that we can build predictive tools that can represent complicated nonlinear models, and can do so with uncertainty quantification. Ideally, we would like to be able to quickly solve the inverse problem. That is, given a desired or undesired output behavior of our system, we would like to be able to bound the region or regions in the input space that lead to the output behavior. We also are working to extend our work to models of different fidelity, as shown in Figure 1. In general, models of lower fidelity are less expensive to run. We would like to be able to compare the behaviors of a low-fidelity and a high-fidelity model, and choose test points in the high-fidelity model that most quickly decrease the uncertainty in the behaviors between the two models. We believe our initial framework gives us a promising path towards these goals.

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]Roache, P., *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, Albuquerque, NM, 1998.

[2]Roache, P. J., "Perspective: Validation—What Does It Mean?" *Journal of Fluids Engineering*, Vol. 131, March 2009, pp. 034503 (4 pages).

[3]Menon, P., Bates, D., and Postelthwaite, L., "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.

[4]Mozdzanowska, A. and Hansman, J. R., "System Transition: Dynamics of Change in the US Air Transportation System," Tech. Rep. ICAT-2008-3, International Center for Air Transportation, Massachussetts Institute of Technology, 2008.

[5]Jacklin, S., 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," *AIAA Infotech*, 2005.

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

[7]Jacklin, S., Lowry, M., Schumann, J., Gupta, 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," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004.

[8]Butler, R. and Finelli, G., "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering*, Vol. 19, No. 1, 1993, pp. 3–12.

[9]Burken, J. J., Williams-Hayes, P., Kaneshige, J. T., and Stachowiak, S. J., "Reconfigurable Control with Neural Network Augmentation for a Modified F-15 Aircraft," Tech. Rep. NASA/TM-2006-213678, NASA, 2006.

[10]Schumann, J. and Liu, Y., editors, *Applications of Neural Networks in High Assurance Systems*, Studies in Computational Intelligence, Springer, Berlin, 2010, pp. 1–19.

[11]Tiwari, A., "Formally Analyzing Adaptive Flight Control," *Numerical Software Verification*, 2009.

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

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

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

[15]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.

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

[17]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.

[18]Menon, P., Bates, D., and Postelthwaite, L., "Nonlinear Robustness Analysis of Flight Control Laws for Highly Augmented Aircraft," *Control Engineering Practice*, Vol. 15, 2007, pp. 665–662.

[19]Callaham, M. and Wieland, F., "Are air traffic models chaotic?" *Digital Avionics Systems, 2001. DASC. 20th Conference*, Vol. 2, oct 2001, pp. 7F4/1 –7F4/8 vol.2.

[20]Gramacy, R. B., *Bayesian Treed Gaussian Process Models*, Ph.D. thesis, University of California at Santa Cruz, Dec. 2005, `http://faculty.chicagobooth.edu/robert.gramacy/papers/gra2005-02.pdf`.

[21]Broderick, T. and Gramacy, R. B., "Treed Gaussian Process Models for Classification," *Proc. 11th Conference of the International Federation of Classification Societies (IFCS-09)*, March 2009, pp. 101–108, `http://www.statslab.cam.ac.uk/~bobby/gfkl-274.pdf`.

[22]Broderick, T. and Gramacy, R. B., "Classification and categorical inputs with treed Gaussian process models," *ArXiv e-prints*, April 2009, `http://arxiv.org/PS_cache/arxiv/pdf/0904/0904.4891v2.pdf`.

[23]Jorgensen, C. C., "Direct Adaptive Aircraft Control Using Dynamic Cell Structure Neural Networks," Tech. Rep. 112198, NASA, 1997.

[24]Kim, B. and Calise, A., "Nonlinear Flight Control Using Neural Networks," *Journal of Guidance, Control, and Dynamics*, Vol. 20, No. 1, 1997, pp. 26–33.

[25]Rysdyk, R. and Calise, A., "Fault Tolerant Flight Control via Adaptive Neural Network Augmentation," *AIAA Guidance, Navigation, and Control Conference*, 1998.

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

[27]Mathworks, "Simulink," `http://www.mathworks.com/products/simulink/`, January 2012, Last accessed 26 March 2012.

[28]Broderick, R., "Statistical and adaptive approach for verification of a neural-based flight control system," *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, Vol. 2, oct. 2004, pp. 6.E.1 – 61–10 Vol.2.

[29]He, Y., Lee, H., and Davies, M., "Predicting Time to Failure in an Adaptive Flight Control Simulation," preprint (2102), please email the authors for a pre-publication copy.

[30]He, Y., Lee, H., and Davies, M., "Predicting Variable-Length Functional Outputs for Emulation of a Flight Simulator," preprint (2102), please email the authors for a pre-publication copy.