Predicting Time Series Outputs and Time-to-Failure for an Aircraft Controller Using Bayesian Modeling

Yuning He*

January 27, 2015

Abstract

Safety of unmanned aerial systems (UAS) is paramount, but the large number of dynamically changing controller parameters makes it hard to determine if the system is currently stable, and the time before loss of control if not. We propose a hierarchical statistical model using Treed Gaussian Processes to predict (i) whether a flight will be stable (success) or become unstable (failure), (ii) the time-to-failure if unstable, and (iii) time series outputs for flight variables. We first classify the current flight input into success or failure types, and then use separate models for each class to predict the time-to-failure and time series outputs. As different inputs may cause failures at different times, we have to model variable length output curves. We use a basis representation for curves and learn the mappings from input to basis coefficients. We demonstrate the effectiveness of our prediction methods on a NASA neuro-adaptive flight control system.

1 Introduction

Most unmanned aircraft (UAS) are equipped with elaborate control systems. In particular, autonomous operations require that the flight control system performs reliably and in the presence of failures. For certification, controller stability and robustness must be demonstrated. UAS controllers can range from simple PID control to advanced adaptive controllers. Their behavior is determined by numerous parameters, many of which can be set during design time. More challenging, however, are adaptive control systems that change their internal parameters (e.g., gains, model parameters, modes) dynamically during the flight in response to a changing mission profile, unexpected environment, or damage.

There exist numerous architectures for adaptive flight control systems. Regardless of its architecture, however, safety requires that certain robustness and stability properties can be guaranteed for the currently active set of parameters. In most cases, such guarantees cannot be provided a priori, so on-board dynamic monitoring techniques are needed to assess if—given the current parameter setting—the aircraft (AC) will remain stable and, if not, when the AC will become unstable. Such information is vital for the on-board autonomous decision making system, especially for UAS.

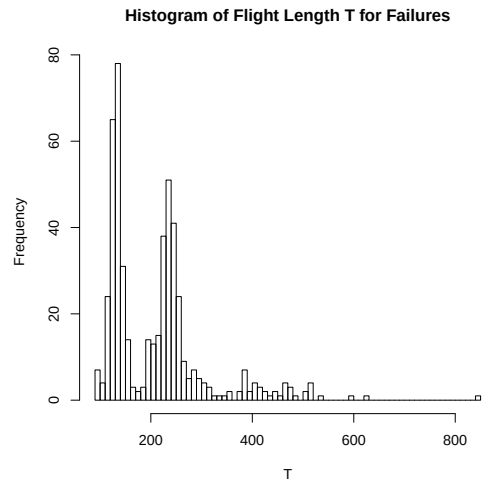

Figure 1: Histogram of T_{fail}. Most failures occur before $T = 600$ time steps. Successfully controlled flights are more common and last for $T = 1901$ time steps.

In this paper, we describe a statistical method which enables us to answer two questions during the flight: (i) given the current values of control system parameters, will the system remain stable within the next T time steps?[1] And, (ii) if not, at which time $T_{fail} < T$ will the system be unstable? We also describe a statistical method for efficient prediction of the actual time-series outputs up to T time steps. This more detailed information can be used for optimal and safe autonomous decision making. For example, a damaged rudder might require a specific parameter setting. If it can be predicted that there needs to be extreme rudder deflections within the near future, then the current situation can be regarded as unsafe and other mitigation techniques must be considered.

Our statistical framework to address these questions is hierarchical. Most simulation runs are successfully controlled and last for $T = 1901$ time steps. Most failure runs have a time-to-failure $T_{fail} < 600$ time steps. When observing the distribution of simulation runs with a specific T_{fail} over numerous parameter settings, the histogram of T_{fail} in Figure 1 suggests the existence of a few easy discernable failure groups. Although, the general appearance of the time series are typically quite different for runs with different

*yuning.he@nasa.gov, UARC, NASA Ames Research Center

[1] For our case study in this paper, one time step is 1/100s.

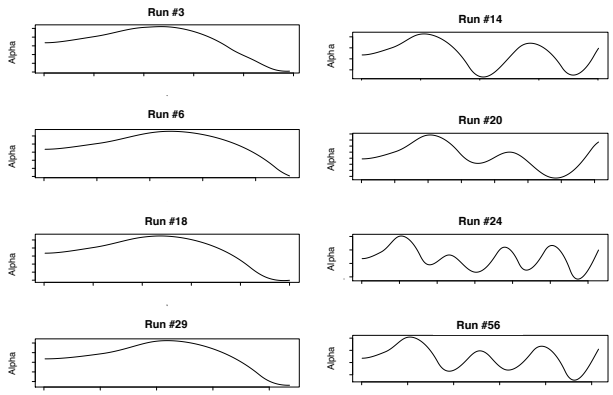

Figure 2: Typical curves for different inputs that result in (left) $T_{fail} \approx 250$ and (right) $T_{fail} \approx 350 \ldots 500$. The horizontal axis is time and the vertical axis is one of the simulator outputs.

T_{fail}, Figure 2 shows that the shape of time-series for runs with $T_{fail} \approx 250$ look relatively similar to each other as do curves with $T_{fail} \approx 350 \ldots 500$ time steps. Although the data in the figure have been obtained with from an experimental setup with a specific adaptive control system (see Section 6), similar behavior can be encountered often.

We therefore developed a hierarchical model, where the entire data first undergo a classification with respect to prominent differences in failure time. Then time-to-failure and time series prediction are performed separately for each of the classes.

We illustrate our approach with a case study, where we have used our statistical framework on a high-fidelity simulation of a neuro-adaptive flight control system – the NASA IFCS (Intelligent Flight Control System). It will be described in Section 6. We have examined classification strategies with one, two, and four classes and provide comparative results as well as an analysis of the quality of time-to-failure estimation in terms of missed and false alarms.

2 Methodology & Architecture

We have implemented a toolchain for time-to-failure and time series prediction using a two-stage hierarchical statistical model (Figure 3A). Our architecture consists of a Categorical Treed Gaussian Process (CTGP) statistical model for the mapping from input to success or failure class, and within each given failure class, a distinct Treed Gaussian Process (TGP) model for the relationship of input to time-to-failure (which is the output curve length). The mapping from input to class is learned through a CTGP supervised classification method (Figure 3A).

In addition to predicting time-to-failure, we also

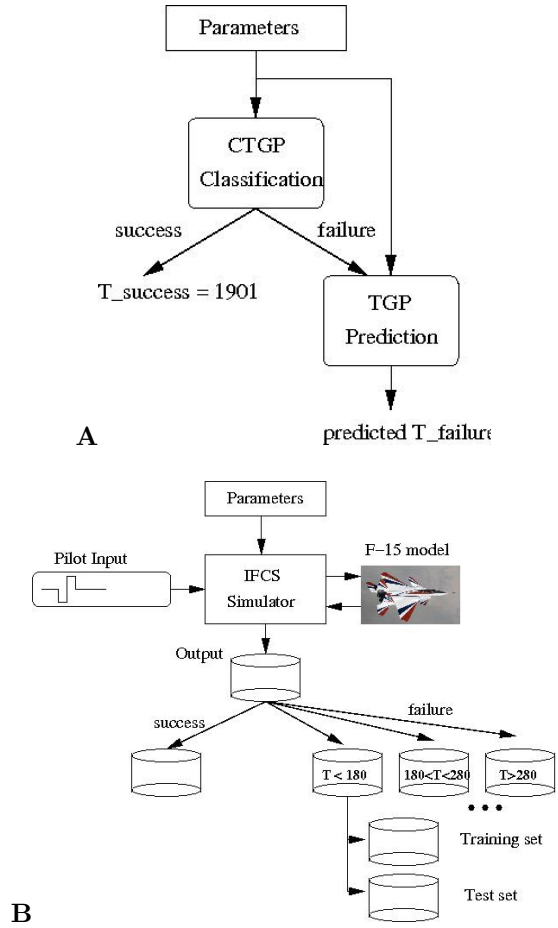

Figure 3: Two-stage prediction of time to failure (A) and supervised learning (B).

predict the actual time series outputs for flight variables in both success and failure cases. As different inputs may cause failures at different times, we have to model variable length output curves. We do so by modeling the mappings from input to a fixed number of coefficients in a curve basis representation, using one TGP model for each real-valued coefficient. Using a curve basis allows us to have a fixed size representation of variable length output curves. We hypothesize that, by fitting distinct models for each of the distinct classes, the overall results of prediction accuracy for time-to-failure and curve basis coefficients will be improved. Our hierarchical model can capture model parameter variation across the success and failure classes.

Next we give some background on TGP models and discuss other Gaussian Process (GP)-related options for modeling output curves to provide context for our statistical modeling decisions.

3 Background

3.1 Treed Gaussian Processes The *Treed Gaussian Process* (TGP) model [9] is more flexible and overcomes the limitations inherent in GP models 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 non-stationary model. Moreover, dividing up the space results in smaller local covariance matrices that are more quickly inverted and thus provide better computational efficiency. Also, partitions can often offer a natural and data-inspired blocking strategy for latent variable sampling in classification. The partitioning approach in the standard TGP model is based on the Bayesian Classification and Regression Trees model developed by Chipman et al. [5, 6].

The TGP subdivision process is 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 TGP parameters Θ include parameters for defining the subdivision process which ultimately determine the number of regions in the subdivision.

3.2 Classification Treed Gaussian Processes It is also useful to be able to predict functions $C : \mathbb{R}^p \rightarrow \{1, \ldots, M\}$ with a categorical output of M possible classes. Recently, an extension of TGP called CTGP (Classification TGP) [2, 1] was developed to use Treed Gaussian Processes for classification. The CTGP model first extends the GP model from regression to classification by introducing M latent variables Z_m, $m = 1, \ldots, M$, to define class probabilities via a softmax function $p(C(\mathbf{x}) = m) \propto \exp(-Z_m(\mathbf{x}))$. In CTGP, each class function $Z_m(\mathbf{x}) \sim \mathrm{TGP}(\Theta_m)$ is modeled using a TGP and each class may use a different tree.

3.3 Curve Prediction with GP One approach to predicting output curves instead of scalar-valued outputs is to extend the GP model to functions $\mathbf{y} : \mathbb{R}^p \rightarrow \mathbb{R}^q$, where the vector output $\mathbf{y}$ represents samples of a curve at $q = T$ time points. In the context of statistical emulation of a simulator computer program, Conti and O'Hagan [7] call this the *Multi-output (MO) emulator* and provide the statistical model for q-dimensional Gaussian Processes:

$$\begin{aligned} y &= f(\cdot)|B, \sigma^2, \mathbf{r} \sim N_q(\mathbf{m}(\cdot), c(\cdot,\cdot)\Sigma), \\ \mathbf{m}(\mathbf{x}) &= B^T h(\mathbf{x}), \\ c(\mathbf{x}, \mathbf{x}') &= \exp\{-(\mathbf{x} - \mathbf{x}')^T R(\mathbf{x} - \mathbf{x}')\}, \end{aligned}$$

where $h : \mathbb{R} \rightarrow \mathbb{R}^m$ is a vector of m regression functions shared by each of the scalar-valued component functions $f_1, \ldots, f_q$, $B \in \mathbb{R}^{m \times q}$ is a matrix of regression coefficients, and $R = \mathrm{diag}(\mathbf{r})$ is a diagonal matrix of p positive roughness parameters $\mathbf{r} = (r_1, \ldots, r_p)$. As in the 1-dimensional GP, a common choice for defining the mean $m(\cdot)$ is the linear specification $h(\mathbf{x}) = (1, \mathbf{x})^T$ for which $m = p + 1$.

In addition to the MO emulator, Conti and O'Hagan [7] outline two other possible approaches for *multi-output* emulation: *Ensemble of single-output (MS) emulators* and the *Time Input (TI) emulator*. In the MS approach, each of the T curve values are predicted independently using T single-output emulators. On the other hand, the TI approach adds the time parameter t to the input $\mathbf{x}$ and builds one, single-output emulator for $y(\mathbf{x}, t) : (\mathbb{R}^p \times \mathbb{R}) \rightarrow \mathbb{R}$. The MO emulator is the simplest from the computational perspective with a computational load that is comparable to a single-output GP emulator, in which the bottleneck is $n \times n$ matrix inversion for n training inputs $\mathcal{S}$. The MS method uses T single-output GP emulators and thus has a computational burden T times more than that of the MO method. A naive implementation of the TI emulator would require nT times the computation of the MO emulator as the training samples are now $\mathcal{S} \times \{1, \ldots, M\}$, but the structure of the problem allows the required $nT \times nT$ matrix inversions to be done via $n \times n$ and $T \times T$ matrix inversions. Of course, this is still more computation than required by the MO emulator.

For prediction, the key component is the correlation between outputs over the input domain as there are only a limited number of training samples available. There are more substantial differences in the MO, MS, and TI methods in terms of their covariance structures, at least for a diagonal covariance matrix R as specified above. For the covariance of outputs $f_t(\mathbf{x}_1)$ and $f_t(\mathbf{x}_2)$ at the same time t, the MS method is most flexible as it estimates different roughness parameters $\mathbf{r}$ for each output time while the MO and TI methods estimate a single $\mathbf{r}$ for all times. In terms of the covariance between different outputs $f_{t_1}(\mathbf{x}_1)$ and $f_{t_2}(\mathbf{x}_2)$ at different locations $\mathbf{x}_1$ and $\mathbf{x}_2$, the MO method is most general because this covariance is $c(\mathbf{x}_1, \mathbf{x}_2) \times \Sigma(t_1, t_2)$ and the matrix entry $\Sigma(t_1, t_2)$ is not constrained. The TI approach is less flexible than the MO approach because the covariance between different outputs is the Gaussian process variance times $\exp(-r_T(t_1 - t_2)^2)$,

essentially replacing the generality of $\Sigma(t_1, t_2)$ with an exponentially decreasing squared time difference. The main disadvantage of the MS approach is a lack of correlation structure over time because it emulates the output curve values at different times independently. For this reason, Conti and O'Hagan [7] dismiss the MS approach for emulation of dynamic computer models.

4 Our Statistical Models

For increased generality in modeling, we do not want to assume stationarity and thus we do not use the GP-based MO approach described above for predicting output curves. One possibility is to build a non-stationary MO approach by extending TGP as described in [8, 9] from 1-dimensional outputs to model multi-dimensional outputs by using multi-dimensional GP models in different regions of the input space. Multi-dimensional outputs are not supported in the current TGP R package [9], so a new implementation would need to be done to test this idea. The current TGP method for scalar output is already quite slow for even modest size problems in terms of the dimension of the input space and the number of training examples. Although the computation for multi-dimensional GPs is comparable to the computation for 1-dimensional GPs in terms of the number of training examples n, there is more computation and space needed for the multi-dimensional case. Thus the TGP-based MO idea does not seem to be a practical one, and we opt for another non-stationary modeling approach.

In our approach to predicting one output variable curve, we represent output curves $\mathbf{y} \in \mathbb{R}^T$ in terms of a linearly independent set of D orthogonal curves $B \in \mathbb{R}^{T \times D}$: $\mathbf{y} \doteq B\mathbf{c}$. We use orthogonal curves which measure different curve characteristics so that the basis coefficients in $\mathbf{c} = (c_i) \in \mathbb{R}^D$ are as uncorrelated as possible. Then we model the coefficients c_i, $i = 1, \ldots, D$, independently using D TGP models. Thus by changing the curve representation, we can use an MS approach without being subject to the criticism of not modeling correlations over time. Now the multiple output values being modeled are not values of the curve at distinct time points but rather the coefficients in a "basis" representation of the curve. In addition to the MS advantages of simplicity and flexibility of modeling correlations of the same coefficient output value over different inputs $\mathbf{x}$, the MS implementation can be parallelized by running each single output emulator, both fitting and prediction, in parallel. In many applications, using $D \ll T$ orthogonal curves will suffice to accurately represent output curves and the use of a good basis provides a substantial data reduction. Although we may not need a full orthogonal basis in which $D = T$, we refer to B as a "basis" or a "reduced basis". Another advantage of our basis representation approach is that it allows us to have a fixed size output representation D for applications which have output curves whose length T vary with input $\mathbf{x}$.

In the case of variable length output curves, modeling the output curve $\mathbf{y}(\mathbf{x})$ requires modeling both the length $T(\mathbf{x})$ as well as the $T(\mathbf{x})$ curve values at different times. In our basis approach, we model the output curve $\mathbf{y}$ by modeling its coefficients in a chosen orthogonal basis of D curves. Thus we have $D+1$ scalar output functions to model: $\mathbf{x} \mapsto T$, $\mathbf{x} \mapsto c_1$, ..., $\mathbf{x} \mapsto c_D$. We use TGP for these scalar-valued prediction problems. The statistical model for our MS, basis approach is given by:

$$\begin{aligned} \mathbf{y}|T, \mathbf{c}, B, V_{\mathbf{y}} &\sim N([B\mathbf{c}]_{1:T}, V_{\mathbf{y}}), \\ T(\mathbf{x}) &\sim \mathrm{TGP}(\Theta_T), \\ c_i(\mathbf{x}) &\sim \mathrm{TGP}(\Theta_i), \quad i = 1, \ldots, D \end{aligned}$$

where $[B\mathbf{c}]_{1:T}$ indicates truncation of the predicted output curve to its correct length T.

5 Our Prediction Method

It is now straightforward to give our full algorithm for emulating variable-length output curves from a simulator. Our strategy for emulation is as follows:

1. *Fitting.* Find coefficients $\{ c_i \}_{i=1}^D$ to approximate the training output curves $\mathbf{y}$ in some basis $\{ \mathbf{b}_i \}_{i=1}^D$:

$$\mathbf{y} \doteq \sum_{i=1}^{D} c_i \mathbf{b}_i \tag{5.1}$$

2. *Learning Coefficient Mappings.* Learn the D mappings from inputs $\mathbf{x}$ to coefficients c_i for $i = 1, \ldots, D$.

3. *Coefficient Prediction.* Predict the coefficients $c_{i,\mathrm{pred}}^{\mathrm{new}}$ for a new input $\mathbf{x}^{\mathrm{new}}$ for $i = 1, \ldots, D$.

4. *Output Curve Prediction.* Predict the output curve $\mathbf{y}^{\mathrm{new}}$ corresponding to input $\mathbf{x}^{\mathrm{new}}$ using the predicted coefficients $c_{i,\mathrm{pred}}^{\mathrm{new}}$:

$$\mathbf{y}_{\mathrm{pred}}^{\mathrm{new}} := \sum_{i=1}^{D} c_{i,\mathrm{pred}}^{\mathrm{new}} \mathbf{b}_i \tag{5.2}$$

6 Results

We will demonstrate our approach using a simulation model of the NASA IFCS adaptive control system. This damage-adaptive control system has been developed by NASA and was originally test-flown on a manned

F-15 aircraft [3]. The underlying architecture (Figure 4) is very generic. The output signals of the model-referencing PID controller are subject to control augmentation that is produced by the neuro-adaptive component (a Sigma-Pi or Adaline neural network). A dynamic inverse converts the required rates into commands for the actuators of the aircraft. The neural network itself is trained online to minimize the error between the desired and the actual aircraft behavior. For details see [4, 3] Although this system has orginally been developed for a manned aircraft, it could be easily used to control a UAS. This system is configured using a considerable number of parameters (e.g., gains, NN learning rate), which makes it an interesting testbed for our approach.

In our experiments, we ran the simulator with a multitude of different parameter settings and levels of damage, recording 12 output signals and the elapsed time T_{fail} if the system went unstable. A maximum simulation time of $19s$ was used.

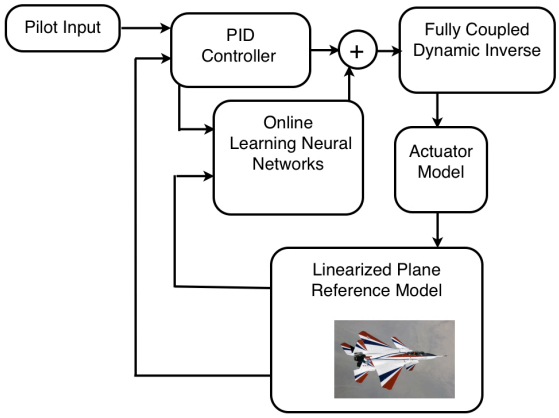

Figure 4: The IFCS adaptive control system

6.1 Classification Results Our experimental data set consists of a total of 967 simulation runs. Table 1 summarizes two different classification strategies: dividing the data into two classes (*Two Class Problem*) and into four classes (*Four Class Problem*). Each strategy separates the data into categories according to the individual time to failure. In our setup, a time of $T = 1901$ (the maximum recorded length in the simulation) indicates as success (S).

We 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 and compared the results for several classifiers (nearest neighbor, several variants of TGP and CTGP, and SVM). As a whole, the CTGP-based classification methods gave, despite relatively long training times, the lowest classification error rates. Table 2 summarizes our CTGP classification results on the 100 test cases.

2-class Prob.		runs	4-class Prob.		runs
F	$0 < T \leq 1900$	569	F_1	$0 < T \leq 180$	257
			F_2	$180 < T \leq 280$	241
			F_3	$280 < T \leq 1900$	71
S	$T = 1901$	398	S	$T = 1901$	398

Table 1: A summary of the data classification strategies we used in our experiments.

2-class	F	S	total
training	507	360	867
test	62	38	100
# errors	3	8	11
error rate	4.8%	21.1%	11.0%

4-class	F_1	F_2	F_3	S	total
training	228	213	66	360	867
test	29	28	5	38	100
# errors	5	9	5	2	21
error rate	17.2%	32.1%	100%	5.3%	21%

Table 2: Overall CTGP performance on the different classification strategies for two and four classes.

The overall CTGP classification error rate for the 4-class case is 27%, making 27 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 S class is lower. In the 4-class case, CTGP misclassified only 2 of the 38 S tests, for a S error rate of just 5.3%. The S class had more training examples than the other three classes. The *failure1* and *failure2* error rates were 17.2% and 32.1%, respectively. All 5 test cases in class *failure3* 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. We expect that the error rate for this failure type would improve with more training samples.

Since our intended application is a safety-critical flight control system, the consequences of misclassification are very different for *successes* and *failures*. In general, safety-critical systems require that there be no *missed failures* (a *failure* misclassified as a *success*), and as few *false alarms* (misclassfied *success*) as possible. In Table 3 we break down our misclassification error rates by these two categories. For the Two Class Problem, we not only have a lower overall error rate (11%), but also a much smaller *missed failure* error rate

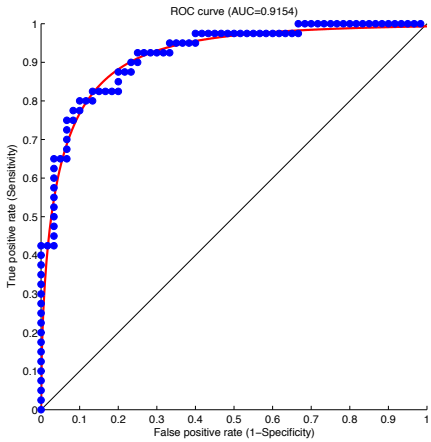

Figure 5: ROC for prediction of time-to-failure.

(4.8%).

	2-class Prob.	4-class Prob.
false alarm rate	21.1%	5.3%
missed failure rate	4.8%	30.6%

Table 3: False Alarm and Missed Failure Percentages

The performance of our time-to-failure estimation with respect to missed and false alarms can be best visualized using a Receiver Operations Characteristics (ROC) curve. Figure 5 shows the curve for our 100 test cases in a coordinate system of true positive rate over false positive rate. The graph of a high-performance system rises sharply, meaning we obtain a good sensitivity (i.e., we can detect most alarms) having to take into account only few false alarms. The diagonal corresponds to a random selection. With a curve close to the upper left corner in Figure 5, our prediction results are good.

In Figure 6, we can see one of the advantages of using our statistical method is that we not only can make predictions, but our Bayesian modeling method also returns a confidence interval/error bar on the prediction. Here the independent axis is the test run index, which we ordered by increasing time to failure. The dependent axis is the time to failure T. The true failure times (dark blue smooth center line—dots connected for better visibility) shows the ground truth values of T where the curve is formed by connecting T values with short line segments. TGP is a Bayesian model, which means it can provide samples from the distribution of $T|\mathbf{x}$. The posterior mean predictions $\hat{T}$ by TGP(circles) are plotted. For perfect predictions, the circles would be along the dark center curve.

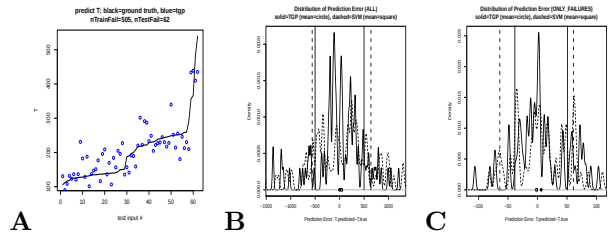

Figure 6: A: Posterior mean T_{fail} prediction using TGP Model for Class F_2. Prediction errors with TGP (solid) and SVM (dashed) for the entire data set (B) and for failure runs only (C).

6.2 Output Curve Prediction In Table 4, we show the prediction errors for each of the PCA, Fourier, and wavelet bases (with dimension $D = 25$) for each of the classification strategies 4 class, 2 class, and 1 class (i.e., just 1 model for all types of output curves). Prediction errors are mean values over the 12 output variables. During basis fitting, here we padded training vectors $\mathbf{y}$ to $T_{\max}$ by repeating the last element in $\mathbf{y}$.

	F_1	F_2	F_3	S	F	S	*all*
μ_{PCA}	0.46	0.45	0.78	0.32	0.63	0.42	1.39
σ^2_{PCA}	0.33	0.29	0.34	0.15	0.30	0.15	1.08
$\mu_{Fourier}$	0.56	0.51	0.82	0.41	0.82	0.41	1.60
$\sigma^2_{Fourier}$	0.36	0.31	0.34	0.15	0.32	0.15	1.00
$\mu_{wavelet}$	0.56	0.52	0.85	0.34	0.84	0.34	1.44
$\sigma^2_{wavelet}$	0.40	0.35	0.34	0.15	0.36	0.15	1.09

Table 4: Output curve prediction errors for different kinds of bases and 4-class, 2-class, and 1-class perfect classification.

The errors reported in the table entries in Table 4 were computed as follows. Let $\mathbf{x}_i$ denote the test inputs with true output curves $\mathbf{y}_i \in \mathbb{R}^{T(\mathbf{x}_i)}$ and predicted output curves $\mathbf{y}_i^{\text{pred}} \in \mathbb{R}^{T(\mathbf{x}_i)}$. We use the standard deviation σ_i for true output curve $\mathbf{y}_i$ to standarize errors across different output curves and different output variables. The error $e_{v,c}$ for output variable v and class c with the set of test output curves $S_{v,c}$ is given by

$$e_{v,c} = \frac{\sum_{i \in S_{v,c}} \frac{||\mathbf{y}_i^{\text{pred}} - \mathbf{y}_i||_1}{\sigma_i}}{\sum_{i \in S_{v,c}} T(\mathbf{x}_i)}. \tag{6.3}$$

The true and predicted output curves $\mathbf{y}_i$ and $\mathbf{y}_i^{\text{pred}}$ are for the given output variable v, but we leave out the dependence on v to simplify the error formula (6.3).The denominator of (6.3) is the total number of output curve values predicted, thus making the reported error an average over all predicted points.

The PCA basis gives the best performance for each of the classification strategies. This can easily be seen

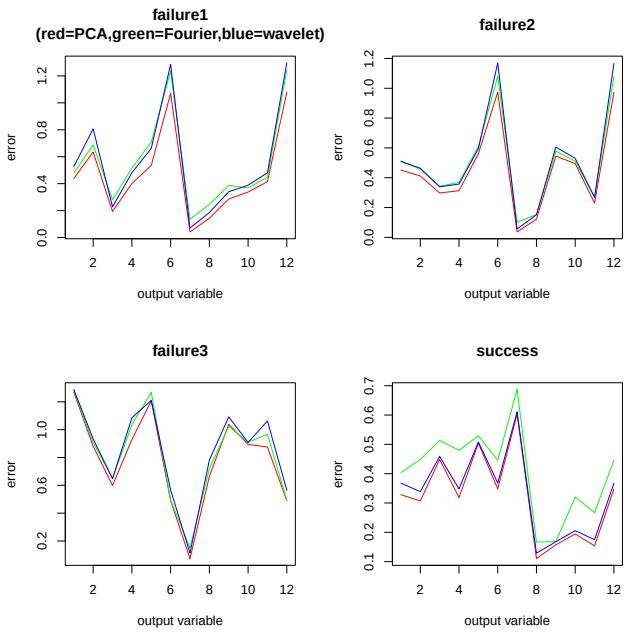

Figure 7: Output curve predicition errors for different bases and output variables.

in the 4 class strategy in the plots in Figure 7. Some representative predictions using the dim $D = 25$ PCA basis for output variable 1 can be seen for the 3 failure classes and the success class in Figure 8. Note that the 4 different classes have differing amounts of training data. Of the 867 training runs, 228 are F_1 cases, 213 are F_2 cases, 64 are F_3 cases, and 360 are S cases. The very small amount of training data for F_3 makes this class the most difficult to predict (Figure 8, bottom left).

We obtained curve prediction results using CTGP to determine, which output class model is applied. For the 4-class model and F_2, we obtained a prediction error of $\mu_{PCA} = 0.49$ with $\sigma^2_{PCA} = 0.28$. With two classes F_1, the prediction error was $\mu_{PCA} = 0.32$, $\sigma^2_{PCA} = 0.15$. Compared to the Table 4, we see that the CTGP results for the two-class classification are considerable better than those in Table 4. Examples of predicted curves (for output variable 1) are shown in Figure 9. In these plots, the black curve is the ground truth curve and is plotted for the correct number of time steps $T(\mathbf{x})$ for test input $\mathbf{x}$. The red curve is the predicted curve using the model for the correct class $C(\mathbf{x})$, and it is plotted for the maximum length $T^{C(\mathbf{x})}_{\max}$ that defines the class $C(\mathbf{x})$. If CTGP incorrectly predicted the class, then the blue curve is the predicted curve using the model for the incorrectly predicted class $C^{\text{pred}}(\mathbf{x})$ and it is plotted for the maximum length $T^{C^{\text{pred}}(\mathbf{x})}_{\max}$ of that class. In the title for each test plot there is an indication of whether the

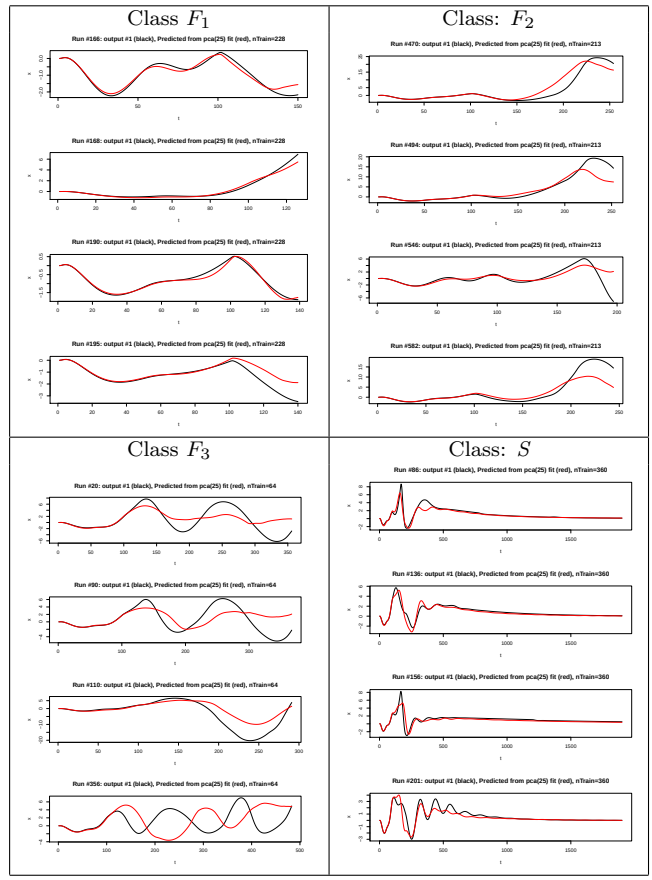

Figure 8: $D = 25$ PCA-based predictions for each class (Output Variable 1).

CTGP classifier predicted the correct class or not, and the incorrect class prediction is given for classification errors.

The red prediction curve for the correct class is always at least as long as the black ground truth curve because the true length $T(\mathbf{x})$ must be less than or equal to $T^{C(\mathbf{x})}_{\max}$ in order for $\mathbf{x}$ to be in the class $C(\mathbf{x})$. (Note that this is different from the plots in Figure 8, in which we truncated the predicted curve to the known correct length $T(\mathbf{x})$.) For tests in which CTGP predicts the wrong class $C^{\text{pred}}(\mathbf{x})$,the predicted blue curve may be shorter or longer than the black ground truth curve. In the third example in the upper left of Figure 9, the correct class is F_1 with $T^{F_1}_{\max} = 180$, but the input was incorrectly classified as S with $T^{S}_{\max} = 1901$ so that the predicted curve is much longer. In the last example in the lower right of Figure 9, the correct class is S with $T^{S}_{\max} = 1901$ but the predicted class was F_3 with $T^{F_3}_{\max} = 600$ so that the predicted curve is shorter.

In Figure 9, we see more examples of excellent curve predictions using the correct output class predicted by

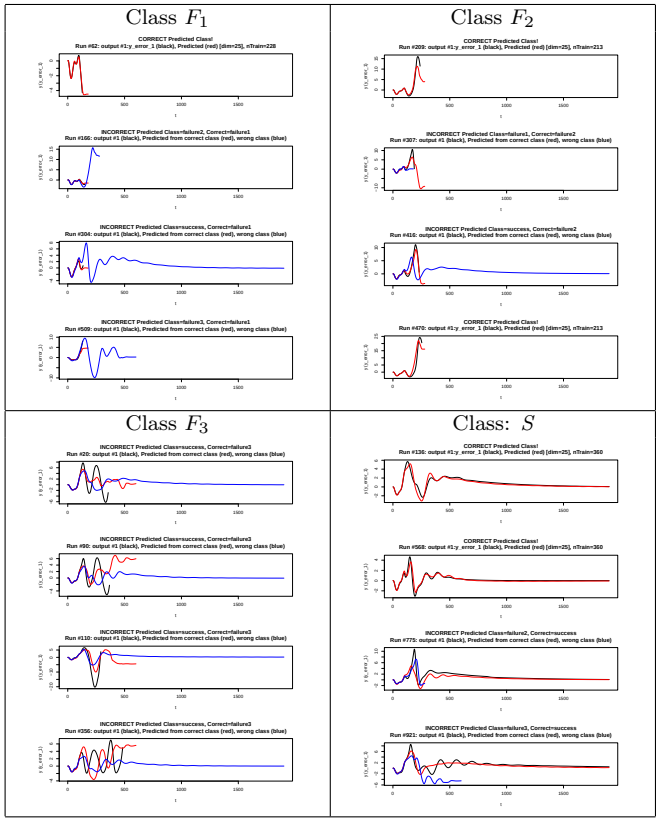

Figure 9: Predicted Output variable 1 curves using CTGP classification. $D = 25$ PCA-based predictions: (upper left) class F_1, (upper right) class F_2, (lower left) class F_3, (lower right) class S.

CTGP. We also see something quite interesting in the examples with incorrect classification: the predicted output curve using the model from an incorrect class is typically quite good near the beginning of the simulation run and often does reasonably well over a significant fraction of the true output curve. As expected, the predicted curve using the correct class is usually better than the one using the incorrect class.

7 Conclusion

In this paper, we discussed a novel statistical toolchain for efficient detection of time-to-failure and prediction of output curves of an adaptive controller. Due to nonlinearity and the large number of parameters in such a controller, it is extremely hard to see which parameter settings yield a stable control system and which ones lead to instability after some time. We predict if the controller is stable, the time-to-failure if unstable, and the output curves. Reliable information is extremely important for autonomous decision making on-board a UAS. For our prediction technique, we used a two-stage statistical approach based on Treed Gaussian Processes (TGP). This method reduces the overall error in the time-to-failure prediction by an order of magnitude for our adaptive control simulator application.

References

[1] T. Broderick and R. B. Gramacy. 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`.

[2] Tamara Broderick and Robert B. Gramacy. Treed gaussian process models for classification. *Proc. 11th Conference of the International Federation of Classification Societies (IFCS-09)*, pages 101–108, March 2009.

[3] John J. Burken, Peggy Williams-Hayes, John T. Kaneshige, and Susan J. Stachowiak. Reconfigurable control with neural network augmentation for a modified F-15 aircraft. Technical Report NASA/TM-2006-213678, NASA, 2006.

[4] A. Calise and R. Rysdyk. Nonlinear adaptive flight control using neural networks. *IEEE T. on Contr. Syst. T.*, 18(6):14–25, 1998.

[5] H. Chipman, E. George, and R. McCulloch. Bayesian cart model search (with discussion). *J. Am. Stat. Assoc.*, 93:935–960, 1998.

[6] H. Chipman, E. George, and R. McCulloch. Bayesian treed models. *Mach. Learn.*, 48(3):303–324, 2002.

[7] Stefano Conti and Anthony O'Hagan. Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140(3):640 – 651, 2010.

[8] Robert B. Gramacy. *Bayesian Treed Gaussian Process Models*. PhD thesis, University of California at Santa Cruz, December 2005. `http://faculty.chicagobooth.edu/robert.gramacy/papers/gra2005-02.pdf`.

[9] Robert B. Gramacy. tgp: An r package for bayesian nonstationary, semiparametric nonlinear regression and design by treed gaussian process models. *Journal of Statistical Software*, 19(9):1–46, June 2007.