Real-Time Detection of In-flight Aircraft Damage

Brenton Blair and Herbert K. H. Lee
Department of Applied Mathematics and Statistics
University of California, Santa Cruz

Misty Davies
NASA Ames Research Center

Abstract

When there is damage to an aircraft, it is critical to be able to quickly detect and diagnose the problem so that the pilot can attempt to maintain control of the aircraft and land it safely. We develop methodology for real-time classification of flight trajectories to be able to distinguish between an undamaged aircraft and five different damage scenarios. Principal components analysis allows a lower-dimensional representation of multi-dimensional trajectory information in time. Random Forests provide a computationally efficient approach with sufficient accuracy to be able to detect and classify the different scenarios in real-time. We demonstrate our approach by classifying realizations of a 45 degree bank angle generated from the Generic Transport Model flight simulator in collaboration with NASA.

1 Introduction

The goal of this research is to develop a real-time aircraft damage detection algorithm. When damage occurs mid-flight, it is critical to be able to warn the pilot as quickly as possible. Because damage can occur in many ways, it is helpful to have multiple systems that can detect damage. We develop here a model to detect and classify damage based on flight trajectory information. The model can monitor trajectory data in real-time to monitor for a sudden occurrence of damage.

As part of work done within NASA's Aviation Safety Program, a non-linear six degree-of-freedom flight dynamics model of a scaled transport aircraft was implemented as a computer simulator. The Generic Transport Model (GTM) sub-scale simulator was designed in part to help train pilots how to properly handle upset conditions in scenarios that would not be feasible in a real-life flight in a transport aircraft such as a Boeing 757 (Jordan et al., 2005). We develop our methodology on this sub-scale simulator, with the intention of being able to eventually apply it to flight data.

The flight simulator enables us to take an experimental design approach where we explore the input space, which includes various physical parameters. There is also the capability to onset damage to the aircraft at any time-step in the simulation and allow the flight trajectory to continue to evolve. The output data of even a few seconds of the simulator is on the order of thousands of observations if you consider the small time-step used and over a dozen kinematic measurements provided.

The general emphasis in the literature has been placed on developing methods to give pilots a few more seconds of control in an upset situation via adaptive-control algorithms. Hovakimyan and Cao (2011) developed and successfully tested an L-1 adaptive control system on over ten flights of the GTM model aircraft. While adaptive control systems have seen great advances in recent years, Holloway and Johnson (2007) caution that such systems may be futile if lacking proper information about the health of related components and systems. Thus the development of a damage detection system would supplement an adaptive control system as well as provide pilots better opportunities for recovery during loss of control scenarios. Indeed, our focus is instead on fast detection of damage. The benefit of this could be twofold as it could be utilized to supplement an adaptive control algorithm or directly be used by a pilot of the aircraft. To explore the development of a real-time damage detection system, we begin with a motivating example generated from the GTM simulator.

Perhinschi et al. (2011) conducted a fault detection study of the GTM simulator utilizing an artificial immune system design and an artificial neural network to predict angular rate output parameters. This is perhaps the most related publication to date in that it utilizes the GTM and statistical learning to identify damage. Our methodology differs in that we fit a model to more Simulator output parameters in an off-line ensemble fashion.

The remainder of the paper explores statistical modeling methodology and is outlined as follows: in the rest of this section we discuss the NASA GTM simulator in detail and outline the motivation of our research. In Section 2 we describe the methodology behind our statistical modeling approach, including the design of a simulation experiment to generate data to train a statistical model, principal component analysis (PCA) as a dimension reduction technique and a Random Forest approach to classification. In Section 3, we expand our methodology to consider the inherent sequential structure of the problem. In Section 4, we present results of our sequential algorithm. In Section 5, we offer conclusions and ideas for future work.

1.1 The Generic Transport Model Simulator

The Generic Transport Model (GTM), a dynamically scaled unmanned aerial vehicle (UAV) developed to research the control of large transport vehicles under upset conditions, is one vehicle in the Airborne Subscale Transport Aircraft Research (AirSTAR) testbed. AirSTAR provides an integrated flight test infrastructure which utilizes remotely piloted, powered sub-scale models for flight testing built as part of NASA's Aviation Safety Program. Its development dates back to the early 2000s at NASA's Langley Research Center. The UAV is a 5.5% sub-scale model of a transport vehicle such as a Boeing 757 (Jordan et al., 2006).

Using data from previous wind tunnel tests along with as-built mass properties data from the fabrication team, a non-linear six degree-of-freedom flight dynamics model of the GTM was developed and incorporated into a real-time pilot simulation tool. This PC-based simulator was used by the pilots to evaluate the flight handling characteristics of the dynamically scaled aircraft. It also allowed the pilots to practice flight procedures during degraded performance conditions. It implements general rigid body equations of motion for the vehicle dynamics and draws aerodynamic forces from a standard coefficient expansion implemented as table lookups. The simulation is coded in Simulink, a model-based environment using a commercial simulation package from Mathworks, Inc. The software makes use of numerical libraries for basic operations as well as the overall time-stepping and numerical integration routines.

The simulator accommodates a large number of inputs, we focus on the 12 inputs the engineers deem most critical. Table 1 (Left) shows these inputs and their lower and upper bounds. Ixx, Iyy, Izz are the moments of inertia about each respective axis, while Ixz, Iyz, Ixy are the corresponding products of inertia. Moments of inertia by definition must be positive, while products of inertia may be negative. Collectively, these six inputs take into consideration the distribution of mass in the airplane and help determine how quickly the aircraft can accelerate or rotate. Along with these inputs, we consider potential variability in the airspeed, angle of attack, weight of the aircraft, altitude, and wind. The range for each input we vary is provided in the table.

For each simulation run, 13 kinematic outputs at each time-step are calculated and can be found in Table 1 (Right). There are three positional outputs: longitude, latitude, and altitude. The one positional velocity output, equivalent airspeed, is defined as the airspeed the aircraft would be flying at under aerodynamic conditions present at sea level. The remaining outputs include six angular measurements and three angular velocities. The angle of attack is defined as the vertical angle between the oncoming air and a reference line on the airplane. The reference line is a line connecting the leading edge and trailing edge at some average point on the wing. A related output, the sideslip angle, is a measurement of the lateral rotation of the aircraft about the reference line from the relative wind. The flight path angle, also known as the climb angle, is measured between

	Lower Bound	Upper Bound	Units
wind speed	0	10	knots
wind direction	0	360	degrees
altitude	600	1000	feet
airspeed	56.25	93.75	knots
angle of attack	2.25	3.75	degrees
gross weight	39	65	pounds
Ixx	0.92	1.53	slug-feet^2
Iyy	3.39	5.65	slug-feet^2
Izz	4.15	6.92	slug-feet^2
Ixz	0.09	0.15	slug-feet^2
Iyz	-0.05	0.05	slug-feet^2
Ixy	-0.05	0.05	slug-feet^2

	Units
longitude	degrees
latitude	degrees
altitude	feet
equivalent airspeed	knots
angle of attack	degrees
sideslip	degrees
flight path angle	degrees
roll angle	degrees
pitch angle	degrees
yaw angle	degrees
roll angular velocity	deg/sec
pitch angular velocity	deg/sec
yaw angular velocity	deg/sec

Table 1: Left: Simulator Inputs, Right:Simulator Outputs

the flight path vector and the horizon. On modern commercial jets it is calculated in reference to the ground.

The roll, pitch, and yaw angles are measurements of rotation between the center of mass of the aircraft and the longitudinal, lateral, and vertical axis respectively. The longitudinal axis passes from the nose to the tail of the aircraft. The displacement about this axis is called bank and occurs when a pilot uses the ailerons and rudder to alter the direction of the aircraft. The lateral axis passes from wingtip to wingtip. Rotation about this axis controls the change in elevation of the aircraft and can be performed by adjusting the elevators. The vertical axis is perpendicular to the wings of the aircraft with its origin at the center of gravity. Rotation about this axis moves the aircraft from side to side and is performed by adjusting the rudder. We also have as output the rate of change with respect to time of these three angular measurements presented as angular velocities in Table 1 (Right).

1.2 Motivation for Real-time Classification

Although modern aircraft are designed to avoid extreme pitch, roll and yaw environments, there are situations such as sensor errors that can lead to an upset for which pilots have not been trained. Airline pilot training is based significantly on full-flight simulators that are calibrated to a limited flight-verified and wind-tunnel tested envelope. Typically there is no data to model handling characteristics in extreme attitudes. Thus, pilots may not be able to train in that setting

using a full-flight simulator. This presents the possibility that pilots may experience a condition in flight that they have not previously experienced in training. The primary motivation of the GTM simulator is to create an environment that enables pilots to realistically train in upset conditions while not risking their safety or unnecessarily causing damage to a commercial sized aircraft. For example, the GTM allows a pilot to train in a simulated environment to recover control and land an aircraft with damage to the wingtip.

The GTM simulator has the capability to consider instantaneous onset of damage to various surfaces on the air vehicle including the wing, elevator, stabilizer, rudder, and tail. In each damage case, the GTM simulator takes into account the change in mass, center of gravity and aerodynamic properties of the aircraft. In certain instances of damage, if detected early enough, the pilot of the aircraft may employ strategies to maximize the possibility of recovery and safe landing of the aircraft. In other instances of damage, the pilot's options are extremely limited. Nonetheless, it is imperative that if damage does occur, a real-time detection system is in place that can notify the pilot specifically where the damage has occurred.

As a motivating example, suppose that an aircraft is making a 45 degree bank angle (roll) in an attempt to make an abrupt change of direction. Using the GTM simulator, we maintain a 45 degree bank angle for 5 seconds and then inflict different damage cases to the aircraft. We plot the resulting trajectories for each of the damage cases (continuing the simulation for 3 seconds) along with the planned flight path in Figure 1. The planned, or "nominal", trajectory is the black line with arrows indicating the flight direction in Figure 1. The distinct damage cases are represented by 5 colored lines that deviate from the nominal flight path.

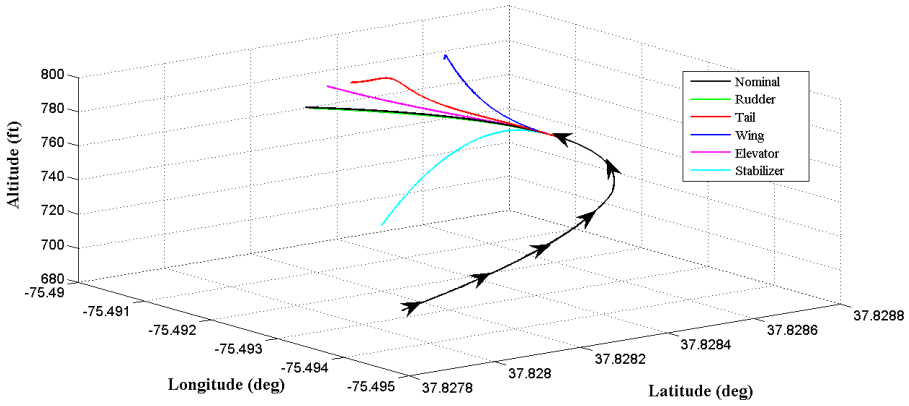

Figure 1: Trajectories for various damage cases.

Perhaps the most interesting damage case involves the rudder in the sense that its deviation from the nominal flight path is significantly less than the other cases. If damage is accurately detected

in this scenario, then the pilot has the best chance of safely landing the aircraft among the damage cases considered. However, as a result of the minimal deviation from the planned flight path, it certainly could go unnoticed by the pilot far longer than damage to other surfaces of the aircraft. If the pilot is unaware where damage actually has occurred, this creates a scenario where continued effort to use that surface for recovery of control could occur. This certainly is a less than ideal situation as the pilot could be wasting time or even making the situation worse, while other fully functional surfaces could be used instead. The stabilizer case is the most distinct from the others, having the worst prognosis of recovery. However, an adaptive-control algorithm may be able to offset the damage and thus provide a pilot with an opportunity of recovery. The elevator and tail cases have similar impact on the resulting trajectory. The impact of the wing damage, although unique from the others, does not initially appear as severe as the stabilizer. While we will use this as our motivating example to build our model, we must consider that there are initial conditions of physical parameters that could vary, resulting in differing trajectories than those plotting in Figure 1. We take that into consideration in our model development and next discuss it further.

2 Model

We proceed with a classification case study including the six scenarios of an aircraft making the 45 degree bank angle discussed above. The six scenarios include the nominal (undamaged) case along with damage cases to the wing, elevator, stabilizer, rudder, and tail of the aircraft. In practice, there is rarely prior knowledge that an aircraft will incur damage during flight, and if so the aircraft remains on the ground until safe to fly. Thus in the extremely rare case that damage occurs in flight, the pilot is not aware until it has occurred or a system has sent a notification. Our goal is to create a system to notify the pilot.

We implement a statistical modeling approach using a Random Forest (Breiman, 2001) to demonstrate that each of the five damage scenarios mentioned can be detected promptly after onset. Before providing details, we first describe the experimental design used to obtain data from the NASA GTM simulator. We next discuss the "sliding window" approach we use to account for the sequential nature of the problem. Principal Component Analysis (PCA) is used as a dimension reduction technique.

2.1 Experimental Design

In our experimental setup, we solely consider damage occurring 5 seconds after the initialization of the plane turning. However when fitting and predicting, we use sliding windows in time because

in practice, damage can occur at any point in time.

Here we consider the five damages cases mentioned along with the nominal case for a 45 degree bank angle. As noted in Table 1, the GTM simulator gives the user the option of varying initial conditions for several inputs. Even in the specific case of a 45 degree bank angle, variability in the wind or aircraft's speed among other inputs can play a significant role in the response of the aircraft's trajectory. To account for this variability, we utilized a Latin Hypercube Sampling (LHS) sampling function available in the statistics toolbox in MATLAB to obtain unique initial conditions of the 12 inputs that were used for 600 flight simulations. For a detailed explanation on LHS, see Santner et al. (2003). The simulations were run for 8 seconds with damage being onset at 5 seconds with the exception of the 100 nominal cases. The default time-step for the simulator is 0.005 seconds and thus there are 1601×13 output values for each of the 600 simulations.

As a result of the size of each output vector from the simulator, we will resort to using Principal Component Analysis as a standard dimension reduction technique. Before discussing that aspect of our model, it is important to outline the implementation of an approach that explores the variability of the simulator output spanning the temporal domain, we refer to as the "sliding window".

2.1.1 The Sliding Window Approach

Realistically, pilots don't know if or when damage will occur to the aircraft. As a result, our goal is to create a single model to continuously monitor for damage to the five different surfaces of the aircraft. To build this model, we want to leverage the variability present in the GTM simulator output that is attributed to aircraft damage, rather than environmental fluctuations possibly present from the LHS sampling of inputs. One effort we could have explored was to train a model simply on the entire set of raw output from the NASA GTM experiment we designed. However, we felt this would not be reasonable because it would be training the model to be detecting damage at only the 5 second mark, and we need the model to be able to pick up damage at an arbitrary time point. Also, with 8 second runs, we don't want to build a model that has to wait for 3 seconds of data past the damage point in order to be able to detect the damage, notification to the pilot needs to occur as quickly as possible.

On the other extreme, it is possible to build a model by considering the 13-dimensional output vector at each time step independently. However, this approach is not sufficient either because of the variability present in the 13-dimensional vector. Thus, we needed to find the optimal subset of information provided in each realization of the GTM training data to build a model, rather than these extreme scenarios mentioned.

We considered the 13 kinematic output over an interval, or window, of time. This window continually shifts as time elapses as we attempt to monitor the flight for damage. We explored different possibilities for the the length of the time windows that included deterministic as well as randomly assigned lengths. We ultimately chose 5 second windows because they provided sufficient information for successful classification in the test phase without requiring an excessive amount of data. For clarity, the information utilized at a given time step in the test phase would include the output of the 13 kinematic outputs at the preceding time steps within 5 seconds. How the data were organized as we trained the model is discussed at the end of this section.

While it was necessary to consider the ideal window length for training a model that continuously monitors for damage in practice, it was also an imperative task to determine an optimal method to train a model that possessed sufficient coverage of the considered temporal domain. Thus, we explored the impact of the damage by "sliding" the window or incrementing the time interval to look at the kinematic outputs as time elapses as we built our model. This is done in an iterative fashion until we have reached the end of our output data, or 8 seconds. While using kinetic output over windows of time was beneficial to classification performance, there remained the question of how to best increment the window to provide sufficient classification performance spanning the entire temporal domain. We found that the performance of the classification was impaired if we used increments larger than 0.5 seconds. For example, if we trained on 5 second windows incremented by 1 second (0-5, 1-6, ..., 3-8 seconds), the classification performance was good near those time points after damage (e.g. 6, 7, 8 seconds). However, at points in time in between (e.g. 5.7, 7.4 seconds), the classification performance deteriorated significantly. Thus, we found that half-second increments provide sufficient temporal spread to give good classification results when testing at times other than the interval endpoints (e.g. 0.7 or 1.2 seconds after damage). While we explored smaller increments, such as 0.25 seconds, we found the results were comparable to using half second increments, but required more storage. There is a trade-off between increasing predictive accuracy when using more increments and extra computational expense from having more increments. The increased predictive accuracy has diminishing returns, and we found the half-second increments gave the best balance between accuracy and computational efficiency.

We now describe how we stored the data into a matrix while implementing the sliding window approach. Although there are alternative ways we could have stored the data, we took an approach we felt most appropriate for this classification problem. Considering we are using 5 second windows, an observation vector can be created by concatenating the 1001 measurements (recall the 0.005 time step) of the 13 output together, giving a vector of length 13013. Each observation vector can be thought of as a row in the matrix we will use for PCA. Next we consider how to store the realizations across the different damage cases. From the simulator we had 100 realizations for each of the six classes (including undamaged). For each of these 600 realizations, we

have 7 unique time windows: 0-5, 0.5-5.5, 1-6, 1.5-6.5, 2-7, 2.5-7.5, 3-8 seconds. We stack the 4200 (100 realizations each $\times$ 6 classes $\times$ 7 time windows) vertically, to create a 4200 $\times$ 13013 matrix. There is some overlap in the time windows, and thus elements of the rows of our data matrix. This resulted from choosing values (discussed earlier) that provided best coverage of the temporal domain in terms of classification performance. By storing the data in a matrix this way, we capture the variability resulting from the different damage onsets over several time windows. This is ideal as it helps discriminate the classes from one another and enables us train a single model to continuously monitor for damage.

2.1.2 Dimension Reduction via PCA

In order to deal with a more reasonably sized data set, we choose to perform Principal Component Analysis (PCA) on the observation matrix created by using the sliding window approach. We choose to use this standard technique because it is imperative we retain as much as variability present in the original data set while reducing its dimension.

Let $\mathbf{X}$ denote our $n \times p$ (4200 x 13013) observation matrix with rank r. We choose to find the PCs via the Singular Value Decomposition (SVD) due to its computational efficiency. We can write the SVD as: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{A}'$. Where $\mathbf{U}$ and $\mathbf{A}$ are $n \times r$, $p \times r$ column orthonormal matrices respectively, and $\mathbf{\Sigma}$ is an $r \times r$ diagonal matrix. By computing the SVD, we are able to obtain the coefficients and standard deviations of the principal components for $\mathbf{X}$. The columns of $\mathbf{U}\mathbf{\Sigma}$ are interpreted as the principal components of $\mathbf{X}$. The diagonal elements of $\mathbf{\Sigma}$, or singular values, are denoted as σ_k for $k = 1 \dots r$ and correspond to the variability associated with each respective PC. For further discussion about PCA, see Jolliffe (2002).

Consider that the cumulative percentage of variation explained (CPVE) can simply be computed using the square of our singular values:

$$CPVE = \frac{\sum_{i=1}^{t} \sigma_i^2}{\sum_{j=1}^{r} \sigma_j^2}$$

Of particular interest is selecting a small subset of the principal component vectors while retaining most of the variance in the original data set. This can be done by selecting the smallest value for t in which the cumulative percentage of variation explained exceeds some threshold. Using a threshold of 0.99 we found $t = 35$ and produced a CPVE plot in Figure 1. Projecting the "sliding" window vectors with the subset of 35 learned PCs, we now have a matrix of 4200 $\times$ 35.

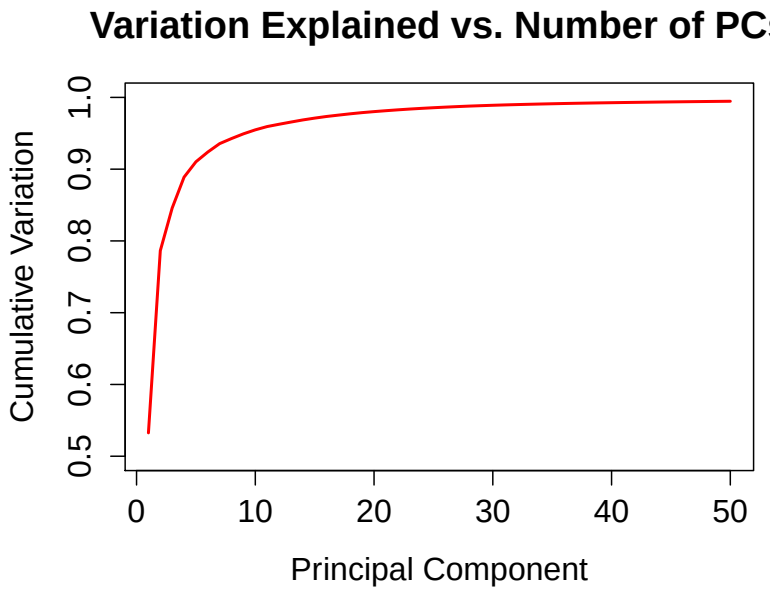

Figure 2: Principal Components.

2.2 Random Forest

Considering the goal to implement an accurate, real-time detection system we elected to explore tree based methods due to their efficiency. While a CART model (Breiman et al., 1984) provided encouraging preliminary results, we found Random Forest models to have much greater predictive accuracy as they avoid the typical high variance found in a single tree. A Random Forest, an ensemble method, is a large collection of tress that are combined to provide for single classification. It can also be thought of as an extension of another popular method, Bootstrapping Aggregation (Bagging) with the difference being that Random Forest considers decorrelated trees by using a random subset of variables at each node split in each tree. Trees were grown deep, with a minimum node size of 1, resulting in the mean number of nodes per tree greater than 700. The subset size of variables (in our case Principal Component scores) to determine node splits was 5, approximately the square root of the original number of variables as Breiman suggests.

Once the forest is built, we can do prediction by considering class probability estimates. Let $\hat{C}_b(x)$ be the class prediction of the bth tree in the forest. The class probability for the kth class is:

$$\hat{p}_k = \frac{1}{B}\sum_{b=1}^{B} I\left(\hat{C}_b(x) = k\right)$$

Observations typically are classified to class $k = \text{argmax}_k \hat{p}_k$

In order to determine the number of trees to include in our forest, we used a common approach by looking at an estimate of the test error rate using Out-of-Bag samples. Because bootstrapping is done via sampling with replacement, each tree grown in the forest will have observations from the original data set that are not used. These samples are known as "Out-of-Bag" (OOB) and can be used accordingly to get an estimate of the error in the forest. In Figure 3 below we can see that the OOB error estimate converges to a minimum after about 200 trees are included in the forest. Thus, we use this value as the number of trees in the forest, as more trees would not necessarily provide better test performance, but would be less efficient.

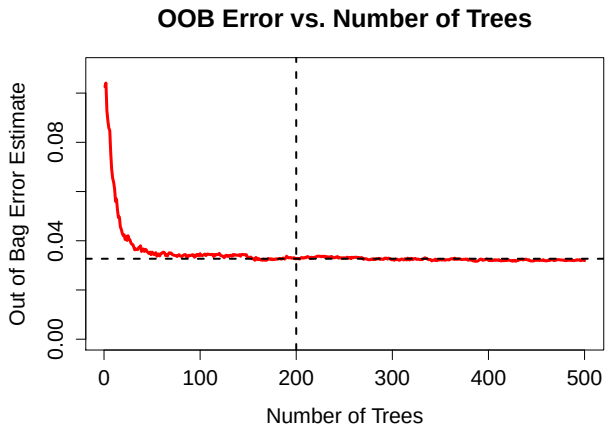

Figure 3: Out of Bag Error

3 Sequential Design

3.1 Motivation

Once we have learned the principal components and random forest, we are able to evaluate our model by classifying test flight trajectories. Because we built a single model (one random forest) to span the temporal domain and we are continuously monitoring for damage, we will use our single classifier repeatedly as we are checking for damage. We seek detection of true damage in minimal time from onset. One possible approach is to simply classify damage to the class that has the maximum empirical probability provided by the random forest. The default state, or null hypothesis, is that there is no damage to the aircraft. Thus, the majority of the time there will be no damage notification because the maximum probability belongs to the nominal class (true negative). However, if we only use a single time step (instantaneous detection) we may have issues with our false positive rate as well as misclassification amongst the damage classes.

To demonstrate these possibilities, empirical probability trace plots were generated by sequentially (sequence of time steps) classifying test cases using the forest previously trained. Trace plots appear in Figures 4-6 that are examples where for a few time steps the forest misclassifies the test observation before classifying the correct class. We are creating the trace plots by obtaining the probability estimates from evaluation of the random forest every 0.05 seconds.

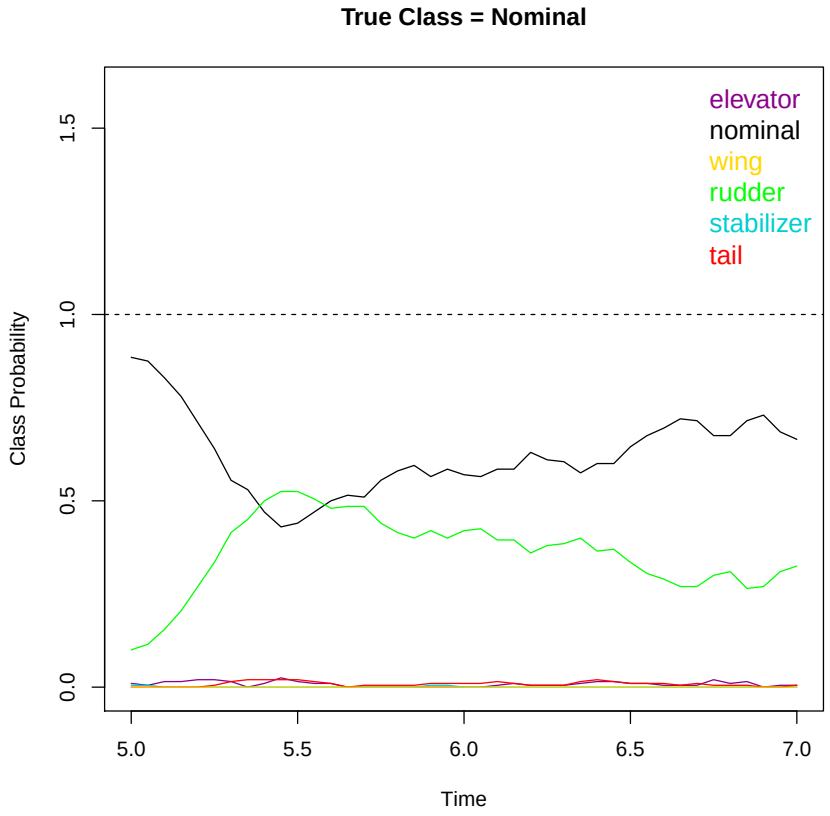

Figure 4: Probability Trace Plots: True Class: Nominal.

Figure 5: Probability Trace Plots: True Class: Wing.

Figure 4 is an example where if we simply look at the class associated with the maximum empirical probability output at a single point in time, we could briefly obtain a false positive (in this case notification of Rudder damage when the aircraft is undamaged). In Figure 5, it can be seen that the same method would briefly misclassify true damage to the wing as damage to the tail. Finally in Figure 6 it can be seen that true damage to the stabilizer would briefly be classified as damage to the rudder.

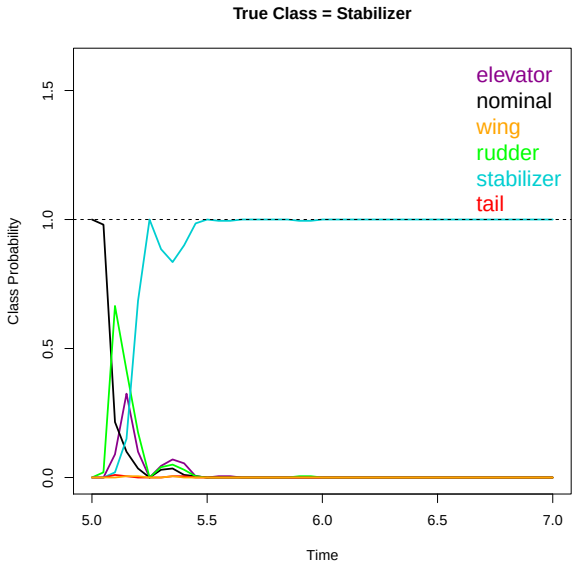

Figure 6: Probability Trace Plot. True Class: Stabilizer.

3.2 Learned Thresholds via Validation Set

The demonstration of misclassifications in the preceding empirical probability trace plots begs the following questions:

1. What is the optimal probability threshold for classifying a damage case?
2. What is the optimal temporal threshold for classifying a damage case?

By optimal, we simply mean the respective values that will minimize predictive classification error. To answer these questions simultaneously, a validation set of 300 cases was created (50 per class). The space explored included threshold probabilities ranging from 0.4 to 0.9 in 0.025 increments and a time window ranging from 2 time steps to 12 time steps (0.1-0.6 seconds). To classify an observation from the validation set as a damage case, it is required to have exceeded the probability threshold consistently for the number of time steps allocated to the time window. All windows of the appropriate number of time steps are sequentially considered until 2 seconds after damage onset. Although there are similarities, this is a separate process from the "sliding window" approach used for PCA that was discussed earlier. If an observation does not exceed the threshold consistently for the number of time steps necessary over all possible windows up to 2 seconds, it is considered a nominal case.
All 300 validation observations are classified over the grid of the probability and temporal thresholds space. For each pair of probability and temporal thresholds considered, the classification error rate is considered. We choose to select the pair of values for the thresholds that minimizes this error rate on the validation set. If the error rate is the same for multiple pairs of thresholds, we select the pair that contains the shortest temporal threshold in consideration of the effort to detect damage in real time. Using this process we found the optimal probability threshold for classification to be 0.575 and the time threshold to be 2 time steps or 0.1 seconds.

4 Results

To evaluate the fit of our model, we now consider an independent test set generated from the GTM flight simulator in MATLAB. The test set contains 600 observations (100 from the six classes). Using the principal components and the random forest learned earlier we generate a matrix of empirical probabilities for each test observation using a time step of 0.05 seconds. With this information we can now use our learned probability and time thresholds to classify each test case. In doing so we see there are only 5 false negatives (in the sense that no damage at all was

detected). Implementation of this is efficient, running as a batch mode in R in seconds. The results are presented in Table 2:

	elevator	Nominal	Rudder	stabilizer	Tail	wingtip
elevator	97	0	0	0	0	0
Nominal	2	98	3	0	0	0
Rudder	1	2	97	0	0	0
stabilizer	0	0	0	100	0	0
Tail	0	0	0	0	100	0
wingtip	0	0	0	0	0	100

Table 2: True class in columns, prediction in rows

While the results thus far seem promising, Table 2 only provides us with the knowledge of how successful we are at detecting damage within 2 seconds of damage onset, but leaves out valuable information. To have a more complete picture, consider Figures 7-9 below. Each figure provides the distribution of times it takes the sequential algorithm to detect the damage to the respective part of the aircraft (only true positives included). In each figure the red vertical line represents the mean time.

The mean time it takes to detect the correct damage class is less than a half of a second in all five damage classes. In the case of damage to the wingtip, the entire distribution is within 1.4 seconds of damage onset. We are most encouraged by the performance of the algorithm when damage occurs to the Rudder as it results in the least affected trajectory among the damage cases. We expected this damage scenario would take longer to diagnose successfully, but our results indicate it is comparable with the other damage cases.

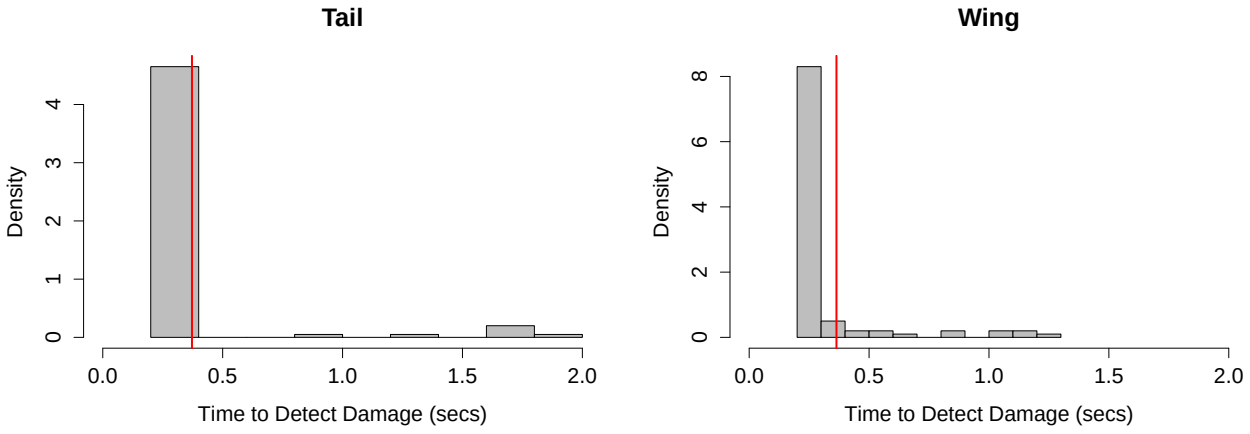

Figure 7: Distribution of time to detect Tail and Wing Damage respectively.

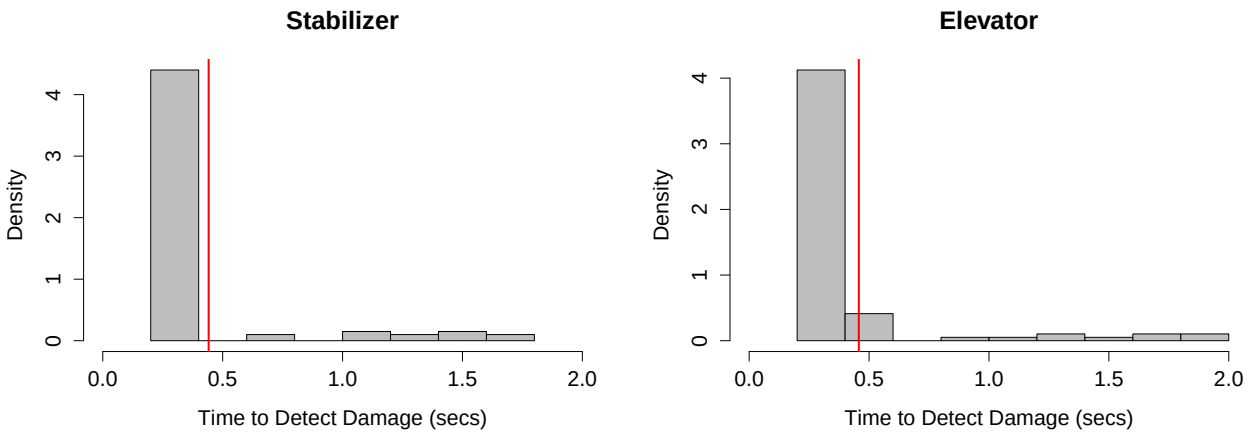

Figure 8: Distribution of time to detect Stabilizer and Elevator damage respectively.

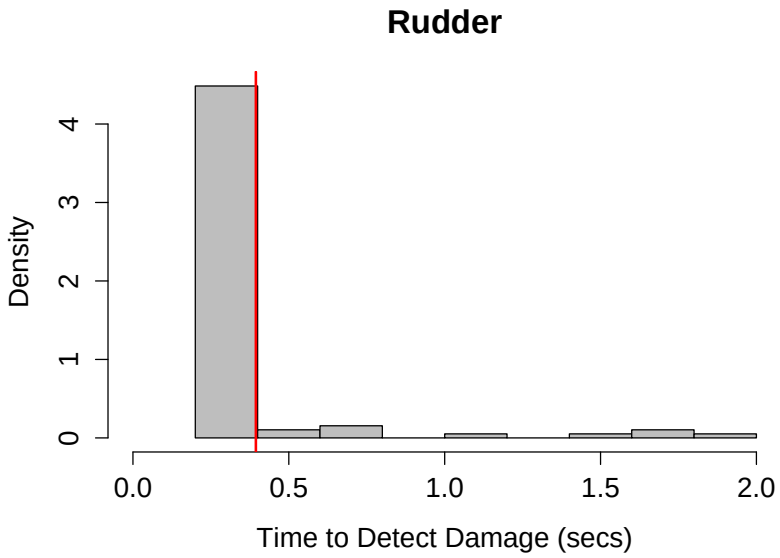

Figure 9: Distribution of time to detect Rudder Damage (Successful cases)

4.1 Timing Profile of Testing Algorithm

Although the classification results we presented thus far seem reasonable as a proof of concept, we have yet to discuss perhaps the most important goal of our model: classification in real time. The test results previously discussed were generated in a batch mode without considering a profile of the time the algorithm took to classify all cases.

If the algorithm were to be realistically implemented in an applied setting, it would require that

the sequential classification scheme at a given instant in time take no longer than the selected time step, 0.05 seconds. Assuming we have kinematic output data stored in an appropriate data structure as well as class probabilities assigned from the previous time step (recall our optimal temporal threshold was two time steps), we must be able to do the following in less than 0.05 seconds:

1. Project the previous 5 second "sliding window" using our learned Principal Components
2. Run our projected test vector through the random forest to get the instantanous class probabilities
3. Concatenate the instantaneous class probabilities to those previously stored
4. Determine if any of the classes have probabilities that exceeded the learned probability threshold (0.575) for 2 time steps (0.1 seconds)
5. Inform the user (i.e. notify the pilot)

To demonstrate a proof of concept of our real time considerations, we iteratively (rather than batch mode) time the outlined tasks for all 600 test cases at 6.5 seconds into the 45 degree bank angle (1.5 seconds after damage when applicable) in the R Studio Development Environment on a laptop computer with the following specifications: Intel® Core™ i3 processor @ 2.26 GHz, 4 GB RAM, Windows 7 64-bit OS. In doing so we have a CPU run time of the outlined process above for each test case. As we can see in Figure 10, the distribution of these run times is entirely less than the chosen time step of 0.05 seconds with a mean of 0.034 seconds. We feel these results are encouraging considering the timing profile results were created in a relatively slow environment, R Studio. Furthermore, if implemented in a faster environment with an upgraded CPU, we would expect to be able to safely use a smaller time step than the one utilized here.

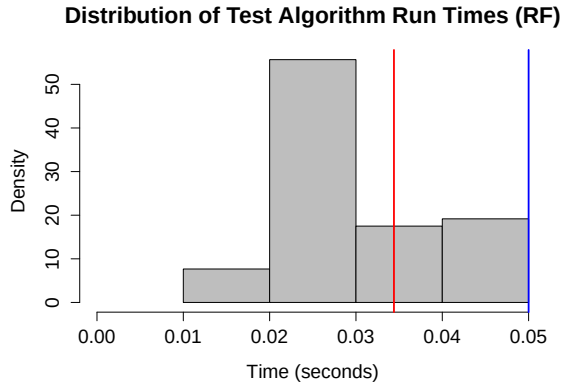

Figure 10: Distribution of Test Classification Run Time

5 Conclusions

Using kinematic data from the NASA GTM simulator we demonstrate a real-time damage classification algorithm. Our methodology combines several statistical techniques including principal component analysis, random forest, and cross-validation to produce a fast and reliable classifier. Using a sliding window approach, we are able to consider the sequential nature of the problem and simultaneously learn probability and temporal thresholds using a validation set from the simulator. This methodology results in an 98.7% predictive accuracy rate and, perhaps more significantly, routinely diagnoses the appropriate damage within a half second of onset on independent test realizations. Even in a scenario where the flight trajectory's deviation from the planned path is minimal, the algorithm is able to detect rudder damage with comparably efficiency to the other classes.

Another encouraging result from our methodology is that the timing of the routine is consistently within the chosen time step of 0.05 seconds implying that it could be implemented in real time. When considering this is done in R studio on an older laptop, we believe in an ideal computing environment it could be much faster and thus a smaller time step could be selected, which could help inform the pilot even more quickly.

Acknowledgment

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

Breiman, L. (2001). "Random Forests." *Machine Learning*, 45, 5–32.

Breiman, L., Friedman, J. H., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.

Holloway, C. and Johnson, C. (2007). "How Past Loss of Control Accidents May Inform Safety Cases for Advanced Control Systems on Commercial Aircraft." In *Proceedings of the IET 1st Int. Conf. on System Safety*.

Hovakimyan, N. and Cao, C. (2011). "L1 Adaptive Control and its Transition to Practice." In *IEEE Conference on Decision and Control*.

Jolliffe, I. (2002). *Principal Component Analysis*. New York: Springer.

Jordan, T., Foster, J., Bailey, R., and Belcastro, C. (2006). "AirSTAR: A UAV Platform for Flight Dynamics and Control System Testing." Tech. rep., NASA.

Jordan, T., Langford, W., , and Hill, J. (2005). "Airborne Subscale Transport Aircraft Research Testbed-Aircraft Model Development." Tech. rep., NASA.

Perhinschi, M. G., Moncayo, H., Wilburn, B., Bartlett, A., Davis, J., and Karas, O. (2011). "Testing of Immunity-Based Failure Detection and Identification Scheme with the NASA Generic Transport Model." In *AIAA Guidance, Navigation, and Control Conference*.

Santner, T. J., Williams, B. J., and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. New York: Springer.