UNIVERSITY OF CALIFORNIA
SANTA CRUZ

Real-time Detection of In-flight Aircraft Damage

A document submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

STATISTICS AND APPLIED MATHMATICS

by

Brenton S. Blair

July 2015

Approved by:

Professor Herbert Lee

Associate Professor Abel Rodriguez

Contents

Real-Time Detection of In-flight Aircraft Damage

Brenton Blair
Department of Applied Mathematics and Statistics
University of California, Santa Cruz

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 three-dimensional trajectories in time. Classification trees 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. 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 in the software environment Simulink. 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 otherwise while traveling through the atmosphere in a transport aircraft such as a Boeing 757. From a researcher's perspective, the simulator provides output that could be explored from various fields of expertise with a common objective of implementing a mechanism to help the pilot identify upset conditions as efficiently as possible.

The flight simulator provides the user with the capability to run simulations one at a time or in a batch mode via MATLAB's command window. This enables us to take an experimental design

approach where we explore the input space, which includes various physical parameters, via a Latin Hypercube Design. 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. From a statistician's perspective, 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 there are over a dozen kinematic measurements provided. The temporal nature of a flight trajectory is also of significance to the approaches considered from the statistical domain.

The remainder of the paper explores statistical learning 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 learning 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 classification tree approach trained via deviance impurity and cost-complexity pruning via minimizing cross-validation error rate. In Section 3, we expand our methodology to consider the inherent temporal structure of the data. 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

Figure 1: GTM-T2 *(Hovakimyan and Cao 2011)*

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. Properties of both vehicles are provided in Table 1 below (*Jordan et al. 2006*).

	Length	Wingspan	Weight	Roll inertia	Airspeed	Altitude
Full Scale Transport	145.5 ft	124 ft	200,000 lbs	2.64e6 sl-ft2	320 mph	13000 ft
5.5% Model	96 in	82 in	49.6 lbs	1.33 sl-ft2	75 mph	1000 ft

Table 1: Comparison of 5.5% dynamically scaled model and full scale properties

To match the aerodynamic performance of a large transport aircraft, similar in shape to NASA's Boeing 757 test aircraft, the GTM T2 weighs 49.6 lb to simulate a $200,000$ lb aircraft. The small size also requires that the aircraft's 16 control surfaces respond more than four times faster than the full-size aircraft. The GTM T2, with a 82 inch wingspan, cruises at 75 mph and $1,000$ feet on twin turbojet engines. On board is a redundant micro inertial reference system, air data probes on each wing tip and potentiometers to measure control surface movements. Commands from the research pilot in a mobile operations station are converted to control surface and engine commands and passed to the aircraft via radio frequency link.

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.

For our research purposes, simulations can run from the command line in MATLAB or by running an m file (MATLAB script). In doing so a Simulink window opens as the simulator is running. Of particular interest is the 13-dimensional kinematic output provided by the simulator at each time step. The output data includes air velocity, longitude, latitude, angular velocities, and Euler angles. A screen shot of a 15 second simulation provides an example of the output varying over time in Figure 2.

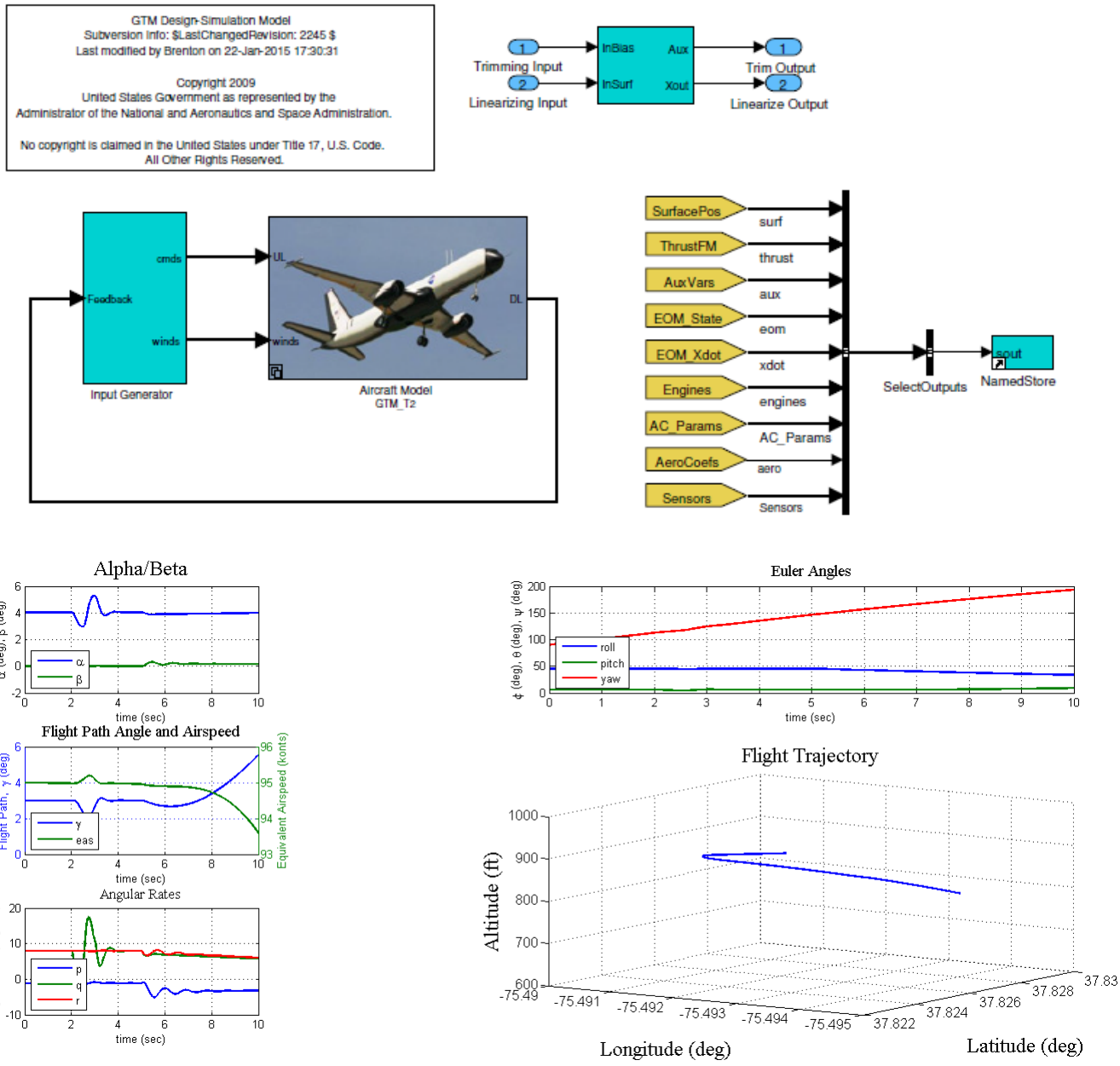

Figure 2: A screen shot of the GTM simulator in Simulink.

1.2 Motivation for Real-time Classification

Although modern aircraft are designed to avoid extreme pitch, roll and yaw environments, situations such as wake turbulence, sensor errors or other unintended consequences can lead to an upset for which pilots have not been trained. Airline pilot training is largely based on full-flight simulators that are calibrated to a limited flight-verified and wind-tunnel tested envelope. Since there is typically no data on which to model handling characteristics in extreme attitudes and

post-stall regions, simulators and the pilots flying them can not train in that setting, creating a situation where pilots may experience a condition in flight for which 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.

While there has been much emphasis in the literature placed on developing methods to give pilots a few more seconds of control in an upset situation via adaptive-control algorithms (for example see Hovakimyan and Cao 2011), our focus is rather on instantaneous 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.

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.

In the spirit of the design of this sub-scale simulator, we will consider a case outside the envelope in which pilots typically train in an aircraft. Suppose that an aircraft is making a 45 degree bank angle (roll) in an attempt to make an abrupt change of direction. The pilot typically would select 'trim' conditions. The change of direction via a steep bank angle requires for the plane to maintain some attitude, which means that the actuators (or control surfaces) of the aircraft (rudder, ailerons, elevators) need to be deflected to some position and held in a steady state. Trimming an aircraft involves finding the positions of these surfaces, and then setting them to be held steady. In the case of the GTM simulator, it requires solving a nonlinear optimization problem to find the desired deflections of the surfaces. Using the trimming capacity with 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 3. The planned, or "nominal", trajectory is the black line with arrows indicating the flight direction in Figure 3. The distinct damage cases are represented by 5 colored lines that deviate from the nominal flight path.

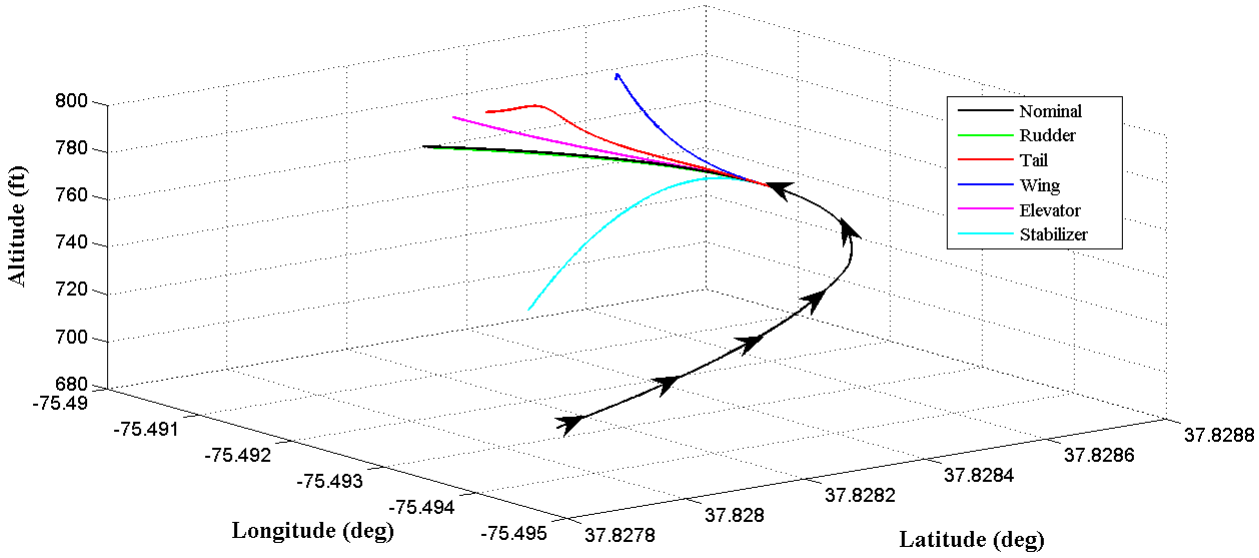

Figure 3: Trajectories for various damage cases.

As we can see, the most interesting damage case is to the rudder in the sense that its deviation from the nominal flight path is significantly less than the other cases. 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 3. We take that into consideration in our model development and now 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 supervised learning approach using Classification Trees (Breiman et al.) to demonstrate that each of the five damage scenarios mentioned can be detected promptly after onset. We fit a single classification tree that can predict instantaneously six classes, including the five damage cases and undamaged case. Details of this are provided in this section, but we first discuss the experimental design used to obtain data from the NASA GTM simulator. This includes a Latin Hypercube Design sampling scheme. Furthermore, we discuss in this section the "sliding window" approach we use to account for the temporal nature of the problem. Principal Component Analysis (PCA) is used as a standard dimension reduction technique and details are provided here.

Just as we focus on the 45 degree bank angle for this case study, we solely consider when damage has occurred 5 seconds after the initialization of the plane turning. In practice, damage can occur at any point in time and there are countless scenarios other than the steep turn we are considering in this case study. For the purpose of our work, we are demonstrating the methodological approach for this specific scenario, but believe it could be extended to other cases.

2.1 Experimental Design

Before we elaborate on statistical learning methodology, we first consider the experimental design we utilized to obtain our training data from the computer simulator. While we specifically consider the 5 damages cases mentioned along with the nominal case for a 45 degree bank angle, the GTM simulator gives the user the option of varying initial conditions for several inputs. In Table 2 below is a small subset of the inputs one could vary with the GTM simulator. For clarity, 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.
Although the GTM is considered a generic model, generation of training data that spans various conditions is necessary. 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 generate 600 (100 per each case) unique eight second simulations by varying the initial conditions of the inputs listed in Table 2.

By utilizing a Latin Hypercube Sampling (LHS) sampling function available in the statistics toolbox in MATLAB, we obtain unique initial conditions that are used for 600 flight simulations. This requires dividing the 12 input parameter domains into 600 even intervals each. The set of

	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

Table 2: Initials Condition Domains

all Cartesian products of the intervals is a partitioning of the sample space into 600^{12} cells. A set of 600 cells is chosen from the 600^{12} considered such that the projections of each of the cells onto each axis is uniformly spread across the axis. For each of the 600 cells selected, a point is then selected from within the cell. That point represents a vector of initial conditions of parameters to be used in the corresponding simulation. The simulations are run using each unique set of initial conditions and as described in the previous section damage is onset at 5 seconds into the simulation with the exception of the 100 nominal cases. For a more detailed explanation on LHS, see Santner et al. 2003.

2.2 Simulation Output

For each simulation run, 13 kinematic outputs at each time-step are calculated and can be found in Table 3 below. For clarity, equivalent airspeed is defined as the airspeed the aircraft would be flying at under aerodynamic conditions present at sea level. 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".

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

Table 3: Simulator Outputs

This output data from the simulator is stored in a four dimensional array in MATLAB. The dimensions are 6 x 100 x 13 x 1601 (scenario x realizations x output x observations). In the following section we will discuss how this data is "folded" in to a matrix suitable for PCA.

2.2.1 The Sliding Window Approach

Due to the temporal nature of our application, consideration must be placed on properly transforming the simulator output data to account for variability of the kinematic measurements over time. It is possible to attempt to classify using only the 13-dimensional output vector. However, this approach is not nearly as sufficient as the variability present in the 13-dimensional vector could be attributed to different initial conditions rather than damage onset. Therefore, we believe it is beneficial to utilize the information available in the simulator outputs after damage onset, but also relative to output calculated beforehand during the nominal flight trajectory. We consider 13 kinematic outputs not at a single point in time, but over an interval, or window, of time. In addition to considering the outputs measured over a given time window, we further explore the impact of the damage by "sliding" the window or incrementing the time interval to look at the kinematic outputs as time progresses after damage onset. This is done in an iterative fashion until we have reached the end of our output data, or 8 seconds.

We explored different possibilities for the the length of the time windows as well as their starting points. This included deterministic window lengths and starting points as well as randomly

assigning starting points. We ultimately decided on 5 second windows with 0.5 increments. 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 was sufficient near those time points after damage (i.e. 6, 7, 8 seconds). However, at any time-steps in between the classification performance dramatically deteriorated. It may seem satisfactory to instantaneously classify the appropriate damage case 1 second after onset, but a closer look reveals we will need a sequential algorithm for optimal performance (see Section 3 for details). Thus, we find that 0.5 seconds increments provides the necessary instantaneously classification results when testing at times other than exactly at a second (e.g. 0.7 or 1.2 seconds after damage instead of 1.0 or 2.0 seconds). 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.

Before we discuss Principal Component Analysis, we now discuss how we perform matrix "folding" of the four dimensional array while implementing the sliding window approach. Although there are plenty of ways to approach collapsing the data into two dimensions, we took an approach we felt most appropriate for this classification problem. Considering that we wish to classify test cases instantaneously, a single observation vector can be created by concatenating the 1001 measurements (from the 5 second window) of the 13 output together, giving a vector of length 13013. By storing the data this way, we retain all the information possible for the selected window length. 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. 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 inevitably improves the training of our classification tree.

2.2.2 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 and element, x_{ij}, corre-

sponding to the entry in row i, column j. The sample covariance matrix is defined as:

$$\mathbf{S} = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)$$

One interpretation of PCA is to let $z_{i1} = \mathbf{a_1'}x_i$ and choose the vector of coefficients $\mathbf{a_1'}$ to maximize the sample variance: $\frac{1}{n-1}\sum_{i=1}^{n}(z_{i1} - \bar{z}_1)^2$ subject to the normalization constraint $\mathbf{a_1'a_1} = 1$. Furthermore, one would continue in an iterative manner finding the $\mathbf{a_k'}$'s, $k = 2, \ldots, p$ by maximizing the respective sample variances of the z_{ik} subject to the normalization constraints as well as being uncorrelated with the other z_{ik} in the sample. We refer to $\mathbf{a_k'x}$ as the kth Principal Component (PC) and z_{ij} as the score for the ith observation on the kth PC. It can be shown that the sample variance of the PC scores for the kth PC is, σ_k^2, the kth largest eigenvalue of $\mathbf{S}$ and the corresponding eigenvector, $\mathbf{a_k}$, is referred to as the loading vector.

We choose to find the PCs via the Singular Value Decomposition (SVD) due to its computational efficiency. We can write the SVD as:

$$\mathbf{X} = \mathbf{U\Sigma A'} = \mathbf{X}\sum_{k=1}^{p} a_k a_k'$$

Where $\mathbf{U}$ and $\mathbf{A}$ are $n \times r$, $p \times r$ column orthonormal matrices respectively, $\mathbf{\Sigma}$ is an $r \times r$ diagonal matrix, and a_k is the kth eigenvector of $\mathbf{X'X}$. By computing the SVD, we are able to obtain the coefficients and standard deviations of the principal components for $\mathbf{S}$. The columns of $\mathbf{U\Sigma}$ are interpreted as the principal components of $\mathbf{X}$. The diagonal elements of $\mathbf{\Sigma}$, or singular values, are denoted as σ_k for $k = 1 \ldots r$ and correspond to the square root of the eigenvalues of $\mathbf{X'X}$. For further discussion about PCA, including an interpretation as the best low rank approximation of the observation matrix, 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 0.995 we found $t = 50$ and produced a CPVE plot in Figure 4. Projecting the "sliding" window vectors with the subset of 50 learned PCs, we now have a matrix of 3600×50. It is possible to use fewer PCs as Figure 4 reveals most of the variation is found in a few PCs. We found (see Section 4) that our classification results were both sufficient and consistently determined in real-time and thus retained 50 PCs, although using fewer may produce similar results.

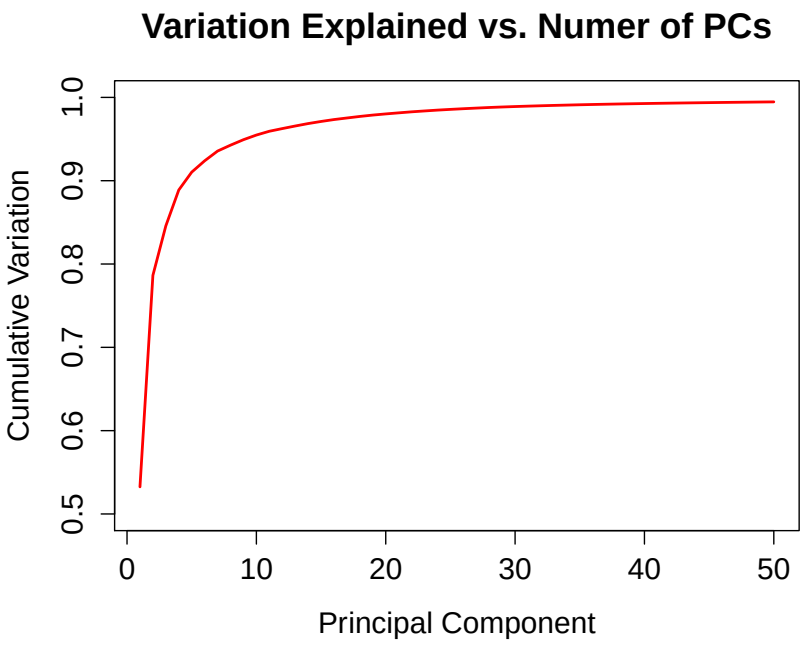

Figure 4: Principal Components.

2.3 Classification Trees

With the goal of a real-time classification method in mind, we turn to Classification Trees (CART Breiman et al. 1984) because of their computationally efficiency. In addition to their efficiency, trees are inherently simple to fit and interpret as well as practically assumption-free. On the other hand, trees tend to have high variance and thus sensitivity to small changes in data reflects their instability. There are methods available that do minimize the effects of this high variance such as Boostrap Aggregation (Bagging) and Random Forests (Breiman 2001). For the time being we do not consider these possible ensemble methods because of concerns of real-time implementation. The main idea behind classification trees is to perform recursive binary partitions of the sample space into sets of rectangles, using observed proportions of classes in each partition to fit a simple model. It is natural to ask: how does one decide to partition the sample space when building the classification tree?

The most common approach is to define a node impurity measure that will be used as a criterion to find optimal splits. We define an impurity measure as:

Definition An impurity function is a function, $Q_m(T)$, defined on the set of all K-tuples of

$$\text{numbers } (p_{m1}, ..., p_{mK}) \text{ satisfying } p_{mk} \geq 0, k = 1, \ldots, K, \sum_k p_{mk} = 1$$

There are three properties which an impurity measure must satisfy:

1. $Q_m(T)$ achieves a maximum only for the uniform distribution, i.e. all p_{mk} are equal
2. $Q_m(T)$ achieves a minimum only when the probability of being in a certain class is 1 and 0 for all the other classes.
3. $Q_m(T)$ is a symmetric function.

There are three common choices for node impurity measures for classification trees (Hastie et al. 2001):

$$\begin{aligned} \text{Misclassification error:} \quad Q_m(T) &= 1 - \hat{p}_{mk(m)} \\ \text{Gini Index:} \quad Q_m(T) &= \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \\ \text{Deviance:} \quad Q_m(T) &= -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \end{aligned}$$

where the proportion of class k observations in our training set that are in the region R_m containing N_m total observations in node m is:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

Observations are classified in node m to class $k(m) = \text{argmax}_k \hat{p}_{mk}$, the majority class in that node.

Now that we have defined the impurity measure, we describe how to use it at a given node to determine the variable j and split point s that gives the best binary partition. Define a pair of half planes as $R_1(j, s) = \{X | X_j \leq s\}$ and $R_2(j, s) = \{X | X_j > s\}$. For each splitting variable, the determination of the split point can be done efficiently by scanning through all of the inputs and determining the pair (j, s) that minimizes:

$$\sum_{x_i \in R_1(j,s)} Q_1(j, s) + \sum_{x_i \in R_2(j,s)} Q_2(j, s)$$

While we defined three commonly used impurity measures, the Gini index and Deviance are differentiable, and therefore easier to optimize numerically than misclassification error. For the purposes of our application, we explored both of these options, but ultimately used deviance as the node impurity measure. Before we discuss the tree we built from our training data, we briefly review cost-complexity pruning as we utilized this mechanism to select the appropriate tree size while minimizing the possibility of overfitting.

2.3.1 Pruning

Cost-complexity pruning is used as a means to determine the appropriate tree size. If a tree is grown too deep, there is a potential for overfitting. On the other hand, if a tree is not grown enough, it may not capture the underlying structure necessary for successful classification. Define our initial tree grown via CART as T_0 and let $T \subset T_0$ be a any subtree created by collapsing any number of terminal nodes. Let $|T|$ be the number of non-terminal nodes in T. Define the cost complexity criterion as

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

where α is a tuning parameter that determines the tradeoff between the size of the tree and its goodness of fit to the data. We wish to find the unique smallest subtree T_α that minimizes $C_\alpha(T)$ by successively collapsing the internal node that produces the smallest per-node increase in $\sum_m N_m Q_m(T)$. We are guaranteed this sequence of subtrees will contain T_α. This requires a value for α, which we select by finding the value that minimizes the 10 fold cross-validation error rate.

While we have 50 principal components, the tree we obtain using CART with cost-complexity pruning only contains 17 terminal nodes with the 35 PC being the largest. Thus, it is only necessary to project our simulator output data with at most 35 PCs under this methodology. The root node is split on the 3rd PC and the set of PCs used are $\{3, 4, 5, 6, 8, 9, 14, 15, 20, 27, 30, 33, 35\}$. While PCs help with the dimensionality of the data, it makes interpretability of the split values difficult and thus we do not include a diagram of the tree that we would otherwise. It is also of note that of the 17 terminal nodes, only one is majority class Rudder while 5 are nominal. While this methodology enables us to efficiently classify damage at an instant in time, we have yet to consider the sequential nature of the problem in terms of classification.

3 Sequential Design

3.1 Motivation

Once we have learned the principal components and classification tree, we are able to evaluate our model by classifying test flight trajectories instantaneously. It is natural to consider the sequential nature of the problem. We seek detection of true damage in minimal time from onset. 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 classifying test cases using the tree previously trained. Trace plots appear in Figures 5-7. Each plot demonstrates an example where for a few time steps the tree in misclassifying the test observation before classifying the correct class. Figures 5 and 6 show true nominal test cases that were misclassified for a duration, but eventually classified correctly. In Figure 7 an elevator damage case is presented. Initially for a few time steps it is classified as rudder damage before being classified correctly. A time step of 0.05 seconds was used in the generated trace plots.

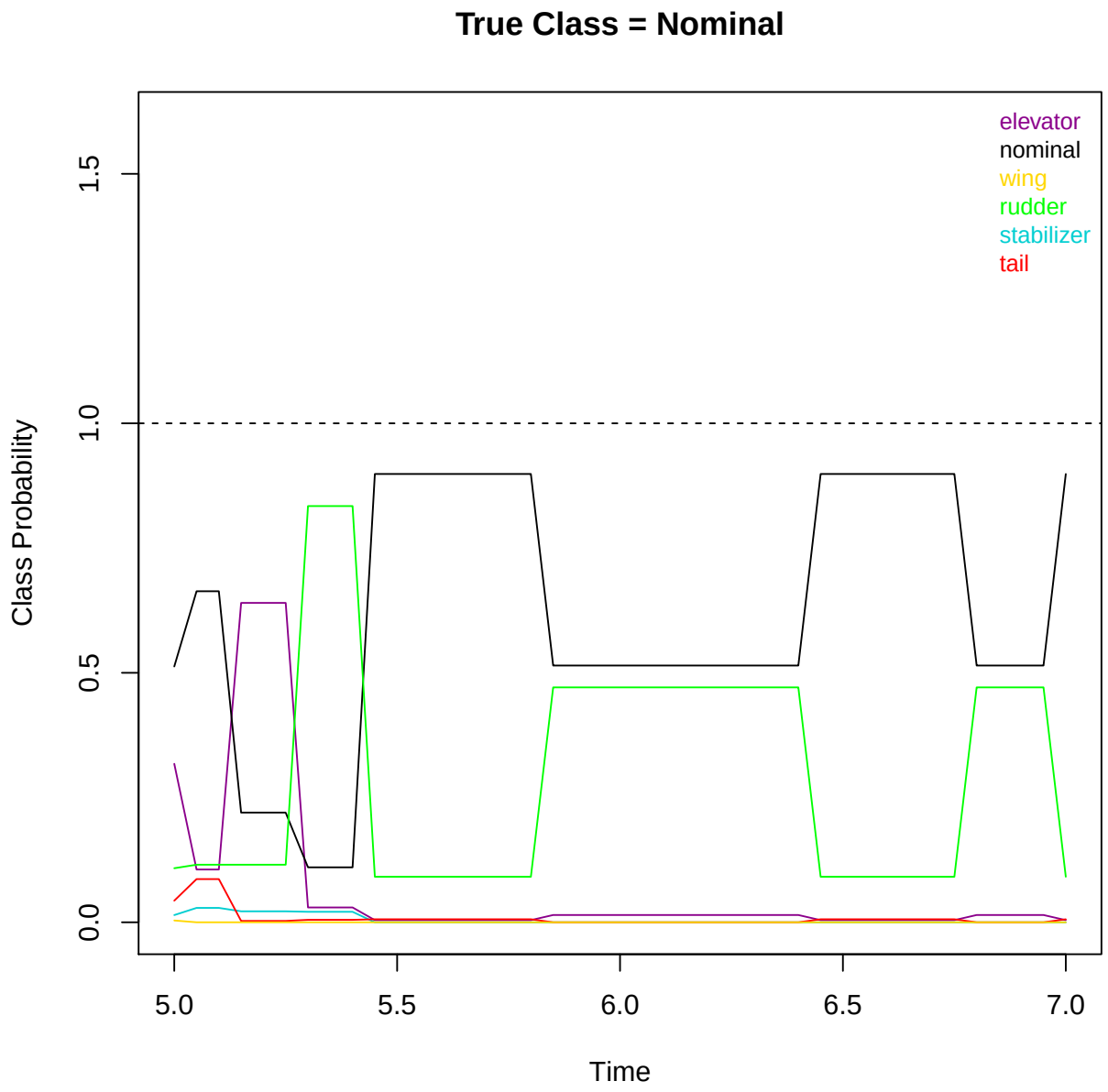

Figure 5: Probability Trace Plots: True Class: Nominal.

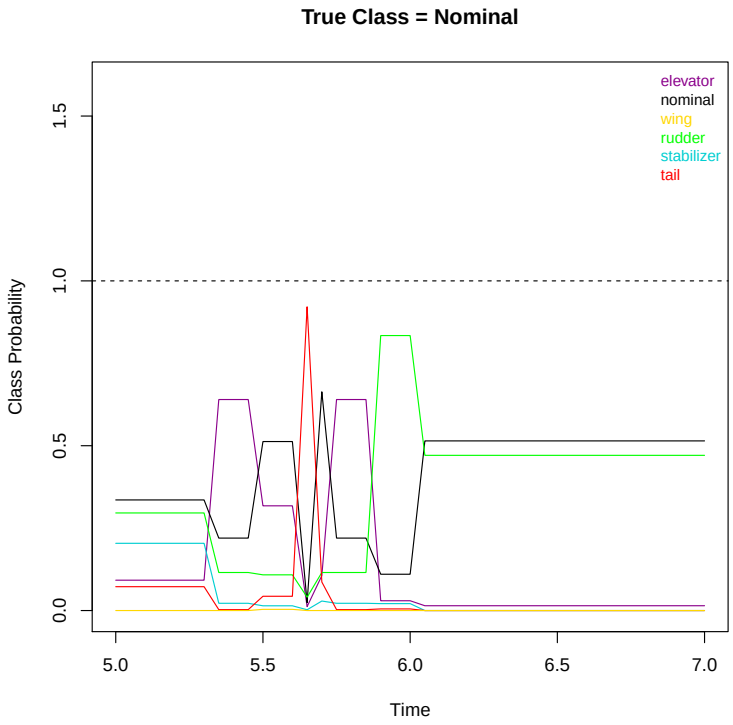

Figure 6: Probability Trace Plots: True Class: Nominal.

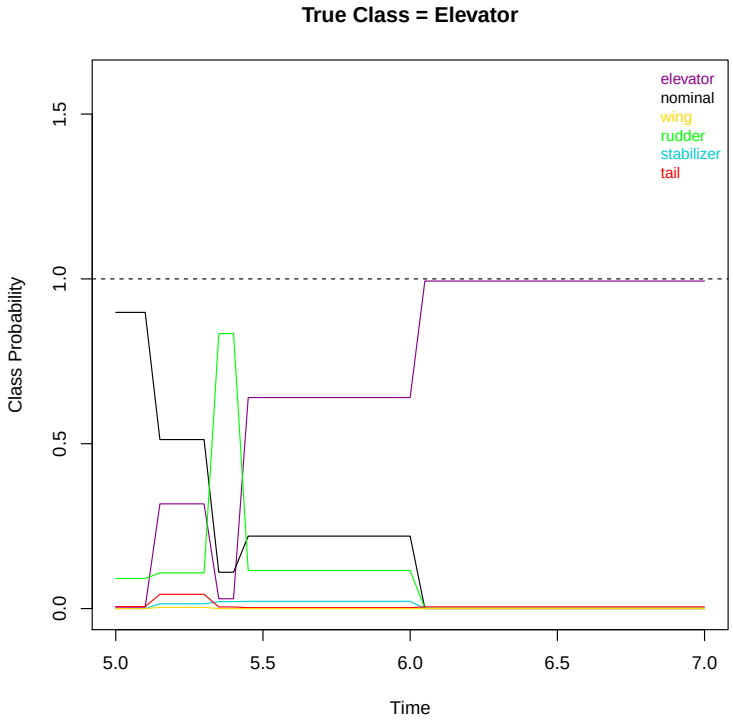

Figure 7: Probability Trace Plots. True Class: Elevator.

3.2 Learned Thresholds via Validation Set

Two questions can be asked when considering the sequential nature of the problem:

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

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 a temporal threshold of minimal value in consideration of the effort to detect damage in real time. Using this process we learned the probability threshold for classification is 0.475 and the time threshold is 6 time steps or 0.3 seconds. While this method appears to be sufficient, there are other schemes that could be considered such as minimizing false positive and false negatives.

4 Results

To evaluate the fit of our sequential model, we now consider a 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 classification tree 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 9 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 4:

	elevator	Nominal	Rudder	stabilizer	Tail	wingtip
elevator	88	0	2	3	0	0
Nominal	1	82	5	3	0	0
Rudder	8	13	88	6	2	0
stabilizer	0	0	0	85	0	0
Tail	2	0	0	0	97	3
wingtip	1	5	5	3	1	97

Table 4: Sequential Classification Experiment

While the results thus far seem promising, Table 1 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 8-10 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 1 second in four of the five damage classes, including less than half a second in the case of wingtip damage. In the case of the rudder damage, the mean time is 1.3 seconds and the distribution appears to be bimodal. Recall that rudder damage results in the least affected trajectory among the damage cases.

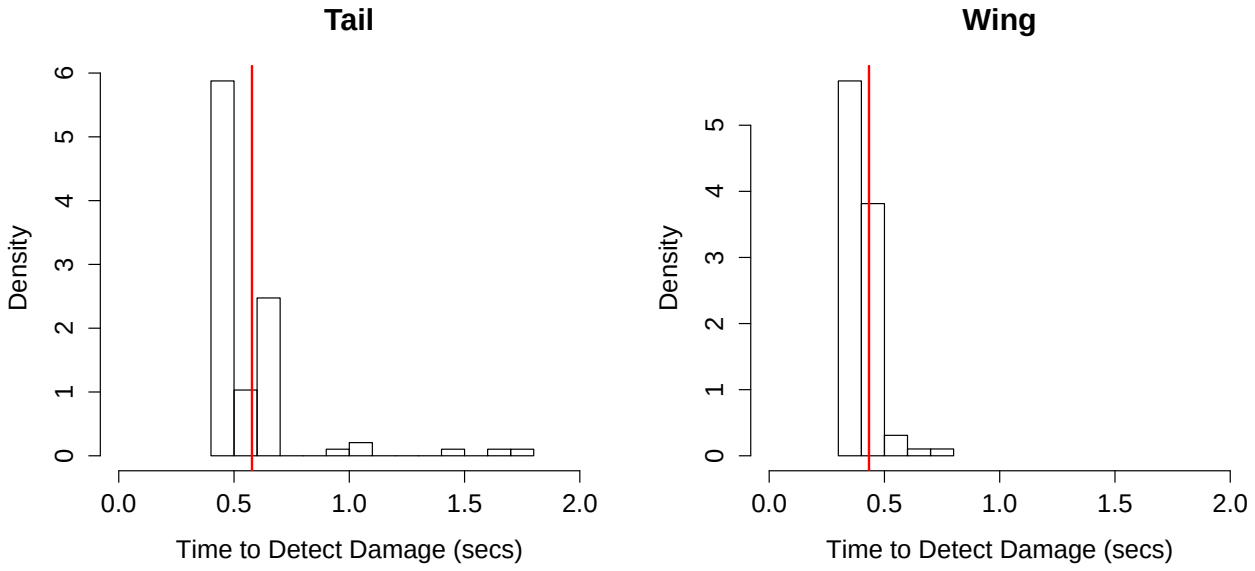

Figure 8: Distribution of time to detect Tail and Wing Damage respectively.

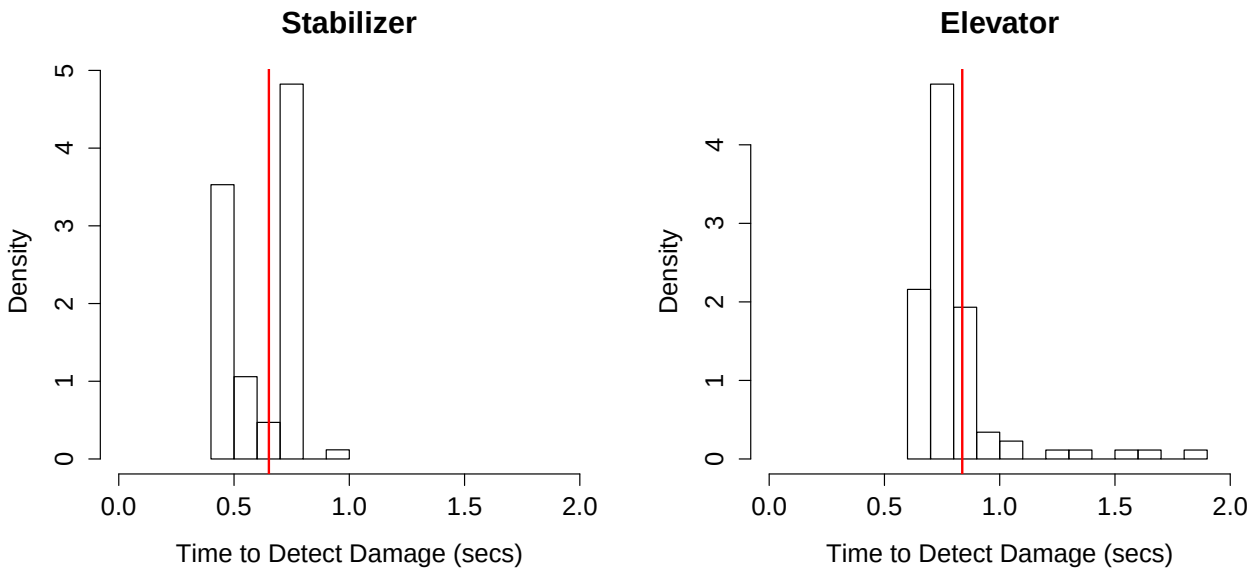

Figure 9: Distribution of time to detect Stabilizer and Elevator damage respectively.

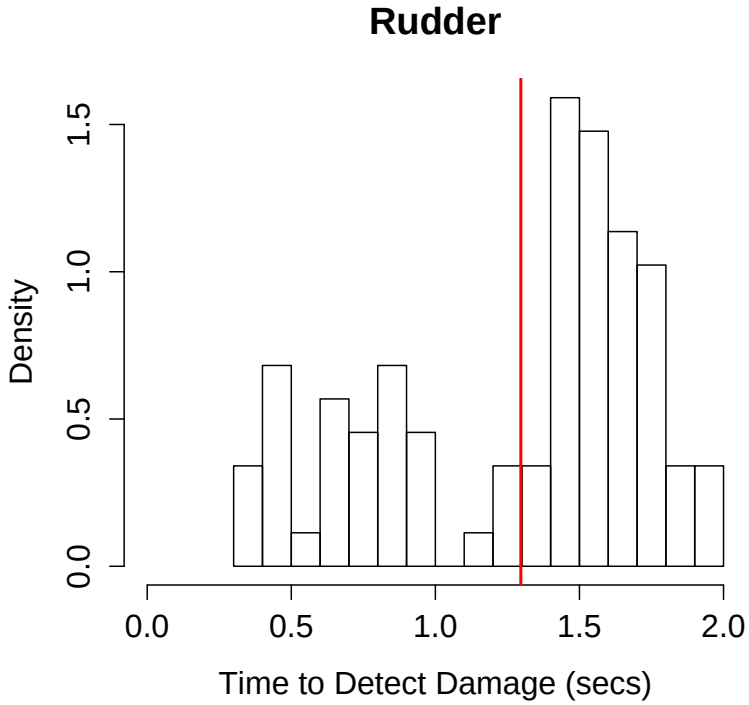

Figure 10: Distribution of time to detect Rudder Damage (Successful cases)

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 5 time steps (remember our optimal temporal threshold was 6 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 decision tree 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 (0.475) for 6 time steps (0.3 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 11, the distribution of these run times is entirely less than the chosen time step of 0.05 seconds with a mean of 0.0177 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.

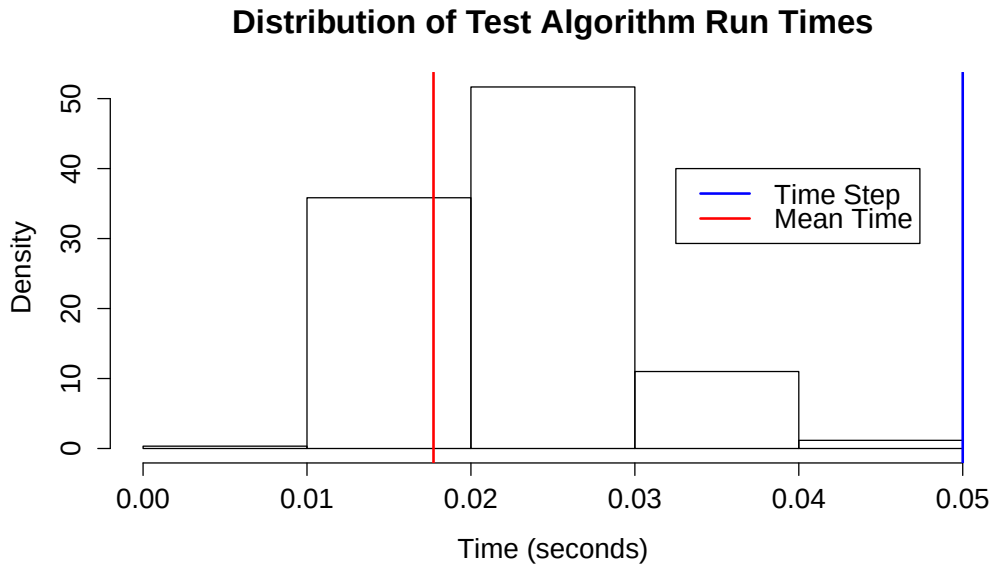

Figure 11: Distribution of Test Classification Run Time

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, classification trees, 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 89.5% predictive accuracy rate and, perhaps more significantly, routinely diagnoses the appropriate damage within 1 second of onset on independent test realizations. As expected, the damage scenario which on average requires the most time to detect involves the rudder. Even in a scenario where the flight trajectory's deviation from the planned path is minimal, the algorithm is able to detect rudder damage within 1.5 seconds.

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, one would believe in an ideal computing environment it could be much faster and thus a smaller time step could be selected.

While our methodology is simple to implement and the preliminary results promising, it is important to note that GTM simulator environment is SIMULINK and can be run from the MATLAB command window. When using the default time step, this leads to a large dataset and storing

it in an appropriate data structure for our methodology was tricky. It resulted in exporting the data into R and performing matrix "folding" to accommodate our methodological goals. While this process served no immediate role in the statistical learning process, is was necessary, time consuming and attentiveness was imperative.

We are optimistic about our results, yet believe there are plenty of ideas that remain to be explored. For example, we could have implemented the damage onset at a random time, rather than deterministically at 5 seconds or considered scenarios other than the 45 degree bank angle. As a result of the learning being done offline, it is possible to use an ensemble approach, although it would not be as efficient in the test phase. By consulting a subject matter expert, we could update the process we used to learn the probability and temporal thresholds by considering the priority between true positive and true negative rate.

References

[1] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone, *Classification and Regression Trees.* Wadsworth, 1984.

[2] Leo Breiman. *Random Forests.* Machine Learning, 45: 5-32, 2001.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman *The Elements of Statistical Learning.* Springer, 2001.

[4] Naira Hovakimyan and Chengyu Cao, *L1 Adaptive Control and its Transition to Practice*, IEEE Conference on Decision and Control, Orlando, FL, 2011.

[5] Richard M. Hueschen, *Development of the Transport Class Model (TCM) Aircraft Simulation From a Sub-Scale Generic Transport Model (GTM) Simulation.* NASA Technical Report, 2011.

[6] I.T. Jolliffe, *Principal Component Analysis.* Springer, 2002.

[7] Thomas Jordan, John Foster, Roger Bailey and Christine Belcastro, *AirSTAR: A UAV Platform for Flight Dynamics and Control System Testing*, NASA Technical Report, 2006.

[8] Thomas Jordan, William Langford, and Jeffrey Hill, *Airborne Subscale Transport Aircraft Research Testbed-Aircraft Model Development*, NASA Technical Report, 2005.

[9] Thomas Jordan, Wiliam Langford, Christine Belcastro, John Foster, Gautam Shah, Gregory Howland, and Reggie Kidd, *Development of a Dynamically Scaled Transport Model Testbed for Flight Research Experiments*, NASA Technical Report, 2004.

[10] B. D. Ripley, *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[11] Thomas J. Santner, Brian J. Williams, and William I. Notz, *The Design and Analysis of Computer Experiments.* Springer, 2003.