

Abstract

NOEL, NANA KATHERINE. Intelligent Event Detection in Aircraft Engines. (Under the guidance of Dr. Gregory Buckner).

Accurate fault detection in gas turbine engines is essential to moving from a calendar-based maintenance program to one that is need-based. Developing condition monitoring that is sensitive to true faults yet robust to false alarms can be a difficult task, but can result in tremendous cost savings and increased safety of an aircraft.

The purpose of this study is to develop an intelligent event detection algorithm for use in a turbofan aircraft engine. One current approach to event detection uses an embedded engine model with an inherent tracking filter, the output of which is then used to determine the fault condition present. This study uses the same tracking filter outputs to explore the possibility of using artificial neural networks for fault detection. A feedforward neural network was trained using backpropagation of error to predict faults in a variety of engine components for eight key points in the flight map; this network was able to determine faults with 100% accuracy using simulated engine data. However, expanding this network to more points in the flight map was problematic; thus, the ability of another type of network—a probabilistic neural network—to characterize faults was also explored. This trained PNN was able to classify engine faults for an expanded flight map (234 points) with an accuracy of 92.4%. This work showed that artificial neural networks can be effective tools for detecting and classifying engine faults.

Intelligent Event Detection in Aircraft Engines

by
Nana K. Noel

A thesis submitted to the Graduate Faculty of
North Carolina State University
In partial fulfillment of the
Requirements for the degree of
Master of Science

Mechanical Engineering

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Gregory Buckner
Committee Chair

Dr. Paul Ro

Dr. M.K. Ramasubramanian

Dedication

To my husband, Christian, for his support and encouragement.

Biography

After obtaining a B.A. in Biology from Warren Wilson College (Swannanoa, NC) in 1993, Nana worked as a Research Associate in the department of Pharmacology and Toxicology at Michigan State University. In 1997 Nana moved to Asheville, NC, where she took employment at a medical diagnostics laboratory. There she worked as a Chemistry Technician, a Research Assistant, and as Project Development Manager. It was while working in the laboratory that Nana became interested in returning to school for a degree in Mechanical Engineering. Nana's husband and three children moved to Durham, NC in 2004 so that she could attend North Carolina State University to complete her B.S.M.E (August 2006) and her M.S.M.E (May 2008).

Acknowledgements

I would like to thank my husband for his love and support, and my children for enduring my return to school.

A special thank you is extended to Dr. Gregory Buckner for his oversight, time, support and patience. Additionally, I would like to thank Dr. Paul Ro and Dr. Ramasubramanian for serving on my committee.

This work was funded by GE Aviation, Cincinnati, Ohio. I would like to express gratitude to Dr. Nathan Gibson of GE Aviation for his time and input into this project.

Also, I would like to thank Dr. Gary Bishop of the University of North Carolina – Chapel Hill for the time he spent with me discussing Kalman filters.

Table of Contents

List of Tables	vi
List of Figures	vii
1 Introduction	1
1.1 Engine Overview.....	2
2 Materials and Methods.....	3
2.1 Materials Provided by General Electric	4
2.2 Fault Detection Methods	8
2.2.1 An Industry Approach to Fault Detection.....	8
2.2.2 Artificial Neural Networks for Fault Detection	12
2.2.2.1 ANN Training Using Eight Flight Map Points	12
2.2.2.1.1 ANN Using Eight Flight Map Points: Results.....	15
2.2.2.2 Cascaded PNNs for Full Flight Map	18
2.2.2.2.1 PNNs for Full Flight Map: Results.....	24
2.2.3 Industry Approach vs. Neural Network Approach.....	25
3 Discussion.....	26
4 Bibliography	27
Appendices	29
Appendix A: Background Preparation for GE Studies	30
Appendix B: Matlab Code Used to Auto-Simulate the CLM.....	38
Appendix C: Matlab Code Used to Generate/Train the ANN.....	44
Appendix D: Matlab Code Used to Generate the PNNs.....	54

List of Tables

Table 1: Possible engine operating conditions and associated ANN targets	15
Table 2: Possible engine operating conditions and associated PNN targets	23

List of Figures

Figure 1: Schematic diagram of a turbofan engine [4]	2
Figure 2: Flight map with 8 points of interest shown.....	5
Figure 3: CLM/TF configuration [7]	6
Figure 4: Typical engine station diagram [7].....	7
Figure 5: Simulink portion of a possible approach to fault detection.....	9
Figure 6: Event signature example.....	9
Figure 7: Schematic of an online event detection strategy.....	11
Figure 8: Feedforward ANN architecture	13
Figure 9: Training performance of a feedforward ANN using 8 flight map points.....	16
Figure 10: Output of trained feedforward ANN (8 flight map points).....	17
Figure 11: Expanded flight map (234 points)	18
Figure 12: Architecture of a PNN	20
Figure 13: Matlab's PNN architecture [10].....	22

1 Introduction

The primary objective of condition monitoring and fault detection is to identify the need for maintenance. With respect to aircraft gas turbine engines, accurate fault detection and diagnosis is critical to passenger safety as well as being important to reducing operating costs. The Department of Defense (DoD) currently spends ~\$4.2B/yr for sustaining the turbine engines of its aircraft fleets. Engine components are prematurely retired and unnecessary maintenance is performed due to sub-optimal engine diagnostic methods. One current industry method for determining maintenance scheduling uses a trending and diagnostic system to diagnose faults in Line Replaceable Units (LRUs). This trending and diagnostic system uses only takeoff data to set fault code bits according to prescribed fault detection logic. Maintenance actions are taken post-flight that are appropriate to the indicated fault. In the current configuration, there is no way of specifying whether the fault is valid and persistent. In fact, this method has a 90% rate of false positive fault diagnoses, which translates to a high rate of unnecessary LRU removals that occur in today's fleet [1]. The goal of this research is to explore intelligent and novel ways to detect and diagnose system anomalies in a gas turbine engine, which, if implemented, would lead to a reduction in DoD spending on parts and maintenance.

One challenge to the development of effective fault detection strategies is the need for sensitivity to faults without triggering false alarms. Model-based approaches frequently suffer from model uncertainties that can lead to the misidentification of faults. Brotherton, et al. addressed this potential problem in aircraft gas turbine engine fault detection by fusing an engine model with an empirical neural network model [2]. Noel, et al. developed and experimentally

demonstrated a similar fault detection strategy for an active magnetic bearing system by augmenting a model-based Kalman filter approach with an artificial neural network [3]. A copy of this paper is presented Appendix A.

1.1 Engine Overview

The basic components of a turbofan engine are illustrated in Figure 1. Specifically, this diagram shows a two-spool, low-bypass turbofan engine. Connected to the low-pressure spool (green) are the fan and low-pressure turbine; the high-pressure compressor is powered by the high-pressure turbine (purple spool).

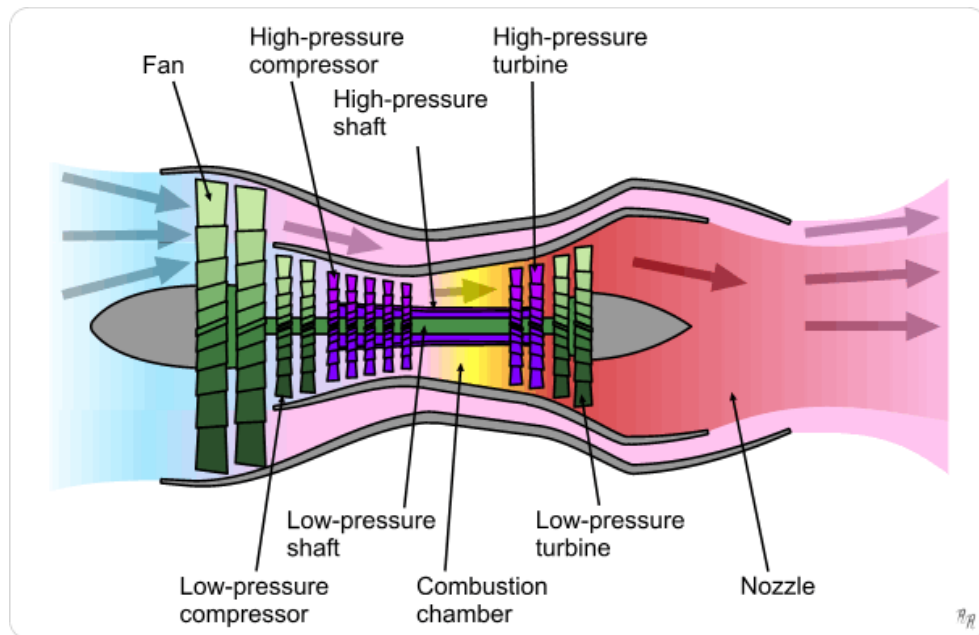


Figure 1: Schematic diagram of a turbofan engine [4]

At the front of the engine is a large fan, which draws air into the engine. Some of the air travels through the center (core) of the engine, while the remaining air bypasses the core. The first core component of the engine is the compressor, which increases the potential energy of the air by compressing it. After exiting the compressor, the air is mixed with fuel and ignited in the combustor. This high-energy airflow then enters the turbine, whose function is to drive the fan and compressor. Finally, at the nozzle, the hot exhaust air from the turbine is combined with the cool air that bypassed the core to produce the thrust that propels the aircraft.

Due to the extreme heat and operating conditions within the engine core, the gas path components, and thus the performance of the engine, deteriorate over time. As it deteriorates, the declined thermodynamic performance of the engine can be measured by component flow capacities and efficiencies, which in turn produce changes in sensed engine parameters such as temperatures, pressures, fuel flow rates, and rotational speeds. Degraded performance reflected by these sensed values can be used to detect faults [5].

2 Materials and Methods

There are two main approaches to fault detection and diagnosis for aircraft engines: model-based and data-driven [6]. The model-based approach compares the outputs of an engine model to those of the real engine. The difference, or residual, between these compared outputs is then used in conjunction with a decision logic system to make a diagnosis. Data-driven fault detection and diagnosis methods, such as artificial neural networks, use historical engine data to determine

fault status.

One approach that offers an improvement to the current trending and diagnostic method is model-based. A component-level model (CLM) of the engine is embedded in the engine monitoring unit (EMU) and uses actual engine inputs to simulate the engine's dynamics. The CLM has an inherent Kalman tracking filter that reduces sensor residuals using adjustable "quality parameters". A pattern-recognition algorithm then uses these tracked parameters to detect gas path anomalies across the flight map.

This research will investigate replacing a model-based approach with a data-driven method for fault detection and diagnosis.

2.1 Materials Provided by General Electric

A CLM of a commercial, two-spool gas turbine engine was provided by General Electric Aviation (GE, Cincinnati, OH). This model was designed to run in the Simulink® (The MathWorks Inc., Natick, MA) computing environment. In order to simulate the engine under both normal and faulty operating conditions, GE supplied fault models for common engine anomalies that could be "injected" into the nominal CLM. Additionally, an assortment of engine test vectors was provided to simulate flights across the flight map. The flight map, also known as the flight envelope, represents the Mach number and altitude under which an aircraft is aerodynamically stable. Figure 2 shows initial points of interest on a flight map: sea-level static

takeoff, standard day takeoff, hot day takeoff, high altitude takeoff, climb, cruise, high-altitude cruise, and descent.

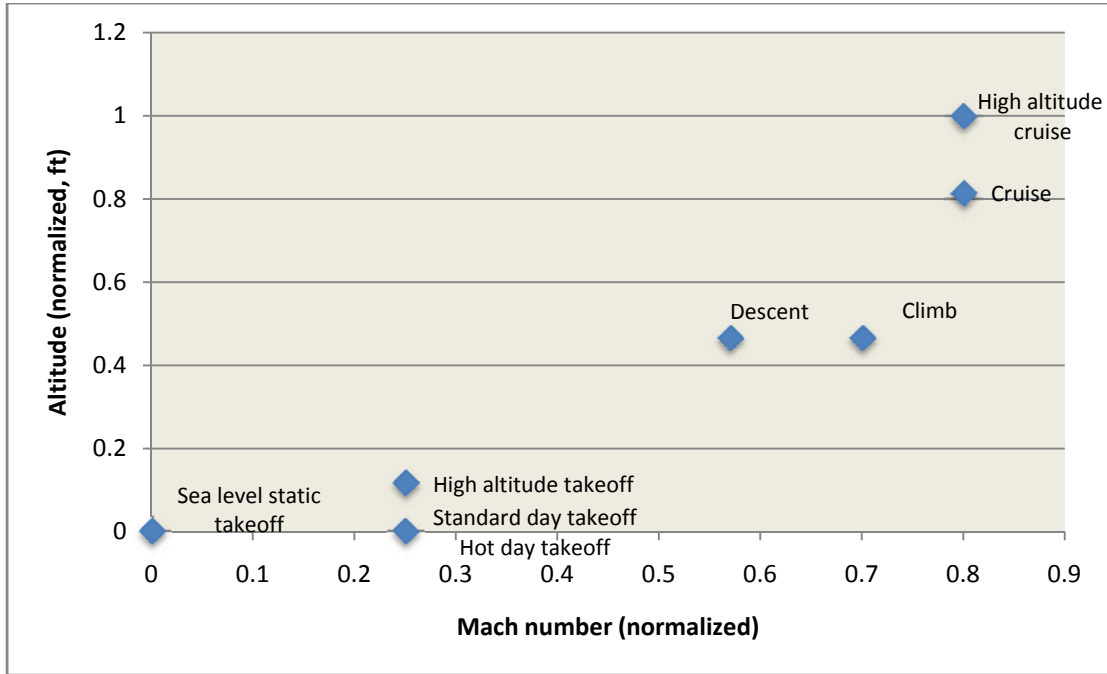


Figure 2: Flight map with 8 points of interest shown

The initial test vectors supplied by GE included engine input parameters needed to run the CLM at these eight unique flight map points. After initializing the CLM at a desired flight map point, it is possible to run the engine nominally or with injected faults. The simulated fault events include anomalies in the fan, booster, high pressure compressor (HPC), combustor, high pressure turbine (HPT), low pressure turbine (LPT), as well as an “unknown event”. These injected faults can be specified as being “small”, “medium” or “large” in magnitude; this is done by scaling the flow and efficiency values for the engine component of interest.

The CLM is configured in parallel with a non-extended, constant-gain Kalman filter (hereafter referred to as the tracking filter), which provides event detection parameters for gas path anomalies (Figure 3).

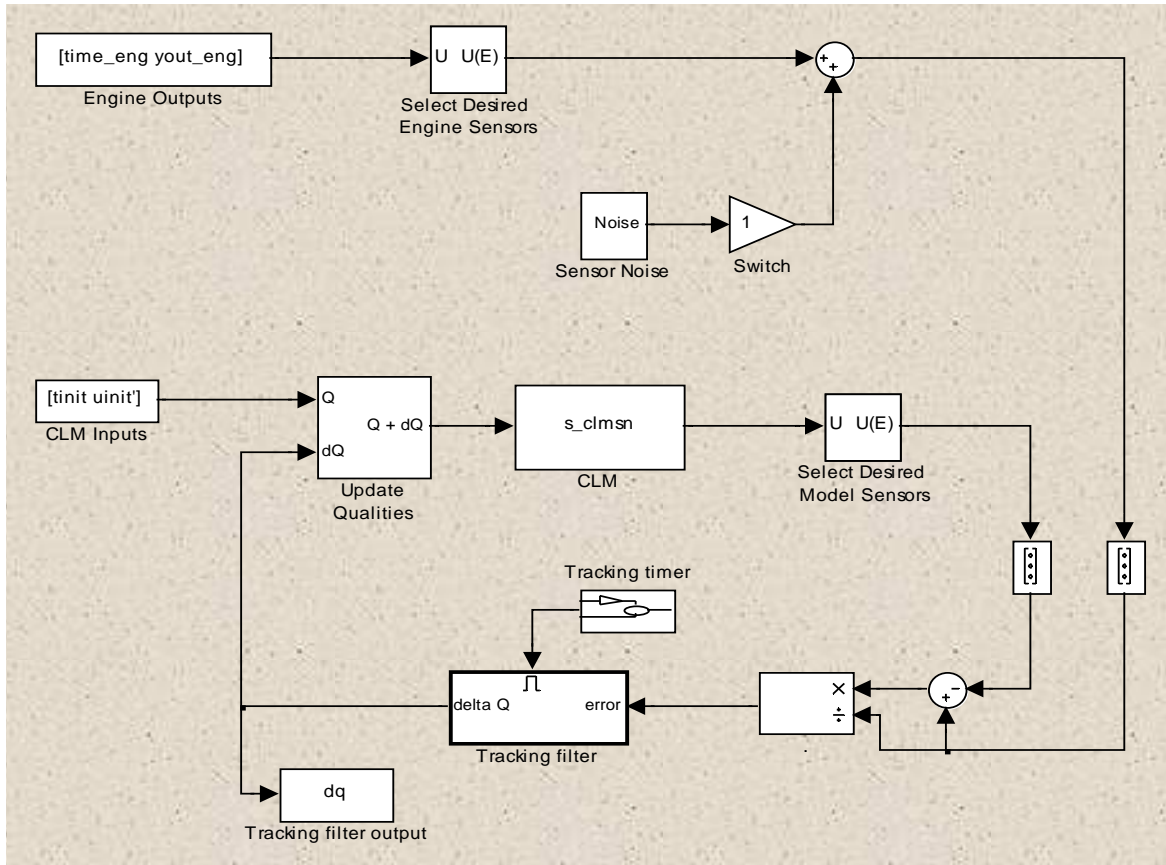


Figure 3: CLM/TF configuration [7]

The CLM of figure 3 is simulated using 63 engine inputs (temperatures, pressures, rotational speeds, valve settings, flows, efficiencies, etc.). The outputs from the real engine are subtracted from those of the CLM; these residuals are then input into the tracking filter to obtain updates to the engine quality parameters. These ten quality parameters are of interest for fault detection:

they are flows and efficiencies for the fan, booster, HPC, HPT and LPT. Not all of these parameters are measured, however. In all, there are seven sensors taking data from the air stream within the engine core (Figure 4). These sensors measure fan speed (N1), core speed (N2), temperature and pressure at the HPC inlet (T25 and P25, respectively), temperature and pressure at the HPC outlet/combustor inlet (T3 and PS3, respectively), and temperature at the LPT inlet (T49). The number in the sensor name corresponds to the placement of the sensor within the engine, i.e. the engine station.

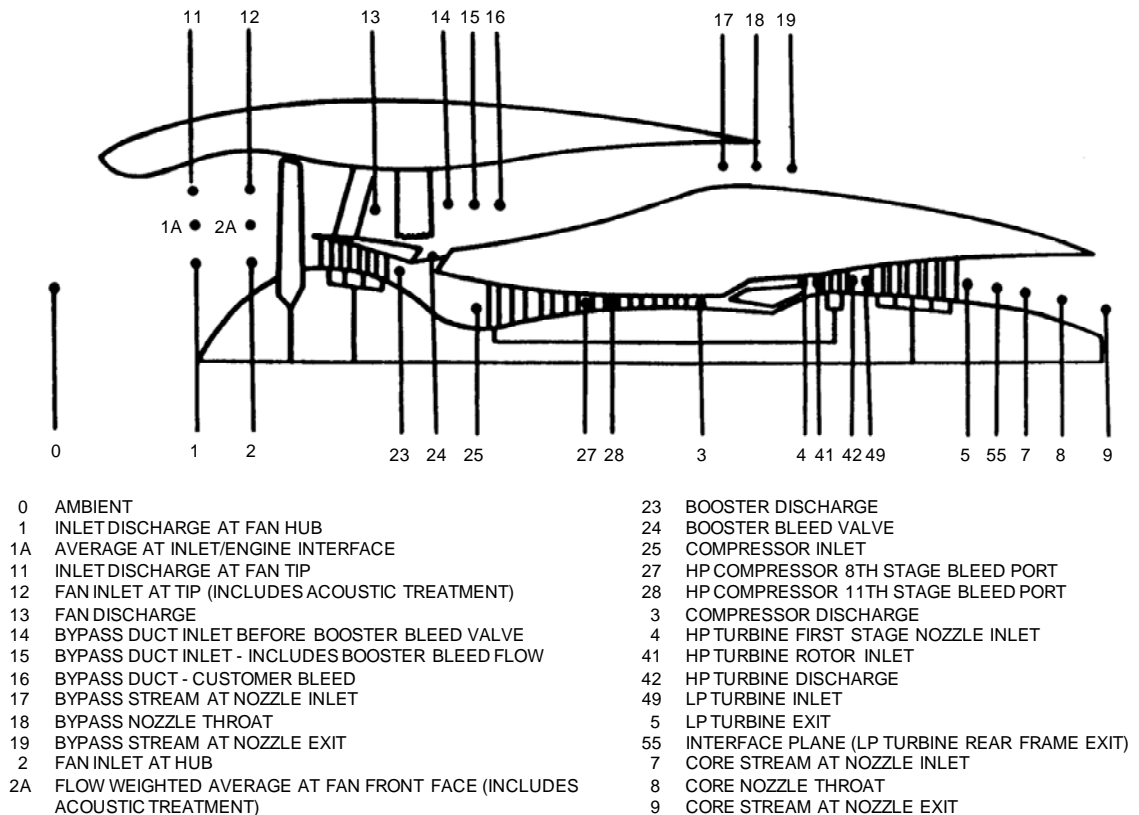


Figure 4: Typical engine station diagram [7]

To address the problem of having an underdetermined system (ten quality parameters but only seven sensors), quality parameters were lumped or eliminated based on both linear dependence between parameters and relatively low observability. The result is a linearly independent set of six quality parameters (ZSWR12, ZS2, ZSWR25, ZSED25, ZSWR41, ZS4) that are mathematically rather than physically meaningful; these are the quality parameter outputs of the tracking filter. The naming convention for these quality parameters is as follows: the “ZS” indicates a tracking filter parameter, “WR” indicates flow, “ED” indicates efficiency, and the number at the end of the parameter name indicates the engine station. Taken together, these six tracking filter parameters form a “signature” that can be used for fault detection and diagnosis.

2.2 Fault Detection Methods

2.2.1 An Industry Approach to Fault Detection

One possible approach to fault detection is illustrated in Figure 5. Here, a k-nearest neighbor method [8] is used to select the appropriate flight phase for given engine input information. Once this flight phase is determined, gas path “signatures” are selected that correspond to “no fault”, an “unknown fault”, and “small”, “medium” and “large” faults in the fan, booster, HPC, HPT and LPT.

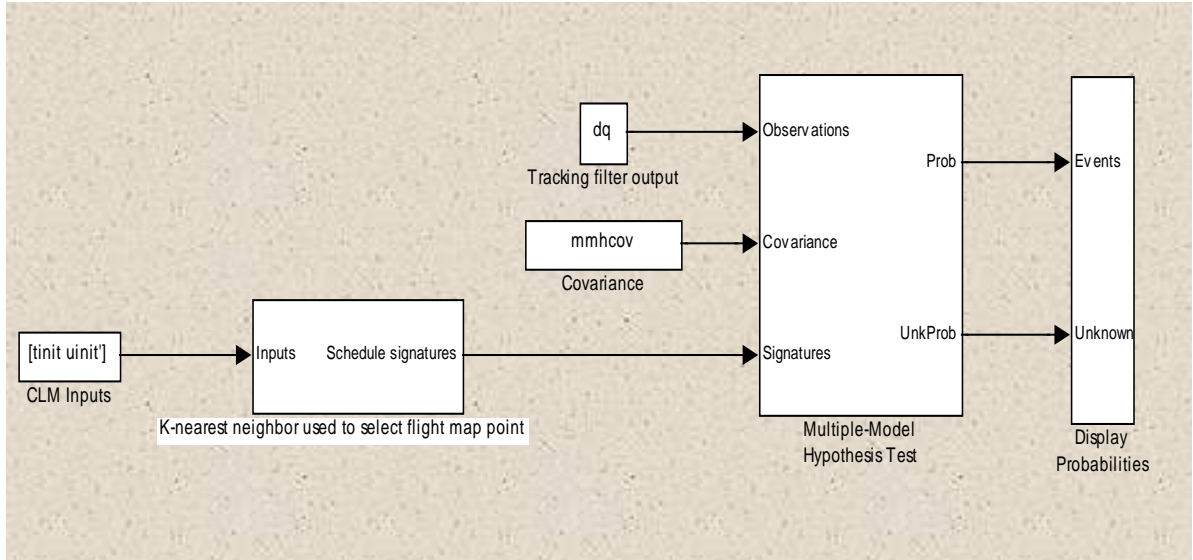
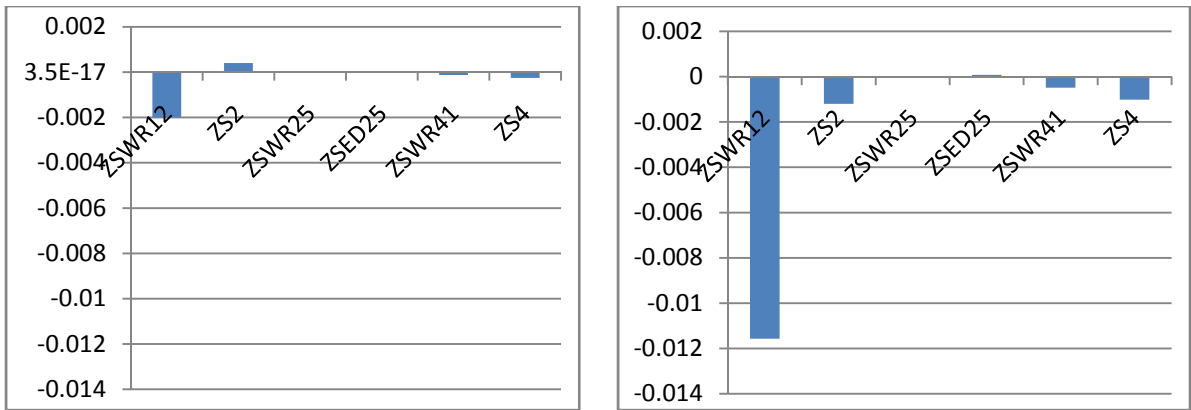


Figure 5: Simulink portion of a possible approach to fault detection

Figure 6 compares event signatures for a “small” and “large” fan anomaly for a specific point in the flight map.



(a) Small fan anomaly

(b) Large fan anomaly

Figure 6: Event signature example

Although the change in ZSWR12 is more significant than the other five quality parameters as this fan fault transitions from “small” to “large”, it is incorrect to assume that this parameter is uniquely associated with this type of fault. In fact, it is not possible to extract physical meaning from any of the six quality parameters; an LPT fault may also greatly affect the ZSWR12 parameter, for instance. What is unique to each fault type is the relative magnitudes of all six quality parameters for given flight map coordinates.

In the example shown in figure 5, a multiple model hypothesis (MMH) test is then used to determine probabilities for the event type and severity. These probabilities are then “fused” with the current trending and diagnostic system fault interpretations to strengthen confidence in fault diagnoses. Figure 7 shows a generalized schematic of this approach to online event detection.

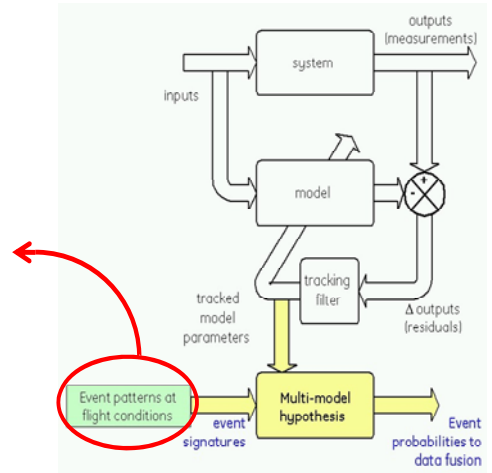
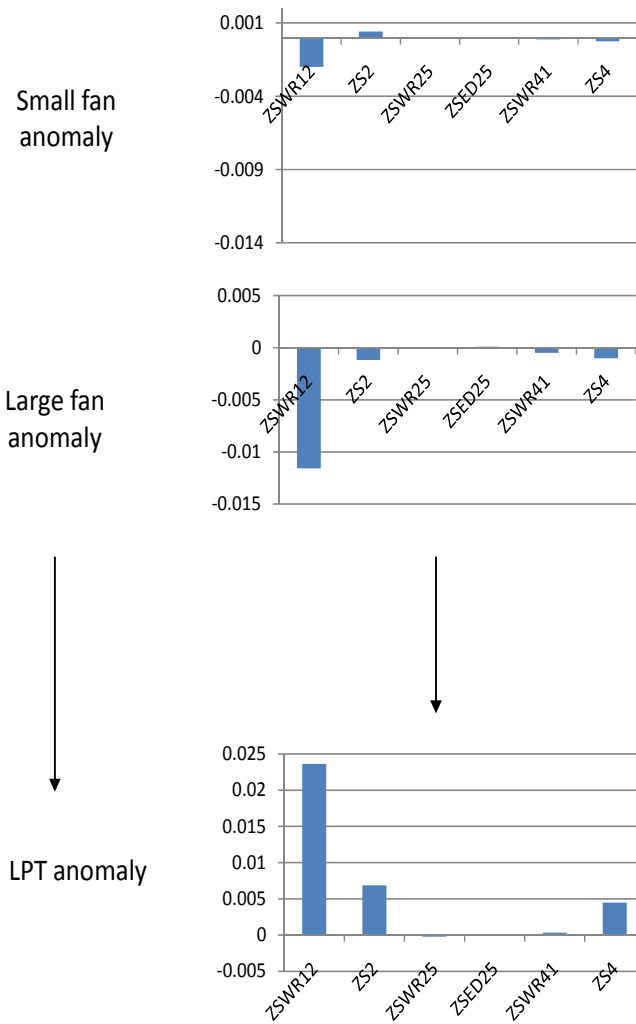


Figure 7: Schematic of an online event detection strategy

While tracking the six quality parameter updates and using a nearest neighbor/multi-model hypothesis approach to supplement the current trending and diagnostic system is an improvement to using the current system alone, the use of only eight flight phases for event

scheduling remains sub-optimal. A more robust and reliable strategy, particularly for fighter aircraft, would define faults using a less generalized and more detailed flight map.

2.2.2 Artificial Neural Networks for Fault Detection

One alternative to the described industry approach to detecting and diagnosing faults is data-driven methods. For example, faults can be detected and diagnosed using artificial neural networks (ANNs). ANNs are highly interconnected processing elements that function as adaptive, non-linear statistical data modeling tools. Artificial analogs of the human brain, they are capable of accurately predicting outputs by “learning” the input/output relationships of complex systems.

2.2.2.1 ANN Training Using Eight Flight Map Points

To examine the feasibility of ANN-based fault detection, a feedforward network was designed and trained using MATLAB’s Neural Network Toolbox. A schematic of this network architecture, frequently used for function approximation and signal classification, is presented in Figure 8.

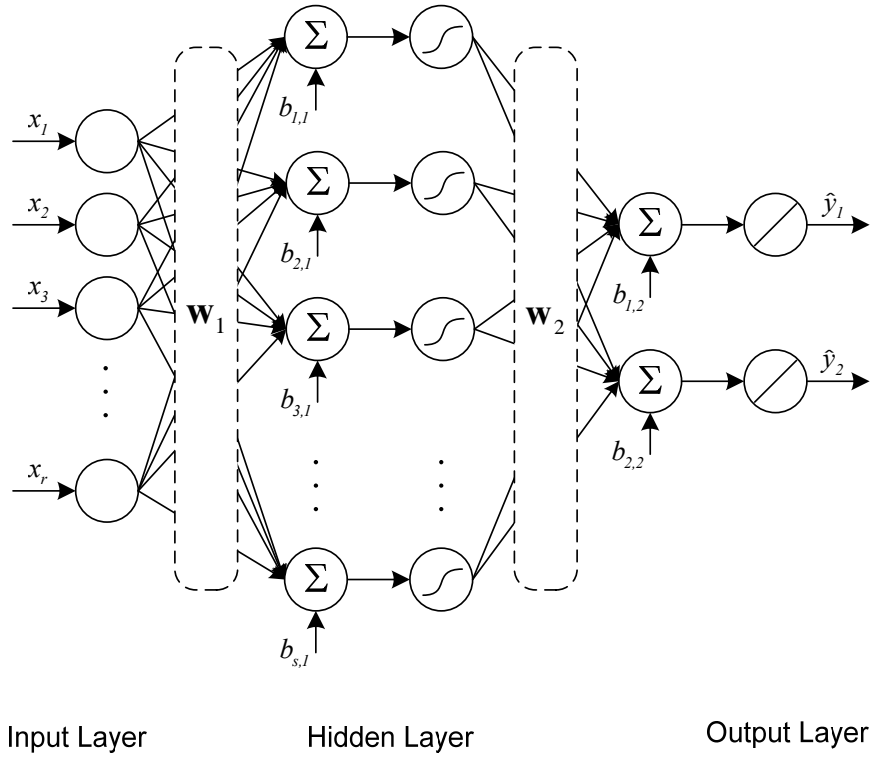


Figure 8: Feedforward ANN architecture

The input vector $\mathbf{x} = [x_1, x_2, x_3, \dots, x_r]^T$ is weighted (by an adaptable vector \mathbf{w}_1) and propagated to neurons in the hidden layer, which consist of biased sigmoidal activation functions ($\mathbf{a} = \frac{2}{1 + e^{-w_1^T \mathbf{x} + b_1}} - 1$), where \mathbf{b}_1 is a hidden layer bias vector. The output vector $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2]^T$ is a weighted and biased sum of the hidden layer outputs ($\hat{\mathbf{y}} = \mathbf{w}_2^T \mathbf{a} + \mathbf{b}_2$)[9]. The network weights and biases ($\mathbf{w}_1, \mathbf{w}_2, \mathbf{b}_1, \mathbf{b}_2$) are adapted using supervised learning, where an

input is presented to the network and a corresponding target output is known. The error between the actual and target network outputs is fed back to the system to adjust the system parameters until the network “learns” the input/output relationship (i.e. prediction errors are minimized).

The most common learning strategy, backpropagation of error, uses input/output training data to minimize an error cost function. Input data is presented to the network and the resulting output ($\hat{\mathbf{y}}_i$) is compared to the target output (\mathbf{y}_i), resulting in a prediction error $\mathbf{E}_i = \mathbf{y}_i - \hat{\mathbf{y}}_i$. A quadratic error cost function is defined according to $\mathbf{J}(\mathbf{E}) = \sum \mathbf{E}_i^2$; it is the goal of backpropagation to minimize this cost for a large number of training pairs. This is achieved iteratively by updating the network’s weight and bias vectors. With each iteration, the network weights are updated according to $\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \frac{\partial \mathbf{J}(\mathbf{E})}{\partial \mathbf{w}}$, where η is the learning rate.

For this specific application, a feedforward ANN with 40 hidden layer neurons was created using MATLAB’s Neural Net Toolbox. Input/output training data were generated by simulating the CLM 160 times: eight flight map points simulated at 20 operating conditions. The eight flight map test points of interest (Figure 2) include sea-level static takeoff, standard day takeoff, high altitude takeoff, hot day takeoff, climb, cruise, high altitude cruise, and descent, all at maximum power. For each of these eight flight conditions there are eight possible events. For the purpose of ANN training, each event was assigned a target output type (numbered 1-8) and a secondary output to indicate fault severity (numbered 1, 2 or 3 for small, medium, and large, respectively), as shown in Table 1.

Table 1: Possible engine operating conditions and associated ANN targets

<i>Anomaly</i>	<i>Designated ANN Fault Targets</i>	<i>Designated ANN Fault Severity Targets</i>
Fan anomaly	1	1, 2, or 3
Booster anomaly	2	1, 2, or 3
HPC anomaly	3	1, 2, or 3
Combustor anomaly	4	1, 2, or 3
HPT anomaly	5	1, 2, or 3
LPT anomaly	6	1, 2, or 3
No event	7	0
Unknown event	8	1

From each 60-second simulation, model inputs and resulting quality parameters were stored for subsequent use in neural network training. Because only seven of the 63 CLM inputs are unique between data sets, the ANN input vector consisted of these seven unique CLM inputs and the corresponding six quality parameter outputs. The ANN target output vector consisted of fault type (1-8) and fault severity (1-3). Prior to training, the ANN input vector was normalized, and input/output data was randomly divided into a training set (60% of the data), a validation set (20% of the data), and a testing set (20% of the data).

Network training was conducted for 100 epochs using Matlab’s ‘trainlm’ function, which updates the weight and bias vectors using Levenberg-Marquardt optimization.

2.2.2.1.1 ANN Using Eight Flight Map Points: Results

The error cost functions associated with the training, validation and testing sets are

presented in Figure 9. This figure reveals that the cost functions (labeled “performance” in the figure) are continually reduced for each training iteration (or “epoch”). After 100 epochs, all three performance indices are below 0.01, indicating successful training without loss of generalization capabilities.

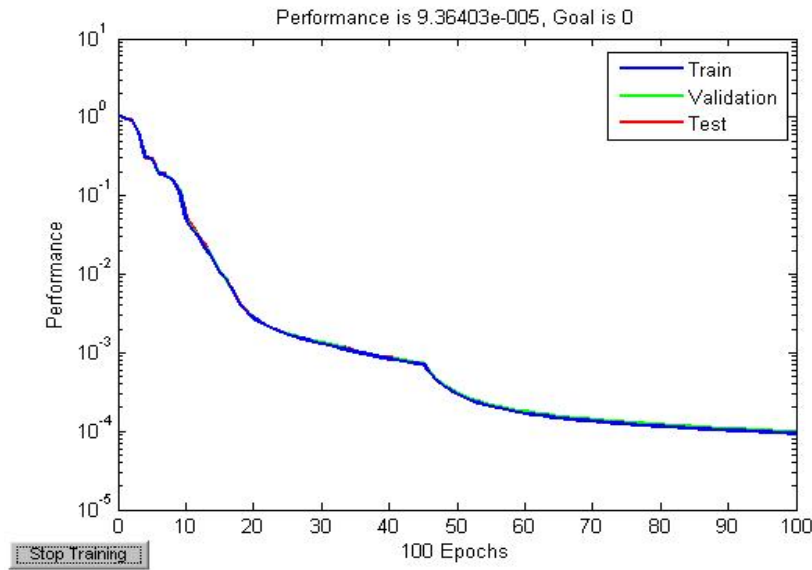


Figure 9: Training performance of a feedforward ANN using 8 flight map points

While simulating the engine and observing resulting multi-model hypothesis fault classifications as well as ANN fault classifications, it was observed that booster faults were consistently misidentified.

With the exception of booster faults, the trained feedforward ANN predicted both the type and severity of all faults with an accuracy of 100%. After discussion with GE, booster faults were omitted from further consideration, reducing the number of fault cases at each operating

point from 20 to 17. The accuracy of ANN fault classification is shown graphically in Figure 10. To generate this figure, the network was simulated using testing data (information not used for network training). The normalized network outputs (fault type and severity) are represented in blue (left column); the “actual” or “target” outputs are shown in green (left column). The non-normalized interpretations of fault type and severity are presented in the right column. Clearly, the predicted fault outputs (blue) are perfectly matched to the target values (green).

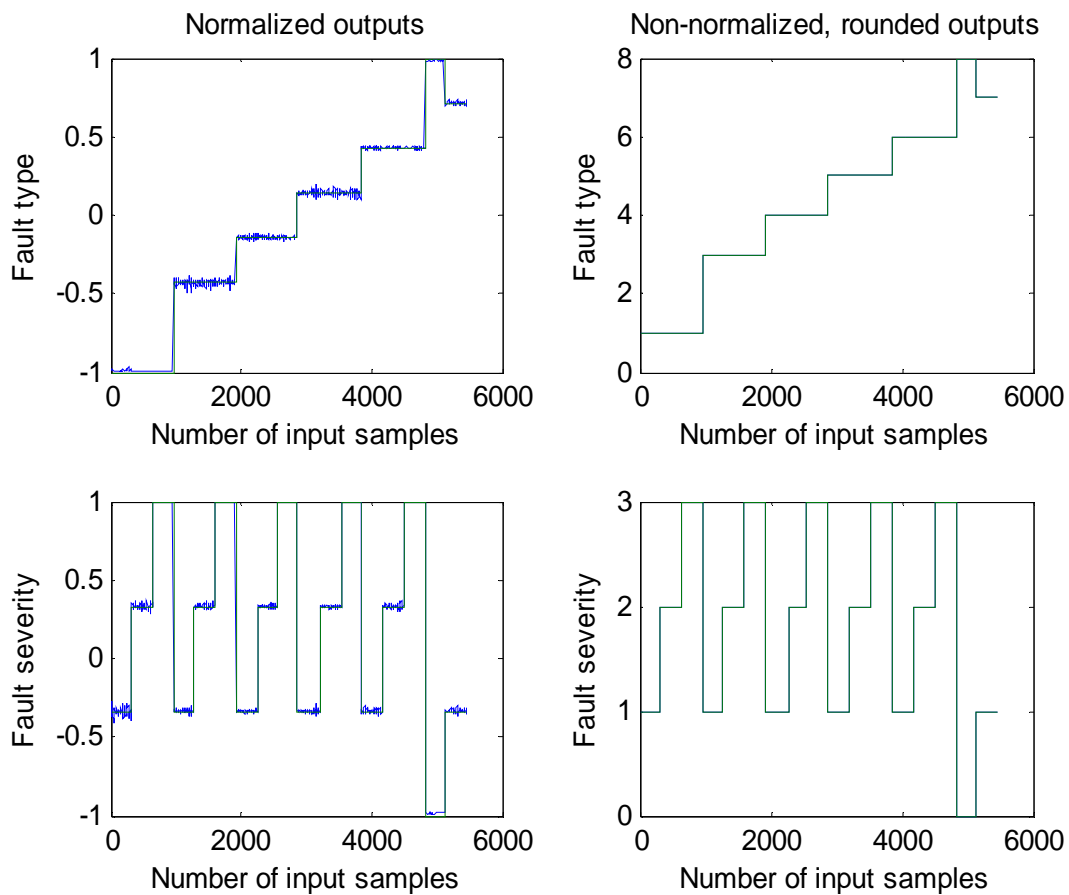


Figure 10: Output of trained feedforward ANN (8 flight map points)

2.2.2.2 Cascaded PNNs for Full Flight Map

The initial test vectors supplied by GE included engine input parameters needed to run the CLM at only eight flight map points. Representing the entire flight map with so few points, however, unrealistically simplified the actual flight conditions; any neural network trained using data from only these points will lack the ability to generalize its fault detection capabilities across the entire flight map. Though the feedforward network performed well when identifying faults associated with these eight flight map points, a more sophisticated network is necessary for expanded flight maps. To expand the fault detection strategy to a more realistic flight map, GE supplied engine input parameters to run the CLM at 234 flight map points (Figure 11). Nine data sets were provided for each of the 26 Mach/Altitude points shown in the figure; these included three engine power settings (idle, intermediate and maximum) as well as three different ambient temperature conditions.

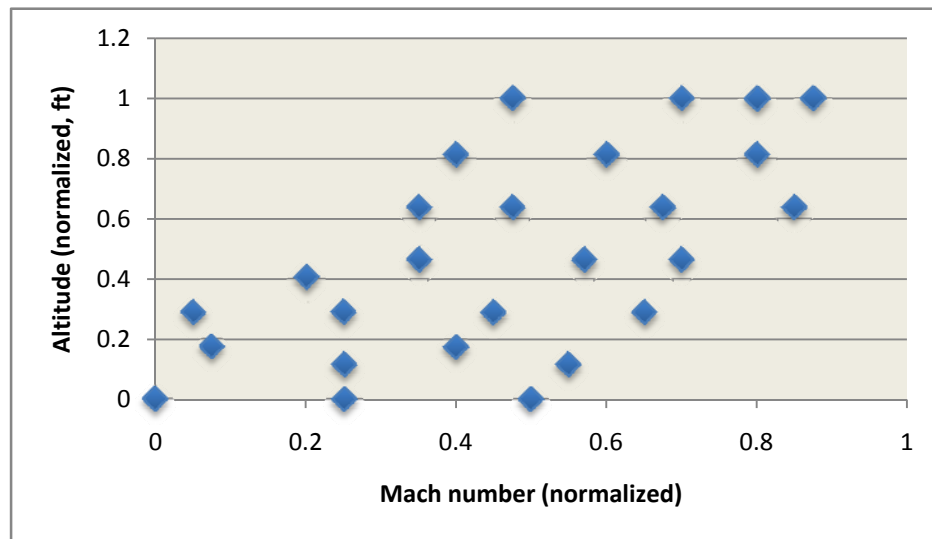


Figure 11: Expanded flight map (234 points)

Computational issues prevented expanding the feedforward network of figure 8 to the expanded flight map of figure 11; the number of inputs to the network increased from over 27,000 to nearly 800,000 when shifting from a network trained using data from the original flight map to the expanded flight map. For this reason, a different network architecture was utilized: cascaded probabilistic neural networks (PNNs). PNNs have several advantages over other feedforward network architectures: most notably they train in up to five orders of magnitude faster and can be easily retrained if data is added or deleted from the training set. With enough training, PNNs are guaranteed to converge to Bayesian classifiers, the accepted definition of optimality [11, 12]. PNNs are capable of learning complex input/output relationships and have the ability to generalize, that is to say, classifications will be correct for inputs that are reasonably similar to the training inputs.

Developed by Donald Specht in the 1960's [13], and later refined in 1990, PNNs are conceptually similar to k-Nearest Neighbor (k-NN) models and are very well suited to categorical classification problems. Derived from Bayesian classifiers – the standard against which all other classification methods are evaluated - PNNs are feedforward networks that learn to approximate the probability density functions of a given training set. The basic idea behind PNNs is that the target value predicted for a given input is likely to be the same as that of other inputs that have predictor variables with similar values. The schematic for a PNN is presented in Figure 12 [11, 12, 13].

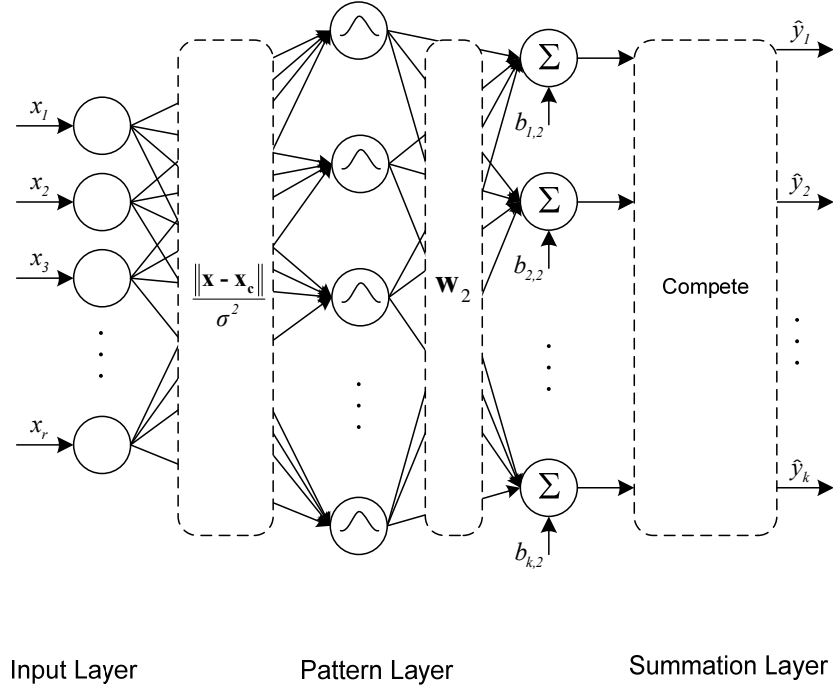


Figure 12: Architecture of a PNN

The input layer consists of one neuron for each predictor variable. After normalizing the input vectors, the input neurons propagate values to neurons in the hidden (or pattern) layer. The pattern layer has one neuron for each case in the training set. Neurons in the pattern layer compute the Euclidean distance of the input from the neuron centers: $\|\mathbf{x}_c - \mathbf{x}\|$. These norms are then passed through activation functions (RBF kernel functions): $\mathbf{a} = \mathbf{w}_2^T e^{-\frac{\|\mathbf{x}-\mathbf{x}_c\|}{\sigma^2}}$, where σ^2 acts as a “spread” or “smoothing parameter” [11, 12, 13, 14]. In general, if the spread is near zero, the network will act as a nearest neighbor classifier, but if the spread is too small the network will not generalize well. As the spread becomes larger the network will take into account more

nearby design vectors, but too large of a spread results in the loss of detail [15].

The third layer - the competitive or summation layer - contains one neuron for each target category. Stored within each hidden neuron is the actual target category of each training case; the weighted output of a hidden neuron is fed only to the summation neuron that corresponds to the hidden neuron's category. The summation neurons add the values for the class they represent, thus they give a weighted vote for that category. At the output, the classification decision is made by comparing the weighted sums from the summation layer for each target category.

As mentioned, PNNs train very quickly as training is completed in one presentation of each training vector. One disadvantage, however, is network size. Because the entire training set is contained within the network, it is slow to execute and is memory-intensive.

For this study, an initial PNN was trained to determine the nearest flight map point ($n=234$). Once the nearest flight map point was identified, a secondary PNN (one trained for each of the 234 flight map points) was used to determine the fault.

Matlab's 'newpnn' function was used to create the networks. The output of the 'newpnn' function is defined by Matlab as a two-layer network, where the previously-described pattern layer is referred to as the 'radial basis layer', and the summation and output layers are combined and termed as the 'competitive layer'. Matlab's architecture for this network is shown in Figure 13.

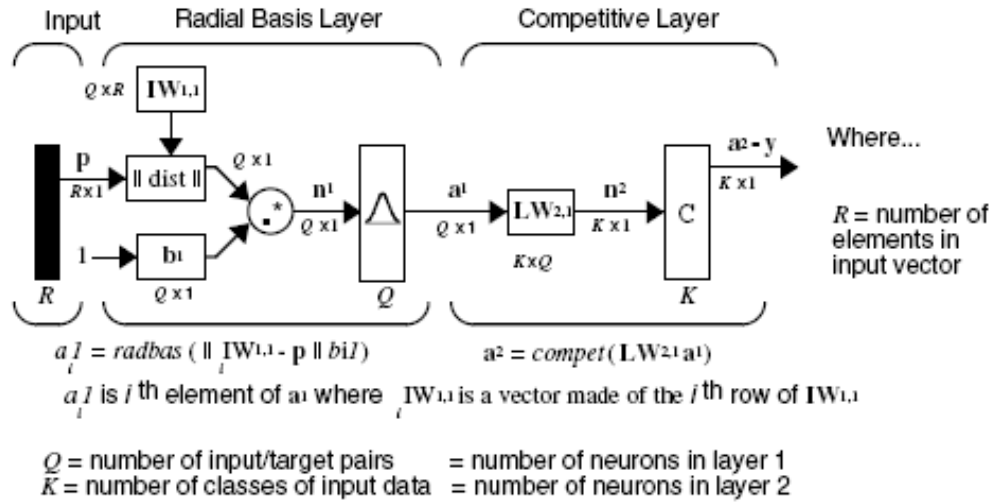


Figure 13: Matlab's PNN architecture [10]

Matlab's implementation replaces the neuron center vector \mathbf{x}_c with weight vector \mathbf{w}_1 , and replaces the spread with a bias vector \mathbf{b}_1 . The resulting output of the radial basis layer is thus:

$\mathbf{a} = \mathbf{w}_2^T e^{-\|\mathbf{x}-\mathbf{w}_1\| \cdot \mathbf{b}_1}$. A *compete* transfer function on the output of the second, or competitive layer, picks the maximum of the summed probabilities, resulting in a 1 for that class and a 0 for the other classes.

Previously, 7 of the 63 inputs to the CLM were unique when only 8 flight map points were considered. With the expanded flight map, the number of unique inputs to the CLM increased to nine. The inputs to the primary, flight map identification PNN were these nine unique engine inputs; the output of this first network was the identified flight map point (1-234). Thus, the first layer contained nine radial basis neurons and the second layer contained 234 neurons, all with an assigned spread of 0.7.

After the flight map point was identified by this first network, one of 234 secondary probabilistic neural nets was then used to determine the fault. Instead of training these networks to output fault type and severity as two separate outputs, the outputs were designated as shown in Table 2.

Table 2: Possible engine operating conditions and associated PNN targets

<i>Anomaly</i>	<i>Designated PNN Targets</i>
Small fan anomaly	1
Medium fan anomaly	2
Large fan anomaly	3
Small booster anomaly	4
Medium booster anomaly	5
Large booster anomaly	6
Small HPC anomaly	7
Medium HPC anomaly	8
Large HPC anomaly	9
Small combustor anomaly	10
Medium combustor anomaly	11
Large combustor anomaly	12
Small HPT anomaly	13
Medium HPT anomaly	14
Large HPT anomaly	15
Small LPT anomaly	16
Medium LPT anomaly	17
Large LPT anomaly	18
No event	19
Unknown event	20

For these 234 networks, the average values from the multiple points collected for each of the six tracking filter outputs (quality parameter updates) were used as training inputs, thus the first layer for each PNN consisted of six radial basis functions. Each of the 234 networks was trained using quality parameter data from simulations for that flight map point only. The second layer

for each of these 234 networks initially had 20 neurons; as mentioned, booster anomalies were omitted from the study after discussion with GE. Thus, the number of neurons in the second layer for each of the 234 networks was reduced from 20 to 17.

2.2.2.2.1 PNNs for Full Flight Map: Results

The primary PNN that was used to determine flight map point did so with an accuracy of 100%. Because the entire set of given flight map points (n=234) was presented to the network for training, a true test of the generalization capabilities of this network would be to present it with data from different flight map points.

The training accuracy for the 234 secondary PNNs – those used to determine the fault for a given flight map point – was 100% for each. Because these networks were each trained using only the average values for each of the six tracking filter outputs (quality parameter updates) , one way to generate additional data for validating the PNN was to use instantaneous (not average) simulation data. Testing with instantaneous data enables trained network simulations using hundreds of thousands of unique, “noisy” data points; not only is the network extensively tested, but it’s ability to generalize is thoroughly demonstrated. Using the ~800,000 instantaneous data points, the mean test accuracy for all 234 nets was 92.4%.

2.2.3 Industry Approach vs. Neural Network Approach

When considering only the original eight flight map points of interest (sea-level static takeoff, standard day takeoff, high altitude takeoff, hot day takeoff, climb, cruise, high altitude cruise, and descent), and omitting booster faults, the k-nearest neighbor method of selecting flight map point paired with the multi-model approach for determining probabilities of faults in each of the engine components resulted in fault diagnoses that were 100% accurate. The feedforward artificial neural network also performed at this level of accuracy.

After expanding the flight map to 234 points and using the multi-model hypothesis method as it is currently designed, this method performed with an accuracy of 44.3%. The PNNs developed for the expanded map were able to classify fault type/severity with an average accuracy of 92.4%. However, evaluating the multiple model hypothesis method in its current configuration for the expanded flight map does not provide a fair assessment of this industry approach. Recall that pre-defined signatures of each fault type are used only for each of the original eight flight map points of interest. The multi-model hypothesis test then compares the signatures that were selected for the identified flight point to the quality parameters (or signatures) from each simulation of the CLM. The output of the MMH is the probability of each fault, (refer back to figures 5 and 7). Thus, without updating this algorithm to include all of the 234 flight map points, as well as defining the fault signatures for use in the MMH, the accuracy of this approach using undefined flight map points is not a meaningful assessment.

3 Discussion

The use of artificial neural networks to predict faults in gas turbine engines was shown to be effective. One described industry approach uses an embedded engine model with an inherent Kalman filter to generate fault “signatures” which are then used for fault diagnosis. When combined with this model-based method, the soft-computing neural network approach was able to identify component faults with a high degree of accuracy.

When simulating the engine with the original eight flight map points of interest, both the MMH and the ANN performed equally well. The use of the currently-configured k-nearest neighbor/MMH approach to identify faults with an expanded flight map, which did not perform well, shows that all flight conditions and event signatures must be incorporated for this approach to be effective; this would be a fairly time-consuming proposition. ANN-based methods, however, are efficient in terms of development time and can easily be updated (retrained) as new data become available from the system. In addition, probabilistic neural networks, in particular, generalize very well and are robust in the presence of noise [11]; that is, inputs that are similar to those used in training the network will most likely be correctly classified.

One possible short-coming to using either of these prediction methods is that both approaches use simulation data, as opposed to real engine data, to define fault signatures. Though the CLM is a fairly accurate depiction of the true engine, it is still a somewhat simplified version of the true aero-thermodynamic engine cycle model. Because of this, “true” event

signatures may be slightly different from those defined by injecting small, medium, and large single component faults into the engine model, as was done for this study. As was demonstrated using the k-NN/MMH approach for the expanded flight map, if the fault signature is not close to one that's already been defined, this method may break down. Both methods, if used in the field, would have to eventually be updated to incorporate real engine data.

Both methods have the ability to predict faults in real-time, but both are only valid (as designed in this study) for steady state operating conditions; transient analysis was not considered.

By incorporating intelligent methods of detecting faults and using the resulting information to proactively schedule engine maintenance when a fault begins to develop, substantial cost savings can be realized as well as benefits from improved engine performance and safety.

4 Bibliography

1. Ed Hindle et al, "A Prognostic and Diagnostic Approach to Engine Health Management", Proceedings of GT2006 ASME Turbo Expo 2006, GT2006-90614.
2. Brotherton, T., Volponi, A., Luppold, R., Simon, D., "eSTORM: Enhanced Self Tuning On-board Real-time Engine Model", Proceedings of the 2003 IEEE Aerospace Conference, Big Sky, MT, March 2003.
3. Noel, N.K., Tammi, K., Buckner, G.D., Gibson, N.S., "Intelligent Kalman Filtering for Fault Detection on an Active Magnetic Bearing System", Proceedings of the 2008 Dynamic Systems and Control Conference, October 20-22, 2008.

4. http://en.wikipedia.org/wiki/Turbofan_engine
5. Y.G. Li, "Performance-analysis-based gas turbine diagnostics: a review", Proceedings of the Institution of Mechanical Engineers, Part A: J Power and Energy, Vol 216, 2002.
6. Dennice Gayme et al, "Fault Diagnosis in Gas Turbine Engines using Fuzzy Logic", IEEE International Conference on Systems, Man, and Cybernetics, Vol 4, (5-8), pp. 3756-3762, October 2003.
7. GE Aviation. One Neumann Way, Cincinnati, OH, 45215
8. http://www.scholarpedia.org/article/K-nearest_neighbor
9. <http://cse.stanford.edu/class/sophomore-college/projects-00/neural-networks/Architecture/feedforward.html>
10. http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf
11. Wasserman, P.D. (1993) *Advanced Methods in Neural Networks*, Van Nostrand Reinhold, New York, Chapter 3, pp. 35-55.
12. Donald F. Specht, "Probabilistic Neural Networks", Neural Networks, Vol. 3, pp. 109-118, 1990.
13. Donald F. Specht, "Probabilistic Neural Networks for Classification, Mapping, or Associative Memory", IEEE International Conference on Neural Networks, 1988 (IEEE Cat. No. 88CH2632-8).
14. <http://www.uta.edu/faculty/sawasthi/Statistics/stneunet.html#pnn>
15. <http://homepages.gold.ac.uk/nikolaev/311pnn.htm>

Appendices

Appendix A: Background Preparation for GE Studies

To investigate and experimentally validate different fault detection approaches for nonlinear dynamic systems, studies conducted using a single-input, single-output active magnetic bearing system. The results of this work are summarized in the following technical paper, which has been accepted for presentation and publication in the Proceedings of the ASME 2008 Dynamic Systems and Control Conference (DSCC 2008), Ann Arbor, MI, October 20-22, 2008.

Intelligent Kalman Filtering for Fault Detection on an Active Magnetic Bearing System

Nana K. Noel¹, Kari Tammi², Gregory D. Buckner¹, Nathan S. Gibson³

¹Department of Mechanical and Aerospace Engineering, North Carolina State University, Raleigh, NC 27695

²VTT – Technical Research Centre of Finland, P.O. Box 1000, FI-02044 VTT, FINLAND

³GE Aviation, One Neumann Way, Cincinnati, OH 45215

ABSTRACT

One of the challenges of condition monitoring and fault detection is to develop techniques that are sufficiently sensitive to faults without triggering false alarms. In this paper we develop and experimentally demonstrate an intelligent approach for detecting faults in a single-input, single-output active magnetic bearing. This technique uses an augmented linear model of the plant dynamics together with a Kalman filter to estimate fault states. A neural network is introduced to enhance the estimation accuracy and eliminate false alarms. This approach is validated experimentally for two types of fabricated faults: changes in suspended mass and coil resistance. The Kalman filter alone is shown to be incapable of identifying all fault cases due to modeling uncertainties. When an artificial neural network is trained to compensate for these uncertainties, however, all fault conditions are identified uniquely.

INTRODUCTION

The primary objective of condition monitoring and fault detection for industrial processes is to identify the true need for maintenance. Condition-based maintenance can lead to safer operation and lower life-cycle cost than calendar-based maintenance. A design challenge is to make condition monitoring systems sufficiently sensitive to true fault situations without triggering false alarms.

One common approach to fault detection is through the use of physical models, both of the plant and of the fault. The basic principle of model-based fault detection involves comparing actual system outputs to simulated responses

generated from a mathematical model of the system. Observers or state estimators may be used to compute differences between the actual and simulated outputs. For accurate plant models, these differences, or residuals, are nearly zero when the system is operating nominally and increase in magnitude when a fault occurs. Various observer-based approaches are discussed by Chen, et al. (1996). Though model-based fault detection can be very effective, model uncertainties can result in non-zero residuals that can be erroneously interpreted as system faults; the fault condition can only be identified with accurate fault models. For this reason, model-based approaches alone can possibly lead to false positive fault identifications, especially if the model is not updated to account for time-varying parameters or plant deterioration.

Another approach to fault detection involves the use of statistical models that “learn” to identify nominal and faulty system behavior after being trained using input-output data. Artificial neural networks are commonly utilized for this purpose (Brotherton 2000). Fuzzy logic, wavelet transforms, genetic algorithms, and Bayesian belief networks are also employed, typically by assessing changes in performance relative to some reference model (Volponi 2004). For a detailed review of statistical condition monitoring and fault diagnostic approaches (as applied to gas turbine engines), see Li (2002).

We present a fault detection method based on Kalman filter state estimation of an augmented linear plant model. Our experiments show that the accuracy and reliability of this approach suffers from model uncertainties associated with the

non-linear plant dynamics. Obvious differences between the plant and its model result in fault conditions that are not uniquely recognizable. For this reason, a neural network is introduced to eliminate false alarms and to make faults uniquely recognizable. This intelligent fault detection strategy is validated experimentally on an active magnetic bearing (AMB) system for combinations of seeded faults.

MATERIALS & METHODS

Here we present the experimental set-up, the identified and augmented plant models, the standard (Kalman filtering) and enhanced (neural network compensated) fault detection strategies.

Experimental Test Rig

The experimental test rig is a single-input, single-output AMB, shown in Figures 1 and 2. This system has one axial electromagnet capable of exerting an attractive vertical force on a 25.4 mm steel ball. This electromagnetic control force is opposed by the gravitational force, the weight of the ball. Real-time control (5 kHz sampling frequency) is achieved using XPC Target™ (The MathWorks Inc., Natick, MA), where a host-target computer configuration allows for the physical system (the AMB test rig) to be interfaced with a Simulink model. The ball's vertical position is measured using a reflectance compensated optical displacement sensor (Philtec RC89, Philtec Inc., Annapolis, MD). This measurement is compared to the tracking reference, resulting in a control voltage which is amplified using a programmable power supply (Kepco BOP 20-10M, Kepco, Inc., Flushing, NY) to power the AMB coil.

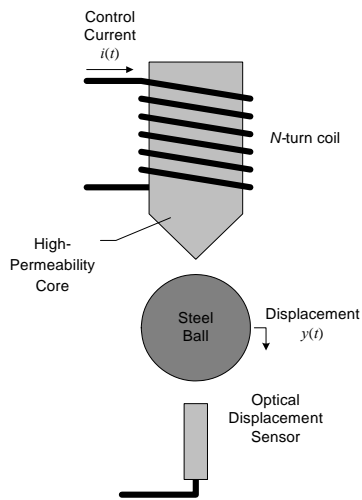


FIGURE 1. SCHEMATIC OF THE AMB SYSTEM

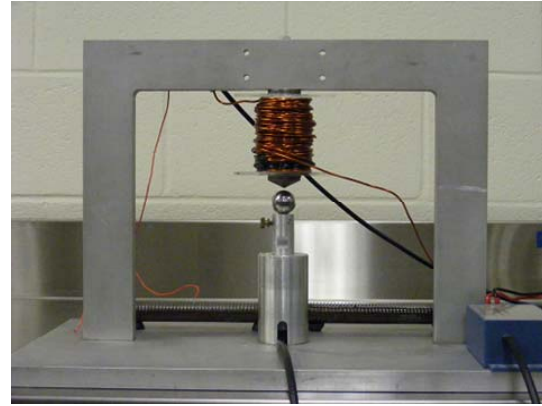


FIGURE 2. PHOTOGRAPH OF THE AMB TEST RIG

Two types of faults were investigated: mass and resistance faults. Mass conditions were varied by drilling holes through two of three 25.4 mm steel balls. Small (11 mm) and large (15.75 mm) bore diameters represented nominal (+0 g) and negative (-16.3 g) mass states, respectively. The positive mass fault condition (+17.8 g) was represented using a ball with no hole. Positive (+0.47 Ω) and negative (-0.46 Ω) resistance faults were introduced using two 1.0 Ω variable power resistors (Ohmite D100K1R0E, 100 W). Table 1 shows the physical parameters of the AMB system and synthesized faults.

TABLE 1. PHYSICAL PARAMETERS OF THE AMB SYSTEM

Ball diameter	25.4 mm
Nominal ball mass (with 11.00 mm through hole)	47.3 g
Negative mass fault (with 15.75 mm through hole)	-16.3 g (31.0 g total)
Positive mass fault (no hole, solid ball)	+17.8 g (65.1 g total)
Nominal air gap	0.3 mm
Minimum air gap	0.2 mm
Maximum air gap	0.4 mm
Coil wiring	175 turns, AWG 12
Nominal coil resistance	2.22 Ω
Positive resistance fault	+0.47 Ω (2.69 Ω total)
Negative resistance fault	-0.46 Ω (1.76 Ω total)

A lead-lag compensator $\left(C(s) = 8000 \frac{0.01s + 1}{0.00025s + 1} \right)$ was experimentally tuned to stabilize the AMB system and enable tracking of a square wave reference input (nominal gap: 0.3 mm, amplitude: 0.15 mm, frequency: 3.33 Hz). Band-limited white noise was injected to ensure persistency of excitation, and time-domain data was acquired for system identification purposes (Figure 3).

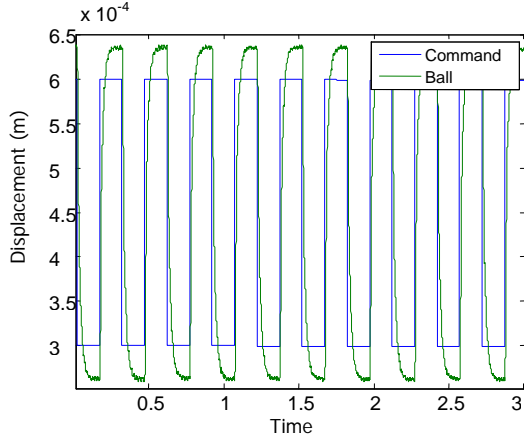


FIGURE 3. REFERENCE SIGNAL (COMMANDED DISPLACEMENT) AND BALL DISPLACEMENT FOR NON-FAULT CONDITIONS WITH NOMINAL AIR GAP

MATLAB's System Identification toolbox was used to identify a fourth-order, discrete-time model (0.2 ms sample time) of the closed-loop AMB plant dynamics:

$$\begin{aligned} x(k+1) &= A_m x(k) + B_m u(k) \\ y(k) &= C_m x(k) \end{aligned} \quad (1)$$

where $x(k)$ represents the state vector ($x_1(k)$ = control voltage, V, $x_2(k)$ = coil current, A, $x_3(k)$ = ball displacement, mm, $x_4(k)$ = ball velocity, mm/s), $u(k)$ represents the tracking reference (mm) and $y(k)$ represents the measured output vector ($y_1(k)$ = voltage, V, $y_2(k)$ = coil current, A, $y_3(k)$ = ball displacement, mm). The identified matrices (the state matrix A_m , the input matrix B_m and the output matrix C_m) represent the closed-loop plant dynamics for a nominal (0.3 mm) air gap:

$$A_m = \begin{bmatrix} 0.8745 & -0.7035 & 4796.4 & -2.091 \\ 0.0727 & 0.7901 & 774.06 & -0.0302 \\ 0.000007 & 0.000019 & 0.8209 & 0.00014 \\ 0.05933 & 0.1516 & -1496.2 & 0.3626 \end{bmatrix},$$

$$B_m = \begin{bmatrix} -6988.2 \\ -651.18 \\ 0.4188 \\ 406.66 \end{bmatrix} \quad C_m = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Table 2 shows the closed-loop pole locations for the identified AMB system.

TABLE 2. CLOSED-LOOP DISCRETE POLES OF THE IDENTIFIED AMB SYSTEM (EQ. 1)

Pole (real \pm imag.)	Damping (%)	Frequency (rad/s)
0.952	100.0	245
0.834	100.0	910
0.531 \pm i 0.572	28.8	4300

Estimation of System Faults

Using a strategy similar to Brotherton et al. (2003), the identified system model (Eq. 1) was augmented to include fault states ΔR and Δm , which represent degradations in resistance and mass, respectively:

$$\begin{aligned} x_{aug}(k+1) &= \begin{bmatrix} x(k+1) \\ \Delta R(k+1) \\ \Delta m(k+1) \end{bmatrix} \\ &= \begin{bmatrix} A_m & A_{RF} & A_{MF} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(k) \\ \Delta R \\ \Delta m \end{bmatrix} + \begin{bmatrix} B_m \\ 0 \\ 0 \end{bmatrix} u(k) \\ y_{aug}(k) &= \begin{bmatrix} C_m & 0 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ \Delta R \\ \Delta m \end{bmatrix} \end{aligned} \quad (2)$$

Here $A_{RF} = [3.5661 \ 0.5517 \ 0.0004 \ 0]^T$ and $A_{MF} = [67.9353 \ 54.4235 \ 0.0059 \ 0]^T$ describe the influence of resistance and

mass faults on the measured states. These matrix coefficients were determined experimentally by measuring the DC shifts in each state measurement resulting from the injection of known faults at nominal operating conditions (0.3 mm air gap). The mass fault condition was “nominal” (no fault) when computing A_{RF} and the resistance fault condition was “nominal” when defining A_{MF} . Because A_{RF} and A_{MF} are linearly independent, mass and resistance faults have unique influences on the measured states.

The augmented model states ($\Delta R(k+1)$ and $\Delta m(k+1)$) are estimated using input-output measurements from the experimental setup (Maciejowski 2002). If a resistance or mass fault occurs, these augmented states compensate for differences (residuals) between the actual plant and model outputs. The state estimator is realized using a discrete-time Kalman filter where the optimal Kalman gain is computed recursively (Sorenson 1985, Welch & Bishop 2006). The measurement noise covariance matrix is computed from actual measurements (voltage, current, and displacement). The process noise covariance matrix requires a velocity signal estimated from the displacement measurement. Standard deviations for the resistance and the mass faults are initialized to one and tuned iteratively; no correlation with other states is assumed.

Compensation With Neural Network

Kalman filter-based fault detection can result in biased state estimates $\hat{x}(k)$, resulting in faults not being uniquely recognizable across the operating space. To address this problem, a simple neural network was trained to compensate for bias in state estimates associated with model uncertainty (Figure 4). Specifically, a linear neural network was trained:

$$\begin{aligned}\tilde{x}(k) &= w(k) \cdot \bar{y}_{ref}(k) + b(k) \\ \hat{x}(k) &= \hat{x}(k) + \tilde{x}(k)\end{aligned}\quad (3)$$

where $w(k) = [-126.3 \ -291.1 \ -0.2 \ 1690.1 \ -390.5 \ 149.6]^T$ represents the neural network’s weight vector and $b(k) = [0.0498 \ 0.1188 \ 0.0001 \ 0.0251 \ 0.1967 \ -0.0749]^T$ represents the network’s bias vector, both of which were updated at each timestep using a backpropagation of error algorithm (10,000 adaptations). The input to the neural network, $\bar{y}_{ref}(k)$, is the moving average of the tracking reference signal. The enhanced state estimate $\hat{x}(k)$ is simply the sum of the Kalman filter estimate $\hat{x}(k)$ and the neural network output $\tilde{x}(k)$.

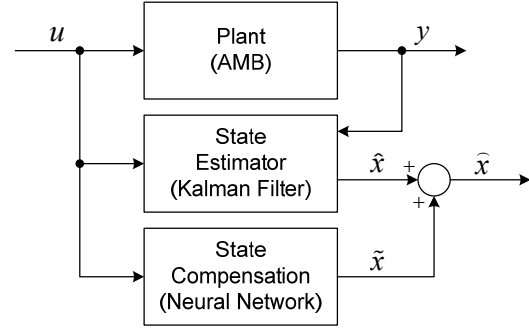


FIGURE 4. AMB PLANT WITH STATE ESTIMATOR (KALMAN FILTER) AND NEURAL NETWORK BASED OFFSET CORRECTION

RESULTS

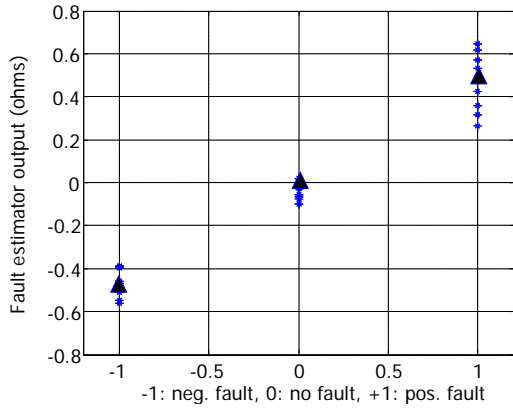
Kalman Filter Based Fault Estimation

Data were collected from the AMB test rig for a variety of air gaps and fault conditions. Table 3 summarizes the experimental cases considered. The Kalman filter implementation was able to identify mass and resistance faults for all cases (Figure 5). However, changing the air gap caused shifts in the fault estimates and made them non-unique. Figures 5a and 5b show the Kalman filter estimates of resistance and mass faults, respectively, for all cases. Table 4 quantifies these results in greater detail.

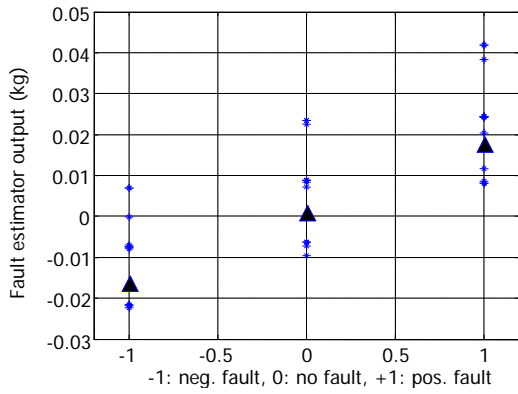
TABLE 3. EXPERIMENTAL CASES

	Mass/ Resistance	-16.3 g	+0 g	+17.8 g
Air gap 0.2 mm	-0.46 Ω	Case 17	Case 11	Case 14
	+0 Ω	Case 16	Case 10	Case 13
	+0.47 Ω	Case 18	Case 12	Case 15
Air gap 0.3 mm	-0.46 Ω	Case 8	Case 2	Case 5
	+0 Ω	Case 7	Case 1	Case 4
	+0.47 Ω	Case 9	Case 3	Case 6
Air gap 0.4 mm	-0.46 Ω	Case 26	Case 20	Case 23
	+0 Ω	Case 25	Case 19	Case 22
	+0.47 Ω	Case 27	Case 21	Case 24

TABLE 4. KALMAN FILTER-BASED FAULT ESTIMATES



(A) RESISTANCE FAULTS



(B) MASS FAULTS

FIGURE 5. ACTUAL (▲) AND ESTIMATED RESISTANCE FAULTS (A) AND MASS FAULTS (B): KALMAN FILTER ESTIMATION

		Resistance fault estimates (Ω)		
Mass/Resistance		-16.3 g	0 g	17.8 g
Air gap 0.2 mm	-0.46 Ω	-0.40	-0.47	-0.56
	+0 Ω	-0.08	-0.07	-0.1
	+0.47 Ω	0.26	0.42	0.62
Air gap 0.3 mm	-0.46 Ω	-0.39	-0.48	-0.55
	+0 Ω	-0.06	-0.02	-0.01
	+0.47 Ω	0.32	0.48	0.65
Air gap 0.4 mm	-0.46 Ω	-0.39	-0.46	-0.51
	+0 Ω	-0.03	0.02	0.02
	+0.47 Ω	0.36	0.53	0.57

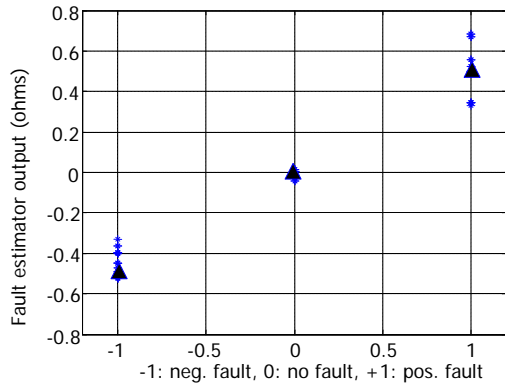
		Mass fault estimates (g)		
Mass/Resistance		-16.3 g	0 g	17.8 g
Air gap 0.2 mm	-0.46 Ω	+7.60	21.7	39
	+0 Ω	7	23.5	41.9
	+0.47 Ω	-0.1	22.5	38.3
Air gap 0.3 mm	-0.46 Ω	-7.4	8.7	24.4
	+0 Ω	-6.9	8.3	24.1
	+0.47 Ω	-7.9	7.1	20.3
Air gap 0.4 mm	-0.46 Ω	-21.6	-6.3	8.2
	+0 Ω	-21.9	-7.3	8.6
	+0.47 Ω	-22.3	-9.5	11.8

The resistance fault estimates (Fig. 5A, Table 4) are reasonably accurate and uniquely recognizable for all cases considered, though the positive fault cases exhibit significant variance. The mass fault estimates (Fig. 5B, Table 4), however, are neither accurate nor uniquely recognizable. Although the estimates trend linearly with mass at a reasonable slope, many of the estimates lead to erroneous fault classifications. Note that several of the Kalman filter estimates for negative mass faults have the wrong sign, with one erroneous estimate (Case 17: +7.60 g) being similar in magnitude and sign to the estimate of a positive mass fault

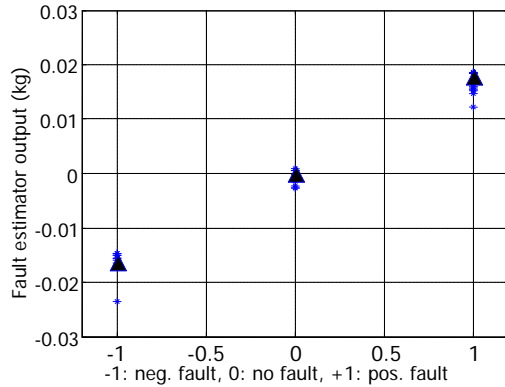
(Case 23: +8.20 g). Hence, changes in operating point cause significant changes in mass fault estimates, making the Kalman filter approach unsuitable for fault detection.

Neural Network Augmented Fault Estimation

The neural network presented in Eq. 3 was trained and implemented for de-biasing the state estimates. All six state estimates were de-biased, but only fault state estimates are shown in Figure 6. Table 5 quantifies these results in greater detail.



(A) RESISTANCE FAULTS



(B) MASS FAULTS

FIGURE 6. ACTUAL (▲) AND ESTIMATED RESISTANCE FAULTS (A) AND MASS FAULTS (B): KALMAN FILTER ESTIMATION WITH NEURAL NETWORK COMPENSATION

TABLE 5. NEURAL NETWORK COMPENSATED FAULT ESTIMATES.

		Resistance fault estimates (Ω)			
		Mass/Resistance	-16.3 g	0 g	17.8 g
Air gap 0.2 mm	-0.46 Ω	-0.33	-0.4	-0.49	
	+0 Ω	-0.01	0	-0.03	
	+0.47 Ω	0.33	0.49	0.68	
Air gap 0.3 mm	-0.46 Ω	-0.36	-0.45	-0.52	
	+0 Ω	-0.03	0	0.01	
	+0.47 Ω	0.35	0.5	0.67	
Air gap 0.4 mm	-0.46 Ω	-0.4	-0.47	-0.53	
	+0 Ω	-0.04	0	0.01	
	+0.47 Ω	0.35	0.52	0.56	

		Mass fault estimates (g)			
		Mass/Resistance	-16.3 g	0 g	17.8 g
Air gap 0.2 mm	-0.46 Ω	-15.9	-2.3	15.2	
	+0 Ω	-17	-0.5	18.3	
	+0.47 Ω	-23.6	-1	14.6	
Air gap 0.3 mm	-0.46 Ω	-15.6	0.6	15.9	
	+0 Ω	-15	-0.2	15.5	
	+0.47 Ω	-16.4	-1.3	12.2	
Air gap 0.4 mm	-0.46 Ω	-14.6	0.8	15.4	
	+0 Ω	-14.8	-0.2	15.7	
	+0.47 Ω	-15.1	-2.7	18.5	

When the neural network compensation is used, all of the fault situations are uniquely recognizable. All compensated fault estimations (above certain thresholds) are correct for all fault conditions. Table 6 further quantifies the improvements in fault detection accuracy resulting from neural network compensation. The Kalman filter-based mass fault estimates exhibit an average error of 250%; when compensated with a neural network, the fault magnitudes are correctly estimated within an accuracy of 30%.

TABLE 6. IMPROVEMENT IN FAULT DETECTION ACCURACY USING ANN COMPENSATION

	Average percent error	
	Uncompensated Kalman estimate	ANN-compensated Kalman estimate
Resistance faults	-1%	0%
Mass faults	250%	-29%

DISCUSSION

The results clearly show that model uncertainties prevent the Kalman filter from estimating mass and resistance faults in an accurate and uniquely recognizable manner. Many of the estimated fault cases, particularly those associated with mass faults, were erroneous; some had the incorrect sign. These errors were primarily associated with differences between the linear model used and the actual AMB plant.

The plant model (Eq. 1) is considered a relatively good linear representation of the plant; a better model might not be achieved via reasonable efforts. Use of a neural network offered a convenient way to compensate for the shortcomings of the linear plant model. The idea was to train the neural network to learn the difference between the linear plant model and the true plant for nominal (no fault) cases. Then, the neural network was used to compensate the difference, which significantly improved the fault detection capabilities of the Kalman filter.

Brotherton et al. (2003) presented an approach where a linear plant model is used and the non-linear characteristics of the true plant are compensated by a neural network in such a way that the fault estimator only observes linear behavior. Only system outputs were compensated in the approach. In the current study, however, full state compensation, including fault estimates was required in order to achieve unique recognition of all the faults. No clear reason was found for the different finding in this study.

REFERENCES

[1] Chen, J., Patton, R.J., and Zhang, H., 1996. "Design of Unknown Input Observers and Robust Fault Detection Filters", *International Journal of Control*, 63(1), pp. 85-105.

[2] Brotherton, T., Jahns, G. Jacobs, J., Wroblewski, D., "Prognosis of Faults in Gas Turbine Engines", 2000.

Aerospace Conference Proceedings, IEEE, 6, pp. 163-171.

[3] Volponi, A.J., Brotherton, T., and Luppold, R., 2004. "Development of an Information Fusion System for Engine Diagnostics and Health Management", AIAA 1st Intelligent Systems Technical Conference, Chicago, Illinois, September.

[4] Li, Y.G., "Performance-Analysis-Based Gas Turbine Diagnostics: a Review", 2002. *Proc. Inst. Mech. Eng., Part A: J Power and Energy*, 216(5).

[5] Brotherton T., Volponi A., Luppold R & Simon D.L., 2003. "eSTORM: Enhanced Self Tuning On-board Real-time Engine Model", Proceedings of the IEEE Aerospace Conference. Big Sky, MT, March.

[6] Maciejowski, J.M., *Predictive Control*, 2002. Pearson Education Limited, Edinburgh Gate, pp. 56-61.

[7] Sorenson, H.W., *Kalman Filtering: Theory and Application*, 1985. IEEE Press, New York.

[8] Welch, G. & Bishop, G. "An Introduction to the Kalman Filter", TR 95-041, 2006. Department of Computer Science, University of North Carolina at Chapel Hill, USA.

Appendix B: Matlab Code Used to Auto-Simulate the CLM

Matlab was used to auto-simulate the CLM. Following is the code that was used to generate a complete data set for all faults (n=20) at all flight map points (n=234).

```
clear;clc;close all
filename={'ss_maxpwr_pnts_1_clminit_s';
'ss_maxpwr_pnts_2_clminit_s';
'ss_maxpwr_pnts_3_clminit_s';
'ss_maxpwr_pnts_4_clminit_s';
'ss_maxpwr_pnts_5_clminit_s';
'ss_maxpwr_pnts_6_clminit_s';
'ss_maxpwr_pnts_7_clminit_s';
'ss_maxpwr_pnts_8_clminit_s';
'ss_maxpwr_pnts_9_clminit_s';
'ss_maxpwr_pnts_10_clminit_s';
'ss_maxpwr_pnts_11_clminit_s';
'ss_maxpwr_pnts_12_clminit_s';
'ss_maxpwr_pnts_13_clminit_s';
'ss_maxpwr_pnts_14_clminit_s';
'ss_maxpwr_pnts_15_clminit_s';
'ss_maxpwr_pnts_16_clminit_s';
'ss_maxpwr_pnts_17_clminit_s';
'ss_maxpwr_pnts_18_clminit_s';
'ss_maxpwr_pnts_19_clminit_s';
'ss_maxpwr_pnts_20_clminit_s';
'ss_maxpwr_pnts_21_clminit_s';
'ss_maxpwr_pnts_22_clminit_s';
'ss_maxpwr_pnts_23_clminit_s';
'ss_maxpwr_pnts_24_clminit_s';
'ss_maxpwr_pnts_25_clminit_s';
'ss_maxpwr_pnts_26_clminit_s';
'ss_maxpwr_pnts_27_clminit_s';
'ss_maxpwr_pnts_28_clminit_s';
'ss_maxpwr_pnts_29_clminit_s';
'ss_maxpwr_pnts_30_clminit_s';
'ss_maxpwr_pnts_31_clminit_s';
'ss_maxpwr_pnts_32_clminit_s';
'ss_maxpwr_pnts_33_clminit_s';
'ss_maxpwr_pnts_34_clminit_s';
'ss_maxpwr_pnts_35_clminit_s';
'ss_maxpwr_pnts_36_clminit_s';
'ss_maxpwr_pnts_37_clminit_s';
'ss_maxpwr_pnts_38_clminit_s';
'ss_maxpwr_pnts_39_clminit_s';
'ss_maxpwr_pnts_40_clminit_s';
```

'ss_maxpwr_pnts_41_clminit_s';
'ss_maxpwr_pnts_42_clminit_s';
'ss_maxpwr_pnts_43_clminit_s';
'ss_maxpwr_pnts_44_clminit_s';
'ss_maxpwr_pnts_45_clminit_s';
'ss_maxpwr_pnts_46_clminit_s';
'ss_maxpwr_pnts_47_clminit_s';
'ss_maxpwr_pnts_48_clminit_s';
'ss_maxpwr_pnts_49_clminit_s';
'ss_maxpwr_pnts_50_clminit_s';
'ss_maxpwr_pnts_51_clminit_s';
'ss_maxpwr_pnts_52_clminit_s';
'ss_maxpwr_pnts_53_clminit_s';
'ss_maxpwr_pnts_54_clminit_s';
'ss_maxpwr_pnts_55_clminit_s';
'ss_maxpwr_pnts_56_clminit_s';
'ss_maxpwr_pnts_57_clminit_s';
'ss_maxpwr_pnts_58_clminit_s';
'ss_maxpwr_pnts_59_clminit_s';
'ss_maxpwr_pnts_60_clminit_s';
'ss_maxpwr_pnts_61_clminit_s';
'ss_maxpwr_pnts_62_clminit_s';
'ss_maxpwr_pnts_63_clminit_s';
'ss_maxpwr_pnts_64_clminit_s';
'ss_maxpwr_pnts_65_clminit_s';
'ss_maxpwr_pnts_66_clminit_s';
'ss_maxpwr_pnts_67_clminit_s';
'ss_maxpwr_pnts_68_clminit_s';
'ss_maxpwr_pnts_69_clminit_s';
'ss_maxpwr_pnts_70_clminit_s';
'ss_maxpwr_pnts_71_clminit_s';
'ss_maxpwr_pnts_72_clminit_s';
'ss_maxpwr_pnts_73_clminit_s';
'ss_maxpwr_pnts_74_clminit_s';
'ss_maxpwr_pnts_75_clminit_s';
'ss_maxpwr_pnts_76_clminit_s';
'ss_maxpwr_pnts_77_clminit_s';
'ss_maxpwr_pnts_78_clminit_s';
'ss_intermedpwr_pnts_79_clminit_s';
'ss_intermedpwr_pnts_80_clminit_s';
'ss_intermedpwr_pnts_81_clminit_s';
'ss_intermedpwr_pnts_82_clminit_s';
'ss_intermedpwr_pnts_83_clminit_s';
'ss_intermedpwr_pnts_84_clminit_s';
'ss_intermedpwr_pnts_85_clminit_s';
'ss_intermedpwr_pnts_86_clminit_s';
'ss_intermedpwr_pnts_87_clminit_s';
'ss_intermedpwr_pnts_88_clminit_s';
'ss_intermedpwr_pnts_89_clminit_s';
'ss_intermedpwr_pnts_90_clminit_s';
'ss_intermedpwr_pnts_91_clminit_s';
'ss_intermedpwr_pnts_92_clminit_s';

'ss_intermedpwr_pnts_93_clmunit_s';
'ss_intermedpwr_pnts_94_clmunit_s';
'ss_intermedpwr_pnts_95_clmunit_s';
'ss_intermedpwr_pnts_96_clmunit_s';
'ss_intermedpwr_pnts_97_clmunit_s';
'ss_intermedpwr_pnts_98_clmunit_s';
'ss_intermedpwr_pnts_99_clmunit_s';
'ss_intermedpwr_pnts_100_clmunit_s';
'ss_intermedpwr_pnts_101_clmunit_s';
'ss_intermedpwr_pnts_102_clmunit_s';
'ss_intermedpwr_pnts_103_clmunit_s';
'ss_intermedpwr_pnts_104_clmunit_s';
'ss_intermedpwr_pnts_105_clmunit_s';
'ss_intermedpwr_pnts_106_clmunit_s';
'ss_intermedpwr_pnts_107_clmunit_s';
'ss_intermedpwr_pnts_108_clmunit_s';
'ss_intermedpwr_pnts_109_clmunit_s';
'ss_intermedpwr_pnts_110_clmunit_s';
'ss_intermedpwr_pnts_111_clmunit_s';
'ss_intermedpwr_pnts_112_clmunit_s';
'ss_intermedpwr_pnts_113_clmunit_s';
'ss_intermedpwr_pnts_114_clmunit_s';
'ss_intermedpwr_pnts_115_clmunit_s';
'ss_intermedpwr_pnts_116_clmunit_s';
'ss_intermedpwr_pnts_117_clmunit_s';
'ss_intermedpwr_pnts_118_clmunit_s';
'ss_intermedpwr_pnts_119_clmunit_s';
'ss_intermedpwr_pnts_120_clmunit_s';
'ss_intermedpwr_pnts_121_clmunit_s';
'ss_intermedpwr_pnts_122_clmunit_s';
'ss_intermedpwr_pnts_123_clmunit_s';
'ss_intermedpwr_pnts_124_clmunit_s';
'ss_intermedpwr_pnts_125_clmunit_s';
'ss_intermedpwr_pnts_126_clmunit_s';
'ss_intermedpwr_pnts_127_clmunit_s';
'ss_intermedpwr_pnts_128_clmunit_s';
'ss_intermedpwr_pnts_129_clmunit_s';
'ss_intermedpwr_pnts_130_clmunit_s';
'ss_intermedpwr_pnts_131_clmunit_s';
'ss_intermedpwr_pnts_132_clmunit_s';
'ss_intermedpwr_pnts_133_clmunit_s';
'ss_intermedpwr_pnts_134_clmunit_s';
'ss_intermedpwr_pnts_135_clmunit_s';
'ss_intermedpwr_pnts_136_clmunit_s';
'ss_intermedpwr_pnts_137_clmunit_s';
'ss_intermedpwr_pnts_138_clmunit_s';
'ss_intermedpwr_pnts_139_clmunit_s';
'ss_intermedpwr_pnts_140_clmunit_s';
'ss_intermedpwr_pnts_141_clmunit_s';
'ss_intermedpwr_pnts_142_clmunit_s';
'ss_intermedpwr_pnts_143_clmunit_s';
'ss_intermedpwr_pnts_144_clmunit_s';

'ss_intermedpwr_pnts_145_clmunit_s';
'ss_intermedpwr_pnts_146_clmunit_s';
'ss_intermedpwr_pnts_147_clmunit_s';
'ss_intermedpwr_pnts_148_clmunit_s';
'ss_intermedpwr_pnts_149_clmunit_s';
'ss_intermedpwr_pnts_150_clmunit_s';
'ss_intermedpwr_pnts_151_clmunit_s';
'ss_intermedpwr_pnts_152_clmunit_s';
'ss_intermedpwr_pnts_153_clmunit_s';
'ss_intermedpwr_pnts_154_clmunit_s';
'ss_intermedpwr_pnts_155_clmunit_s';
'ss_intermedpwr_pnts_156_clmunit_s';
'ss_idle_pnts_157_clmunit_s';
'ss_idle_pnts_158_clmunit_s';
'ss_idle_pnts_159_clmunit_s';
'ss_idle_pnts_160_clmunit_s';
'ss_idle_pnts_161_clmunit_s';
'ss_idle_pnts_162_clmunit_s';
'ss_idle_pnts_163_clmunit_s';
'ss_idle_pnts_164_clmunit_s';
'ss_idle_pnts_165_clmunit_s';
'ss_idle_pnts_166_clmunit_s';
'ss_idle_pnts_167_clmunit_s';
'ss_idle_pnts_168_clmunit_s';
'ss_idle_pnts_169_clmunit_s';
'ss_idle_pnts_170_clmunit_s';
'ss_idle_pnts_171_clmunit_s';
'ss_idle_pnts_172_clmunit_s';
'ss_idle_pnts_173_clmunit_s';
'ss_idle_pnts_174_clmunit_s';
'ss_idle_pnts_175_clmunit_s';
'ss_idle_pnts_176_clmunit_s';
'ss_idle_pnts_177_clmunit_s';
'ss_idle_pnts_178_clmunit_s';
'ss_idle_pnts_179_clmunit_s';
'ss_idle_pnts_180_clmunit_s';
'ss_idle_pnts_181_clmunit_s';
'ss_idle_pnts_182_clmunit_s';
'ss_idle_pnts_183_clmunit_s';
'ss_idle_pnts_184_clmunit_s';
'ss_idle_pnts_185_clmunit_s';
'ss_idle_pnts_186_clmunit_s';
'ss_idle_pnts_187_clmunit_s';
'ss_idle_pnts_188_clmunit_s';
'ss_idle_pnts_189_clmunit_s';
'ss_idle_pnts_190_clmunit_s';
'ss_idle_pnts_191_clmunit_s';
'ss_idle_pnts_192_clmunit_s';
'ss_idle_pnts_193_clmunit_s';
'ss_idle_pnts_194_clmunit_s';
'ss_idle_pnts_195_clmunit_s';
'ss_idle_pnts_196_clmunit_s';

```

'ss_idle_pnts_197_clminit_s';
'ss_idle_pnts_198_clminit_s';
'ss_idle_pnts_199_clminit_s';
'ss_idle_pnts_200_clminit_s';
'ss_idle_pnts_201_clminit_s';
'ss_idle_pnts_202_clminit_s';
'ss_idle_pnts_203_clminit_s';
'ss_idle_pnts_204_clminit_s';
'ss_idle_pnts_205_clminit_s';
'ss_idle_pnts_206_clminit_s';
'ss_idle_pnts_207_clminit_s';
'ss_idle_pnts_208_clminit_s';
'ss_idle_pnts_209_clminit_s';
'ss_idle_pnts_210_clminit_s';
'ss_idle_pnts_211_clminit_s';
'ss_idle_pnts_212_clminit_s';
'ss_idle_pnts_213_clminit_s';
'ss_idle_pnts_214_clminit_s';
'ss_idle_pnts_215_clminit_s';
'ss_idle_pnts_216_clminit_s';
'ss_idle_pnts_217_clminit_s';
'ss_idle_pnts_218_clminit_s';
'ss_idle_pnts_219_clminit_s';
'ss_idle_pnts_220_clminit_s';
'ss_idle_pnts_221_clminit_s';
'ss_idle_pnts_222_clminit_s';
'ss_idle_pnts_223_clminit_s';
'ss_idle_pnts_224_clminit_s';
'ss_idle_pnts_225_clminit_s';
'ss_idle_pnts_226_clminit_s';
'ss_idle_pnts_227_clminit_s';
'ss_idle_pnts_228_clminit_s';
'ss_idle_pnts_229_clminit_s';
'ss_idle_pnts_230_clminit_s';
'ss_idle_pnts_231_clminit_s';
'ss_idle_pnts_232_clminit_s';
'ss_idle_pnts_233_clminit_s';
'ss_idle_pnts_234_clminit_s';
};
%=====
for j=1:234, %flight phases (use 8 or 234)
    filename{j}
    run(filename{j});
    Flight_Map_Point=j;
        for i=1:20, %fault possibilities (including no fault)
            Engine_fault=i;
            init
            sim engine_sim_runfirst
            sim model_MMH_GP_runsecond
            Q=dqall(2000:4000,:);
            Savefile1=strcat('map',num2str(j),'fault',num2str(i),'.mat')

```

```
        save(savefile1, 'Q', 'uinit');
        savefile2=strcat('GEmap',num2str(j),'fault',num2str(i),'.mat')
        save (savefile2,'unknownprob','Probabilities');
    end
end
%=====
```

Appendix C: Matlab Code Used to Generate/Train the ANN

After generating a complete set of fault data for all flight map points, a reduced set of input/output data for training and testing the ANN was needed. The following code selects input/output data, trains and tests a feedforward ANN.

```
clear;clc;close all

%=====TARGETS FOR ANN=====
targets1=[ones(1,16008);ones(1,16008)]; %small fan anomaly
targets2=[ones(1,16008);2*ones(1,16008)]; %med fan anomaly
targets3=[ones(1,16008);3*ones(1,16008)]; %large fan anomaly

targets4=[2*ones(1,16008);ones(1,16008)]; %small booster anomaly
targets5=[2*ones(1,16008);2*ones(1,16008)]; %med booster anomaly
targets6=[2*ones(1,16008);3*ones(1,16008)]; %large booster anomaly

targets7=[3*ones(1,16008);ones(1,16008)]; %small HPC anomaly
targets8=[3*ones(1,16008);2*ones(1,16008)]; %med HPC anomaly
targets9=[3*ones(1,16008);3*ones(1,16008)]; %large HPC anomaly

targets10=[4*ones(1,16008);ones(1,16008)]; %small combustor anomaly
targets11=[4*ones(1,16008);2*ones(1,16008)]; %med combustor anomaly
targets12=[4*ones(1,16008);3*ones(1,16008)]; %large combustor anomaly

targets13=[5*ones(1,16008);ones(1,16008)]; %small HPT anomaly
targets14=[5*ones(1,16008);2*ones(1,16008)]; %med HPT anomaly
targets15=[5*ones(1,16008);3*ones(1,16008)]; %large HPT anomaly

targets16=[6*ones(1,16008);ones(1,16008)]; %small LPT anomaly
targets17=[6*ones(1,16008);2*ones(1,16008)]; %med LPT anomaly
targets18=[6*ones(1,16008);3*ones(1,16008)]; %large LPT anomaly

targets19=[8*ones(1,3208);zeros(1,3208)]; %no event
targets20=[7*ones(1,3208);ones(1,3208)]; %unknown event

%=====FAN ANOMALIES=====
%small fan anomaly, all flight phases
filenames={'data11' 'data21' 'data31' 'data41' 'data51' 'data61' 'data71'
'data81'};
```

```

for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);

    Q=Q';
    in=[inputs;Q];
    in1(:,:,i)=[in];
end

% inputs1=[in1(:,:,1) in1(:,:,2) in1(:,:,3) in1(:,:,4) in1(:,:,5)
in1(:,:,6) in1(:,:,7) in1(:,:,8)];
inputs1=[in1(:,1:10:2001,1) in1(:,1:10:2001,2) in1(:,1:10:2001,3)
in1(:,1:10:2001,4) in1(:,1:10:2001,5) in1(:,1:10:2001,6)
in1(:,1:10:2001,7) in1(:,1:10:2001,8)];

%medium fan anomaly, all flight phases
filenames={'data12' 'data22' 'data32' 'data42' 'data52' 'data62' 'data72'
'data82'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in2(:,:,i)=[in];
end

% inputs2=[in2(:,:,1) in2(:,:,2) in2(:,:,3) in2(:,:,4) in2(:,:,5)
in2(:,:,6) in2(:,:,7) in2(:,:,8)];
inputs2=[in2(:,1:10:2001,1) in2(:,1:10:2001,2) in2(:,1:10:2001,3)
in2(:,1:10:2001,4) in2(:,1:10:2001,5) in2(:,1:10:2001,6)
in2(:,1:10:2001,7) in2(:,1:10:2001,8)];

%large fan anomaly, all flight phases
filenames={'data13' 'data23' 'data33' 'data43' 'data53' 'data63' 'data73'
'data83'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in3(:,:,i)=[in];
end

% inputs3=[in3(:,:,1) in3(:,:,2) in3(:,:,3) in3(:,:,4) in3(:,:,5)
in3(:,:,6) in3(:,:,7) in3(:,:,8)];
inputs3=[in3(:,1:10:2001,1) in3(:,1:10:2001,2) in3(:,1:10:2001,3)
in3(:,1:10:2001,4) in3(:,1:10:2001,5) in3(:,1:10:2001,6)
in3(:,1:10:2001,7) in3(:,1:10:2001,8)];

```

```

%=====BOOSTER ANOMALIES=====
%small booster anomaly, all flight phases
filenames={'data14' 'data24' 'data34' 'data44' 'data54' 'data64' 'data74'
'data84'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in4(:,:,i)=[in];
end
inputs4=[in4(:,1:10:2001,1) in4(:,1:10:2001,2) in4(:,1:10:2001,3)
in4(:,1:10:2001,4) in4(:,1:10:2001,5) in4(:,1:10:2001,6)
in4(:,1:10:2001,7) in4(:,1:10:2001,8)];

%medium booster anomaly, all flight phases
filenames={'data15' 'data25' 'data35' 'data45' 'data55' 'data65' 'data75'
'data85'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in5(:,:,i)=[in];
end
inputs5=[in5(:,1:10:2001,1) in5(:,1:10:2001,2) in5(:,1:10:2001,3)
in5(:,1:10:2001,4) in5(:,1:10:2001,5) in5(:,1:10:2001,6)
in5(:,1:10:2001,7) in5(:,1:10:2001,8)];

%large booster anomaly, all flight phases
filenames={'data16' 'data26' 'data36' 'data46' 'data56' 'data66' 'data76'
'data86'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in6(:,:,i)=[in];
end
inputs6=[in6(:,1:10:2001,1) in6(:,1:10:2001,2) in6(:,1:10:2001,3)
in6(:,1:10:2001,4) in6(:,1:10:2001,5) in6(:,1:10:2001,6)
in6(:,1:10:2001,7) in6(:,1:10:2001,8)];

```

```

%=====HPC ANOMALIES=====
HPC anomaly, all flight phases
filenames={'data17' 'data27' 'data37' 'data47' 'data57' 'data67' 'data77'
'data87'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in7(:, :, i)=[in];
end
inputs7=[in7(:, 1:10:2001,1) in7(:, 1:10:2001,2) in7(:, 1:10:2001,3)
in7(:, 1:10:2001,4) in7(:, 1:10:2001,5) in7(:, 1:10:2001,6)
in7(:, 1:10:2001,7) in7(:, 1:10:2001,8)];

medium HPC anomaly, all flight phases
filenames={'data18' 'data28' 'data38' 'data48' 'data58' 'data68' 'data78'
'data88'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in8(:, :, i)=[in];
end
inputs8=[in8(:, 1:10:2001,1) in8(:, 1:10:2001,2) in8(:, 1:10:2001,3)
in8(:, 1:10:2001,4) in8(:, 1:10:2001,5) in8(:, 1:10:2001,6)
in8(:, 1:10:2001,7) in8(:, 1:10:2001,8)];

large HPC anomaly, all flight phases
filenames={'data19' 'data29' 'data39' 'data49' 'data59' 'data69' 'data79'
'data89'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in9(:, :, i)=[in];
end
inputs9=[in9(:, 1:10:2001,1) in9(:, 1:10:2001,2) in9(:, 1:10:2001,3)
in9(:, 1:10:2001,4) in9(:, 1:10:2001,5) in9(:, 1:10:2001,6)
in9(:, 1:10:2001,7) in9(:, 1:10:2001,8)];

%=====COMBUSTOR ANOMALIES=====
combustor anomaly, all flight phases
filenames={'data110' 'data210' 'data310' 'data410' 'data510' 'data610'
'data710' 'data810'};

```

```

for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in10(:, :, i)=[in];
end
inputs10=[in10(:, 1:10:2001, 1) in10(:, 1:10:2001, 2) in10(:, 1:10:2001, 3)
in10(:, 1:10:2001, 4) in10(:, 1:10:2001, 5) in10(:, 1:10:2001, 6)
in10(:, 1:10:2001, 7) in10(:, 1:10:2001, 8)];

%medium combustor anomaly, all flight phases
filenames={'data111' 'data211' 'data311' 'data411' 'data511' 'data611'
'data711' 'data811'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in11(:, :, i)=[in];
end
inputs11=[in11(:, 1:10:2001, 1) in11(:, 1:10:2001, 2) in11(:, 1:10:2001, 3)
in11(:, 1:10:2001, 4) in11(:, 1:10:2001, 5) in11(:, 1:10:2001, 6)
in11(:, 1:10:2001, 7) in11(:, 1:10:2001, 8)];

%large combustor anomaly, all flight phases
filenames={'data112' 'data212' 'data312' 'data412' 'data512' 'data612'
'data712' 'data812'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in12(:, :, i)=[in];
end
inputs12=[in12(:, 1:10:2001, 1) in12(:, 1:10:2001, 2) in12(:, 1:10:2001, 3)
in12(:, 1:10:2001, 4) in12(:, 1:10:2001, 5) in12(:, 1:10:2001, 6)
in12(:, 1:10:2001, 7) in12(:, 1:10:2001, 8)];

%=====HPT ANOMALIES=====
%small HPT anomaly, all flight phases
filenames={'data113' 'data213' 'data313' 'data413' 'data513' 'data613'
'data713' 'data813'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';

```

```

        in=[inputs;Q];
        in13(:, :, i)=[in];
end
inputs13=[in13(:, 1:10:2001,1) in13(:, 1:10:2001,2) in13(:, 1:10:2001,3)
in13(:, 1:10:2001,4) in13(:, 1:10:2001,5) in13(:, 1:10:2001,6)
in13(:, 1:10:2001,7) in13(:, 1:10:2001,8)];

%medium HPT anomaly, all flight phases
filenames={'data114' 'data214' 'data314' 'data414' 'data514' 'data614'
'data714' 'data814'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=uinit*ones(1,2001);

    Q=Q';
    in=[inputs;Q];
    in14(:, :, i)=[in];
end
inputs14=[in14(:, 1:10:2001,1) in14(:, 1:10:2001,2) in14(:, 1:10:2001,3)
in14(:, 1:10:2001,4) in14(:, 1:10:2001,5) in14(:, 1:10:2001,6)
in14(:, 1:10:2001,7) in14(:, 1:10:2001,8)];

%large HPT anomaly, all flight phases
filenames={'data115' 'data215' 'data315' 'data415' 'data515' 'data615'
'data715' 'data815'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=uinit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in15(:, :, i)=[in];
end
inputs15=[in15(:, 1:10:2001,1) in15(:, 1:10:2001,2) in15(:, 1:10:2001,3)
in15(:, 1:10:2001,4) in15(:, 1:10:2001,5) in15(:, 1:10:2001,6)
in15(:, 1:10:2001,7) in15(:, 1:10:2001,8)];

%=====LPT ANOMALIES=====
%small LPT anomaly, all flight phases
filenames={'data116' 'data216' 'data316' 'data416' 'data516' 'data616'
'data716' 'data816'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=uinit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in16(:, :, i)=[in];
end

```

```

inputs16=[in16(:,1:10:2001,1) in16(:,1:10:2001,2) in16(:,1:10:2001,3)
in16(:,1:10:2001,4) in16(:,1:10:2001,5) in16(:,1:10:2001,6)
in16(:,1:10:2001,7) in16(:,1:10:2001,8)];

%medium LPT anomaly, all flight phases
filenames={'data117' 'data217' 'data317' 'data417' 'data517' 'data617'
'data717' 'data817'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in17(:, :, i)=[in];
end
inputs17=[in17(:,1:10:2001,1) in17(:,1:10:2001,2) in17(:,1:10:2001,3)
in17(:,1:10:2001,4) in17(:,1:10:2001,5) in17(:,1:10:2001,6)
in17(:,1:10:2001,7) in17(:,1:10:2001,8)];

%large LPT anomaly, all flight phases
filenames={'data118' 'data218' 'data318' 'data418' 'data518' 'data618'
'data718' 'data818'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in18(:, :, i)=[in];
end
inputs18=[in18(:,1:10:2001,1) in18(:,1:10:2001,2) in18(:,1:10:2001,3)
in18(:,1:10:2001,4) in18(:,1:10:2001,5) in18(:,1:10:2001,6)
in18(:,1:10:2001,7) in18(:,1:10:2001,8)];

%=====NO ANOMALY=====
%no anomaly, all flight phases
filenames={'data119' 'data219' 'data319' 'data419' 'data519' 'data619'
'data719' 'data819'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in19(:, :, i)=[in];
end
inputs19=[in19(:,1:10:2001,1) in19(:,1:10:2001,2) in19(:,1:10:2001,3)
in19(:,1:10:2001,4) in19(:,1:10:2001,5) in19(:,1:10:2001,6)
in19(:,1:10:2001,7) in19(:,1:10:2001,8)];

```

```

%=====UNKNOWN ANOMALY=====
%no anomaly, all flight phases
filenames={'data120' 'data220' 'data320' 'data420' 'data520' 'data620'
'data720' 'data820'};
for i=1:8,
    filenames{i};
    load(filenames{i});
    inputs=unit*ones(1,2001);
    Q=Q';
    in=[inputs;Q];
    in20(:, :, i)=[in];
end
inputs20=[in20(:,1:10:2001,1) in20(:,1:10:2001,2) in20(:,1:10:2001,3)
in20(:,1:10:2001,4) in20(:,1:10:2001,5) in20(:,1:10:2001,6)
in20(:,1:10:2001,7) in20(:,1:10:2001,8)];

%===== NN training =====

inputs=[inputs1 inputs2 inputs3 inputs7 inputs8 inputs9 inputs10 inputs11
inputs12 inputs13 inputs14 inputs15 inputs16 inputs17 inputs18 inputs19
inputs20];
inputs=removeconstantrows(inputs);
targets=[targets1(:,1:1608) targets2(:,1:1608) targets3(:,1:1608)
targets7(:,1:1608) targets8(:,1:1608) targets9(:,1:1608)
targets10(:,1:1608) targets11(:,1:1608) targets12(:,1:1608)
targets13(:,1:1608) targets14(:,1:1608) targets15(:,1:1608)
targets16(:,1:1608) targets17(:,1:1608) targets18(:,1:1608)
targets19(:,1:1608) targets20(:,1:1608)];

[pin,ps]=mapminmax(inputs);
[tin,ts]=mapminmax(targets);
[trainV,valV,testV]=dividevec(pin,tin,0.2,0.2)

%training the NN
net=newff(minmax(pin),[40 2]);
net=train(net,trainV.P,trainV.T,[],[],valV,testV);
save 822net net ps ts %net trained on all data EXCEPT booster (removed),
used every 10th point. looks great!
%===== NN testing =====
%testing net (normalized using mapminmax)
nettrain=(sim(net,trainV.P)); %test net using original training data
t=(1:16402);
figure(1)
subplot(2,2,1)
plot(t,nettrain(1,:),'t,trainV.T(1,:)')
title('Training normalized')
ylabel('fault')
xlabel('Number of input samples')
subplot(2,2,3)
plot(t,nettrain(2,:),'t,trainV.T(2,:)')

```

```

ylabel('severity')
xlabel('Number of input samples')

actual=mapminmax('reverse',trainV.T,ts);
nettrain2=mapminmax('reverse',nettrain,ts);
nettrain2=round(nettrain2);
% actual2=mapminmax('reverse',testV.T,ts);
% nettest2=mapminmax('reverse',nettest,ts);

%plotting actual (un-normalized)
% figure(2)
subplot(2,2,2)
t=(1:16402);
plot(t,nettrain2(1,:),'t,actual(1,:)')
title('Training actual')
ylabel('fault')
xlabel('Number of input samples')

subplot(2,2,4)
plot(t,nettrain2(2,:),'t,actual(2,:)')
ylabel('severity')
xlabel('Number of input samples')

%=====
nettrain=(sim(net,testV.P)); %test net using testing data
t=(1:5467);
subplot(2,2,1)
plot(t,nettrain(1,:),'t,testV.T(1,:)')
title('Testing normalized')
ylabel('fault')
xlabel('Number of input samples')
subplot(2,2,3)
plot(t,nettrain(2,:),'t,testV.T(2,:)')
ylabel('severity')
xlabel('Number of input samples')

actual=mapminmax('reverse',testV.T,ts);
nettrain2=mapminmax('reverse',nettrain,ts);
nettrain2=round(nettrain2);
% actual2=mapminmax('reverse',testV.T,ts);
% nettest2=mapminmax('reverse',nettest,ts);

%plotting actual (un-normalized)
% figure(2)
subplot(2,2,2)
t=(1:5467);
plot(t,nettrain2(1,:),'t,actual(1,:)')
title('Testing actual')
ylabel('fault')
xlabel('Number of input samples')
subplot(2,2,4)

```

```
plot(t,nettrain2(2,:)',t,actual(2,:))  
ylabel('severity')  
xlabel('Number of input samples')
```

Appendix D: Matlab Code Used to Generate and Train the PNNs

After generating a complete set of fault data for all flight map points, a reduced set of input/output data for training and testing the PNN was needed. The following code selects input/output data, trains and tests a feedforward PNN.

```
%PNN used to detect and diagnose faults (234 flight map points)
%2-23-08

clear;clc;close all

map=[1:234]; %234 flight map points
%=====FAN ANOMALIES=====
%small fan anomaly, all flight phases

for k=1:234 %change all back to 234 for ALL flight map points/power
settings
    filenames(k)=[ 'map' int2str(k) 'fault1' ];
end

for i=1:234,
    load(filenames{i});
    Q=Q';%Q is the vector of qualities

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];%PNN will be trained using the mean quality values
    in=[unit;R];%69x1
    in1(:,:,i)=[in]; %69x1x234
    Q1(:,:,i)=[Q];%6x2001x234 Q1 is matrix of qualities for each sim
of 234 flight map points
end
inputs1=reshape(in1,69,234); %69x234 (1-63 neighbor info, 64-69 fault
signature)

inputs1=removeconstantrows(inputs1);%13x234 1-9 neighbor info, 10-15 fault
signature
fault1=ones(1,234);%1x234

%medium fan anomaly, all flight phases
for k=1:234
    filenames(k)=[ 'map' int2str(k) 'fault2' ];
end
```

```

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in2(:,:,i)=[in];
end

inputs2=reshape(in2,69,234);
inputs2=removeconstantrows(inputs2);
fault2=2*ones(1,234);

%large fan anomaly, all flight phases
for k=1:234
    filenamees(k)={['map' int2str(k) 'fault3'}];
end

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in3(:,:,i)=[in];
end

inputs3=reshape(in3,69,234);
inputs3=removeconstantrows(inputs3);
fault3=3*ones(1,234);

%=====BOOSTER ANOMALIES=====
%small booster anomaly, all flight phases
for k=1:234
    filenamees(k)={['map' int2str(k) 'fault4'}];
end

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];

```

```

    in4(:, :, i)=[in];
end

inputs4=reshape(in4,69,234);
inputs4=removeconstantrows(inputs4);

fault4=4*ones(1,234);

%medium booster anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault5'}];
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1, :));mean(Q(2, :));mean(Q(3, :));mean(Q(4, :));mean(Q(5, :));mean(Q
(6, :))];
    in=[uinit;R];
    in5(:, :, i)=[in];
end

inputs5=reshape(in5,69,234);
inputs5=removeconstantrows(inputs5);
fault5=5*ones(1,234);

%large booster anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault6'}];
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1, :));mean(Q(2, :));mean(Q(3, :));mean(Q(4, :));mean(Q(5, :));mean(Q
(6, :))];
    in=[uinit;R];
    in6(:, :, i)=[in];
end

inputs6=reshape(in6,69,234);
inputs6=removeconstantrows(inputs6);

fault6=6*ones(1,234);

```

```

%=====HPC ANOMALIES=====
%small HPC anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault7']};
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];

    in7(:,:,i)=[in];
end

inputs7=reshape(in7,69,234);
inputs7=removeconstantrows(inputs7);
fault7=7*ones(1,234);

%medium HPC anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault8']};
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in8(:,:,i)=[in];
end

inputs8=reshape(in8,69,234);
inputs8=removeconstantrows(inputs8);
fault8=8*ones(1,234);

%large HPC anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault9']};
end

```

```

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in9(:,:,i)=[in];
end

inputs9=reshape(in9,69,234);
inputs9=removeconstantrows(inputs9);
fault9=9*ones(1,234);

%=====COMBUSTOR ANOMALIES=====
%small combustor anomaly, all flight phases

for k=1:234
    filenamees(k)={['map' int2str(k) 'fault10'}];
end

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in10(:,:,i)=[in];
end

inputs10=reshape(in10,69,234);
inputs10=removeconstantrows(inputs10);
fault10=10*ones(1,234);

%medium combustor anomaly, all flight phases
for k=1:234
    filenamees(k)={['map' int2str(k) 'fault11'}];
end

for i=1:234,
    load(filenamees{i});
    Q=Q';

```

```

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];

    in11(:,:,i)=[in];
end

inputs11=reshape(in11,69,234);
inputs11=removeconstantrows(inputs11);
fault11=11*ones(1,234);

%large combustor anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault12'}];
end

for i=1:234,
    load(filenames{i});

    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];

    in12(:,:,i)=[in];
end

inputs12=reshape(in12,69,234);
inputs12=removeconstantrows(inputs12);
fault12=12*ones(1,234);
%=====HPT ANOMALIES=====
%small HPT anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault13'}];
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in13(:,:,i)=[in];
end

```

```

inputs13=reshape(in13,69,234);
inputs13=removeconstantrows(inputs13);
fault13=13*ones(1,234);

%medium HPT anomaly, all flight phases
for k=1:234

    filenames(k)={['map' int2str(k) 'fault14'}];
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in14(:, :, i)=[in];
end

inputs14=reshape(in14,69,234);
inputs14=removeconstantrows(inputs14);
fault14=14*ones(1,234);

%large HPT anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault15'}];
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in15(:, :, i)=[in];
end

inputs15=reshape(in15,69,234);
inputs15=removeconstantrows(inputs15);
fault15=15*ones(1,234);

%=====LPT ANOMALIES=====
%small LPT anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault16'}];
end

```

```

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];

    in16(:,:,i)=[in];
end

inputs16=reshape(in16,69,234);
inputs16=removeconstantrows(inputs16);
fault16=16*ones(1,234);

%medium LPT anomaly, all flight phases
for k=1:234
    filenamees(k)={['map' int2str(k) 'fault17'}];
end

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in17(:,:,i)=[in];
end

inputs17=reshape(in17,69,234);
inputs17=removeconstantrows(inputs17);

fault17=17*ones(1,234);

%large LPT anomaly, all flight phases
for k=1:234
    filenamees(k)={['map' int2str(k) 'fault18'}];
end

for i=1:234,
    load(filenamees{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in18(:,:,i)=[in];
end

```

```

inputs18=reshape(in18,69,234);
inputs18=removeconstantrows(inputs18);
fault18=18*ones(1,234);

%=====NO ANOMALY=====
%no anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault19'}];
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in19(:,:,i)=[in];
end

inputs19=reshape(in19,69,234);
inputs19=removeconstantrows(inputs19);
fault19=19*ones(1,234);

%=====UNKNOWN ANOMALY=====
%no anomaly, all flight phases
for k=1:234
    filenames(k)={['map' int2str(k) 'fault20'}];
end

for i=1:234,
    load(filenames{i});
    Q=Q';

R=[mean(Q(1,:));mean(Q(2,:));mean(Q(3,:));mean(Q(4,:));mean(Q(5,:));mean(Q
(6,:))];
    in=[uinit;R];
    in20(:,:,i)=[in];
end

inputs20=reshape(in20,69,234);
inputs20=removeconstantrows(inputs20);
fault20=20*ones(1,234);

```

```

%=====
%Generating actual quality parameters (all faults) for each map point

for m=1:234,
    for f=1:20,
        files(m,f)={['map' num2str(m) 'fault' num2str(f)}];%creates 234x20
cell of filenames
    end
end

for m=1:234
    for i=1:20,
        load(files{m,i});
        Q=Q';
        f=(i)*ones(1,2001);
        in=[Q;f];
        a(:,:,i)=[in];%7x2001x20
    end

faultdata=strcat('map',num2str(m),'Quals234');
    m=reshape(a,7,40020);
    save(faultdata,'m')%each map point has qualities saved for all faults.
files named 'map#Quals'
end

%===== PNN training =====

inputs=[inputs1 inputs2 inputs3 inputs7 inputs8 inputs9 inputs10 inputs11
inputs12 inputs13 inputs14 inputs15 inputs16 inputs17 inputs18 inputs19
inputs20];
mapinputs=inputs(1:9,:);

maptargets=[map map map map map map map map map map map map map map
map map];
Tc1=ind2vec(maptargets);

%===map network=====
mapnet=newpnn(mapinputs,Tc1,0.7)
save MapNet mapnet
%simulating map point identification
M = sim(mapnet,mapinputs)
Mc = vec2ind(M)

isequal(maptargets,Mc)

% %generating sampling points for testing map network
% sample=randperm(1326);
% for i=1:1326
%     maptestin(:,i)=[mapinputs(:,sample(i))];

```

```

% %      maptesttargets(i)=[maptargets(sample(i))];
% end
% Mtest=sim(mapnet,maptestin)
% Mt=vec2ind(Mtest)
% isequal(maptesttargets,Mt)
%=====
%=====

%fault PNNs
%this loop generates training sets for each of the 234 fault PNNs
for n=1:234,
    PNNin(:,:,n)=[inputs1(10:15,n) inputs2(10:15,n) inputs3(10:15,n)
inputs7(10:15,n) inputs8(10:15,n) inputs9(10:15,n) inputs10(10:15,n)
inputs11(10:15,n) inputs12(10:15,n) inputs13(10:15,n) inputs14(10:15,n)
inputs15(10:15,n) inputs16(10:15,n) inputs17(10:15,n) inputs18(10:15,n)
inputs19(10:15,n) inputs20(10:15,n)];
end

faults=[1:3,7:20];
Tc2=ind2vec(faults);
for j=1:234

net=newpnn(PNNin(:,:,j),Tc2,0.7);%creates PNN for map point j
    savenet=strcat('234faultnet',num2str(j),'.mat');
    save (savenet, 'net');
end

%testing each fault net
for i=1:234
    nets(i)={'234faultnet' int2str(i)};
end

%simulating fault identification using training qualities (mean qualities)
total=0;
for i=1:234,
    count=0;
    load(nets{i})
    F = sim(net,PNNin(:,:,i));
    Fc = vec2ind(F)
    isequal(Fc,faults)
    for k=1:17
        true=isequal(Fc(k),faults(k));
        count=count+true;
    end
    accuracy=count/17*100; %percent accuracy of each PNN

total=total+accuracy;
AvgTrainAccuracy=total/234 %average percent accuracy of all 234 PNNs

```

```

end

% %simulating fault identification using ACTUAL qualities
% total=0;
% for i=1:234,
%     count=0;
%     load(nets{i})
%     F = sim(net,PNNin(:, :, i));
%     Fc = vec2ind(F)
%     isequal(Fc,faults)
%     for k=1:17
%         true=isequal(Fc(k),faults(k));
%         count=count+true;
%     end
%     accuracy=count/17*100 %percent accuracy of each PNN
%     total=total+accuracy;
%     AvgAccur=total/234 %average percent accuracy of all 234 PNNs
% end

%=====
%trying to loop the testing of all nets using actual qualities
%currently set to test each manually!

clear;clc
for j=1:234,
    map(j)={['map' num2str(j) 'Quals234']};
    nets(j)={['234faultnet' int2str(j)]};
end
total=0;
for i=1:234,
    count=0;count2=0;count3=0;
    count4=0;count5=0;count6=0;
    count7=0;count8=0;count9=0;count10=0;count11=0;count12=0;count13=0;
    count14=0;count15=0;
    count16=0;count17=0;count18=0;count19=0;count20=0;
    load(nets{i});
    load(map{i});

%     load faultnet2
    F = sim(net,m(1:6,:));
    Fc = vec2ind(F);
    out=[m(7,:) ' Fc'];
    %     isequal(m(7,1:6003),Fc(1:6003))
    %     isequal(m(7,12007:40020),Fc(12007:40020))
    for i=1:2001
        true=isequal(Fc(:,i),[1]);
        count=count+true;
    end
end

```

```

for i=2002:4002
    true=isequal(Fc(:,i),[2]);

    count2=count2+true;
end

for i=4003:6003
    true=isequal(Fc(:,i),[3]);
    count3=count3+true;
end
for i=6004:8004
    true=isequal(Fc(:,i),[4]);
    count4=count4+true;
end

for i=8005:10005
    true=isequal(Fc(:,i),[5]);
    count5=count5+true;
end

for i=10006:12006
    true=isequal(Fc(:,i),[6]);
    count6=count6+true;
end
for i=12007:14007
    true=isequal(Fc(:,i),[7]);
    count7=count7+true;
end

for i=14008:16008
    true=isequal(Fc(:,i),[8]);
    count8=count8+true;
end
for i=16009:18009
    true=isequal(Fc(:,i),[9]);
    count9=count9+true;
end
for i=18010:20010
    true=isequal(Fc(:,i),[10]);
    count10=count10+true;
end

for i=20011:22011
    true=isequal(Fc(:,i),[11]);
    count11=count11+true;
end
for i=22012:24012
    true=isequal(Fc(:,i),[12]);
    count12=count12+true;
end

```

```

end

for i=24013:26013
    true=isequal(Fc(:,i),[13]);
    count13=count13+true;
end
for i=26014:28014
    true=isequal(Fc(:,i),[14]);
    count14=count14+true;
end

for i=28015:30015
    true=isequal(Fc(:,i),[15]);
    count15=count15+true;
end
for i=30016:32016
    true=isequal(Fc(:,i),[16]);
    count16=count16+true;
end

for i=32017:34017
    true=isequal(Fc(:,i),[17]);
    count17=count17+true;
end

for i=34018:36018
    true=isequal(Fc(:,i),[18]);
    count18=count18+true;
end
for i=36019:38019
    true=isequal(Fc(:,i),[19]);
    count19=count19+true;
end
for i=38020:40020
    true=isequal(Fc(:,i),[20]);
    count20=count20+true;
end

TESTaccuracy=(count+count2+count3+count7+count8+count9+count10+count11+count12+count13+count14+count15+count16+count17+count18+count19+count20)/34017*100;
    total=total+TESTaccuracy;
end
AvgTestAccuracy=total/234 %average percent accuracy of all 234 PNNs

```