NASA/CR—2008-215273

Model-Based Fault Tolerant Control

Aditya Kumar and Daniel Viassolo
GE Global Research, Niskayuna, New York

September 2008

NASA/CR—2008-215273

Model-Based Fault Tolerant Control

Aditya Kumar and Daniel Viassolo
GE Global Research, Niskayuna, New York

Prepared under Contract NAS3–01135, Task order 3

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

September 2008

Acknowledgments

This project was funded by the NASA Aviation Safety and Security Program under the RASER contract NASA3-01135. We gratefully acknowledge the project supervision and feedback of Don Simon and Tak Kobayashi at NASA Glenn Research Center, and Kiyoung Chung and Matt Wiseman at General Electric Aircraft Engines.

Trade names and trademarks are used in this report for identification only. Their usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Level of Review: This material has been technically reviewed by NASA technical management.

Available from

NASA Center for Aerospace Information
7115 Standard Drive
Hanover, MD 21076–1320

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161

Available electronically at http://gltrs.grc.nasa.gov

Table of Contents

List of Tables

List of Figures

Model-Based Fault Tolerant Control

Aditya Kumar and Daniel Viassolo
GE Global Research
Niskayuna, New York 12301

1. Abstract

The Model Based Fault Tolerant Control (MBFTC) Phase II program was conducted under the NASA Aviation Safety and Security Program (AvSSP). The goal of the MBFTC is to develop and demonstrate aircraft engine Model-Based Fault Tolerant Control strategies. The specific objective is to accommodate anomalous behavior such as sensor faults, actuator faults, or turbine gas-path component damage that can lead to in-flight shutdowns (IFSD) or aborted takeoffs (ATO), asymmetric thrust/loss of thrust control (LOTC) or engine surge/stall events. Such events, often when coupled with pilot error, can compromise aircraft safety. In this program, we selected four different faults as prime candidates for MBFTC on-line detection and accommodation. These faults included a sensor (Compressor discharge pressure, PS3), an actuator (variable geometry, VG) and two component (high pressure compressor, HPC, and high pressure turbine, HPT) faults. For these faults, we developed a suite of model-based fault detection algorithms and evaluated their performance over the entire flight envelope and in the presence of engine-to-engine variation and deterioration. Based on the performance and maturity of the developed algorithms two approaches were selected: (i) multiple-hypothesis testing and (ii) neural networks; both used residuals from an Extended Kalman Filter (EKF) to detect the occurrence of the selected faults. A simple fusion algorithm was also implemented to combine the results from each algorithm to obtain an overall estimate of the identified fault type and magnitude. This fusion algorithm combined the complementary performances of the two algorithms to yield improved performances for HPC and HPT faults in comparison to either individual algorithm. Moreover, the fusion algorithm reduced the incidence of false alarms and miss-classified faults compared to either individual algorithm. The identification of the fault type and magnitude thus enabled the use of an online fault accommodation (FA) to correct for the adverse impact of these faults on the engine operability thereby enabling continued engine operation in the presence of these faults. FA restored thrust and component stall margin through control accommodation actions. We developed and implemented these fault detection and fault accommodation algorithms in a FADEC Simulation (FSIM) environment. Finally, we modified the implemented algorithms, especially the implementation of the EKF for fault detection and accommodation to address real-time implementation on the actual FADEC. The performance of the fault detection and accommodation algorithms were extensively tested with the high fidelity Cycle Workstation (CWS) model and simplified Component Level Model (CLM).

2. Introduction

2.1 Objective

The goal of the Model-Based Fault Tolerant Control (MBFTC) program is to improve the operation of aircraft engines in the presence of faults or anomalous behavior, and increase overall aviation safety. In particular, the objective of the Phase II of the MBFTC program is

1. To develop technologies for on-wing detection of critical faults in an aircraft engine system that may lead to in-flight shut downs (IFSD), aborted take-offs (ATO), asymmetric thrust/loss of thrust control (LOTC) or engine stall/surge.
2. To automatically employ fault tolerant control to mitigate the adverse impact of identified faults, thereby allowing continued and safe operation of the engine and aircraft.
3. To demonstrate the application of the developed model-based fault detection and accommodation technologies in a simulation/rig environment.

2.2 Tasks

This was a 4-year (2002–2005) MBFTC Phase II program continuing from MBFTC Phase I. In Phase I of the program, GEAE conducted a field event study and identified engine fault conditions, which have led to anomalous engine behavior. A single fault was then selected, modeled, and used for the simulation demonstration of a fault accommodating control strategy. This work was expanded to address additional faults, enhance the model-based diagnostic and control logic, and demonstrate the MBFTC technology on a ground-based (FADEC Simulation, FSIM) engine simulation using appropriate real-time engine models. To this end, the overall program was organized according to the following high-level tasks:

Task 1—Engine Model Generation. This task entailed the development of a suitable real-time transient engine model to form the basis of MBFTC algorithms. Specifically, a component level model (CLM) was generated from a high-fidelity cycle workstation (CWS) model, which enabled the implementation and testing of developed MBFTC algorithms through simulations.

Task 2—Engine Fault Selection and Modeling. This task entailed the identification of a few key engine sensor, actuator and gas path component faults based on historical data on the occurrence and significance of these faults. The identified faults were included in the engine model (CLM and CWS) to simulate the impact of these faults and enable MBFTC algorithm development and validation.

Task 3—Robust Fault Tolerant Control Development. This is the core technical task that entailed (i) development of model-based fault detection algorithms for on-wing detection of selected faults, and (ii) development of fault tolerant control algorithms to adapt the existing FADEC logic to accommodate the identified fault, i.e., mitigate the adverse impact of the fault on engine operability and performance to enable continued and safe engine operation.

Task 4—Demonstration of MBFTC Technology on a Relevant Test Environment. This task was to entail the maturation of the developed MBFTC algorithms and implementation on a test platform, i.e., a dry rig, with the developed algorithms running on a FADEC in real time. To this end, the developed algorithms were modified and implemented in a FADEC simulation (FSIM) environment with the CLM used to simulate the engine and actual FADEC software verified through closed-loop simulations. The FSIM implementation was modified to address computational limitations of the FADEC hardware towards real-time implementation. Due to facility access issues, the MBFTC algorithms were not tested in a dry rig environment.

2.3 Report Layout

This report is organized into individual sections documenting the technology development and results of this program for each task mentioned above.

Section 3 describes the overall MBFTC approach developed and implemented in this program. Details of the specific elements in the MBFTC approach are provided in the subsequent sections. Section 4 discusses the two main models used in this program: (i) the high-fidelity cycle workstation (CWS) model, and (ii) component level model (CLM) simplified from the CWS model to enable real-time implementation as an embedded engine model. Section 5 describes the selection of the four key engine sensor, actuator and gas-path component faults addressed in this program, and modeling of these faults in the CWS and CLM. Section 6 describes the various model-based fault detection algorithms and their performance assessed through extensive CWS simulations. Section 7 describes the overall fault accommodation strategy and documents the results for fault accommodation for the selected faults through CWS simulations.

Section 8 describes the implementation and validation of the developed models, model-based fault detection and fault accommodation algorithms in a FADEC simulation (FSIM) environment, and the modifications implemented to address real-time implementation issues for the FADEC hardware. Finally, section 9 concludes the results of this program, with some general thoughts on future directions/extension of the developed technology.

3. Overall MBFTC Approach

In this section, we present the overall MBFTC approach developed and implemented in this program for model-based detection and isolation of faults and automated fault accommodation based on the identified fault. It should be mentioned that the developed technology builds upon the existing FADEC logic that provides the nominal control logic for an un-faulted engine system. The developed approach modifies this existing FADEC logic upon identification of a fault to adapt the control logic for that particular identified fault.

Figure 1 shows the overall MBFTC approach. The developed algorithms are implemented in the FADEC hardware on top of the existing FADEC logic for un-faulted engine. The new MBFTC technology elements for model-based fault detection and accommodation are highlighted with blue blocks. Specifically, the developed approach uses an Extended Kalman Filter (EKF) with an embedded engine model (CLM) to generate residuals for all sensed engine outputs. These sensor residuals from the EKF are then processed, by one or more fault detection algorithms, to detect and isolate the fault type, confidence and fault magnitude. The results from each individual fault detection algorithm are combined through a fusion algorithm to obtain the overall fault type, confidence and magnitude. Based on the identified fault type an appropriate accommodation strategy is employed. In particular, for non-sensor (engine/actuator) faults, optimum FADEC adjustments are evaluated through pre-computed look-up tables based on the fault type and magnitude and added to the baseline FADEC outputs (control actuator commands) for an un-faulted engine. The resulting net command for the control actuators is optimal for mitigating the adverse impact of the identified fault type and magnitude. On the other hand, for sensor faults, e.g., a pressure sensor fault, another EKF is used to optimally estimate the correct value from the remaining sensors and used in the existing FADEC logic instead of the faulty pressure sensor value. The details on the fault selection, modeling, fault detection and fusion algorithms and the fault accommodation strategy are described in the following sections.

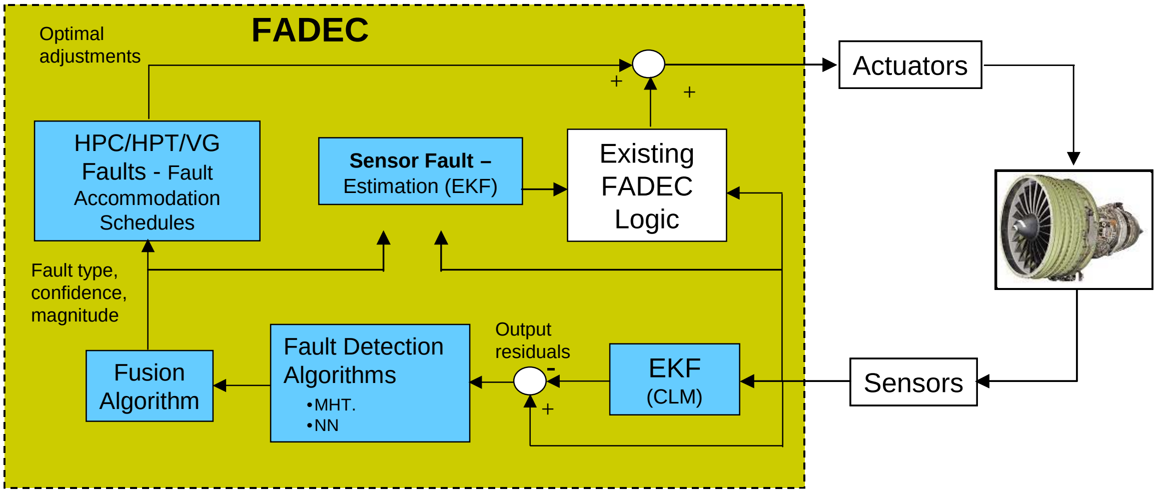

FADEC
Optimal adjustments
+
+
Actuators
HPC/HPT/VG Faults - Fault Accommodation Schedules
Sensor Fault – Estimation (EKF)
Existing FADEC Logic
Fault type, confidence, magnitude
Output residuals
-
+
Fusion Algorithm
Fault Detection Algorithms
•MHT.
•NN
EKF (CLM)
Sensors

Figure 1.—Overall MBFTC approach.

4. Engine Models

This section describes the transient engine models used in this program to develop and test the MBFTC algorithms for selected faults. In particular, we used two kinds of transient models (i) a high-fidelity cycle workstation model (CWS) and (ii) a component level model (CLM). While the CWS model is the highest-fidelity model available, it is computationally very intensive and not amenable to real-time implementations. Thus, the CWS model was simplified to generate a CLM that avoids expensive computations (e.g., iterative solutions) to enable real-time implementations with a minimal impact on simulation accuracy. The CWS model was used extensively to generate fault data used for development and testing of fault detection algorithms. The developed MBFTC algorithms used the CLM as an embedded engine model for online fault detection and accommodation.

4.1 Cycle Workstation (CWS) Model

The CWS model is a high-fidelity non-real-time model of the aircraft engine. It is a detailed physics-based transient model of the engine, and is validated against extensive engine data. While the CWS model is very accurate for steady state and transient simulation, its complexity prohibits real-time implementation. In particular, it involves an iterative solution of algebraic equations that is not amenable for real-time environments. The CWS model was used extensively in this program to generate data from nominal and faulted engines over the entire flight envelope, and this data was used to develop and validate the fault detection algorithms. In order to simulate the CWS with variation sources like engine-to-engine variation and engine degradation/deterioration, appropriate engine-to-engine variation and deterioration models were incorporated in the baseline CWS model. The following parameters are modified to simulate the effects of engine-to-engine variation in the engine.

The existing deterioration model altered certain parameters including HPT scalers and adders. A more accurate model replaces the aforementioned variables and modifies the cold clearance to propagate deterioration through the gas path. The following variables are altered to simulate deterioration.

Also, models for the selected faults were added to the CWS model to enable simulations with these faults—the fault models are described later in section 5.

TABLE 1.—ENGINE-TO-ENGINE VARIATION PARAMETERS

Description	Variable
Efficiency adder for fan	DE13D
Efficiency adder for booster	DE23D
HPC efficiency adder	DE3D
HPT efficiency adder	DE42D
LPT efficiency adder	DE5D
CDP Seal	D3W0
CDP Front Seal	D3W42
Fan Flow Scalar	SW2AR
Booster Flow Scalar	SW2R
HPC flow multiplier	SW25R
Delta Cold Clearance	DTCLS1

TABLE 2.—DETERIORATION PARAMETERS

Description	CWS Input
HPC Efficiency	DE3D
LPT Efficiency	DE5D
HPC Flow Scalar	SW25R
Front Seal Flow Adder	D3W0
CDP Seal Flow Adder	D3W42
Cold clearance	CLRC (1)
Cold clearance	CLRC (2)

4.2 Component Level Model (CLM)

Since the program is focused on model-based fault tolerant control, a model is at the heart of our control and detection algorithms. For this purpose we are using a component level model. The model used for this program is a nonlinear simulation of an advanced commercial high-bypass twin-spool turbofan engine. This model can be run both in a transient and steady state modes. An interface from Matlab to call the CLM was created to enable rapid development and testing of the fault detection and accommodation algorithms. Some of the details of the model are described in the following sections.

4.2.1 CLM Model Structure

The top-level inputs and outputs to the CLM are shown in figure 2.

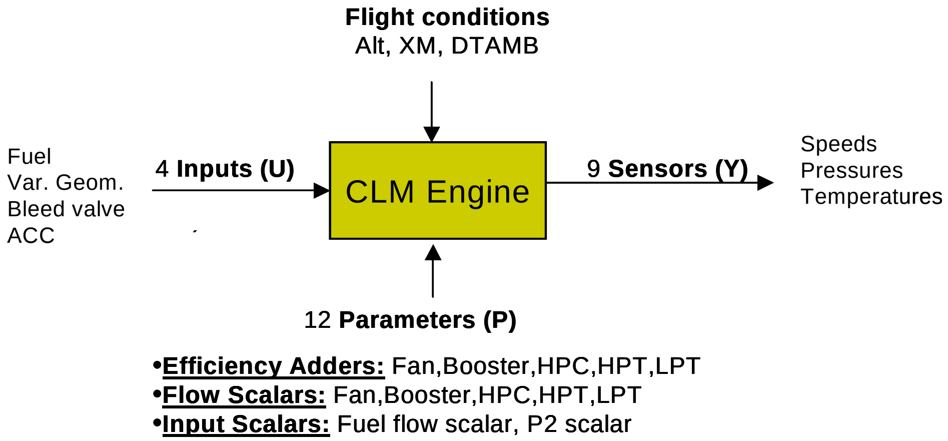


Figure 2.—CLM inputs, parameters and sensors.

The nonlinear CLM model structure is represented by

$$\begin{aligned} \dot{x} &= f(x,z,u,p) \\ y_r &= g(x,z,u,p) = 0 \\ y &= h(x,z,u,p) \end{aligned}, \qquad (1)$$

where *x* is the states, *z* is the guesses, *u* is the inputs, and *p* are the health parameters.

4.2.2 Inputs

Table 3 describes the inputs needed to drive the model for transient and steady state operation.

TABLE 3.—MODEL INPUTS

Name	Description
WF	Fuel flow
BV	Bleed Valve
VG	Variable Geometry Guide Vanes
ACC	Clearance control
ALT	Altitude
XM	Free stream mach number
DTAMB	Deviation from iso standard day temperature

4.2.3 States

The CLM contains, as dynamic states, the speeds for low pressure (LP) and high pressure (HP) rotors, as well as heat soak metal temperatures in individual engine components, and metal temperatures for the thermal clearance model in the HP turbine. In addition to these states in the engine, there are dynamic lag states associated with the pressure and temperature sensors.

4.2.4 Parameters

Table 4 lists all of the health parameters that are used as inputs to the CLM.

TABLE 4.—MODEL PARAMETERS

Name	Description
SCLWF	Scalar multiplier for fuel flow
SCLP2	Scalar multiplier for P2 measurement
DE13D	Efficiency adder for fan
SW12R	Flow multiplier for fan
DE23D	Efficiency adder for booster
SW2R	Flow multiplier for booster
DE3D	HPC efficiency adder
SW25R	HPC flow multiplier
DE42D	HPT efficiency adder
SW41R	HPT flow multiplier
DE5D	LPT efficiency adder
SW49R	LPT flow multiplier

4.2.5 Sensors

The engine has a total of 9 sensors, including the 2 rotor speeds and pressures and temperatures at different points in the engine. The CLM includes these sensors as outputs.

4.2.6 CLM Solver

The guesses (*z*) are due to the algebraic loops inside of the CLM and the values can be calculated using methods like Newton-Raphson (what GEAE calls SLAM) or approximations using a constant matrix (what GEAE calls xiiter). The top level Newton-Raphson method is shown in equation (2). Here the function is linearized about z_n and a Newton step is accomplished to determine the new value of the guess at z_{n+1}. Using SLAM this process is continued until the value of the error term (y_r) is within some tolerance. For this model the CLM is using xiiter, which is a non-iterative approximation to SLAM. Instead of re-linearizing or determining the sensitivities at each point a constant matrix *L* is used for all points, and instead of continuing the iteration until the tolerance is below some pre-specified value the CLM uses only a single (or small user defined number (parameter nxiter in the CLM)) iteration. Both of these simplifications reduce the accuracy of the model but they speed up the operation of the CLM.

$$\begin{aligned} & y_r = f(z) \\ & \Delta y_r = \left.\frac{\partial f}{\partial z}\right|_{z_n} \Delta z = L\Delta z, \quad \Delta y_r = y_{n+1} - y_n, \quad \Delta z = z_{n+1} - z_n \\ & z_{n+1} = L^{-1}\Delta y_r + z_n \end{aligned} \tag{2}$$

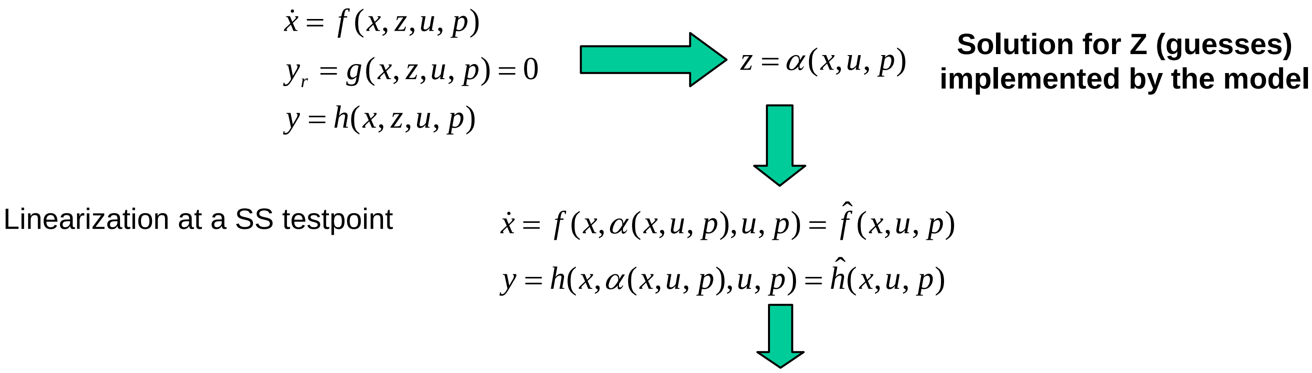
4.2.7 Linearization

For the MBFTC algorithms developed in this program, a linear state space model of the system is required, which can be obtained by linearizing the nonlinear CLM. However, problems arise with getting the linearized state-space (SS) model and calculating gain matrices due to the presence of the algebraic equations and variables (guesses) besides the standard state differential equations. The linearization approach needs to account for these algebraic equations to obtain a correct linear SS model. Two methods have been developed to create a linear model on-line at each time sample that addresses this issue in different manners. Approach #2 was selected as the method to use going forward with the program.

4.2.7.1 Linearization—Approach #1

In this approach, the linear SS model is obtained by directly perturbing the states *x*, the inputs *u*, and the parameters *p* one at a time. During each of the calls to the CLM with these perturbations, this approach relies on the model to solve for the guesses (*z*'s) correctly to keep the residuals of algebraic equations at 0. Accuracy problems occurred for this method if we only used one xiiter iteration (parameter nxiter in the CLM), since one iteration was insufficient to correctly update the *z*'s for each perturbation. We had to do between 40 to 100 iterations for each state and input to get an accurate result. This results in very long computation time for the creation of the linear models.

Figure 3 details the development of the linear model using approach #1.



$$\dot{x} = f(x, z, u, p)$$
$$y_r = g(x, z, u, p) = 0$$
$$y = h(x, z, u, p)$$

$$z = \alpha(x, u, p)$$

Solution for Z (guesses) implemented by the model

Linearization at a SS testpoint

$$\dot{x} = f(x, \alpha(x, u, p), u, p) = \hat{f}(x, u, p)$$
$$y = h(x, \alpha(x, u, p), u, p) = \hat{h}(x, u, p)$$

Figure 3.—Linearization approach #1.

4.2.7.2 Linearization—Approach #2

The second approach involves linearizing the system with respect to the algebraic equations and variables (guesses) as well, and obtaining the linearized SS model by solving the linearized equations for the algebraic variables (guesses). Since the algebraic equations are also linearized in this approach, xiiter is in fact disabled to disable updates in the algebraic variables (guesses *z*'s). Thus, this approach is much more computationally efficient.

Again, the state space linear model is determined by perturbing *x*, *u*, *p*, and in this case *z* as well using central differencing. Then the guesses (*z*) are solved for explicitly from the linearized algebraic equations and substituted back into the xdot and y (output) equations resulting in linear models that are functions of *x*, *u*, and *p*. Figure 4 details the development of the state space linear model using approach #2. It was found that approach #2 is more accurate and faster computationally so it was selected as the linearization process going forward with this program.

4.2.8 Variation, Deterioration, Bias, Noise

This section will discuss the various aspects of the engine operation that vary either between engines, as engines age, or through the measurement processes.

4.2.8.1 Engine-to-Engine Variation

An engine-to-engine variation model was created for the engine model. The engine-to-engine variation accounts for manufacturing and assembly variation found in new engines. These variations can be described or modeled by adding normally distributed variation to the engine component health parameters listed in table 4.

4.2.8.2 Deterioration

A deterioration model was created for the engine model. After evaluating analytic engine teardowns, production test data, development test data, and overhaul engine findings it was determined that the parameters shown in table 5 must be modified to model engine deterioration.

$$\begin{array}{l} \dot{x} = f(x,z,u,p) \\ y_r = g(x,z,u,p) = 0 \\ y = h(x,z,u,p) \end{array} \Longrightarrow \begin{array}{l} \dot{x} = Ax + B_z z + B_u u + B_p p \\ y_r = Kx + L_z z + L_u u + L_p p = 0 \\ y = Cx + D_z z + D_u u + D_p p \end{array} \Longrightarrow z = -(L_z)^{-1}(Kx + L_u u + L_p p)$$

$$\Longrightarrow \begin{array}{l} \dot{x} = (A - B_z L_z^{-1} K)x + (B_u - B_z L_z^{-1} L_u)u + (B_p - B_z L_z^{-1} L_p)p \\ y = (C - D_z L_z^{-1} K)x + (D_u - D_z L_z^{-1} L_u)u + (D_p - D_z L_z^{-1} L_p)p \end{array} \Longrightarrow \begin{array}{l} \dot{x} = \hat{A}x + \hat{B}_u u + \hat{B}_p p \\ y = \hat{C}x + \hat{D}_u u + \hat{D}_p p \end{array}$$

Figure 4.—Linearization approach #2.

TABLE 5.—DETERIORATION MODEL PARAMETERS

Description	CLM input	Units
HPC Efficiency	DE3D	pts
HPT Efficiency	DE42D	pts
LPT Efficiency	DE5D	pts
HPC Flow Scalar	SW25R	%
HPT Flow Scalar	SW41R	%
LPT Flow Scalar	SW49R	%
Front Seal	G3W0	%W25
CDP Seal Distress	G3W42	%W25
LPT Aft brush seal	G27W5	%W25

4.2.9 Sensor Accuracy

There are several factors that affect the measurement accuracy of any given sensor. Factors considered here are:

1. Signal conditioning—accounts for excitation, A/D conversion, filtering,
2. Sensor bias—accounts for sensor to sensor variation,
3. Profile error—accounts for radial and circumferential variation in the measured parameters,
4. Noise—accounts for noise in the system.

Figure 5 shows a diagram of a generic measurement system in a gas turbine to illustrate how the different components of the system are related and how each of the factors affecting accuracy could enter the measurement process.

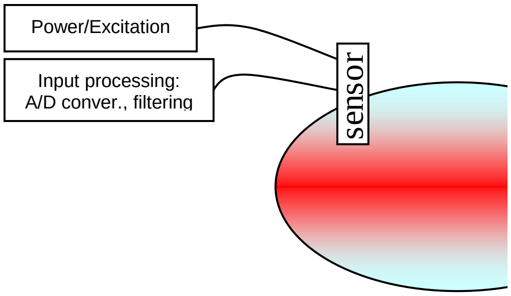


Figure 5.—Sensor accuracy diagram.

5. Fault Selection and Modeling

In this section, we document the selection of the four key sensor/actuator/engine component faults and their modeling in the CWS and CLM models.

5.1 Fault Selection

The purpose is to identify, and ultimately select, faults that are relevant for the MBFTC program. To this end, it was imperative to determine the potential faults that can be considered in that context and to which degree they are relevant to the safety of aircraft operation.

5.1.1 Selected Faults

A list of fault candidates for the propulsion system was generated, and the faults were grouped at a high level into six categories including actuator, FADEC, fuel system, gas path, oil system, and sensors. For this program, we are interested in the Actuator, Gas Path Turbomachinery, and Sensor system categories. The faults that were selected for consideration in the AvSSP program are:

1. Variable Geometry guide vanes (VG) – Actuators,
2. HPC Damage – Gas Path,
3. HPT Damage – Gas Path,
4. PS3 Pressure Sensor – Sensors.

Figure 6 shows the overall structure of selected faults in the engine-control system.

5.2 Modeling of Selected Faults

In this section, we describe in more detail the type of fault considered for each of the sensor/actuator/turbo-machinery fault types and their modeling in CWS and CLM models.

5.2.1 Variable Geometry (VG) Fault

For the variable geometry guide vanes, position errors and other losses of system actuation account for the majority of the system failures. The remaining faults result from localized failures at individual lever arm/vane sites. Since the lever arm failures would require three dimensional flow models to characterize and our current VG model is a lumped system level model, we cannot model the lever arm failures and excluded them from this project. The VG position error failures are modeled as a steady in-range bias and a drifting in-range bias. Table 6 shows the models for VG position error for small/medium/large faults.

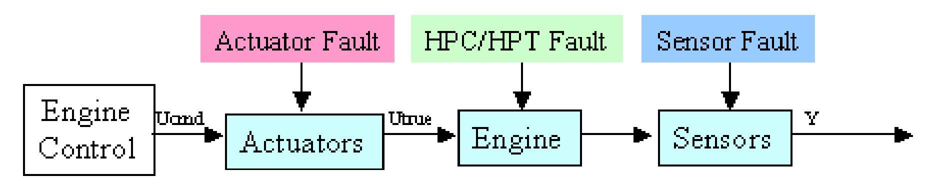


Figure 6.—Fault block diagram.

TABLE 6.—MODELING OF SMALL/MEDIUM/LARGE VG POSITION FAULTS

	CLM Inputs	Small Fault	Medium Fault	Large Fault
VG Fault	VG Bias	1.3%	4.2%	8.3%

5.2.2 HPC/HPT Fault

The HPC and HPT faults were modeled as changes in the corresponding component's efficiency adders and flow scalers. While on the GE military engine Survivable Engine Controls Algorithm Development (SECAD) program a large damage was considered of the order of 25% reduction in efficiency, for a commercial engine program a large efficiency reduction is on the order of 3 to 5%. Table 7 shows the damage models for small, medium, and large HPC and HPT faults.

TABLE 7.—HPC AND HPT FAULT MODEL PARAMETERS

		CLM inputs	Small Fault	Medium Fault	Large Fault
Compressor fault	Efficiency	DE3D	-1.50%	-3%	-5%
	Flow	SW25R	-1.50%	-3%	-5%
HPT fault	Efficiency	DE42D	-1.50%	-3%	-5%
	Flow	SW41R	-1.50%	-3%	-5%

5.2.3 PS3 Pressure Sensor Fault

A primary fault mode for the PS3 pressure sensor is a leak. The PS3 pressure sensor leak fault is modeled as a negative bias on the sensor output compared to the true value, and the model values for small, medium and large faults are given in table 8.

TABLE 8.—PRESSURE SENSOR FAULT MODELS

	Small	Medium	Large
Pressure Bias (%)	2.33	6.67	10.00

6. Model-Based Fault Detection

In this section, we describe the development and testing of fault detection algorithms for the four selected faults.

6.1 Fault Data Generation

We generated extensive data using the CWS/CLM models for engines with and without the selected faults. Initially, we generated data in the so-called "open-loop" setup where the control actuators were held constant at values for an un-faulted engine, despite the presence of a fault. However, this is not realistic since the FADEC logic does respond to a fault and consequently the control actuators vary with time. To capture this aspect, we re-generated the data in "closed-loop" setup allowing the FADEC to respond to the fault and vary the control actuators in a closed-loop configuration. In all we generated an extensive set of Monte Carlo simulations to reflect the presence of various sources of variation, and the presence of faults or different types and magnitudes. In all simulation runs, we initialized the engine at a steady-state point, and then injected a random fault type and magnitude, and simulate until reaching a new steady state corresponding to the particular fault.

6.1.1 Open-Loop Setup

We generated extensive data for an engine in the presence of variation arising from

- TRA, Alt, XM, DTAMB (point in flight envelope)
- Engine-to-Engine Variation
- Engine deterioration
- Sensor noise

for an un-faulted engine as well as a faulted engine for each combination of the four selected fault types and small, medium, large magnitude. To this end, initially, we ran the true engine in the so-called “open-loop” mode. More specifically, the true engine (CWS model) was run to steady-state conditions in a set of Monte Carlo runs in the presence of above-mentioned variation, but without any fault. For each case, the control actuators were then fixed and a random fault type and magnitude combination was injected to generate the full Monte Carlo simulation with 13 cases (1 no fault case + 12 cases with combinations of 4 fault types and 3 fault magnitudes). This data was then processed through an Extended Kalman Filter (EKF) using the CLM as the embedded engine model; the embedded CLM was set to correspond to a half-deteriorated engine to reflect the average engine over the entire fleet. The overall structure of this “open-loop” data generation setup is shown in figure 7. However, we realized that this data was not entirely correct, since it did not include the “closed-loop” response of the faulted engine as the FADEC logic responded to the injected fault, thereby changing the control actuators.

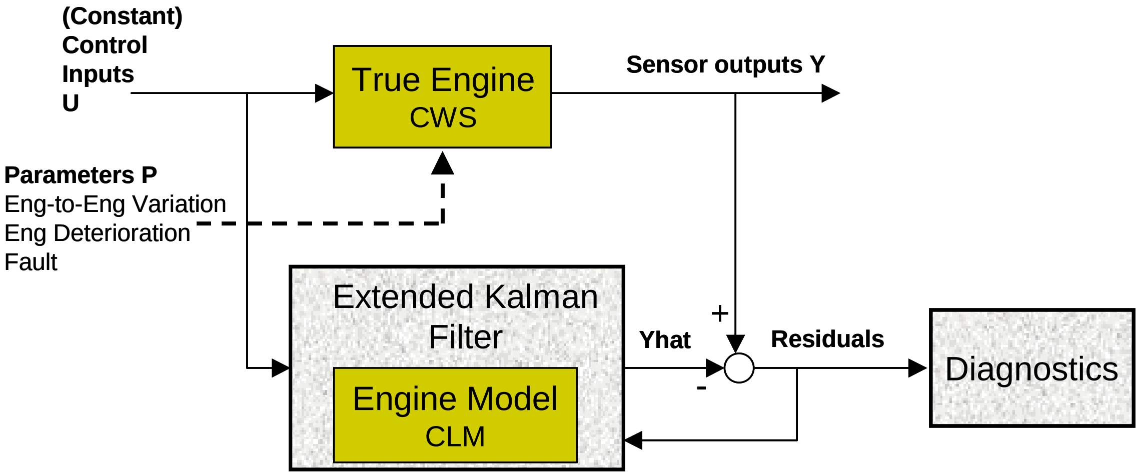


Figure 7.—Open-loop fault data generation setup.

6.1.2 Closed-Loop Setup

To more accurately reflect the fact that in the presence of a fault, the FADEC logic will respond to the impact of the fault on the sensed outputs, we re-generated the data in “closed-loop” configuration as shown in figure 8. In this closed-loop configuration, we ran a Monte Carlo simulation with 13 cases (1 no-fault case plus 12 cases with combinations of the 4 fault types and 3 fault magnitudes), with 2047 runs for each case reflecting random combinations of engine-to-engine variation (normally distributed variation of component health parameters listed in table 1), engine deterioration (uniformly distributed variation of component health parameters listed in table 5), and flight conditions (Alt, XM, DTAMB, TRA) over the flight envelope as shown in figure 9.

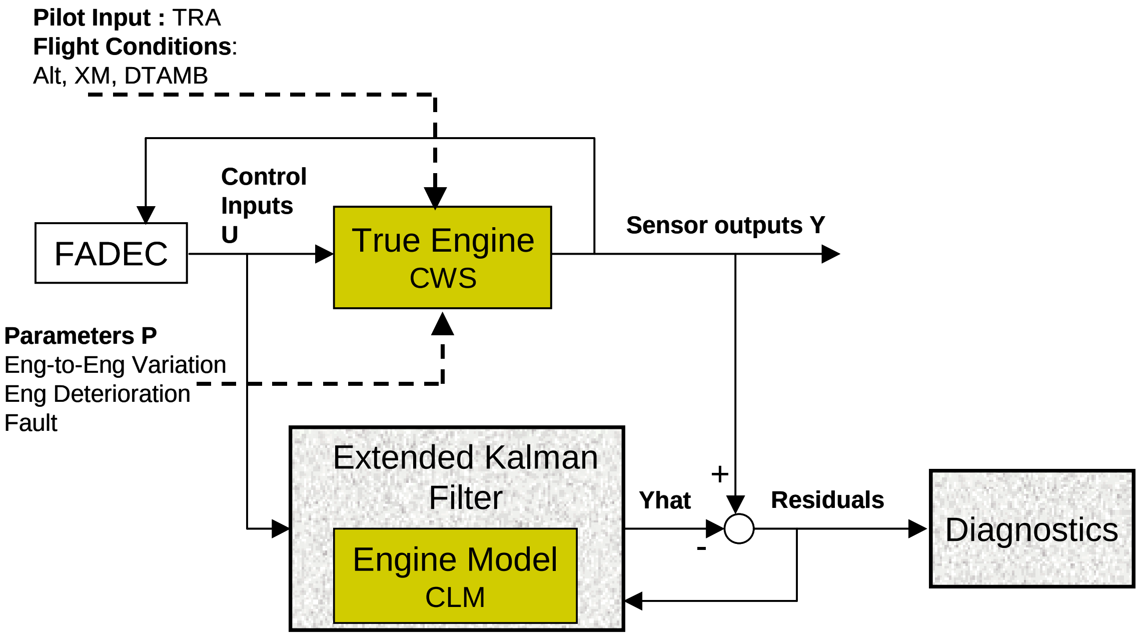


Figure 8.—Closed-loop fault data generation setup.

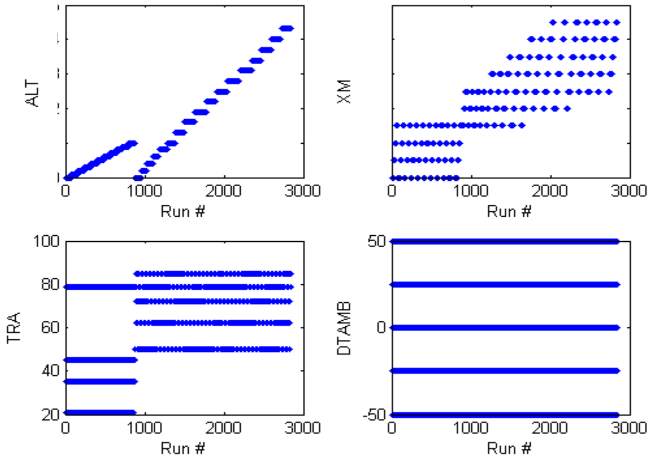


Figure 9.—Flight envelope points in fault data generation.

The generated data and the EKF residuals for each of these Monte Carlo runs were used to develop and test the performance of the fault detection algorithms.

6.2 Extended Kalman Filter

As mentioned before in section 3 and section 6.1, the model-based fault detection algorithms rely on using an Extended Kalman Filter to generate residuals for all sensed outputs, i.e., the mismatch between measured values and EKF estimates, which reflect the presence or absence of a fault. In particular, the fault detection has to be performed in the presence of variations arising from (i) flight conditions over the flight envelope, (ii) engine-to-engine variation, (iii) engine deterioration, and (iv) sensor noise. The use of a nonlinear embedded model (CLM) and the EKF accounts for the nonlinearities over the entire flight envelope. However, in this program, we are not adapting the embedded engine model (CLM) to match a specific engine, i.e., account for engine-to-engine variation and deterioration. Thus, the embedded engine model uses fixed health parameters corresponding to an average engine in the fleet, i.e., a half-deteriorated engine to avoid any systematic bias in the residuals. The residuals will have a non-zero mean and variance arising due to the un-matched model (engine-to-engine variation and deterioration). In the presence of a fault in the true engine, the residuals will vary in a manner dependent on the fault type and magnitude, thereby allowing the detection and isolation of the fault type and magnitude.

6.2.1 EKF implementation

The EKF is based on the *Component-Level Model* (CLM). This model is parameterized by states *x* and *z*, inputs *u*, and parameters *p*. Thus,

$$\begin{aligned} \dot{x}_t &= f(x_t, z_t, u_t, p_t), \\ 0 &= g(x_t, z_t, u_t, p_t). \end{aligned} \tag{3}$$

The constrained state variables, *z*, are defined implicitly by the second equation. Thus, the state equation is continuous-time, is in descriptor form and is inherently nonlinear. The output equation also is nonlinear.

$$y_t = h(x_t, z_t, u_t, p_t). \tag{4}$$

The state, *x*, comprises the spool speeds and heat soak temperatures followed by many clearance model states and a few sensor states. In final implementations of the EKF, we implement a reduced-order EKF estimating only 9 states, excluding any heat soak, clearance model and sensor lag states that are un-observable or have too fast dynamics.

The input, *u*, has five parts: 7 “guesses” for *z*, 3 unused elements, 4 control inputs, the environmental variables of altitude, mach number and ambient temperature, and the set of 12 parameters.

The CLM output, *y*, contains many intermediate variables, the three environmental variables, followed by the 9 sensed outputs. The update for the guess state variables, *z*, is reported in *y* at elements.

The parameter vector, *p*, contains the coefficients representing the engine wear state. This should capture the engine deterioration and the engine-to-engine variability. As mentioned above, these parameters are set to correspond to a half-deteriorated engine for the embedded CLM used in the EKF.

For the purposes of the EKF, we shall run the CLM in discrete time, with time index $t = k\,dt$. The time-update section of the EKF is given in standard form for the *x*-part of the equations

$$\hat{x}_{k+1|k} = \hat{x}_{k|k} + dt\; f(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k). \tag{5}$$

Here we have used Euler integration to move from continuous to discrete time.

To derive the update formula for the *z*-part of the state, we use a Newton-Raphson iteration to attempt to move *z* closer to satisfying the constraint. Thus,

$$\hat{z}_{k+1|k} = \hat{z}_{k|k} - \left(\left. \frac{\partial g}{\partial z} \right|_{\hat{x}_{k+1|k}, \hat{z}_{k|k}, u_k, p_k} \right)^{-1} g(\hat{x}_{k+1|k}, \hat{z}_{k|k}, u_k, p_k). \tag{6}$$

This results in the same algorithm as the alternative approach of attempting to keep *z* on the manifold defined by the constraint as *x* updates. This latter approach presumes that the constraint was satisfied at time *k* and then evolves *z* to stay on the constraint.

$$\begin{aligned}
0 &= g(\hat{x}_{k+1|k}, \hat{z}_{k+1|k}, u_k, p_k), \\
0 &= g(\hat{x}_{k|k} + \tilde{x}_k, \hat{z}_{k|k} + \tilde{z}_k, u_k, p_k), \\
0 &= g(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k) + \frac{\partial g}{\partial x}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k)\tilde{x}_k + \frac{\partial g}{\partial z}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k)\tilde{z}_k, \\
0 &= \frac{\partial g}{\partial x}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k)\tilde{x}_k + \frac{\partial g}{\partial z}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k)\tilde{z}_k, \\
\hat{z}_{k+1|k} - \hat{z}_{k|k} &= -\left(\frac{\partial g}{\partial z}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k) \right)^{-1} \frac{\partial g}{\partial x}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k)\tilde{x}_k, \\
&= -\left(\frac{\partial g}{\partial z}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k) \right)^{-1} \left[g(\hat{x}_{k+1|k}, \hat{z}_{k|k}, u_k, p_k) - g(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k) \right], \\
\hat{z}_{k+1|k} &= \hat{z}_{k|k} - \left(\frac{\partial g}{\partial z}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k) \right)^{-1} g(\hat{x}_{k+1|k}, \hat{z}_{k|k}, u_k, p_k).
\end{aligned} \tag{7}$$

The preference for the former derivation lies in its not assuming the constraint satisfaction and its clear connection to the Newton-Raphson method to minimize the deviation from the constraint. The *g* terms are the residuals representing the error in constraint satisfaction.

The measurement update section requires the linearized model for the computation of the Σ-matrix using the partial derivatives

$$F_k = \frac{\partial f}{\partial x}(\hat{x}_{k|k}, \hat{z}_{k|k}, u_k, p_k), \quad H_k = \frac{\partial h}{\partial x}(\hat{x}_{k|k-1}, \hat{z}_{k|k-1}, u_k, p_k). \tag{8}$$

[Note the differing indices on the state estimate arguments in these two terms.]
The Kalman gain is computed as follows.

$$\begin{aligned} K_k &= \Sigma_{k|k-1} H_k^T \left(H_k \Sigma_{k|k-1} H_k^T + R_k \right)^{-1} H_k \Sigma_{k|k-1} + Q_k, \\ \Sigma_{k+1|k} &= F_k \Sigma_{k-1|k} F_k^T - F_k \Sigma_{k|k-1} H_k^T \left(H_k \Sigma_{k|k-1} H_k^T + R_k \right)^{-1} H_k \Sigma_{k|k-1} F_k^T + Q_k. \end{aligned} \tag{9}$$

The state estimate measurement update is

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \left(y_k - \hat{y}_{k|k-1} \right), \\ &= \hat{x}_{k|k-1} + K_k \left(y_k - h(\hat{x}_{k|k-1}, \hat{z}_{k|k-1}, u_k, p_k) \right). \end{aligned} \tag{10}$$

In the global scheme of the EKF, one should also consider updating the constrained states to produce $\hat{z}_{k|k}$ from $\hat{x}_{k|k}$ and $\hat{z}_{k|k-1}$. However, in place of the linearized approach of EKF, we prefer to rely on the Newton-Raphson step above as the only adjustment. To do more would require computation of more terms such as $g\left(\hat{x}_{k|k}, \hat{z}_{k|k-1}, u_k, p_k\right)$ and a new derivative term. We shall rely on a single Newton update stage and take $\hat{z}_{k|k} = \hat{z}_{k|k-1}$.

In summary, the EKF algorithm implemented is as follows.

$$\begin{aligned} \hat{x}_{k+1|k} &= \hat{x}_{k|k} + dt\ f(\hat{x}_{k|k}, \hat{z}_{k|k-1}, u_k, p_k), \\ \hat{z}_{k+1|k} &= \hat{z}_{k|k-1} - \left(\left. \frac{\partial g}{\partial z} \right|_{\hat{x}_{k+1|k}, \hat{z}_{k|k-1}, u_k, p_k} \right)^{-1} g(\hat{x}_{k+1|k}, \hat{z}_{k|k-1}, u_k, p_k), \\ F_k &= \frac{\partial f}{\partial x}(\hat{x}_{k|k-1}, \hat{z}_{k|k-1}, u_k, p_k), \quad H_k = \frac{\partial h}{\partial x}(\hat{x}_{k|k-1}, \hat{z}_{k|k-1}, u_k, p_k), \\ K_k &= \Sigma_{k|k-1} H_k^T \left(H_k \Sigma_{k|k-1} H_k^T + R_k \right)^{-1} H_k \Sigma_{k|k-1} + Q_k, \\ \Sigma_{k+1|k} &= F_k \Sigma_{k-1|k} F_k^T - F_k \Sigma_{k|k-1} H_k^T \left(H_k \Sigma_{k|k-1} H_k^T + R_k \right)^{-1} H_k \Sigma_{k|k-1} F_k^T + Q_k, \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \left(y_k - h(\hat{x}_{k|k-1}, \hat{z}_{k|k-1}, u_k, p_k) \right). \end{aligned} \tag{11}$$

The central approximations made are in the computation of derivatives only at one point and the single Newton-Raphson correction of the constrained states, *z*.

6.2.2 EKF Design for Fault Detection

The ideal residuals for a given fault type and magnitude would all be “large”, i.e., distinguishable from a no-fault case, and lay on top of each other with very little variation. In addition, the residuals should be “different” for different types of faults and fault magnitudes to enable fault detection and isolation. However, the presence of engine-to-engine variation and deterioration in the true engine causes an engine-model mismatch (the embedded engine model in EKF uses a fixed set of health parameters) that leads to non-zero residuals even in the un-faulted case. This undesired variation in the EKF residuals masks the residuals corresponding to an actual fault, thereby making the fault detection and isolation difficult. Figure 10 shows the residuals for nine sensors for closed-loop data with a large HPT fault. These results are the baseline before doing any development to optimize the EKF tuning and generate residuals correlated with fault type and magnitude. Clearly, the sensor residuals are not very structured and have significant variability despite a fixed fault type (HPC) and fault magnitude (large).

Figure 11 shows an example set of fan speed sensor residuals in a specific location in the flight envelope for HPT fault. In particular, the four subplots to the left show the residuals for no fault (labeled as level 0) and small, medium and large faults labeled as levels 1, 2, and 3, respectively. All these 4 cases are plotted together in the subplot on the right side. It is clear, that there is no systematic pattern in the residual correlated with the presence and magnitude of this fault, thereby making its detection difficult.

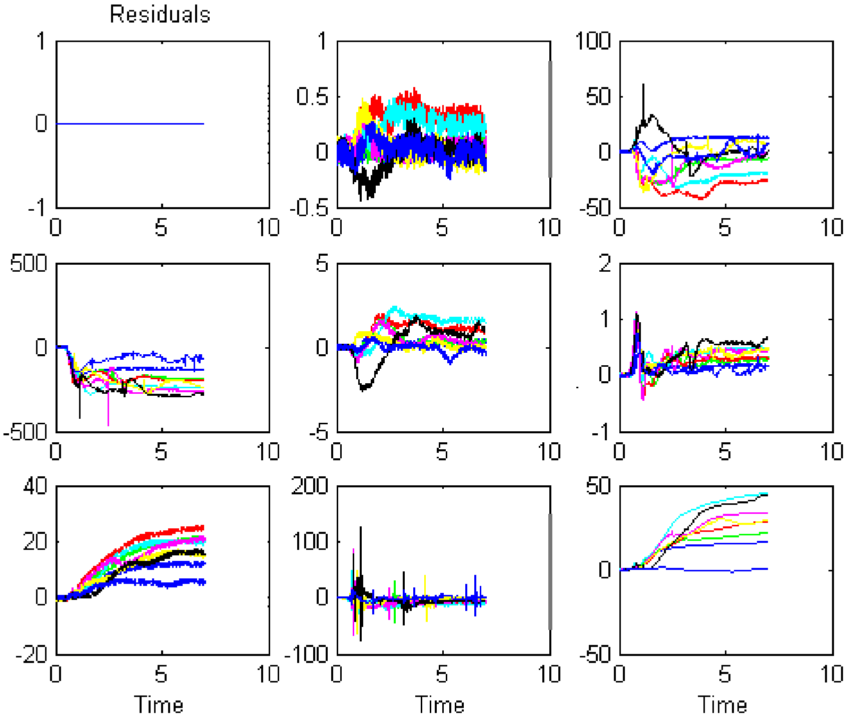


Figure 10.—EKF residuals for one regime for large HPT faults before optimal design.

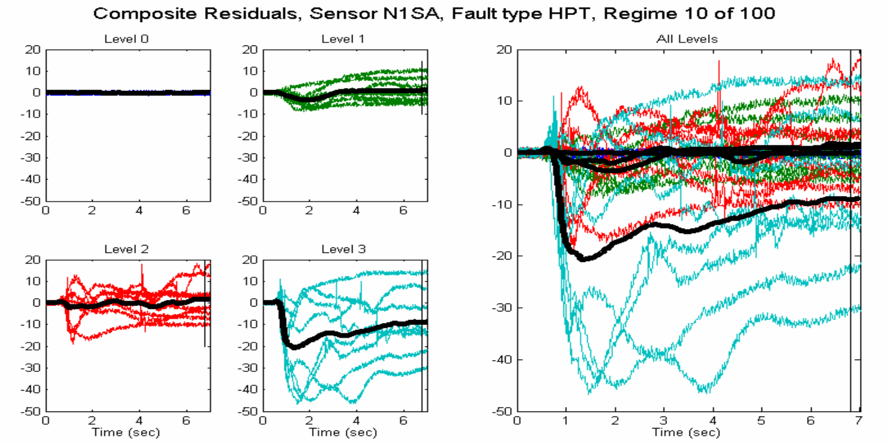


Figure 11.—EKF residuals in N1 sensor for HPT fault before optimal design.

As we stated, residuals corresponding to a given fault type and magnitude should all be “large” and lay on top of each other with very little variation, irrespective of flight conditions, engine-to-engine variations and deterioration. In addition, residuals should be different for different types of faults and magnitudes to allow fault isolation and identification of the fault magnitude. Note that the isolation of the fault type and fault magnitude is critical in enabling a correct fault accommodation. For the un-optimized EKF, the residuals do not show these desirable features due to the random engine-to-engine variation and deterioration we put into the closed-loop CWS runs when generating the fault data. Therefore, our goal now is to redesign the EKF to make it:

- Less sensitive to deterioration and engine-to-engine variation, and
- More sensitive to faults.

The EKF was optimized to meet the above objectives. The results of the residuals obtained with the optimally designed EKF are shown in figure 12. Clearly, we see large residuals in specific sensors for the large HPT fault, with small variation in these residuals despite variation from engine-to-engine variation and engine deterioration.

Figure 13 shows an example set of fan speed sensor residuals for HPC faults of different magnitudes in a specific section of the flight envelope, obtained by this optimally designed EKF. Clearly, it is apparent that the residuals for no fault case (blue) is zero mean and has a small variance, while the residuals for small, medium and large HPC faults show a systematic trend associated with the fault magnitude, as desired.

6.3 Fault Detection Algorithms

In this section, we describe the multiple fault detection algorithms that were developed in this program. These methods:

1. Crisp Rule (or abrupt change detection) method
2. Neural Network,
3. Rapid Fault detection
4. Multiple Hypothesis Testing

with the exception of the first method, rely on analyzing the EKF residuals obtained from the EKF described in the previous section, to detect and isolate a fault. The individual algorithms are described in detail in the following sections.

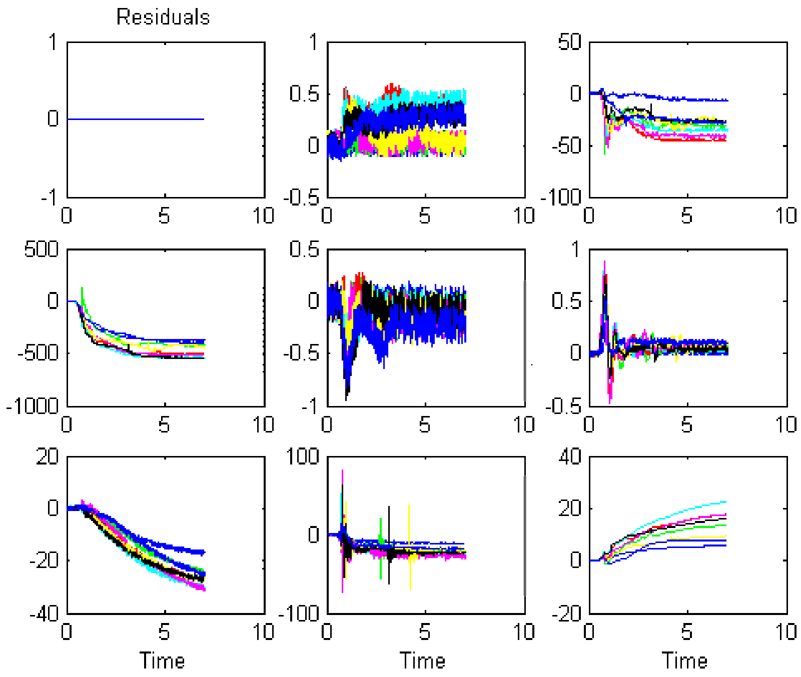


Figure 12.—EKF residuals for one regime for large HPT faults after optimal design.

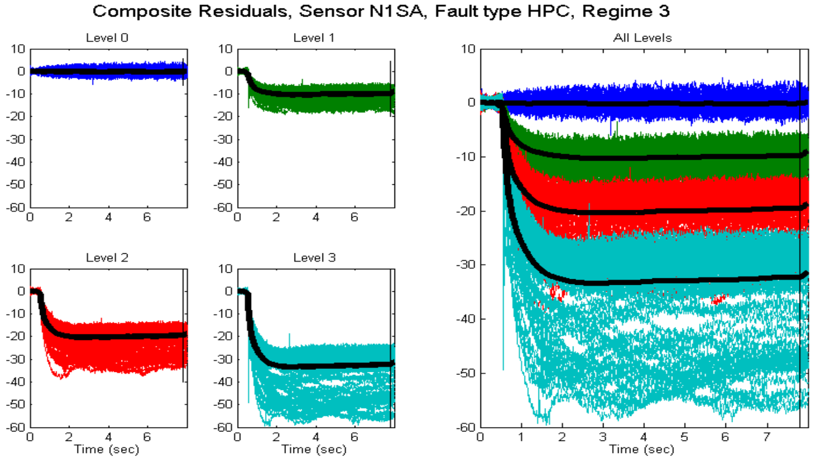


Figure 13.—EKF residuals in N1 sensor for HPC fault after optimal design.

6.3.1 Crisp-Rule Classification

Even quite subtle faults can be detected given enough time; however, for fault accommodation, faults generally need to be detected in a minimum amount of time. Thus, fault detection is a tradeoff between detection time and accuracy. Techniques that take a long time have a much higher criterion for classification accuracy, while techniques that are very fast necessarily have some tolerance for misclassification between faults. Crisp rule (cf. fuzzy rule) classification (CRC), is designed to use a fixed set of rules to detect faults quickly, with a bias toward classifying faults either correctly or as nominal (i.e., rather than misclassifying them as another fault), and with no tolerance for false alarms.

This approach operates on the raw sensed data (i.e., it relies on neither the model nor the EKF). The idea is to compare the sensed values from the current time step to the sensed values in the interval from two to three seconds ago[1].

The data are *z*-transformed, using the mean and standard deviation from the data in the interval from two to three seconds ago:

$$s_z = \frac{s - \overline{s}}{\sigma}.$$

The *z* transformation removes any engine variation, deterioration and flight envelope effects, leaving only sensor noise (with unit variation across all sensors) and the effect of faults in the data.

Figure 14, shows the performance of the fault detection using the closed-loop fault data.

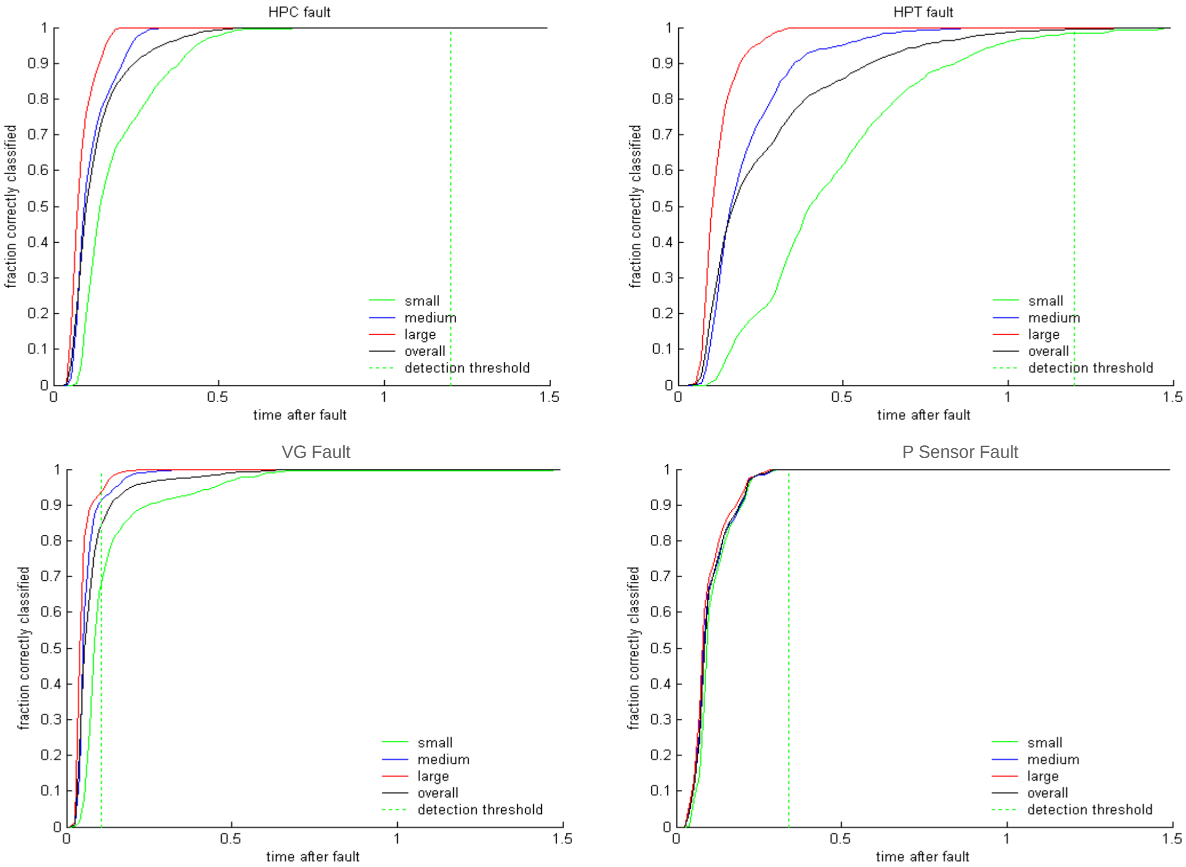


Figure 14.—Percent of faults correctly classified as a function of time for the closed-loop data.

[1] This particular window was chosen to reflect the faults in this study, but it is not inherent to the technique – a longer or shorter window will detect faults with slower or quicker onset, respectively

This method is very simple in that it is designed to detect rapid changes in the engine sensors at a steady-state operating condition, arising due to the faults. Clearly, the method relies heavily on understanding the precise mechanism for the onset of a fault, i.e., how fast or slowly it is manifested. Moreover, the rules for processing these rapid changes in the sensors and identifying/isolating a particular fault type are determined through manual inspection of the fault data. Finally, this method offers no measure of certainty (a fault is either detected, or it is not) or the magnitude of the fault.

6.3.2 Neural Network Classification

To address some of the potential limitations of the very fast detection algorithms, several “slow” algorithms were developed which detect the fault based on the nature of the residuals several seconds after the fault. Unlike the fast detection approaches, these approaches do not rely on the fault onset being modeled precisely. Thus, if the fast algorithms miss a fault because it evolves in a way that was not modeled during the development of the fast algorithms, one of the slow algorithms, which are tuned based on the steady-state effect of the faults, may detect it. The slow detection algorithm described in this section relies on neural networks for fault identification.

An artificial neural network (NN) is a biologically inspired system that attempts to mimic, in a greatly simplified manner, the function of the brain. A neural network is composed of neurons (also referred to as nodes). A neuron has one or more inputs ($X_1…X_n$), which are modified by the corresponding weights ($w_1… w_n$). The product of the inputs and weights are summed (represented by the Σ), along with the bias (*b*, a constant), and evaluated by a typically nonlinear transfer function (represented by the $\int$); the result is the output, y. Mathematically, a neuron is described by:

$$y = \int\left(b + \sum_{i=1}^{n} w_i x_i\right).$$

All of the neural networks used here employ the hyperbolic tangent sigmoid transfer function:

$$\int(x) = \frac{e^{2x}-1}{e^{2x}+1}.$$

Individual neurons are arranged in layers to form a network, typically comprised of an input layer, one or two hidden layers, and an output layer. For example, the NN in figure 16 has an unspecified number (*n*) of inputs, five neurons in the first hidden layer, three neurons in the second hidden layer, and two output neurons (corresponding to the two outputs, y_1 and y_2).

The process of determining the weights and biases is known as training. A series of patterns is evaluated by the network, and the weights and biases are adjusted using Bayesian regularization backpropagation[2] so as to minimize the error between the predicted value and the desired value, but not over fit the training data (i.e., to generalize well).

[2] For details, see:
Foresee, F.D., and M.T. Hagan, "Gauss-Newton approximation to Bayesian regularization," *Proceedings of the 1997 International Joint Conference on Neural Networks*, 1997.
MacKay, D.J.C., "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415-447, 1992.

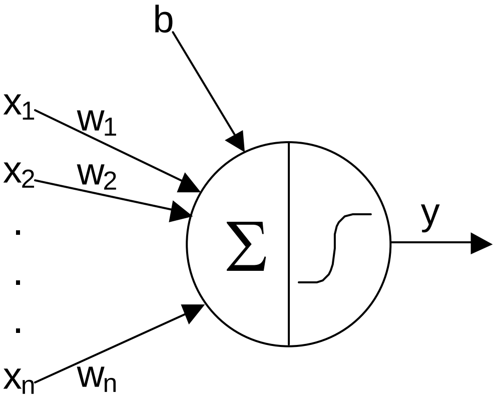


Figure 15.—An individual neuron.

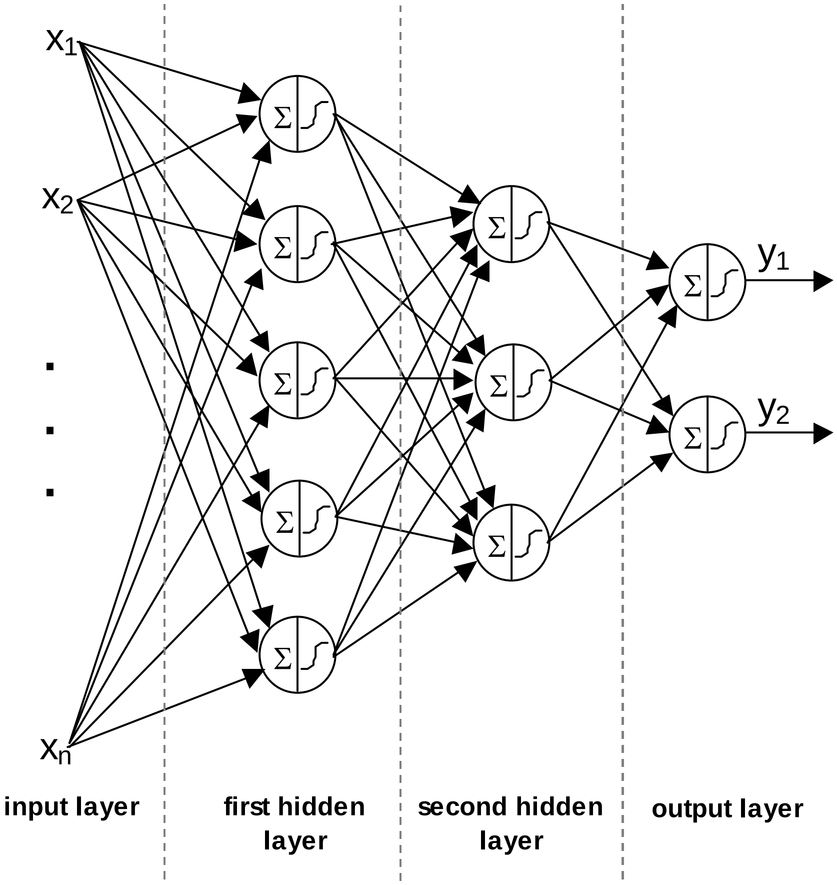


Figure 16.—A NN with n inputs, five neurons in the first hidden layer, three neurons in the second hidden layer, and two output neurons (corresponding to the two outputs, y1 and y2). The bias for each neuron is not shown.

A neural network fault detection system was developed, operating on the EKF residuals (with EKF bias removed). The mean of a subset of the sensed data a few seconds after fault onset, as well as flight envelope information (mach number, ambient temperature, throttle angle, and altitude) was used as network inputs. There were seven neurons in the single hidden layer, and five neurons in the output layer. All the EKF generated residual input data was normalized such that it ranged from –1 to 1. The data was split into two groups, with 85% of the data used to train the network, and 15% of the data used to validate the network.

Each of the five output neurons corresponds to one of the four fault types, plus the nominal condition. The output for each neuron was 1 if the case corresponded to the neuron type (e.g., VG fault), or –1 otherwise. The case is assigned the fault type (or nominal condition) corresponding to the neuron with the largest value above a threshold.

The results of the NN detection system at the end of each run are presented in table 9. Note that 99.9% of all faults are correctly classified at the end of the run, with no false alarms, and only a few misclassifications.

TABLE 9.—RESULTS FOR VALIDATION DATA FOR NEURAL NETWORK CLASSIFIER

Overall	Classification				
Fault	HPC	HPT	VG	P Sensor	Nom
HPC	893	2	0	0	0
HPT	1	992	0	0	0
VG	0	0	954	0	0
P Sensor	0	0	0	971	0
Nom	0	0	0	0	325
Small	Classification				
Fault	HPC	HPT	VG	P Sensor	Nom
HPC	271	0	0	0	0
HPT	1	324	0	0	0
VG	0	0	324	0	0
P Sensor	0	0	0	325	0
Nom	0	0	0	0	325
Medium	Classification				
Fault	HPC	HPT	VG	P Sensor	Nom
HPC	318	1	0	0	0
HPT	0	351	0	0	0
VG	0	0	320	0	0
P Sensor	0	0	0	326	0
Nom	0	0	0	0	325
Large	Classification				
Fault	HPC	HPT	VG	P Sensor	Nom
HPC	304	1	0	0	0
HPT	0	317	0	0	0
VG	0	0	310	0	0
P Sensor	0	0	0	320	0
Nom	0	0	0	0	325

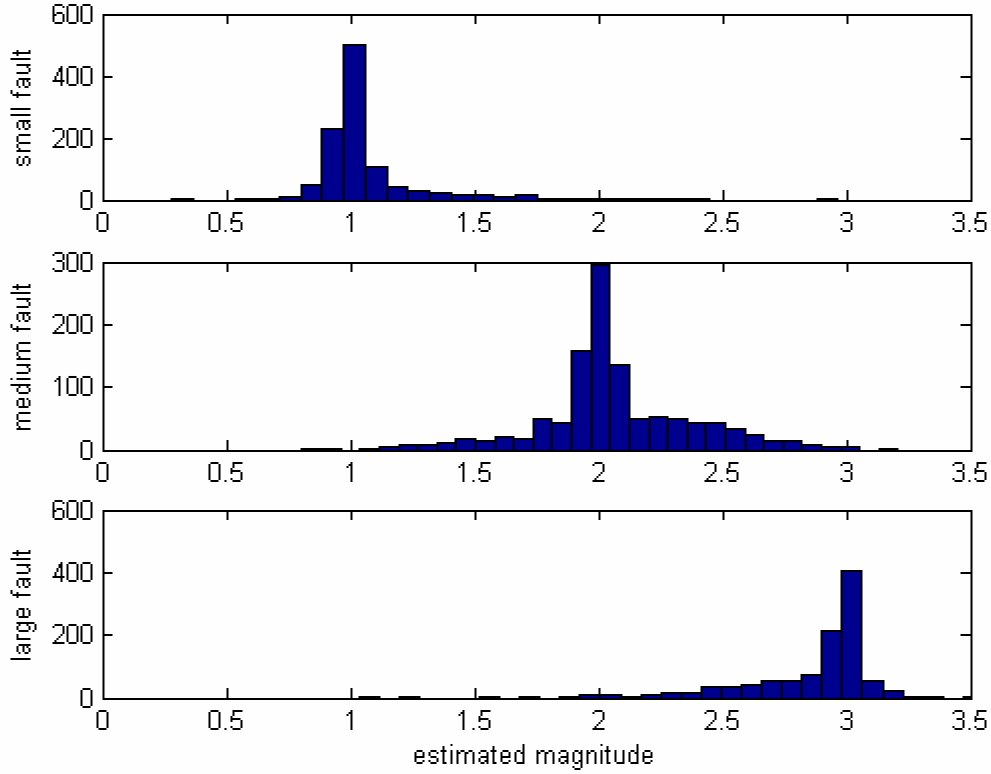


Figure 17.—Accuracy of fault magnitude estimator. Small faults were assigned a value of 1, medium faults a value of 2 and large faults a value of 3.

In addition to the identification and classification of the faults, another neural network was also trained to identify the fault magnitude. This second network uses a subset of the transformed sensed variables as well as flight envelope information (mach number, ambient temperature, throttle angle, and altitude) as inputs, has two hidden layers with 17 and 13 nodes, respectively, and one output i.e., the fault magnitude. Figure 17 is a plot of the accuracy of the NN fault magnitude estimator, which is quite good.

Some advantages to this detection system are that, because of the encoding system, it allows for both a positive nominal and, conversely, for the detection of faults of an unknown type. That is, if the nominal condition is indicated, it is likely that the engine is truly nominal; conversely, if the value of the nominal neuron output does not reach the threshold, but none of the other neurons reach the threshold as well, then a fault of unknown nature is detected. Moreover, the closeness of the dominant output value to 1 provides an indication of the confidence in the fault type – a feature that can be used to fuse the results from other fault detection/classification results and get an improved overall detection/classification performance.

6.3.3 Rapid Fault Detections

This method was designed and tested only using the initially generated “open-loop” fault data. Based on considerations of maturity of this approach relative to the others, we did not pursue it when we regenerated the “closed-loop” data. In this section, we describe the general approach and its results for the “open-loop” data only.

Detection accuracy of engine faults is typically predicated by the magnitude of the fault signature seen in sensor measurements and time to detection. That is, typically, fault detection assumes a sufficiently large fault signature and enough time to come up with the decision. In addition, noise, engine-to-engine variation, deterioration, model uncertainty, and closed loop controller effects further encumber the detection algorithms. Because we need to support automatic fault accommodation, another requirement imposed on the detection includes a zero false positive rate to avoid taking remedial action when no fault exists. It is therefore imperative to remove as much noise as possible within the allowed detection interval.

To allow for the detection of small changes in engine components, a suite of techniques was employed to address certain aspects of these issues in parallel. Since emphasis was placed on the quick detection, feature extraction underwent special attention. First, the effects of fault signature variability of the tracking residuals due to operating conditions were investigated using a generic baselining approach. In addition, several advanced features were developed that indicate a deviation from normal operation including some techniques that are more commonly found in data trending. Finally, a bank of binary classifiers determines the presence of the fault as resolved by a maximum likelihood hypothesis test. We show performance results for four different faults at various levels of severity (small, medium, and high) at steady state with no change in actuator inputs or flight conditions and continuously for several seconds after fault introduction.

In support of the rapid detection, we introduce here an elbowing operator that responds to abnormal changes. Because changes do not to show up as a step change, the detection is encumbered. This is true for most measured parameters because mechanical and thermodynamical inertia acts as a dampening agent such that changes come to light with some lag only. What hampers reliable early recognition further is the noise in the system, which is the primary potential source for false-positives. It is vital to find the point at which the change may have been initiated. Traditional regression techniques do not do well because only a few data points are available for the change detection and small changes are drowned in the noise. That in turn may result in drastically different regression results as well as very unreliable determination of the elbow point. Generally these techniques can be found within the trending domain where similar problems of quick detection occur (although the sampling time may be off by several orders of magnitude). To address quick-change recognition, we carry out a quasi-sliding hypothesis test that evaluates whether a change over a window of observations meets certain change criteria. These criteria are based on change persistence as well as change magnitude. If these criteria are met, an “elbow” point, i.e., the point at which the potential change has first occurred, is identified and retained. The difference between the elbow point and the current smoothed observation is then used as a feature for the classifier. The window size for smoothing is determined by the time to detection requirements. One advantage of this feature is that it is always baselined to 0 because it operates on the differences only. The operative equation for the elbowing variable *x(k)* is

$$x(k) = m(k) - x_{elbow}$$

with

$$x_{elbow}(k) = \begin{cases} m(k) & \text{if } \left| res(k) - res(k - k_{elbow} + 1) \right| < d_{th} \\ x_{elbow}(k-1) & \text{otherwise} \end{cases}$$

where

$$m(k) = mean\left(res(k), res(k-1), res(k-2), res(k-3), res(k-4)\right) \quad res(k) = \hat{y}(k) - y(k)$$

k_{elbow} is the time at which elbowing was observed
d_{th} is the threshold for elbowing

Figure 18 shows the temperature residual and corresponding elbowing where the fault was injected at time *t = 60*.

The overall classification approach followed the scheme shown in figure 19. Inputs are the flight envelope data (FE data), sensor data, and EKF estimates. After a preliminary variable selection, residuals are computed from which further features are calculated including the elbowing features. After a further feature selection process, the features are subjected to the classification (here multiple binary classification). The classifiers were trained as binary classifiers using the fault data as one class and the normal and other fault classes as the other class. The last step is the hypothesis test, which selects the final fault state.

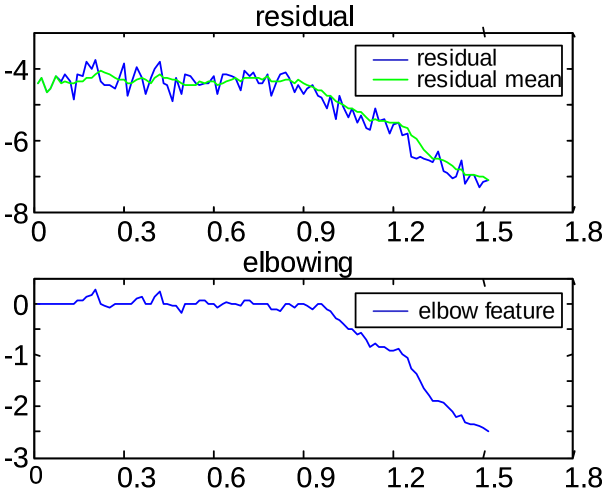


Figure 18.—Small HPT fault example; (a) residual and mean; (b) “elbow” feature.

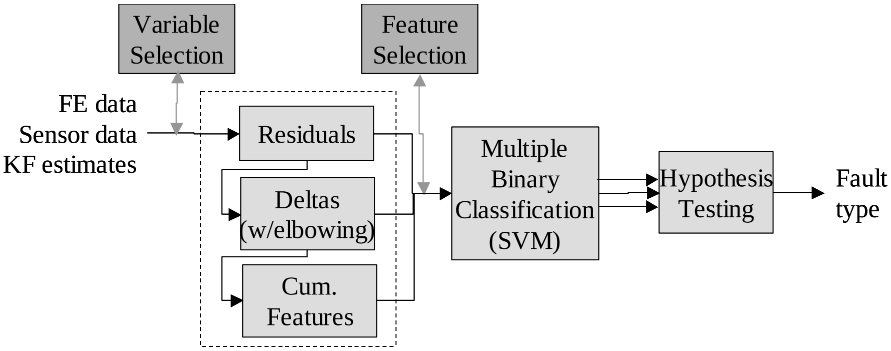


Figure 19.—Fault classification scheme.

Following are a number of Receiver Operating Characteristic (ROC) curves that summarize fault specific results. Because performance is generally good, we show only a small portion of the ROC curve from the true positive (TP) point where the false positive (FP) rate was zero. This means that the axes of the curves are different in the cases presented to provide a better appreciation of the performance.

The ROC in figure 20 shows the results for detection of an HPC fault. For a false positive rate of zero, the true negative (TN) rate is 0.998. The somewhat choppy line is due to the number of cases used, which at this level of granularity is not sufficient to provide a smooth curve.

The ROC in figure 21 shows the results for HPT fault. For a false positive rate of zero, the TN rate is 0.9998.

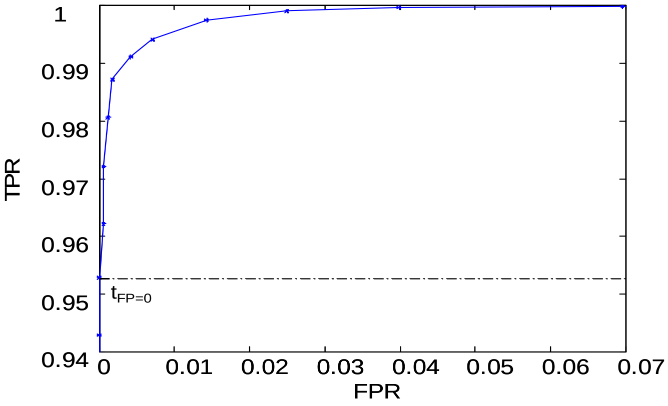


Figure 20.—ROC curve for HPC fault classification.

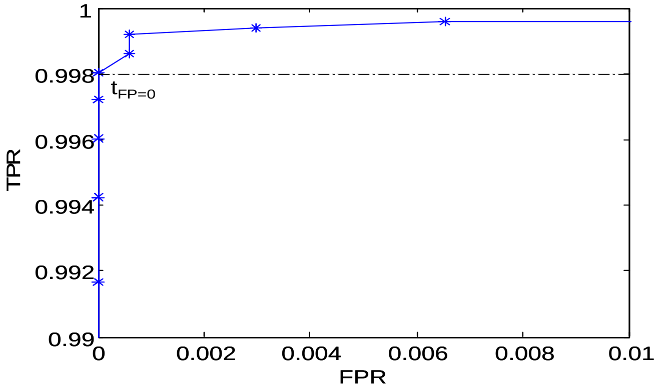


Figure 21.—ROC curve for HPT fault classification.

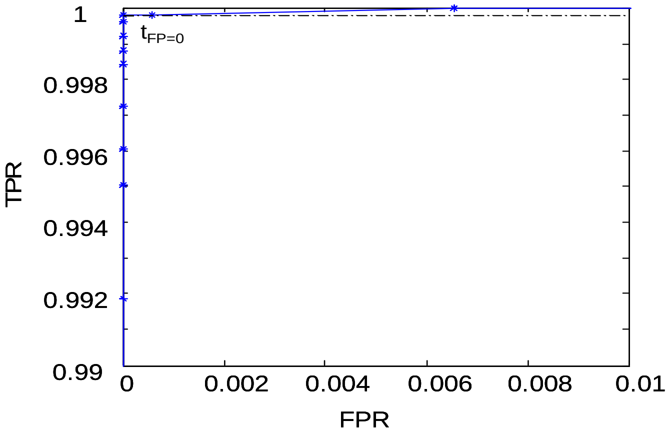


Figure 22.—ROC curve for VG fault classification.

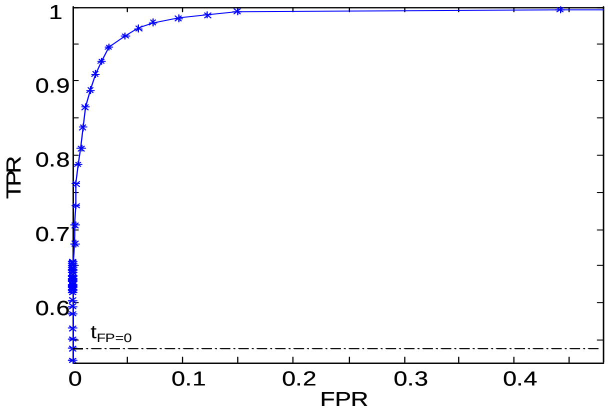


Figure 23.—ROC curve for pressure sensor fault classification.

The VG fault results are shown in figure 22. For a zero false positive rate, the corresponding TN rate is 0.953.

For zero false positives for the pressure sensor fault, the TN rate is 0.52. Figure 23 shows the associated ROC.

One of the main distinctions of this study is that results are compiled for faults that can occur at any point in the flight envelope with any level of noise and engine deterioration. With the mandated zero false positive rate (to avoid any un-commanded accommodation), results are still very reasonable as shown in table 10, which shows the confusion matrix for the output of the maximum likelihood fault selection logic with sensor residuals alone.

TABLE 10.—CONFUSION MATRIX FOR RAPID DETECTION OF SELECTED FAULTS (FP FORCED TO ZERO)—SENSORS ONLY

	No fault	HPC	HPT	VG	PS3 sensor
No fault	1	0	0	0	0
HPC	0.200	0.799	0.001	0	0
HPT	0.166	0.009	0.825	0	0
VG	0.152	0	0	0.848	0
PS3 sensor	0.480	0	0	0	0.520

The performance improves considerably over the detection with sensor measurements alone when the elbowing feature is used as well. Results are summarized in the confusion matrix in table 11. Specifically, all faults are detected at zero false positive levels upward of 90%, where in particular the P sensor fault improved from 0.52 to 0.951 TP rate. At the same time, all falsely classified faults drop considerably. In both cases, only HPT and HPC faults are misclassified. If accommodation mandates that no misclassifications be made at all, then the results will deteriorate considerably because the particular combination of deterioration, flight envelope, and fault signature are completely overlapping. However, the misclassifications occur in large part only at the smallest fault level with very few misclassifications at the medium fault level and no misclassifications at the large fault level. That also implies that larger faults can be accommodated safely. Results for that case are summarized in table 11.

TABLE 11.—CONFUSION MATRIX FOR RAPID DETECTION OF SELECTED FAULTS (FP FORCED TO ZERO) WITH ELBOW FEATURE

	No fault est.	**HPC est.**	**HPT est.**	**VG est.**	**PS3 sensor est.**
No fault	1	0	0	0	0
HPC	0.036	0.959	0.005	0	0
HPT	0.041	0.007	0.952	0	0
VG	0.090	0	0	0.910	0
PS3 sensor	0.049	0	0	0	0.951

Figure 24 shows the individual fault indicators as well as the overall fault indicator for HPC fault. At time t = 0.5s, an HPC fault is injected. It can be seen that the fault indicator shows the presence of the fault within the desired time window, with some chattering.

Figure 25 shows a VG fault injected at 0.5 s. It is notable that the detection commences rapidly within only a few samples and well within the required detection interval.

Figure 26 shows the effect of fault size on detection capability. The three graphs show a small, medium, and large pressure sensor fault. In this particular instance, a fault at a point in the flight envelope with noise conditions was chosen for display that could not be picked up for small faults because it falls under the detection threshold. Medium and large faults are detected as seen in the second and third graph. Not surprisingly, fault magnitude seems to have a strong impact on detection capability.

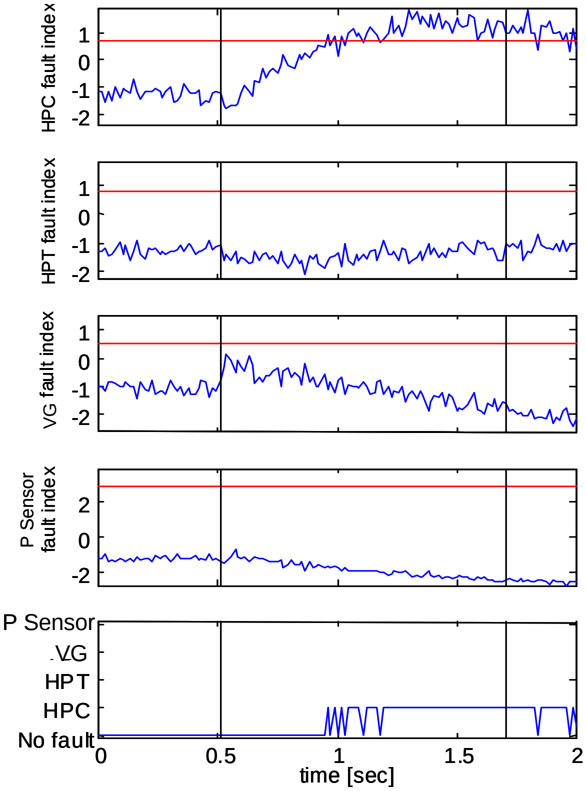


Figure 24.—HPC fault injected at 0.5 s.

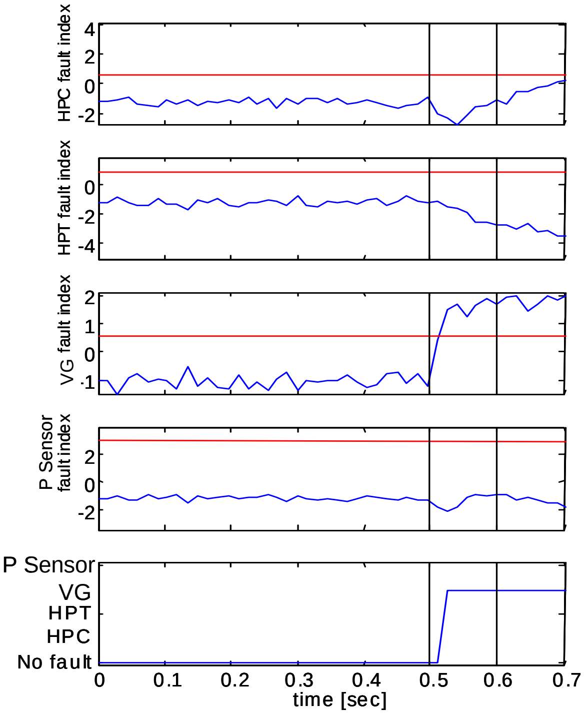


Figure 25.—VG fault injected at 0.5 s.

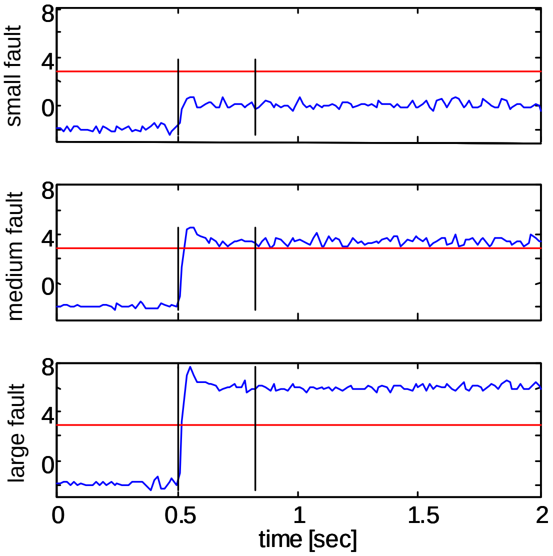


Figure 26.—Effect of fault size on detection capability: PS3 sensor for same flight envelope point.

6.3.4 Multiple Hypothesis Testing

In this section, we describe the "multiple hypothesis testing" method that we have developed to detect all four faults for the VG, HPC, HPT, and PS3 sensor failures.

In this approach, the various fault types and levels, as well as the no-fault condition, are expressed in terms of their signatures in a "measurement space" defined by the EKF residuals. This space is of dimension *p*, corresponding to the number of sensors. The values attained by the residuals when the various fault conditions are imposed are thus represented as positions in the space, and these positions are compared to real-time measurements to assess the health of the engine.

Because the sensors are subject to noise and unknown biases, the fault conditions actually give rise not to discrete positions in the measurement space, but rather to probabilistic distributions in the space. More precisely, these are conditional probability functions defined on the *p*-dimensional space, given the various fault hypotheses. If the sensor noise is considered to be Gaussian, white, and hypothesis-independent, the fault hypotheses may be represented as uniform ellipsoidal functions centered on various mean positions in the space, with the axes of the ellipsoid sized according to the noise variance in each dimension.

Under a further assumption of independence among the sensors, it is customary to normalize the component dimensions of the measurement space by the standard deviation of the corresponding sensor noise, so that the ellipsoidal probability density functions degenerate to spherical functions of uniform radius, and the hypotheses are distinguished solely by the locations of their correspondingly normalized, vector-valued means. Further, in the absence of *a-priori* knowledge of engine faults, the various fault hypotheses are assumed to be equally likely over a given time interval.

Under the full set of assumptions outlined above, the implementation of an optimal Bayesian Hypothesis test that minimizes the probability of error (false positives, false negatives, and misclassifications) can accomplished by a relatively simple computation. Namely, a distance, in the standard Euclidean sense, is computed between the real-time instantaneous value of the residuals and the various *p*-dimensional reference hypotheses, after dividing each residual component by the standard deviation of the sensor noise; the hypothesis with the smallest distance gives the "maximum likelihood" and is chosen as the result. This simplified approach is possible because the conditional probabilities can be expressed as monotonically decreasing functions of distance alone.

If one or more of the assumptions on sensor noise—namely whiteness, Gaussian distribution, fault-independence, and independence across sensors—does not hold, the method of Bayesian hypothesis testing remains applicable, though its implementation becomes more complex. Generally, what is required in such cases is the more computationally expensive direct computation of the conditional probabilities.

One of the challenges encountered in applying the described method to fault estimation in aircraft engines is compensating for unmodeled variations due to deterioration, engine-to-engine variation, and sensor accuracy bias. To mitigate this, we implemented a straightforward, slowly updating bias estimator that determines the DC component of the difference between the engine output and the EKF output, i.e., the EKF residuals. Figure 27 shows where in the detection algorithm the computed bias is removed.

Figures 28 and 29 show the effect of removing the bias for data obtained at a representative operating point within the flight envelope.

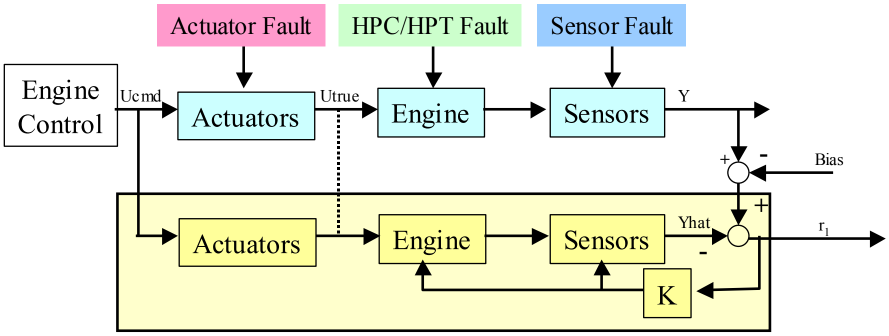


Figure 27.—Engine model and EKF, showing location of bias removal.

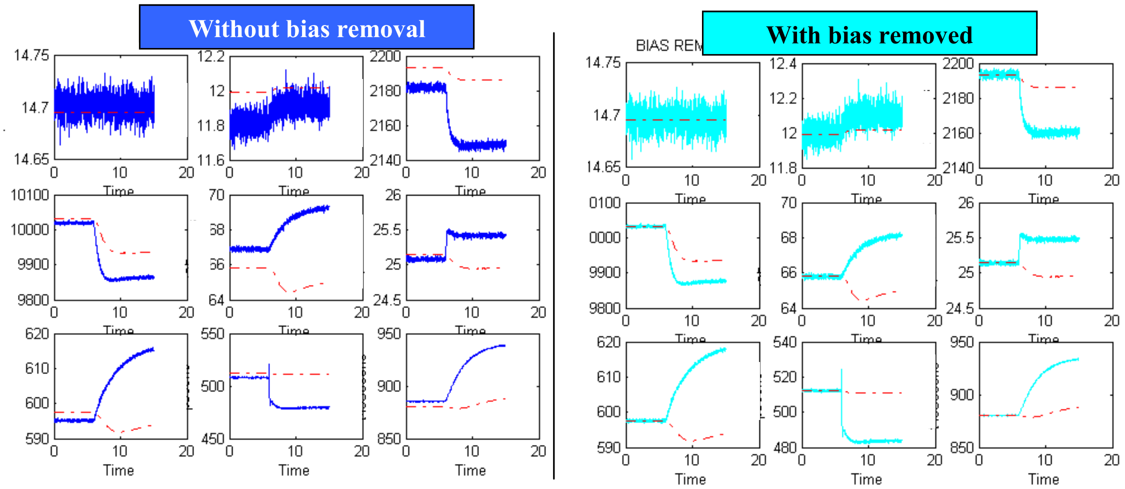


Figure 28.—Bias removal in sensor outputs: engine outputs (blue), EKF (red).

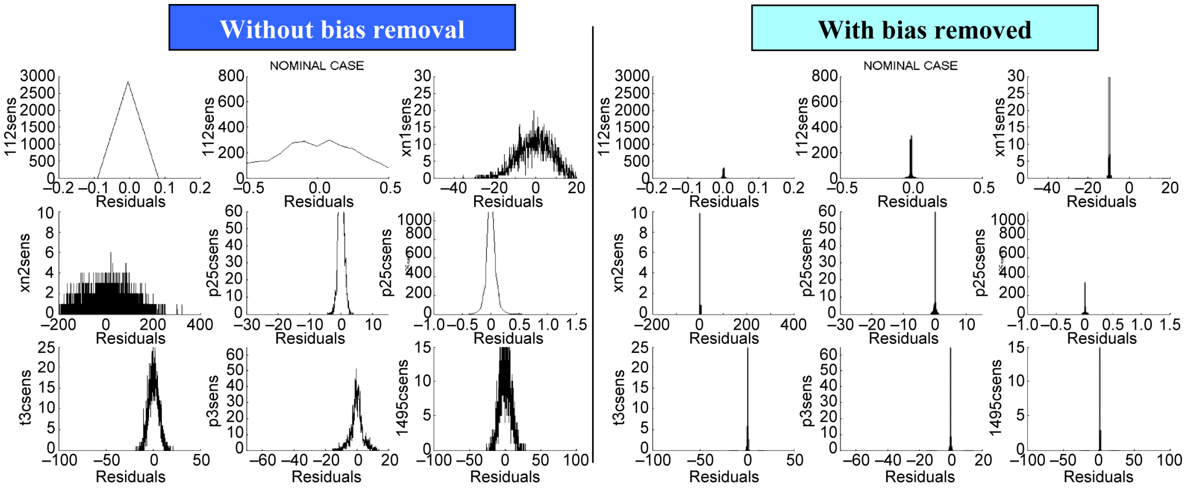


Figure 29.—Histograms of sensor residuals for nominal case, before and after bias removal.

Figure 30 depicts the overall training and implementation phases of the hypothesis testing approach. The estimator is first trained offline with the engine model (bottom of figure) to determine the noise variances and the final values of the residuals under each of the representative fault types and levels, plus the nominal (no-fault) case. To account for variation in engine behavior over the flight envelope (defined by thrust level, altitude, mach number, and ambient temperature), the domain of possible variation in the envelope parameters is divided into representative *regimes*; the final values of the sensor residuals are logged for each of these regimes by averaging over all test cases pertaining to a particular regime. The sensor noise variance is similarly segregated by regime. However, the noise is assumed to be independent of the faults. Therefore, the computation of the standard deviations is performed only on the no-fault training data, again after segregating the data by regime.

Operating in real time, the engine and the EKF (top of figure) provide sensor residuals, which are processed by the Hypothesis Testing Algorithm (right of figure). The algorithm first reads the flight envelope parameters to determine which regime the engine is operating in. It then normalizes the residuals by the standard deviation values, and compares the resulting vector against the available hypotheses to determine (a) if there is a fault at all, and (b) the type and level of any detected fault. Mathematically, the choice is determined by minimizing the distance between the hypothesis and the normalized residual vector.

Actual fault magnitudes are continuous, not discrete (small, medium, large). This warrants that the hypothesis test be modified somewhat to a) recognize the variability in the fault magnitude, and (b) provide a real and continuous-valued estimate of that magnitude. To visualize the needed modification, consider figure 31.

The cubes in the figure represent the measurement space defined by three out of the *p* sensor residuals, after normalization by the standard deviation values. Plotted in each is a representation of the four fault types (HPC, HPT, VG, and PS3 Sensor leak), as manifested for an engine operating in one of *R* possible regimes within the flight envelope. At the left is a plot of the end values attained by the normalized residuals (i.e., the hypotheses corresponding to small, medium, and large values of the four fault types). At the right is a plot of the “mean directions” for the contours in the left-hand plot.

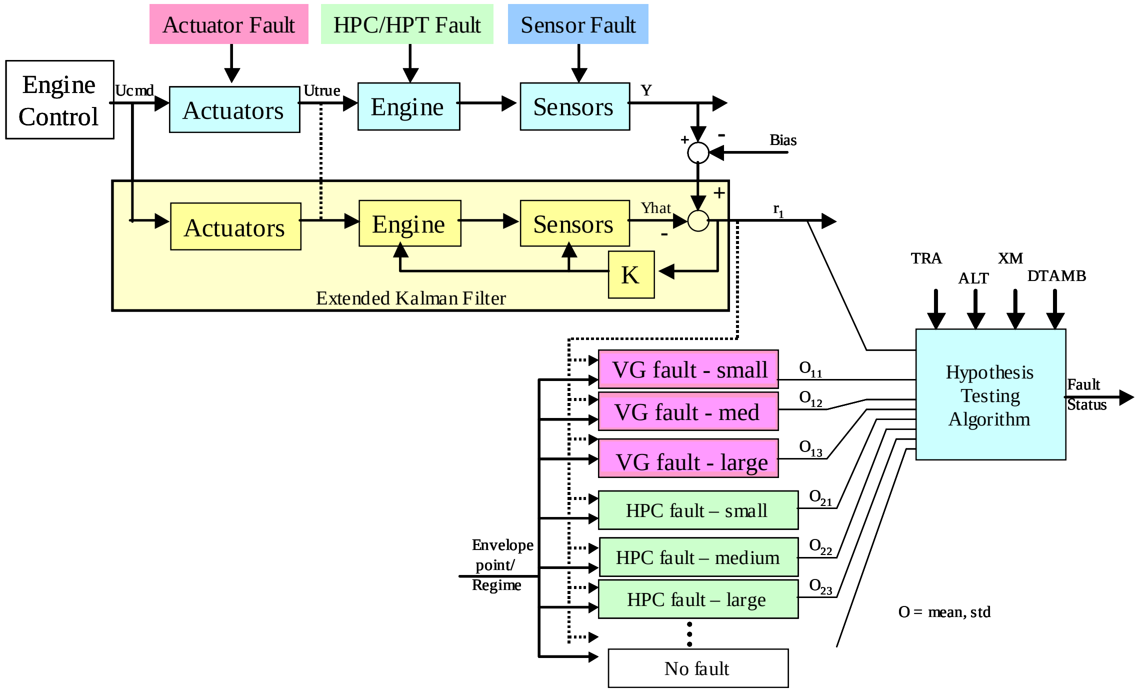


Figure 30.—MHT Algorithm block diagram.

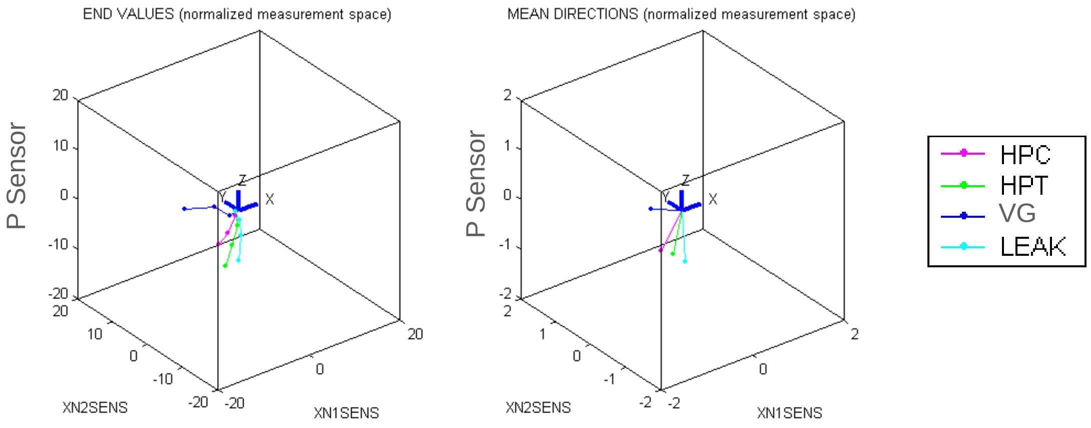


Figure 31.—Fault hypotheses for a single regime.

The “mean directions” are unit vectors approximating the contours defined by the end values of the residuals for the various fault hypotheses. Since the fault magnitude is unknown, we assume for each possible fault type the maximum-likelihood value, which corresponds to the orthogonal projection of the residual vector onto each of the candidate hypothesis vectors. The distances to be compared in the hypothesis test thus become the distances to the projections. Upon selection of the maximum-likelihood fault type, the estimate of fault magnitude corresponds to the length of the selected projection.

Equivalently, an inner product or *correlation* computation is performed between the residual vector and the unit-length “mean directions,” with the maximum correlation determining the fault type. The fault level is subsequently determined via a lookup table that associates fault levels with length along the appropriate fault contour. Continuous values are obtained by interpolating (or extrapolating) from the three fault levels used in the training process.

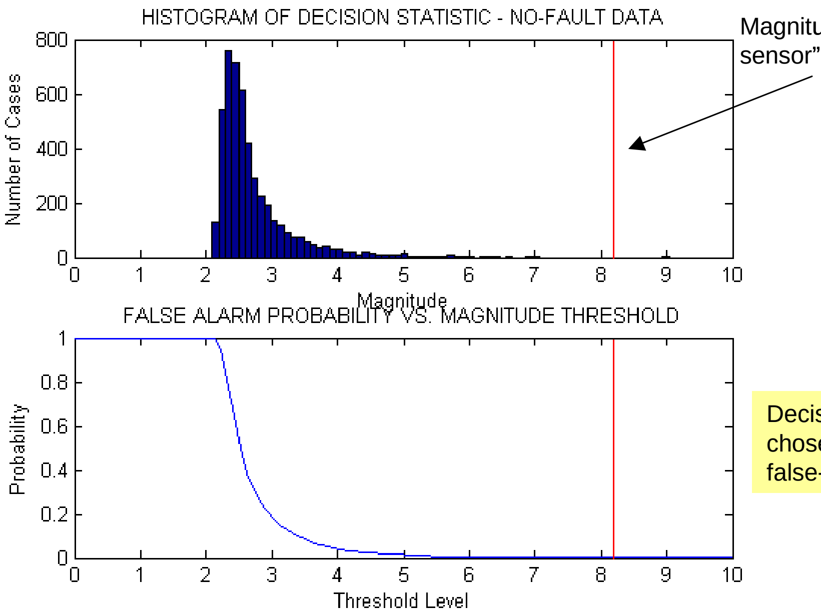
Allowing the fault levels to be continuous necessitates distinguishing the nominal, no-fault condition from very small faults, since fault values arising from the previously described calculation can attain any value, including zero. The distinction can be accomplished by applying a threshold test on the magnitude of the received residual set. If normalization and computation of the vector magnitude results in a value below the threshold, a no-fault decision is declared and no classification (i.e., correlation) is performed.

To determine an appropriate value for this threshold, it is necessary to examine the statistics of the no fault data. Figure 32 (upper plot) shows a histogram of the “decision statistic” defined by the magnitude of the normalized residuals. The vertical line represents the decision threshold, which in this case was chosen to reduce to near zero the probability of asserting a fault under normal operating conditions (i.e., the false alarm rate). The lower plot shows the dependence of the expected false alarm rate on the choice of decision threshold. In practice, the choice of decision threshold is a balance between the false alarm rate and the fault detection rate: too low a setting increases the false alarm rate, while too high a value increases the likelihood that actual faults will go undetected.

This tradeoff is more clearly evidenced in figure 33, which shows probability density functions of the decision statistic under the no-fault condition (blue) and the condition that any of the faults, at severity level 1, 2, or 3, has occurred (green). The red line indicates a threshold value, whose value directly influences both the false-alarm probability Pf and the detection probability Pd (see two plots at bottom right). The “receiver operating characteristic” curve (bottom left) shows the direct relationship between Pf and Pd as parameterized by threshold value (red circle). Here we have chosen to maximize Pd while constraining Pf to be no greater than 0.001. (This is sometimes referred to as a Neyman-Pearson criterion.) The detection probability that results, for the assumed fault levels, is 0.97.

Note that the ROC curve “crowds” the upper left corner, where detection is maximized and false alarms are minimized. This is indicative of a relatively crisp separation between the “no-fault” and “fault” conditional probability density functions. When the sensor noise or other forms of degradation increase, the curve will trend away from the upper left corner, making it more difficult to achieve the desired detection and false-alarm rates.

As discussed, the training process consists of exercising the engine model to determine sensor noise levels and final residual values. Per the previous discussion, it then enables computing the mean directions for faults, the mappings between fault levels and vector magnitudes, and the assignment of a decision threshold.



Magnitudes in “normalized sensor” space (8-dim).

Decision threshold of 8.2 was chosen to achieve < 0.1% false-alarm rate.

Figure 32.—Histogram of decision statistic.

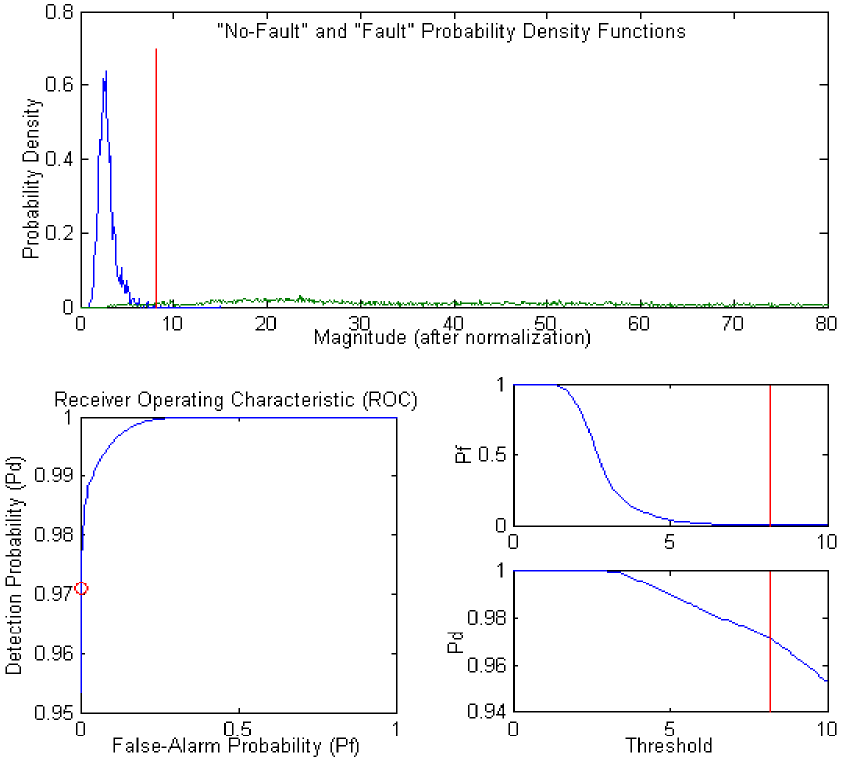


Figure 33.—R.O.C. and choice of decision threshold.

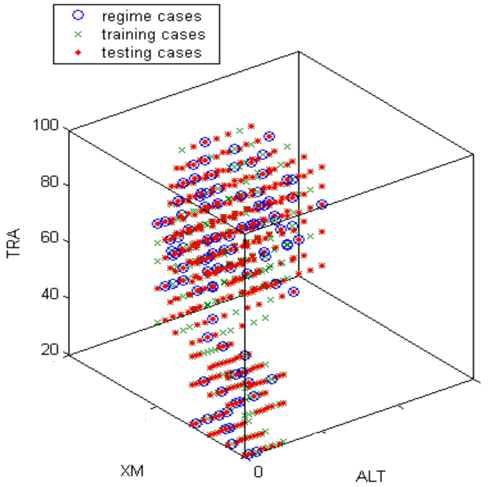


Figure 34.—Flight envelope regime, training, and testing cases.

Figure 34 is a plot of flight-envelope test points chosen during one set of simulation trials (though only 3 of the 4 flight-envelope parameters are represented in this 3D plot). Our exercise of the engine model and the EKF for each fault type and fault level in this set of trials consisted of 2846 cases spanning the region shown in the figure.

Division of the region into discrete regimes can be accomplished in many ways. In our previous implementation of the method, we chose 100 of the 2846 cases at random, and assigned a regime to each of the remaining cases according to their vicinity to the 100 chosen points. These points are shown circled in blue. We then extracted 2000 points for estimator training (green symbols), and designated the remaining 746 for algorithm testing (red symbols).

This method gave satisfactory results during “open-loop” investigation of the aircraft engine model, but this performance was not retained for trials performed under “closed-loop” conditions (i.e., inclusive of engine controller logic). This performance degradation was due to a greatly increased level of variability in the EKF residuals, which were seen to exhibit a fault-type and fault-level dependent random noise component and time-varying biases not previously observed.

In strict theoretical terms, the method might be deemed inapplicable to the closed-loop case, because the fault-dependence of the measurement noise violates an assumption made earlier. However, we were able to ignore this fact and yet recover much of the earlier open-loop performance by reducing the variability through averaging of larger training subsets. This was accomplished by reducing the number of flight-envelope regimes from 100 to a much smaller number in the range 7 to 13. In concert with this, we adopted a deterministic approach in the assignment of regimes, rather than the randomized approach used earlier.

Figure 35 shows the flight-envelope points of figure 34 partitioned deterministically into 7, 10, and 13 regimes. The partitioning is based on three regions in the ALT-XM plane per discrete power level (TRA), except at the lowest power level, where only one of the regions was represented in the trials. The fourth parameter, ambient temperature, was not included in the partitioning.

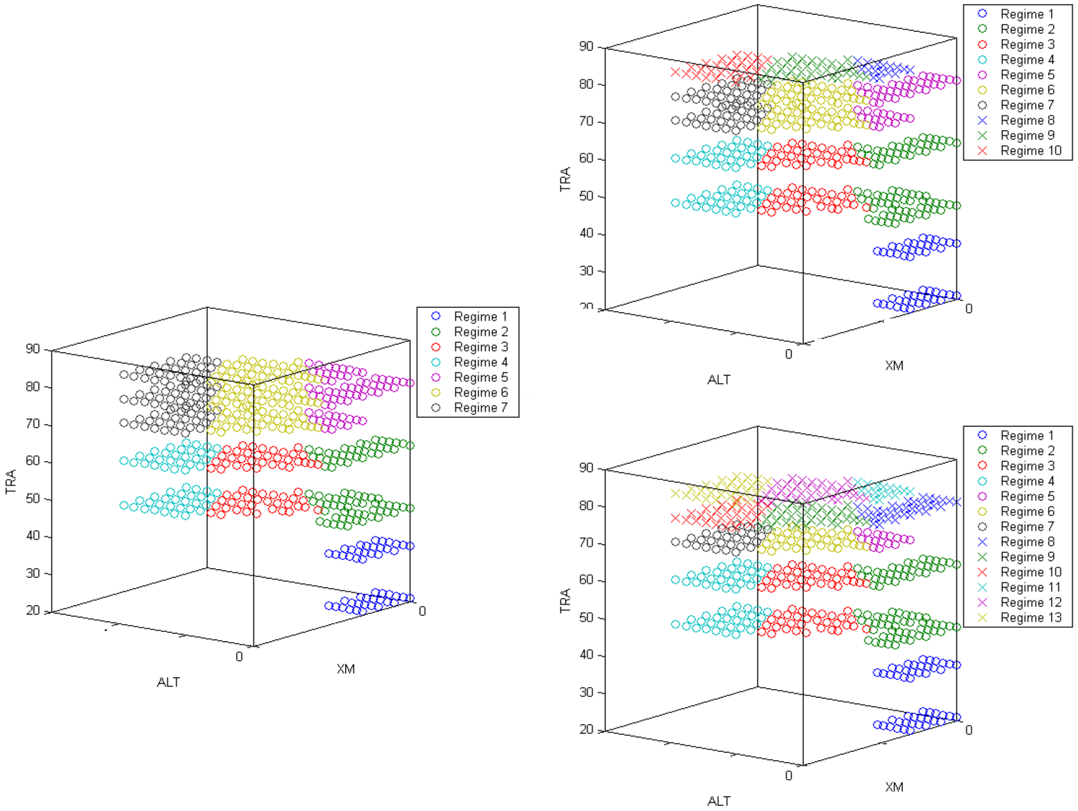
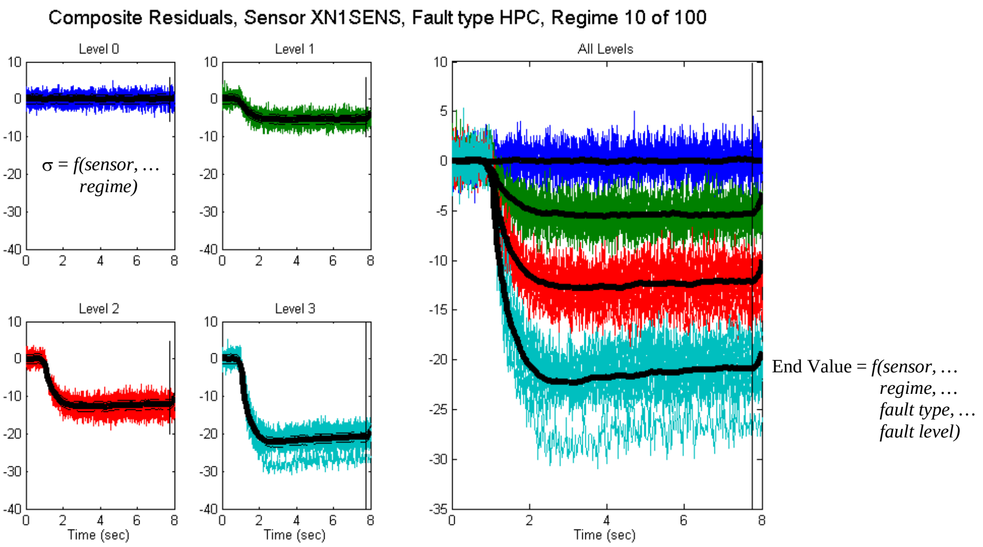


Figure 35.—Deterministic assignment of 7, 10, or 13 regimes.

Figures 36 and 37 illustrate the procedure for determining noise levels and end values for a given sensor. The plots show overlays of the residuals versus time for all training cases corresponding to a particular regime point, under conditions of no fault, and for three levels of the HPC fault type. The noise level for this sensor (N1) and regime is computed simply as the standard deviation of all time points in the no-fault case, shown at the upper left. The end values are obtained by averaging the overlaid plots over time and across the ensemble of related cases. The vertical line within each plot shows the time at which the end values were taken. (The time occurs somewhat before the end of the trajectory, to avoid edge effects from the filter.) This yields the no-fault end value and the end values corresponding to small, medium, and large HPC faults. This procedure is repeated for each sensor, for each fault type, and for each regime reference point in the flight envelope.

The difference between the two figures is that the former represents results with open-loop data, presented earlier in the program, and the latter represents closed-loop results using the optimally tuned EKF with the described adjustments in the regime assignment and training method. Note that the retuned EKF improved the residuals with a clean structure similar to the open-loop case.



σ = *f(sensor, … regime)*

End Value = *f(sensor, … regime, … fault type, … fault level)*

Figure 36.—Residuals versus time for all training cases corresponding to a regime. (Open-loop data, 100 regimes chosen randomly.)

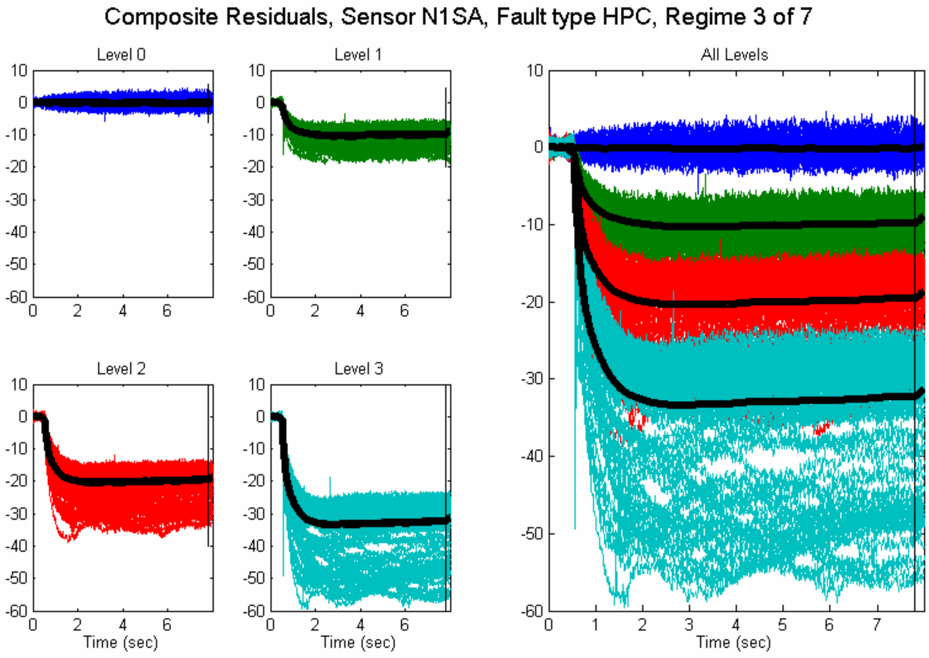


Figure 37.—Residuals versus time for all training cases corresponding to a regime. (Closed-loop data, 7 regimes chosen deterministically.)

The estimator is embodied by three components: (i) a set of *p*-dimensional vectors representing the mean direction for each fault type in the normalized measurement space, as a function of regime, (ii) a set of lookup tables that map vector magnitudes to fault levels as a function of fault type and regime, and *iii) a decision threshold. The estimation algorithm itself follows the sequence:

1) Divide each residual by the noise standard deviation and construct a measurement vector.
2) Compute the magnitude of the vector and compare to the decision threshold
3) If below threshold, declare no fault.
4) Otherwise, determine the operating regime from the flight envelope. From this determination, extract the mean directions and fault level mappings for all fault types.
5) Correlate the received measurement vector with the mean directions. From the maximum correlation, declare the fault type.
6) Compute the projection of the measurement vector onto the mean direction for the declared fault type, and map this value to a continuous-valued fault level via the corresponding lookup table for the determined operating regime.

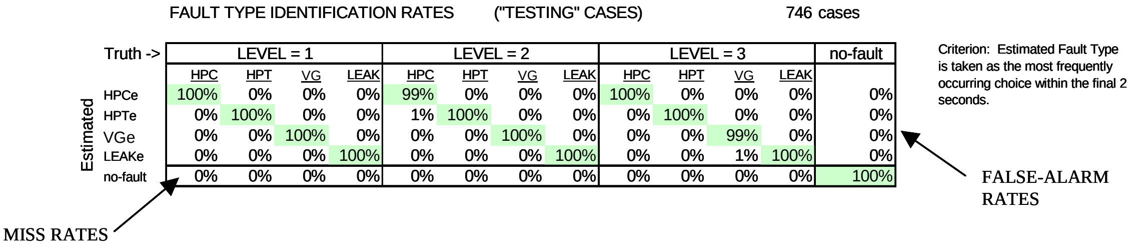
The results to be presented below represented the following scenarios:

1) Open-loop model data, simplified estimator, randomized regime selection, as presented in prior reports.
2) Closed-loop data, simplified estimator, with deterministic regime selection and a small number of regimes.

Table 12 presents the results for the 746 open-loop test points in the flight envelope. A single classification decision was attained for each case by considering a time window of 2 s duration starting 5 s after imposition of the fault, and determining the most frequently occurring choice.

Overall, the estimator attained: 0 false alarms, 0 missed faults, and 1% or fewer misclassified faults. Note that the test cases include variation due to engine-to-engine variation, engine deterioration, sensor bias, and sensor noise.

TABLE 12.—CLASSIFICATION RESULTS – CONFUSION MATRIX (OPEN-LOOP)



FAULT TYPE IDENTIFICATION RATES ("TESTING" CASES) 746 cases

Truth ->	LEVEL = 1				LEVEL = 2				LEVEL = 3				no-fault
Estimated	HPC	HPT	VG	LEAK	HPC	HPT	VG	LEAK	HPC	HPT	VG	LEAK	
HPCe	100%	0%	0%	0%	99%	0%	0%	0%	100%	0%	0%	0%	0%
HPTe	0%	100%	0%	0%	1%	100%	0%	0%	0%	100%	0%	0%	0%
VGe	0%	0%	100%	0%	0%	0%	100%	0%	0%	0%	99%	0%	0%
LEAKe	0%	0%	0%	100%	0%	0%	0%	100%	0%	0%	1%	100%	0%
no-fault	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%

Criterion: Estimated Fault Type is taken as the most frequently occurring choice within the final 2 seconds.

MISS RATES

FALSE-ALARM RATES

The performance with respect to fault level determination is shown in figure 38. Actual fault levels are represented by color (small = blue, medium = green, large = red). The estimated fault levels are represented as histograms. On average the estimated fault levels converge to their actual values, though a variance of up to one fault level is observed.

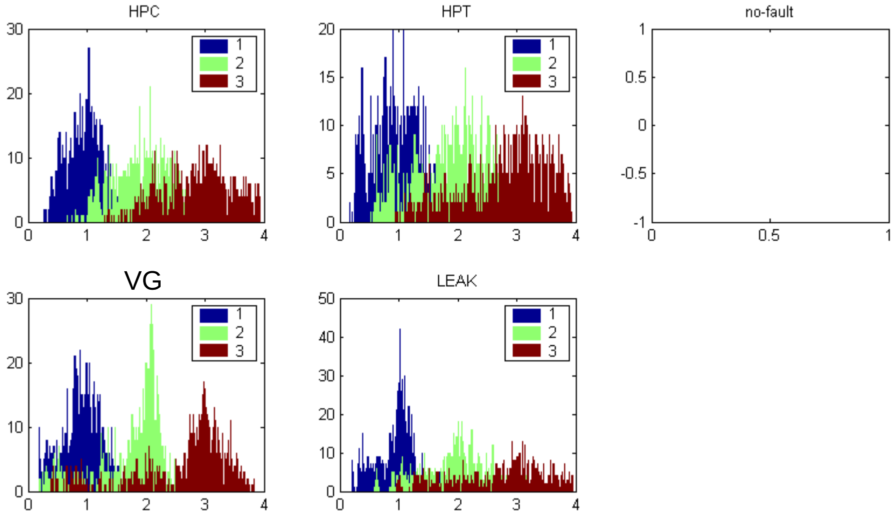
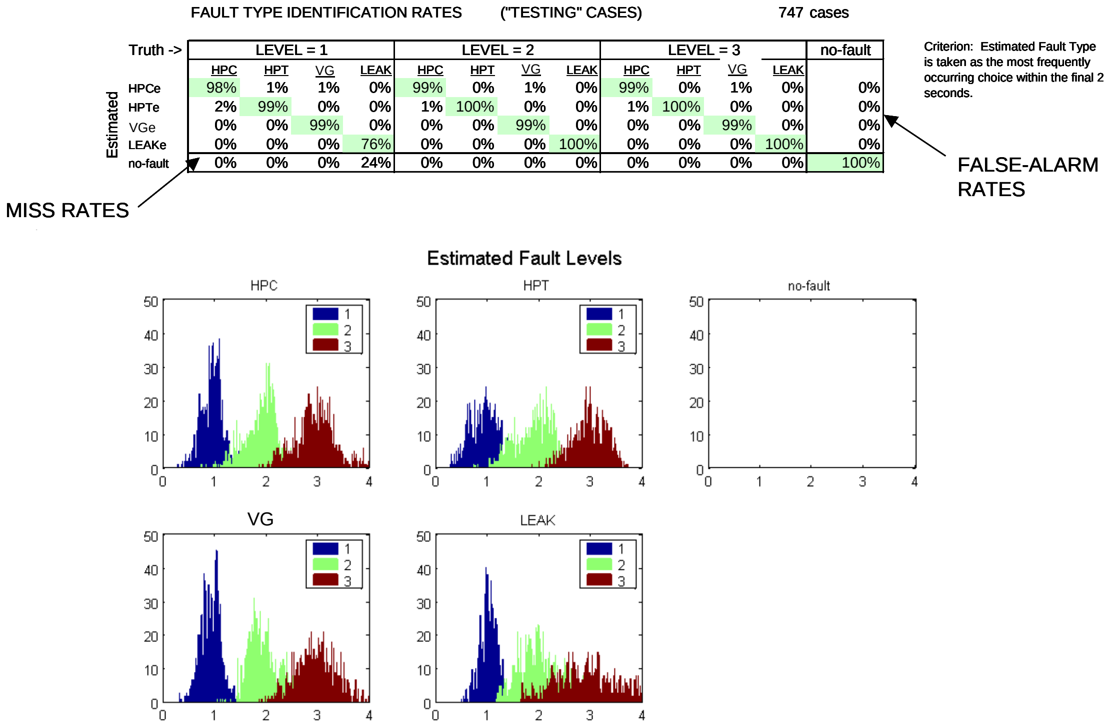


Figure 38.—Estimated fault levels with open-loop data (abcissa is fault level in all plots).



FAULT TYPE IDENTIFICATION RATES ("TESTING" CASES) 747 cases

Truth ->	LEVEL = 1				LEVEL = 2				LEVEL = 3				no-fault
Estimated	HPC	HPT	VG	LEAK	HPC	HPT	VG	LEAK	HPC	HPT	VG	LEAK	
HPCe	98%	1%	1%	0%	99%	0%	1%	0%	99%	0%	1%	0%	0%
HPTe	2%	99%	0%	0%	1%	100%	0%	0%	1%	100%	0%	0%	0%
VGe	0%	0%	99%	0%	0%	0%	99%	0%	0%	0%	99%	0%	0%
LEAKe	0%	0%	0%	76%	0%	0%	0%	100%	0%	0%	0%	100%	0%
no-fault	0%	0%	0%	24%	0%	0%	0%	0%	0%	0%	0%	0%	100%

Criterion: Estimated Fault Type is taken as the most frequently occurring choice within the final 2 seconds.

FALSE-ALARM RATES

MISS RATES

Estimated Fault Levels

Figure 39.—Results using the modified estimator (closed-loop data, 13 regimes).

The results obtained using the updated estimator on the closed-loop data are shown in figure 39. True to our design and choice of threshold, none of 747 cases tested resulted in a false alarm. We observe a somewhat degraded classification performance, as noted in the 1 to 2% rates of misclassification of HPC, HPT, and VG faults, and the 24% miss-rate for level-1 PS3 Sensor leak faults. Interestingly, however, we also observe an improved accuracy in the estimation of fault level. These results represent a flight-envelope partitioned using 13 regimes, though other trials using 7 and 10 regimes (fig. 35) resulted in essentially the same performance.

6.4 Implementation of Fault Detection and Fusion Algorithms

6.4.1 Overall MHT, NN and Fusion Algorithm Architecture for Fault Detection

Out of the four fault detection algorithms described in the previous section, we down-selected the Neural Network (NN) and Multiple Hypothesis Testing (MHT) algorithms for implementation based on considerations for algorithm maturity, performance, ability to be implemented online in a FADEC and ability to obtain confidence of the identified fault type as well as the fault magnitude. The last two requirements are especially important since the confidence estimate allows fusing the results from multiple algorithms based on the individual algorithms' confidence, and the fault magnitude estimate is necessary to engage the correct fault accommodation. Moreover, the MHT and NN algorithms provided complementary performance allowing us to obtain an overall improved result through fusion.

Figure 40 shows the overall architecture for implementing the two selected detection algorithms, MHT and NN, and the fusion algorithm to combine the results from each algorithm and identify the fault type and magnitude. The two algorithms run in parallel and use the flight condition data (TRA, Alt, XM, DTAMB) as well as the EKF residuals as inputs. The EKF residuals were pre-filtered with a simple first-order low-pass filter to further reduce noise in order to suppress chattering in the results obtained from the individual detection algorithms.

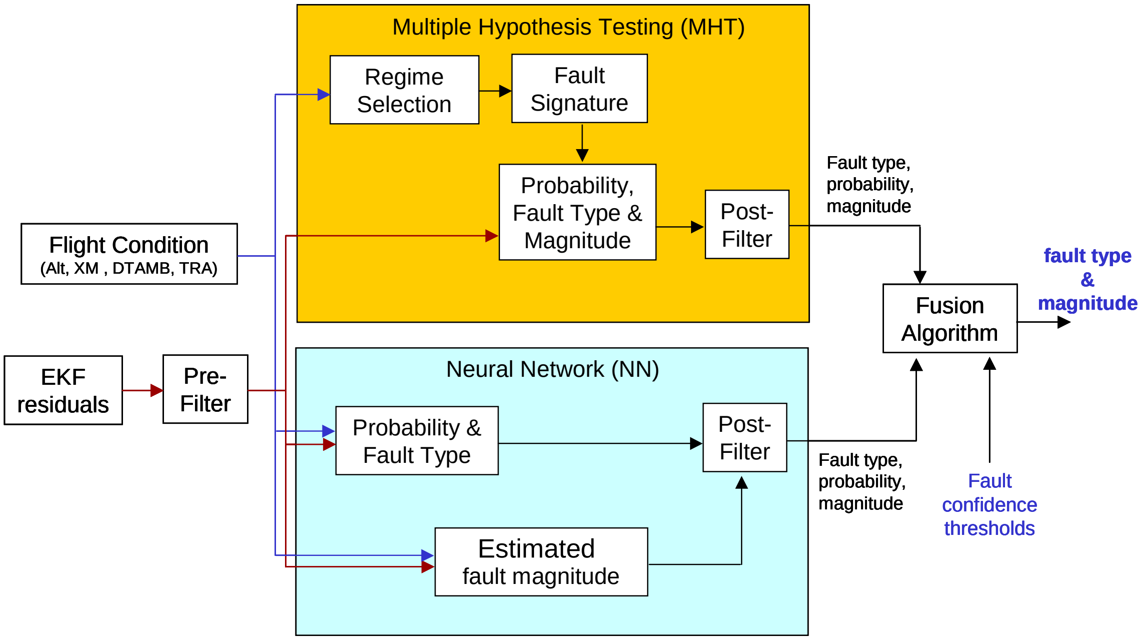

Multiple Hypothesis Testing (MHT)
Regime Selection
Fault Signature
Probability, Fault Type & Magnitude
Post-Filter
Fault type, probability, magnitude
Flight Condition (Alt, XM , DTAMB, TRA)
EKF residuals
Pre-Filter
Fusion Algorithm
fault type & magnitude
Neural Network (NN)
Probability & Fault Type
Post-Filter
Fault type, probability, magnitude
Fault confidence thresholds
Estimated fault magnitude

Figure 40.—MHT and NN fault detection and fusion algorithms.

6.4.2 Online Implementation of MHT Algorithm

The MHT algorithm was modified for implementation in an online version, i.e., process data one sample at a time in real time; the original version as developed processed all transient data in batch mode. The instantaneous flight condition data is used to identify one of the 10 flight regimes, and pick the corresponding fault signature. Upon close inspection of the sensor residuals and the fault signatures, it was determined that two sets of pressure and temperature sensors had no information about the selected faults and they were eliminated from the sensor set, thereby using only the remaining 5 sensor residuals scaled by the corresponding 1-sigma variation. In this method, the magnitude (i.e., 2-norm) of the scaled sensor residuals is used initially to decide between no-fault or presence of a fault. Based on these 5 sensor residuals, the threshold for declaring a fault was determined to be 6.7 corresponding to a false alarm probability of 0.1 or less.

Once a fault is indicated, i.e., the residual norm crosses above this threshold, the method calculates the probabilities of individual faults. To this end, first the probability of no fault is calculated based on the magnitude of the scaled EKF residual vector and the plot in figure 41. Given that a no fault is declared below the threshold of 6.7, to be consistent with this decision, the probability of no fault is scaled to a value of 0.5 so that it is the one with highest probability compared to each fault. The remaining probability, i.e. (1-probability of no fault), is then assigned to each fault type in proportion to the cosine of the angle between the scaled EKF residual vector and the direction associated with each fault type (fig. 41). Moreover, the original algorithm was tuned to achieve small false alarm rates, which was achieved at the expense of missing a significant fraction, ~24%, of the small PS3 sensors leak faults (fig. 39). Now that the results of the two algorithms will be fused to generate the overall fault detection and classification result, we can be more aggressive in terms of improving the performance for small PS3 sensor leak faults while tolerating a relatively higher false alarm rates. Motivated by this, we modified the probability calculations to favor the PS3 sensor leak fault if present. More specifically, in cases where the PS3 sensor fault has the highest probability among all 4 fault types (or equivalently, the cosine of the angle between the scaled EKF residual and the PS3 sensor fault direction is the largest), the probability for no fault was scaled to 0.3 instead of 0.5 mentioned above. Finally, the instantaneous probabilities calculated for no fault and the 4 fault types were filtered using the same past horizon used for filtering the fault type detected. The filtered probabilities were calculated as weighted averages over this horizon using the frequencies of each fault type declared at each instant over this horizon as the relative weight for the corresponding fault probability. This yields filtered probabilities for all 4 faults consistent with the filtered fault type calculation based on the “voting” filter.

6.4.3 Online Implementation of NN Algorithm

During the development phase, the NN algorithm was tested using only the final steady state value of the EKF residuals obtained after the fault occurrence. We modified the algorithm for online implementation where each instantaneous sample of the EKF residuals is processed sequentially. The final layer of the neural network for fault detection and isolation yields a set of 5 numbers in the range –1 to 1 corresponding to no fault and each of the 4 fault types. The no fault/fault type with the number closest to +1 was declared as the final decision for that instantaneous time sample. In order to enable the fusion of the results from each algorithm, we scaled this number between –1 to 1 to 0 to 1 and normalized them to obtain a measure of probability/confidence of the faults/no fault. Finally, to reduce the chattering of fault/no fault probabilities, they were filtered by a simple, first-order recursive filter to obtain the filtered probabilities for no fault and each fault type.

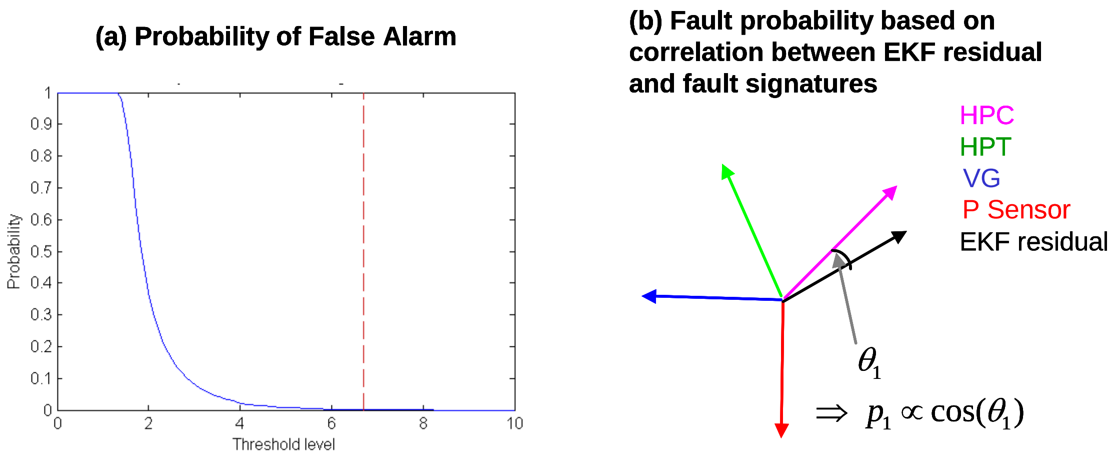


Figure 41.—Probability of (a) false alarm versus magnitude of residual vector and (b) each fault type based on EKF sensor residual.

6.4.4 Implementation of Fusion Algorithm

The results for the fault type, probabilities and magnitudes from the individual MHT and NN algorithms are fused to obtain the overall results on the fault type and magnitude. Figure 40 shows the overall architecture of the individual fault detection algorithms based on the EKF residuals and the fusion of the results from each algorithm. The final fault type and magnitude is used subsequently for fault accommodation—figure 1.

The objectives of the fusion algorithm are:

- Improve the overall false alarm performance compared to each individual algorithm especially given that the incorrect fault accommodation in the case of a false alarm would degrade the performance of a nominal engine.
- Reduce the occurrence of incorrect fault classification—again given that the fault detection/classification will be used for online fault accommodation, it is better to miss a fault (especially small faults) than miss-classify the fault and take a wrong corrective action in the accommodation step.

Figure 42 shows the overall architecture of the fusion algorithm. In the first step, the filtered probabilities for no fault and each fault type obtained from the two algorithms are combined to obtain the overall fused probabilities for no fault and each fault type. The fusion of the fault/no fault probabilities involves calculating an average probability from the two algorithms and then enhancing/suppressing the resulting probability above/below 0.5 respectively, using a suitable exponent. More specifically, given the individual probabilities *pj*, MHT and pj,NN for jth event (no fault/fault types 1 to 4), the average probability is calculated as *pj*,avg = 0.5(*pj*,MHT +*pj*,NN). Thereafter, the average probability is enhanced/suppressed by using an exponent, i.e.,

$$p_{j,\,fused} = p_{j,avg}^{(1-p_{j,avg})}. \qquad (12)$$

In the next step, the event (fault type/no fault) with maximum fused probability is identified. If the event with the maximum fused probability is one of the fault types, then its probability is compared against a threshold before declaring that fault type as the instantaneous fault type. In the final step, a persistency check is performed to ensure that the instantaneous decision on a

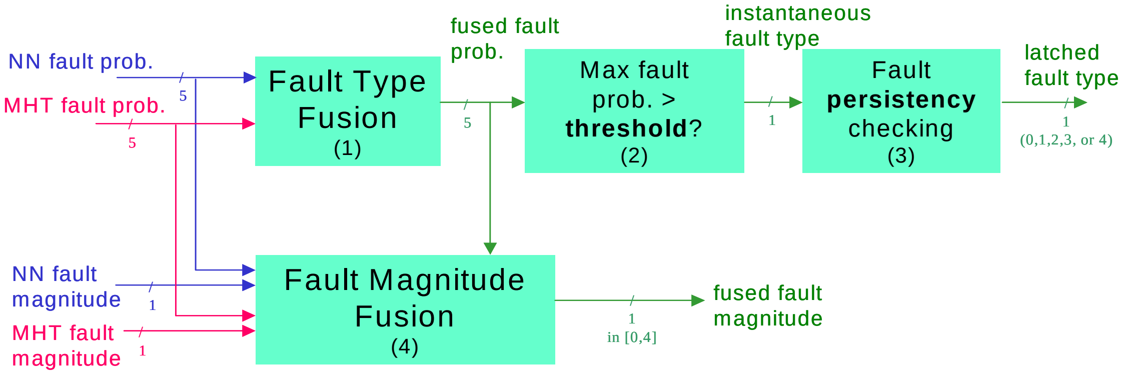


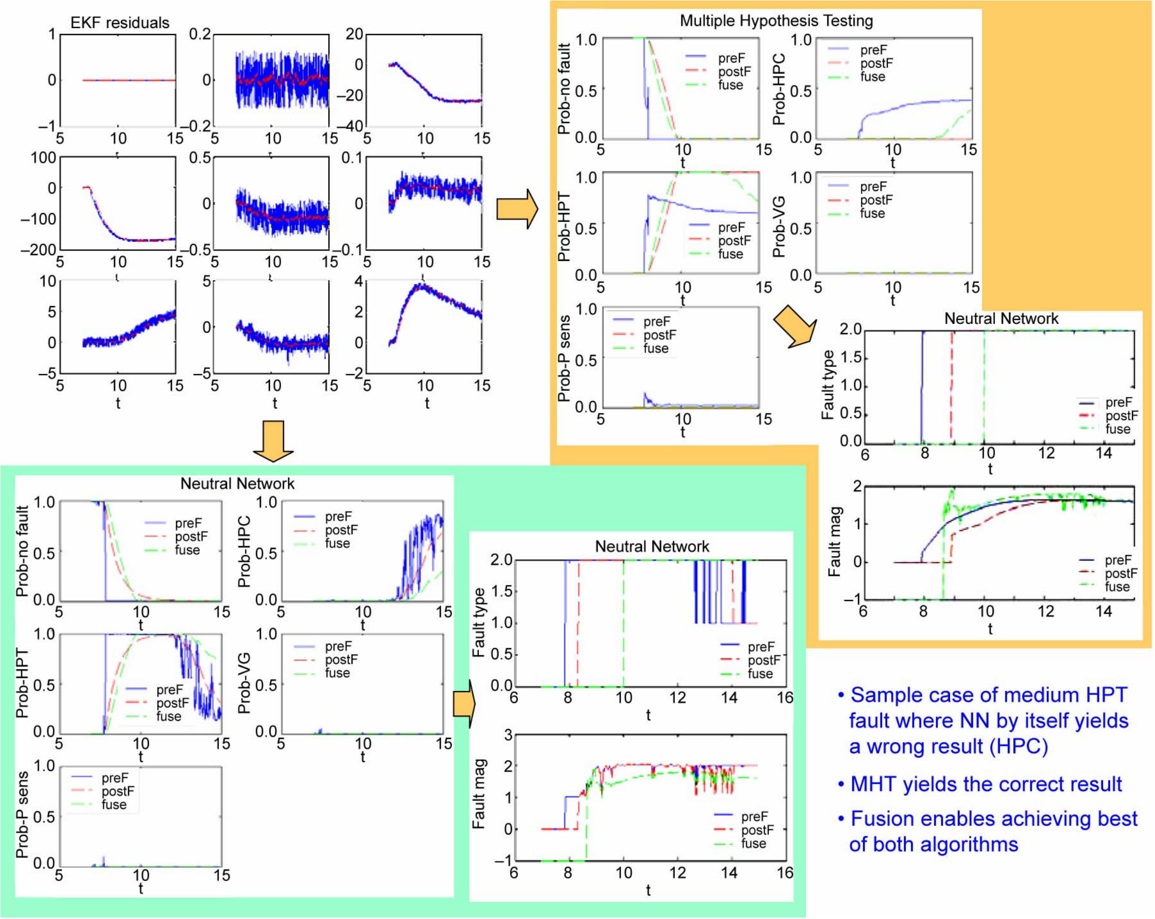
Figure 42.—Fusion algorithm architecture for fault type and magnitude calculation.

particular fault type persists consistently for some minimum time duration before declaring and latching the fault type to engage the corresponding fault accommodation. In parallel, the fault magnitudes from the individual algorithms are also fused using a weighted average using the corresponding probabilities from the respective algorithms as the relative weights. In cases where the fault types identified in the two algorithms differ, the fault magnitude identified in the algorithm with the same fault type as the overall fused fault type is used as the fused fault magnitude.

6.4.5 Fault Detection and Fusion Results

During our initial design phase for creating the online versions of the two fault detection algorithms and the fusion algorithm, we used a random subset of 50 runs from the full set of 2047 runs for the 13 cases of no fault and each fault type and magnitude. The specific parameters of the two algorithms and the fusion algorithm were tuned using these 50 runs to obtain the overall fused result.

Figure 43 shows the results obtained from the two fault-detection and the fusion algorithms for a sample run with medium HPT fault. In this sample run, while MHT algorithm correctly identifies the HPT fault, the NN algorithm yields an incorrect classification as HPC fault. However, the fusion algorithm enables combining the results from the individual algorithms to obtain the correct result, demonstrating the desired capabilities of the fusion algorithm. The plots on the top left corner of the figure show the EKF residuals (after the bias elimination) in blue and the filtered EKF residuals in red. These filtered EKF residuals along the flight condition parameters are used as inputs in the two algorithms. The plots for the MHT algorithm are highlighted with the yellow background. The first set of plots shows the probabilities for each fault type and no fault. The plots in blue are the instantaneous probabilities while the plots in red are the probabilities obtained after the post-filtering step where the instantaneous probabilities are filtered using a frequency-weighted average. Clearly, the post-filtering step yields a significant enhancement of the probability for the HPT fault type compared to the instantaneous probabilities, while suppressing the probabilities for the other fault types and no fault to yield a correctly identified HPT fault. The plots in green are the corresponding probabilities for no fault/fault from the fusion algorithm. The fusion algorithm yields correctly the highest probability for HPT fault for a consistently long enough period to latch to the HPT fault type. The second set of plots show the results for the fault type and magnitude from the MHT and the fusion algorithms. The plots in blue and red show the instantaneous and post-filtered results respectively from the MHT algorithm and the plots in green show the results from the fusion. The corresponding plots for the NN classification are at the bottom and highlighted with the light blue background.



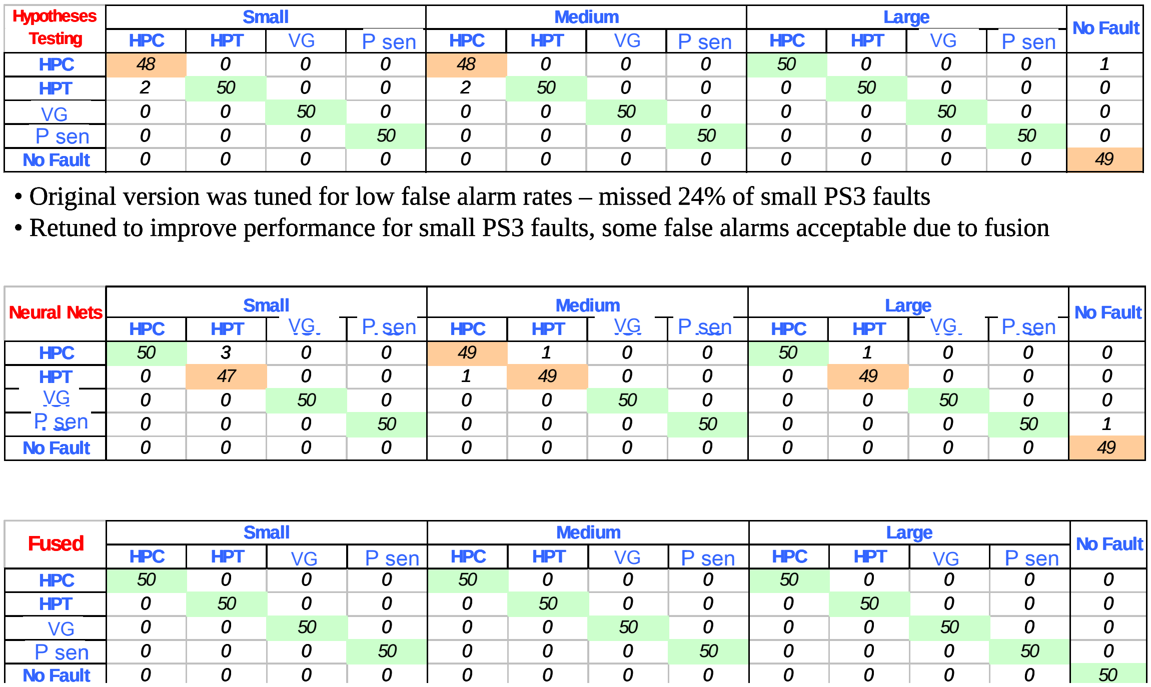
- Sample case of medium HPT fault where NN by itself yields a wrong result (HPC)
- MHT yields the correct result
- Fusion enables achieving best of both algorithms

Figure 43.—Sample result for multiple hypothesis testing, neural network classification and fusion algorithms for a medium HPT fault.

Figure 44 shows the overall confusion matrices from the individual algorithms and the fusion algorithm obtained using the subset of 50 runs for all 13 cases. From these results it is clear that the MHT algorithm has better performance than the neural network classification for HPT fault type, while the reverse holds true for HPC fault type. Note also, that the performance of the MHT algorithm for PS3 sensor faults, especially small faults, is very good compared to.

Figure 39—this is consistent with the retuning of the algorithm to improve the performance for PS3 sensor faults while sacrificing some performance in terms of false alarms. The fusion algorithm enables combining the complimentary features of each algorithm to obtain the overall improved results. The fusion algorithm was tuned to get the perfect results as shown in the last confusion matrix for this subset of 50 runs.

Once the individual fault detection algorithms and the fusion algorithm were tuned using the subset of 50 random runs, their performance was tested for the full set of 2047 runs for all 13 cases. The corresponding confusion matrices are shown in figure 45. The results remain similar for this full set of 2047 runs. In particular, the MHT and NN classification algorithms have complementary performances for HPC and HPT faults and similar performance for false alarms. Note also that with the retuned parameters, the MHT algorithm has a significantly improved performance for small PS3 sensor fault compared to figure 39.



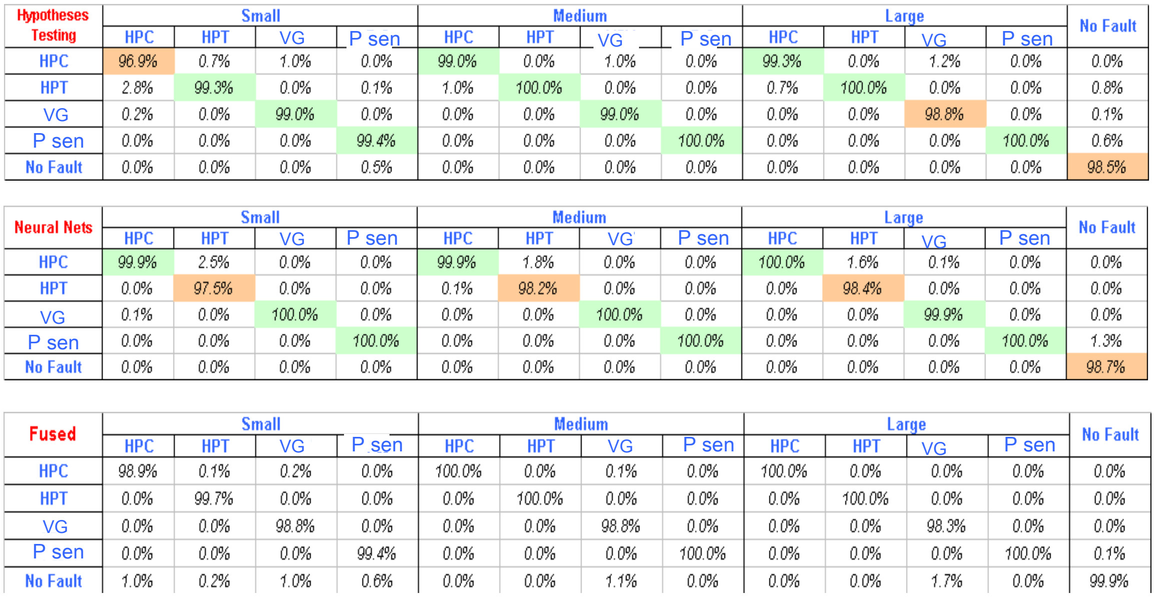
Hypotheses Testing	Small				Medium				Large				No Fault
	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	
HPC	48	0	0	0	48	0	0	0	50	0	0	0	1
HPT	2	50	0	0	2	50	0	0	0	50	0	0	0
VG	0	0	50	0	0	0	50	0	0	0	50	0	0
P sen	0	0	0	50	0	0	0	50	0	0	0	50	0
No Fault	0	0	0	0	0	0	0	0	0	0	0	0	49

- Original version was tuned for low false alarm rates – missed 24% of small PS3 faults
- Retuned to improve performance for small PS3 faults, some false alarms acceptable due to fusion

Neural Nets	Small				Medium				Large				No Fault
	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	
HPC	50	3	0	0	49	1	0	0	50	1	0	0	0
HPT	0	47	0	0	1	49	0	0	0	49	0	0	0
VG	0	0	50	0	0	0	50	0	0	0	50	0	0
P sen	0	0	0	50	0	0	0	50	0	0	0	50	1
No Fault	0	0	0	0	0	0	0	0	0	0	0	0	49

Fused	Small				Medium				Large				No Fault
	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	
HPC	50	0	0	0	50	0	0	0	50	0	0	0	0
HPT	0	50	0	0	0	50	0	0	0	50	0	0	0
VG	0	0	50	0	0	0	50	0	0	0	50	0	0
P sen	0	0	0	50	0	0	0	50	0	0	0	50	0
No Fault	0	0	0	0	0	0	0	0	0	0	0	0	50

Figure 44.—Confusion matrices for multiple hypothesis testing, neural network classification and fusion algorithms for subset of 50 runs used for tuning the algorithms.



Hypotheses Testing	Small				Medium				Large				No Fault
	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	
HPC	96.9%	0.7%	1.0%	0.0%	99.0%	0.0%	1.0%	0.0%	99.3%	0.0%	1.2%	0.0%	0.0%
HPT	2.8%	99.3%	0.0%	0.1%	1.0%	100.0%	0.0%	0.0%	0.7%	100.0%	0.0%	0.0%	0.8%
VG	0.2%	0.0%	99.0%	0.0%	0.0%	0.0%	99.0%	0.0%	0.0%	0.0%	98.8%	0.0%	0.1%
P sen	0.0%	0.0%	0.0%	99.4%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	100.0%	0.6%
No Fault	0.0%	0.0%	0.0%	0.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	98.5%

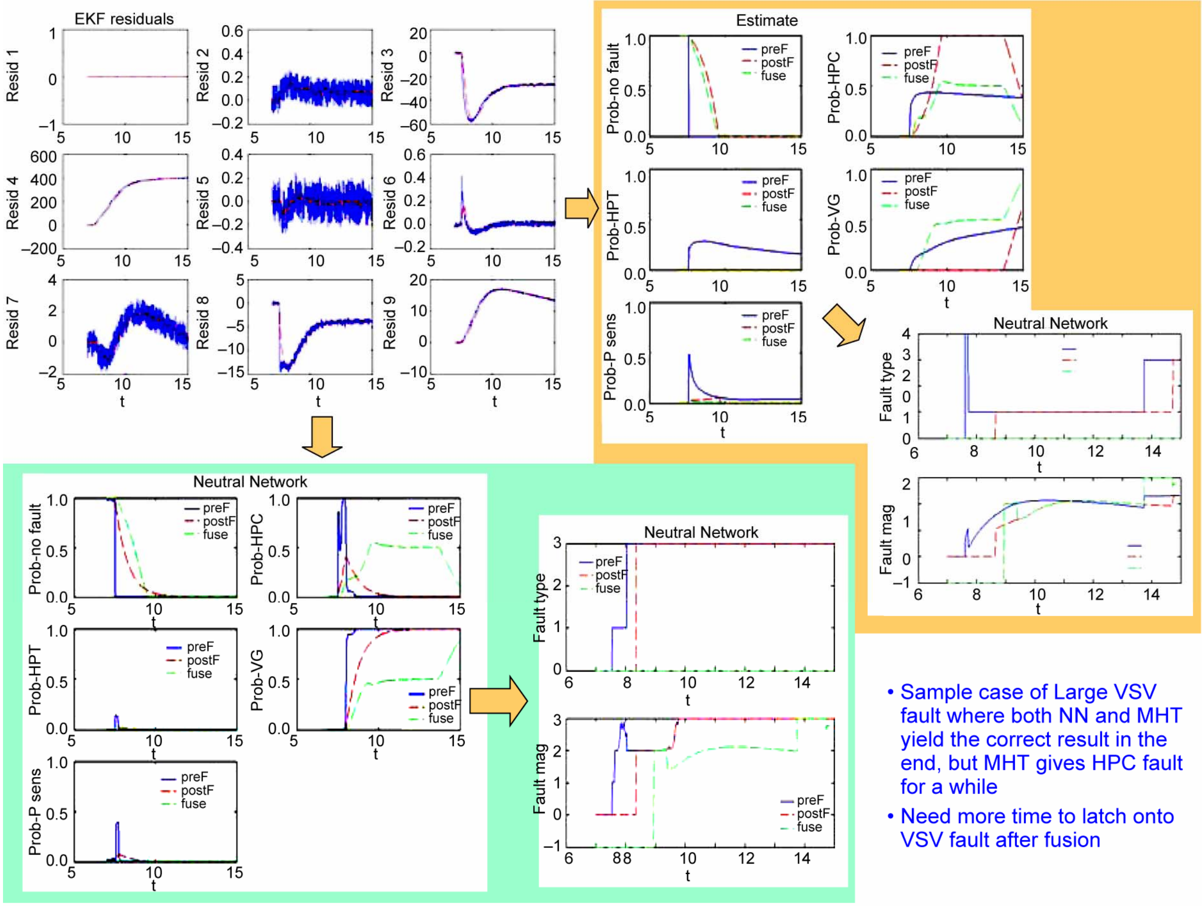
Neural Nets	Small				Medium				Large				No Fault
	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	
HPC	99.9%	2.5%	0.0%	0.0%	99.9%	1.8%	0.0%	0.0%	100.0%	1.6%	0.1%	0.0%	0.0%
HPT	0.0%	97.5%	0.0%	0.0%	0.1%	98.2%	0.0%	0.0%	0.0%	98.4%	0.0%	0.0%	0.0%
VG	0.1%	0.0%	100.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	99.9%	0.0%	0.0%
P sen	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	100.0%	1.3%
No Fault	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	98.7%

Fused	Small				Medium				Large				No Fault
	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	HPC	HPT	VG	P sen	
HPC	98.9%	0.1%	0.2%	0.0%	100.0%	0.0%	0.1%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%
HPT	0.0%	99.7%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%
VG	0.0%	0.0%	98.8%	0.0%	0.0%	0.0%	98.8%	0.0%	0.0%	0.0%	98.3%	0.0%	0.0%
P sen	0.0%	0.0%	0.0%	99.4%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	100.0%	0.1%
No Fault	1.0%	0.2%	1.0%	0.6%	0.0%	0.0%	1.1%	0.0%	0.0%	0.0%	1.7%	0.0%	99.9%

Figure 45.—Confusion matrices for multiple hypothesis testing, neural network classification and fusion algorithms for all 2047 runs.

Finally, the fusion algorithm enables the combination of complementary performances of the two algorithms to yield the desired improved performances for HPC and HPT faults as well as false alarms in comparison to either individual algorithm. Moreover, the fusion algorithm reduces the incidence of false alarms (last column) and miss-classified faults (off-diagonal terms) compared to either individual algorithm. This is consistent with our design of the fusion algorithm to favor missed detection (bottom row) over false alarms and miss-classification to avoid undesired effects of wrong fault accommodation. The fusion algorithm performance for VG faults, on the other hand, may appear to be worse than either algorithm. However, the confusion matrices for the individual algorithms are based on the final detection and classification result at the end of each simulation run. Often, the MHT algorithm miss-classifies the VG fault as HPC fault for a long duration until finally converging on the VG fault at the end of the run. The fusion algorithm result lags the individual algorithms, and thus, would declare the VG fault correctly if there had been some more time in these simulation runs.

Figure 46 shows the results from the individual algorithms and the fusion results for a sample run with large VG fault. In this case, clearly the MHT algorithm incorrectly identifies the fault type as HPC fault for a fairly long time and then converges to the VG fault at the end of the simulation run. The NN algorithm, on the other hand, correctly identifies the fault type as VG. However, owing to the large mismatch in the individual fault probabilities from the two algorithms, the overall fused result yields a fairly low probability for VG fault, and thus doesn’t detect a fault. At the end of the run, the fused VG fault probability rises above the threshold of 0.8 owing to the concurrence between the two algorithms. However, it is not above the threshold for enough time to latch onto VG fault before the simulation run ends. There are several such cases where, given some more time, the fusion would latch on correctly to the VG fault without incurring a false classification. On the other hand, there are some runs where the MHT algorithm correctly identifies the VG fault, but the NN algorithm miss-classifies the fault as HPC fault. Again, in such cases, owing to this discrepancy in the results from the two algorithms, the fusion algorithm yields a low VG fault probability and misses the fault detection rather than detecting a fault but miss-classifying it.



- Sample case of Large VSV fault where both NN and MHT yield the correct result in the end, but MHT gives HPC fault for a while
- Need more time to latch onto VSV fault after fusion

Figure 46.—Sample case of missed-detection of large VG fault with fusion algorithm due to delayed classification as VG fault in multiple hypothesis testing.

7. Fault Accommodation

Following the overall MBFTC approach shown in figure 1, once a fault is detected, an appropriate fault accommodation has to be employed to mitigate the adverse effects of the fault and allow continued and safe operation of the engine. The key objective for fault accommodation is to maintain operability in engines experiencing the selected faults through control actions. Challenges to be addressed include the highly non-linear response of the existing FADEC controller and engine, the uncertainty of operability measures, and variations in flight conditions and engine health. The following sections detail the general requirements for fault accommodation, impact of the selected faults on engine performance and operability, the developed fault accommodation approach and results.

7.1 Operability Requirements

The critical measures we are using for engine operability are the stall margins for the fan, booster, and high-pressure compressor. The existing FADEC logic is designed to protect certain stall margins required for safe operation at different points in the flight envelope for an un-faulted engine. It is our goal to achieve steady state stall margins in a faulted engine that are equal to or greater than the pre-fault values. The accommodation action may increase the exhaust gas temperatures but must not cause the exceedence of exhaust gas temperature limits. On the other hand, it is also desirable to maintain the net engine thrust close to pre-faulted value as much as possible, to achieve the least impact on the aircraft operation.

7.2 Closed-Loop Engine Response

We studied the impact of each of the four faults on engine performance and operability at steady state. To this end, for each fault type, we ran a small Monte Carlo run with engines with engine-to-engine variation and deterioration over the flight envelope, wheren the specific fault type was injected and we monitored the percent change in important engine parameters as a result of the fault. In general, the biggest impact of the HPC, HPT and VG faults was on the fan, booster and compressor stall margins, especially booster and compressor SM. On the other hand, the specific pressure sensor is used in the FADEC only during transient operation, and thus, as expected, the PS3 sensor fault had no impact on the engine at steady state.

7.3 Fault Accommodation Strategy

The *strategy* developed for Fault Accommodation (FA) includes the following steps:

1. Estimate fault type and magnitude from detection and fusion algorithms
2. If the fault is a pressure sensor fault
    a. Replace sensed pressure value by estimated value (using EKF)
3. For HPC, HPT or VG faults
    a. Use fault type, fault magnitude, and flight envelope (FE) data to determine FADEC adjustments from a pre-computed Look-Up Table
    b. Apply FADEC adjustments, which in turn change the control actuators to desired values that mitigate the adverse impact of the fault on engine performance and operability.

We initially focus on the fault accommodation for HPC, HPT and VG faults. As mentioned above, the approach for these faults is to first compute off-line model-based optimum solutions for key FADEC adjustments that minimize the impact of the fault at few chosen points in the flight envelope. These individual optimum point solutions are then used to create the look-up

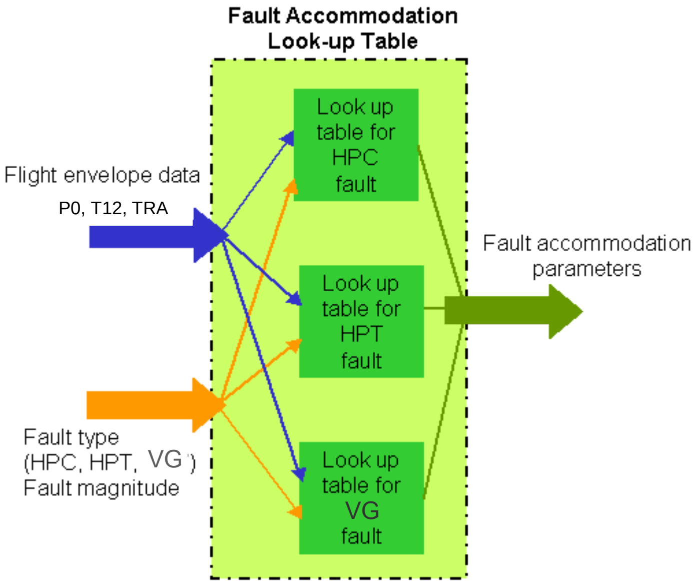

Fault Accommodation
Look-up Table
Flight envelope data
P0, T12, TRA
Fault type
(HPC, HPT, VG)
Fault magnitude
Look up table for HPC fault
Look up table for HPT fault
Look up table for VG fault
Fault accommodation parameters

Figure 47.—Look-up Table for FA of HPC, HPT, and VG faults.

tables. The look up table is actually composed of 3 separate look-up tables, one for each fault type (fig. 47). Inputs to the look up tables are flight envelope (FE) data (TRA, and sensed inlet temperature T12 and pressure P0), and fault data (fault type and magnitude). Outputs from the look up table are the FADEC adjustments, which are added to the ordinary FADEC signals as shown, for example, in figure 1.

The individual look-up tables are constructed offline—by fixing table inputs (fault data and flight condition data), and finding the best FADEC adjustments to achieve our FA goals via global model-based optimization. These are the FADEC adjustments for the table *nodes or design points*. For a generic input (fault data and FE data) in the online implementation, the corresponding FADEC adjustments are computed through interpolation of the look-up tables. More details on the optimization/interpolation steps follow.

Initially, we identified 9 FADEC adjustments as potential candidates to be used for FA. In general terms, the optimal adjustments are found by solving an optimization problem as follows

$$\begin{gathered}\textit{Minimize}_{\textit{FADEC adjustments}}\ \textit{Cost}\\ \textit{s.t. Constraints}\end{gathered}$$

In the *cost* we penalize losses in fan, booster, and HPC stall margins as well as loss in thrust. In the *constraints* we explicitly enforce EGT max limit, and use the existing FADEC logic to enforce other constraints, e.g., actuator constraints. For the optimization we use a global optimizer (Genetic Algorithm) together with a FADEC+engine model (CWS simulation), which enforces constraints like actuator magnitudes/rates, etc.; figure 48.

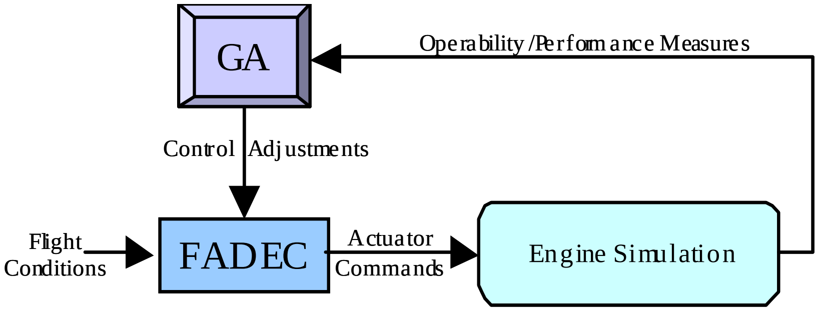


Figure 48.—Schematic showing the global optimizer (GA) in connection with the CWS simulation (FADEC + engine).

Our offline optimization approach evolved as we learned more about the structure of the problem. The final formulation used only 4 *optimization variables* or FADEC adjustments

1. Bleed valve adder
2. VG adder
3. TRA
4. Compressor Bleed Switch

A previous formulation of this optimization used all 9 adjustments, which resulted in good recovery of pre-fault stall margins and thrust for table nodes, but poor FA for other generic FE points. Too many optimization variables resulted in over-parameterization and thus non-unique optimal solutions, which lead to highly discontinuous adjustments—very different sets of adjustments were assigned to two node points very close in the FE, leading to non-smooth tables with poor interpolation capabilities for points in between these nodes. Reducing the number of adjustments to what seems to be the minimum possible, together with including in the *cost* a penalty on the adjustments' departures from their nominal values, gave us good and smooth solutions for both table nodes amenable to online interpolation for generic FE points in between the table nodes.

Figure 49 shows the design and testing points, in the Altitude-Mach (ALT-XM) plane used in the FA approach. FADEC adjustments for the design points (or table nodes) are computed off-line. Adjustments for testing points (which are generic FE points) are computed online via interpolation. Notice that most of the design points are located on the boundaries of the FE. Those set of points can be mapped into the inlet sensor plane P0-T12 (fig. 50).

7.4 Fault Accommodation for HPC Fault

7.4.1 Fault Accommodation for Large HPC Fault

Let us focus for now on the HPC fault type and large fault magnitude. By solving optimization problems (as explained before) we obtain the optimal adjustments for the design points. The optimal adjustments for TRA = 50 are shown in table 13 and for TRA = 75 in table 14. FA adjustments for other TRAs are obtained through interpolations. Notice that optimal TRA adjustments are very close to their nominal values of 50 or 75; therefore, for simplicity, we are not going to be using TRA adjustment for FA (this is valid for HPC faults, but it remains to be seen if it is valid for VG and HPT faults).

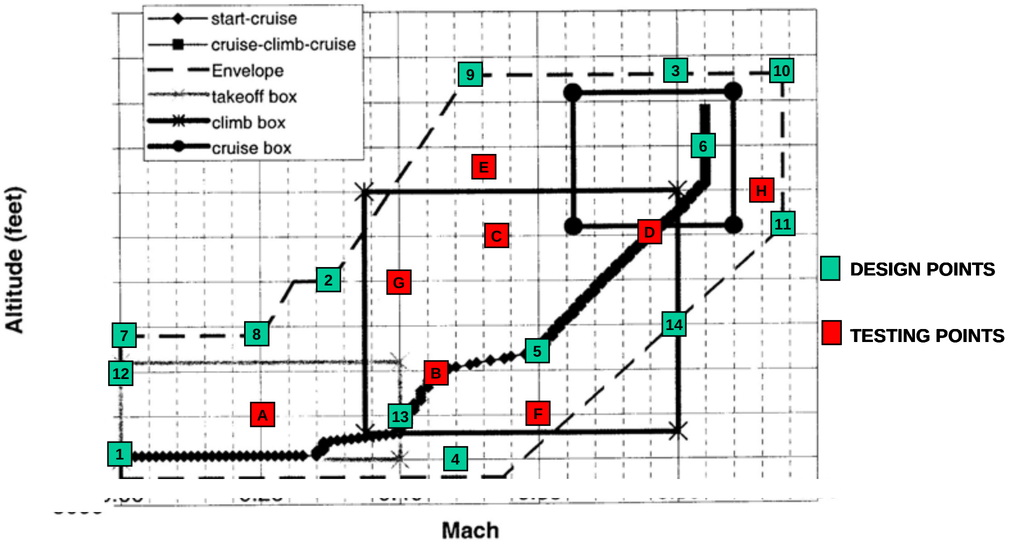


Figure 49.—Location of the 14 Design Points and 8 Testing Points in the Flight Envelope (ALT-Mach plane).

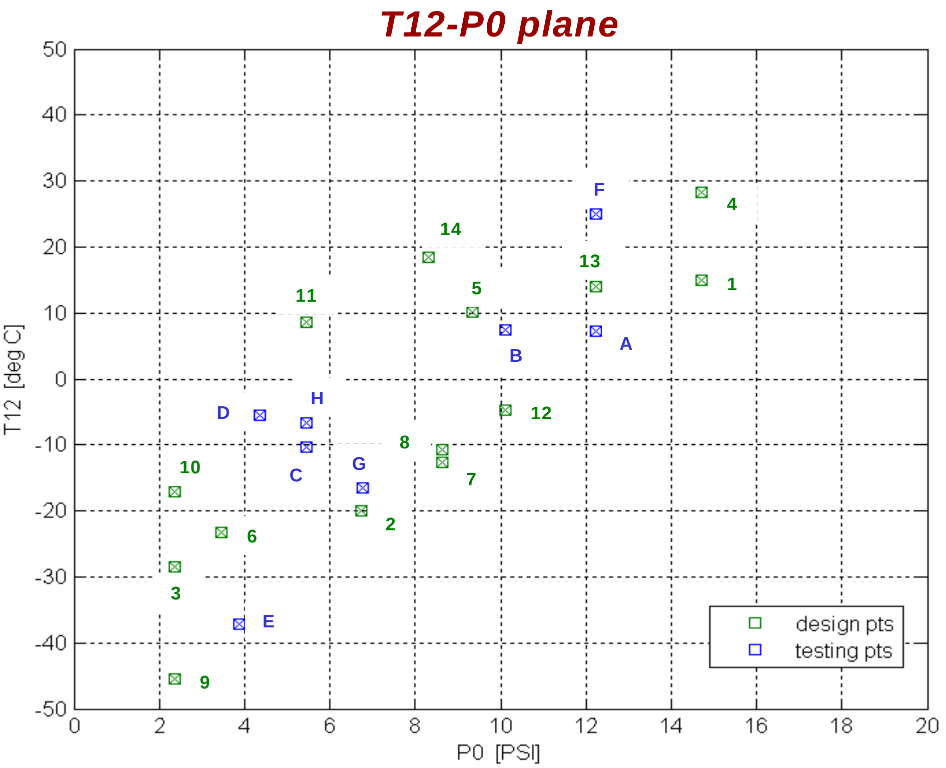


Figure 50.—Design and testing points in the P0-T12 plane (P0 and T12 are inlet variables sensed by the engine).

The particular interpolation method to be used in our FA approach is a key "parameter" of our technique. The interpolation problem is to determine, from the adjustments in tables 13 and 14 the set of 4 adjustments needed to accommodate a large HPC fault for a given value of P0, T12 and TRA (i.e., we are solving an interpolation in the 3D space P0-T12-TRA).

TABLE 13.—OPTIMAL FADEC ADJUSTMENTS FOR DESIGN POINTS FOR TRA = 50

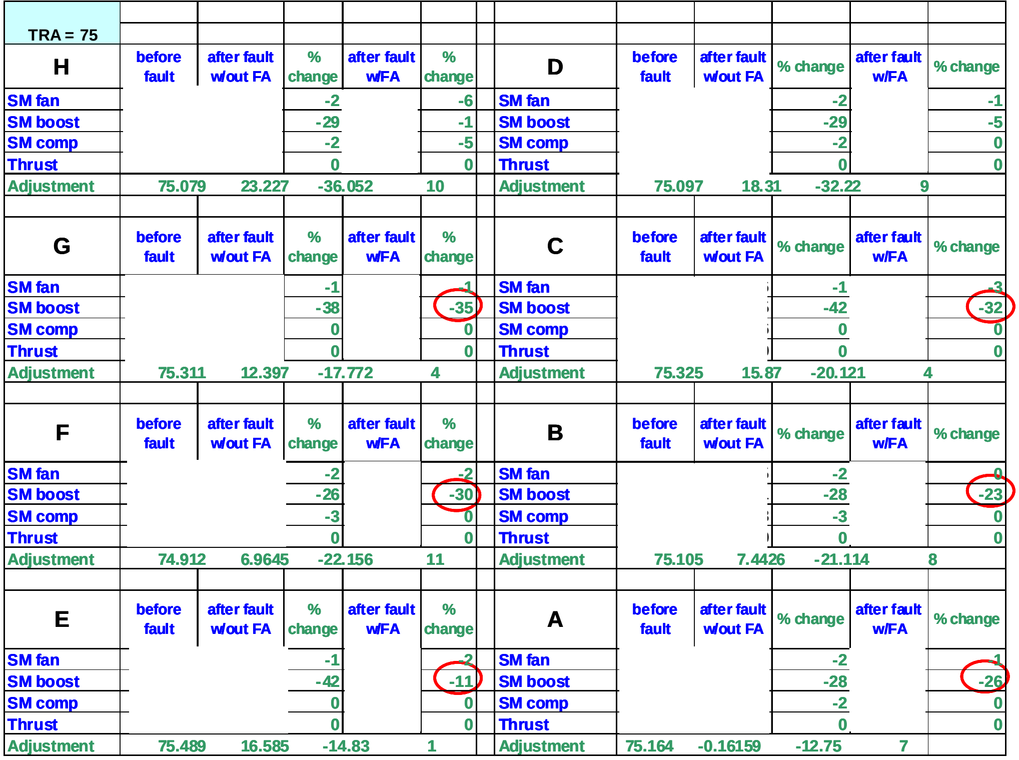
FE pt #	TRA	P0	T12	ftype	fmag	TRA	Bleed adj	VG adj	switch
1	50.00	14.70	14.88	1	3	49.93	17.87	1.67	1.00
2	50.00	6.75	-20.02	1	3	49.98	36.40	0.03	1.00
3	50.00	2.35	-28.56	1	3	50.02	27.78	-0.06	1.00
4	50.00	14.70	28.20	1	3	50.01	15.84	-1.52	2.00
5	50.00	9.35	10.22	1	3	50.01	12.89	0.27	1.00
6	50.00	3.46	-23.23	1	3	50.02	18.40	-0.38	3.00
7	50.00	8.63	-12.65	1	3	50.02	11.52	-0.08	1.00
8	50.00	8.63	-10.64	1	3	50.01	14.90	0.13	1.00
9	50.00	2.35	-45.53	1	3	50.01	20.02	0.48	1.00
10	50.00	2.35	-17.13	1	3	50.01	41.27	-0.25	1.00
11	50.00	5.45	8.61	1	3	50.00	17.24	0.58	1.00
12	50.00	10.11	-4.73	1	3	50.00	15.36	0.71	2.00
13	50.00	12.23	13.91	1	3	50.00	38.49	-1.30	1.00
14	50.00	8.31	18.38	1	3	49.97	25.02	-0.29	1.00

TABLE 14.—OPTIMAL FADEC ADJUSTMENTS FOR DESIGN POINTS FOR TRA = 75

FE pt #	TRA	P0	T12	ftype	fmag	TRA	Bleed adj	VG adj	switch
1	75.00	14.70	14.88	1	3	75.09	15.05	-3.90	2
2	75.00	6.75	-20.02	1	3	75.00	20.49	-3.59	1
3	75.00	2.35	-28.56	1	3	74.89	19.49	-4.51	2
4	75.00	14.70	28.20	1	3	75.00	21.27	-2.84	1
5	75.00	9.35	10.22	1	3	74.99	17.79	-2.07	2
6	75.00	3.46	-23.23	1	3	74.79	22.73	-2.35	2
7	75.00	8.63	-12.65	1	3	75.12	20.68	-5.61	2
8	75.00	8.63	-10.64	1	3	75.01	20.76	-2.88	3
9	75.00	2.35	-45.53	1	3	75.10	22.02	-4.21	9
10	75.00	2.35	-17.13	1	3	75.23	24.67	-4.30	3
11	75.00	5.45	8.61	1	3	75.11	20.05	-1.92	1
12	75.00	10.11	-4.73	1	3	74.95	23.76	-4.71	1
13	75.00	12.23	13.91	1	3	74.96	22.16	-4.37	2
14	75.00	8.31	18.38	1	3	74.97	18.24	-2.15	1

The first approach we proposed was to use the linear interpolation provided by the Matlab function *interpn.* To be able to use *interpn* (see its Matlab help) we needed to create a grid that contains the design points as some of the grid points. For the remaining grid points, we proposed to compute the adjustments via *linear regression*. Table 15 shows the quality of the FA via this approach (for TRA = 75). For design point A through H, table 15 shows the 2 columns showing the percent change in the 3 stall margins and net thrust (FN) with respect to the no fault case, for fault without accommodation and fault with accommodation, respectively. We observe that the Booster SM is the operability parameter most affected by the fault. From table 15, it is obvious that the proposed FA approach performs quite poorly.

TABLE 15.—FA PERFORMANCE FOR TESTING POINTS USING INTERPOLATION BASED ON "LINEAR REGRESSION" (TRA = 75, HALF DETERIORATED ENGINE, LARGE HPC FAULT)



TRA = 75												
H	before fault	after fault w/out FA	% change	after fault w/FA	% change		**D**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-6		SM fan			-2		-1
SM boost			-29		-1		SM boost			-29		-5
SM comp			-2		-5		SM comp			-2		0
Thrust			0		0		Thrust			0		0
Adjustment	75.079	23.227	-36.052	10			Adjustment	75.097	18.31	-32.22	9	
G	before fault	after fault w/out FA	% change	after fault w/FA	% change		**C**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-1		SM fan			-1		-3
SM boost			-38		-35		SM boost			-42		-32
SM comp			0		0		SM comp			0		0
Thrust			0		0		Thrust			0		0
Adjustment	75.311	12.397	-17.772	4			Adjustment	75.325	15.87	-20.121	4	
F	before fault	after fault w/out FA	% change	after fault w/FA	% change		**B**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-2		SM fan			-2		0
SM boost			-26		-30		SM boost			-28		-23
SM comp			-3		0		SM comp			-3		0
Thrust			0		0		Thrust			0		0
Adjustment	74.912	6.9645	-22.156	11			Adjustment	75.105	7.4426	-21.114	8	
E	before fault	after fault w/out FA	% change	after fault w/FA	% change		**A**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-2		SM fan			-2		-1
SM boost			-42		-11		SM boost			-28		-26
SM comp			0		0		SM comp			-2		0
Thrust			0		0		Thrust			0		0
Adjustment	75.489	16.585	-14.83	1			Adjustment	75.164	-0.16159	-12.75	7	

In an attempt to improve this, we proposed to compute the adjustments for the remaining grid points via “local” linear regression; i.e., using only the design points that are “close” to the grid point under consideration. Table 16 shows that this new variation performs only slightly better than its predecessor.

The third approach proposed for interpolation is a much simpler one, which does not use Matlab’s *interpn*. We propose to compute the FA adjustments as the weighted average of the adjustments corresponding to the *N* closest design points (in the P0-T12-TRA space). The weights are given by the normalized inverses of the distances to the corresponding design points. Figure 51 shows a schematic to help explain our interpolation approach.

Table 17 displays the FA results for the 8 testing points. It is clear that this “nearest neighbors” approach yields the best results so far. Due to its relative simplicity and effectiveness, this is the approach we selected for interpolation. Results for FA for testing points for different TRAs are shown in table 18 (TRA 50), table 19 (TRA 35), table 20 (TRA 60), and table 21 (TRA 85). As we see from these tables, some cases cannot be run straightforward in CWS. In general, however, the FA performance is excellent, with almost perfect recovery of SMs and Thrust in the vast majority of the testing points.

TABLE 16.—FA PERFORMANCE FOR TESTING POINTS USING INTERPOLATION BASED ON "LOCAL LINEAR REGRESSION" (TRA = 75, HALF DETERIORATED ENGINE, LARGE HPC FAULT)

TRA = 75 (local LUT)											
H	before fault	after fault w/out FA	% change	after fault w/FA	% change	**D**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-7	SM fan			-2		-4
SM boost			-30		0	SM boost			-29		-20
SM comp			-3		-9	SM comp			-3		-4
Thrust			0		2	Thrust			0		1
Adjustment	73.741	29.21	-48.196	7		Adjustment	74.027	17.436	-48.436	5	
G	before fault	after fault w/out FA	% change	after fault w/FA	% change	**C**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-4	SM fan			-1		-1
SM boost			-39		-7	SM boost			-42		0
SM comp			0		-1	SM comp			0		0
Thrust			0		0	Thrust			0		0
Adjustment	75.64	20.048	-21.606	7		Adjustment	73.858	17.142	-48.068	2	
F	before fault	after fault w/out FA	% change	after fault w/FA	% change	**B**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-7	SM fan			-2		-2
SM boost			-26		0	SM boost			-28		-30
SM comp			-5		-10	SM comp			-4		0
Thrust			0		0	Thrust			0		1
Adjustment	74.864	49.61	-13.388	14		Adjustment	75.142	-93.398	-48.723	4	
E	before fault	after fault w/out FA	% change	after fault w/FA	% change	**A**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		-5	SM fan			-2		0
SM boost			-43		-16	SM boost			-28		-27
SM comp			0		-4	SM comp			-3		0
Thrust			0		1	Thrust			0		1
Adjustment	74.703	20.23	-5.0529	4		Adjustment	75.802	-91.356	-47.51	3	

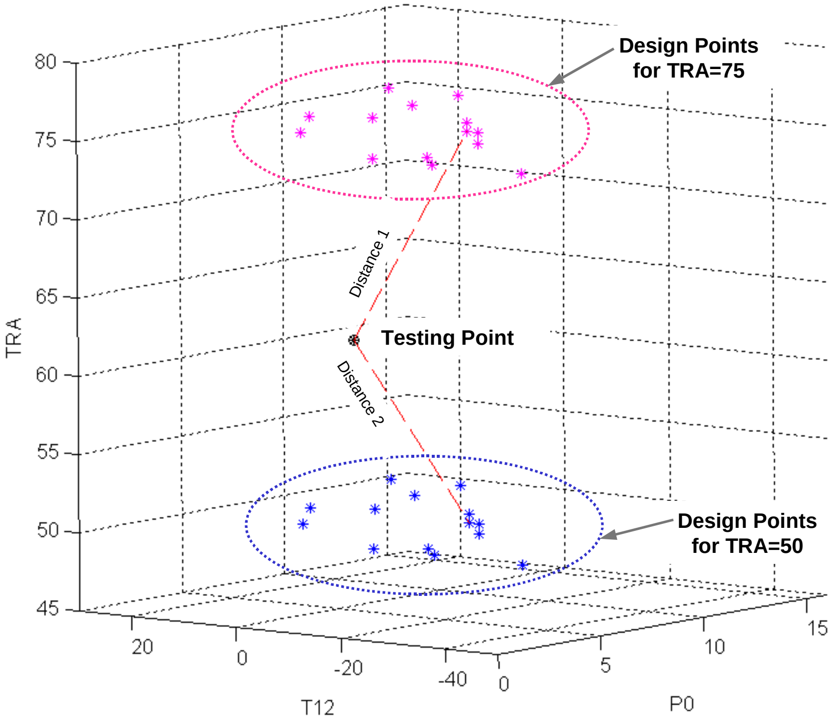
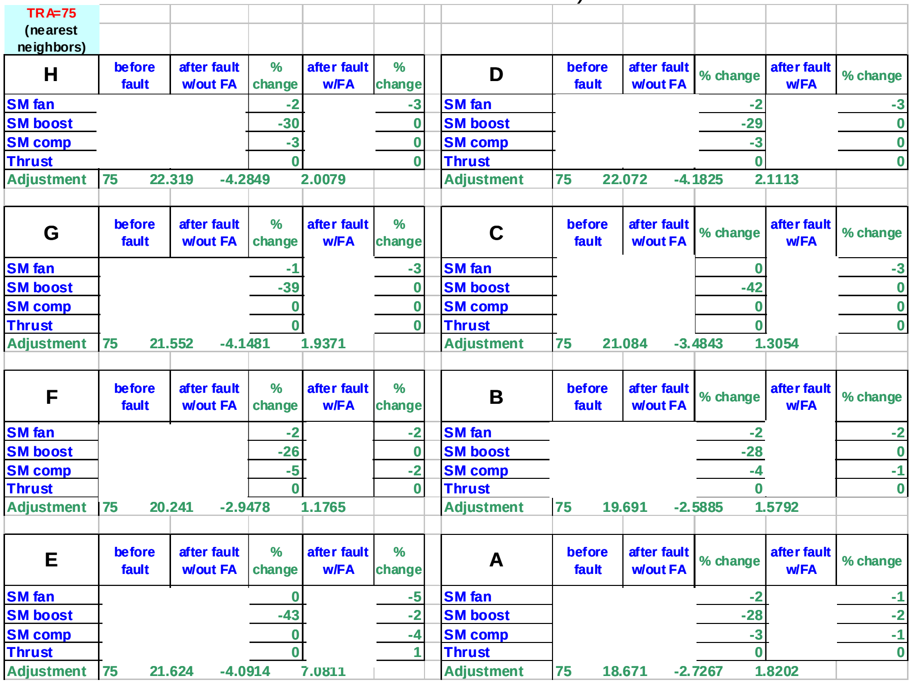


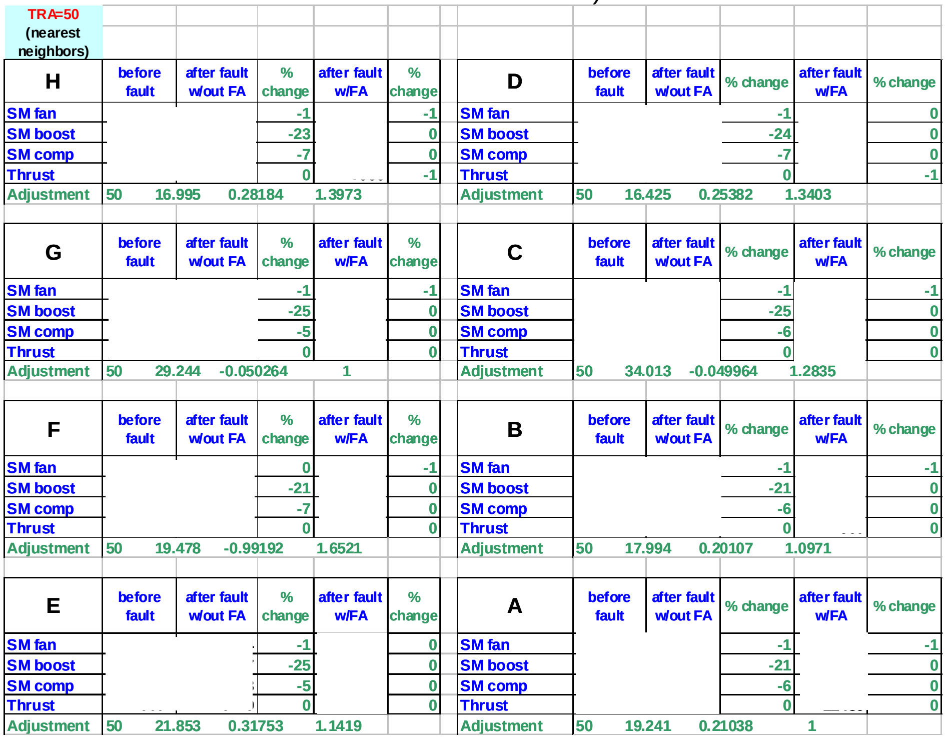
Figure 51.—Schematic for interpolation by weighted average of adjustments for nearest neighbors.

TABLE 17.—FA PERFORMANCE FOR TESTING POINTS USING INTERPOLATION BASED ON "NEAREST NEIGHBORS" (TRA = 75, HALF DETERIORATED ENGINE, LARGE HPC FAULT)



TRA=75 (nearest neighbors)											
H	before fault	after fault w/out FA	% change	after fault w/FA	% change	D	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-3	SM fan			-2		-3
SM boost			-30		0	SM boost			-29		0
SM comp			-3		0	SM comp			-3		0
Thrust			0		0	Thrust			0		0
Adjustment	75	22.319	-4.2849	2.0079		Adjustment	75	22.072	-4.1825	2.1113	
G	before fault	after fault w/out FA	% change	after fault w/FA	% change	C	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-3	SM fan			0		-3
SM boost			-39		0	SM boost			-42		0
SM comp			0		0	SM comp			0		0
Thrust			0		0	Thrust			0		0
Adjustment	75	21.552	-4.1481	1.9371		Adjustment	75	21.084	-3.4843	1.3054	
F	before fault	after fault w/out FA	% change	after fault w/FA	% change	B	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-2	SM fan			-2		-2
SM boost			-26		0	SM boost			-28		0
SM comp			-5		-2	SM comp			-4		-1
Thrust			0		0	Thrust			0		0
Adjustment	75	20.241	-2.9478	1.1765		Adjustment	75	19.691	-2.5885	1.5792	
E	before fault	after fault w/out FA	% change	after fault w/FA	% change	A	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		-5	SM fan			-2		-1
SM boost			-43		-2	SM boost			-28		-2
SM comp			0		-4	SM comp			-3		-1
Thrust			0		1	Thrust			0		0
Adjustment	75	21.624	-4.0914	7.0811		Adjustment	75	18.671	-2.7267	1.8202	

TABLE 18.—FA PERFORMANCE FOR TESTING POINTS USING INTERPOLATION BASED ON "NEAREST NEIGHBORS" (TRA = 50, HALF DETERIORATED ENGINE, LARGE HPC FAULT)



TRA=50 (nearest neighbors)											
H	before fault	after fault w/out FA	% change	after fault w/FA	% change	D	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-1	SM fan			-1		0
SM boost			-23		0	SM boost			-24		0
SM comp			-7		0	SM comp			-7		0
Thrust			0		-1	Thrust			0		-1
Adjustment	50	16.995	0.28184	1.3973		Adjustment	50	16.425	0.25382	1.3403	
G	before fault	after fault w/out FA	% change	after fault w/FA	% change	C	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-1	SM fan			-1		-1
SM boost			-25		0	SM boost			-25		0
SM comp			-5		0	SM comp			-6		0
Thrust			0		0	Thrust			0		0
Adjustment	50	29.244	-0.050264	1		Adjustment	50	34.013	-0.049964	1.2835	
F	before fault	after fault w/out FA	% change	after fault w/FA	% change	B	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		-1	SM fan			-1		-1
SM boost			-21		0	SM boost			-21		0
SM comp			-7		0	SM comp			-6		0
Thrust			0		0	Thrust			0		0
Adjustment	50	19.478	-0.99192	1.6521		Adjustment	50	17.994	0.20107	1.0971	
E	before fault	after fault w/out FA	% change	after fault w/FA	% change	A	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		0	SM fan			-1		-1
SM boost			-25		0	SM boost			-21		0
SM comp			-5		0	SM comp			-6		0
Thrust			0		0	Thrust			0		0
Adjustment	50	21.853	0.31753	1.1419		Adjustment	50	19.241	0.21038	1	

TABLE 19.—FA PERFORMANCE FOR TESTING POINTS FOR TRA = 35, HALF DETERIORATED ENGINE, LARGE HPC FAULT

TRA=35 (nearest neighbors)												
H	before fault	after fault w/out FA	% change	after fault w/FA	% change		D	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			######		#####		SM fan			#DIV/0!		#DIV/0!
SM boost			[illegible]		#####		SM boost			#DIV/0!		#DIV/0!
SM comp			######		#####		SM comp			#DIV/0!		#DIV/0!
Thrust			######		#####		Thrust			#DIV/0!		#DIV/0!
Adjustment	35	19.482	0.16452	1.2808			Adjustment	[illegible]	19.464	0.15754	1.2712	
G	before fault	after fault w/out FA	% change	after fault w/FA	% change		C	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		-1		SM fan			0		0
SM boost			-14		0		SM boost			-23		0
SM comp			-6		0		SM comp			-11		0
Thrust			0		-1		Thrust			0		9
Adjustment	35	26.335	-0.045197	1			Adjustment	35	31.075	-0.16778	1.5281	
F	before fault	after fault w/out FA	% change	after fault w/FA	% change		B	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		0		SM fan			0		0
SM boost			-6		-1		SM boost			-5		0
SM comp			-7		0		SM comp			-5		0
Thrust			0		13		Thrust			0		-1
Adjustment	35	23.469	-0.449	1.3076			Adjustment	35	20.563	0.082228	1.2122	
E	before fault	after fault w/out FA	% change	after fault w/FA	% change		A	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-1		SM fan			-1		0
SM boost			-28		0		SM boost			-5		0
SM comp			-10		0		SM comp			-6		0
Thrust			0		5		Thrust			0		0
Adjustment	35	24.289	0.10327	1.3796			Adjustment	35	21.452	0.27987	1	

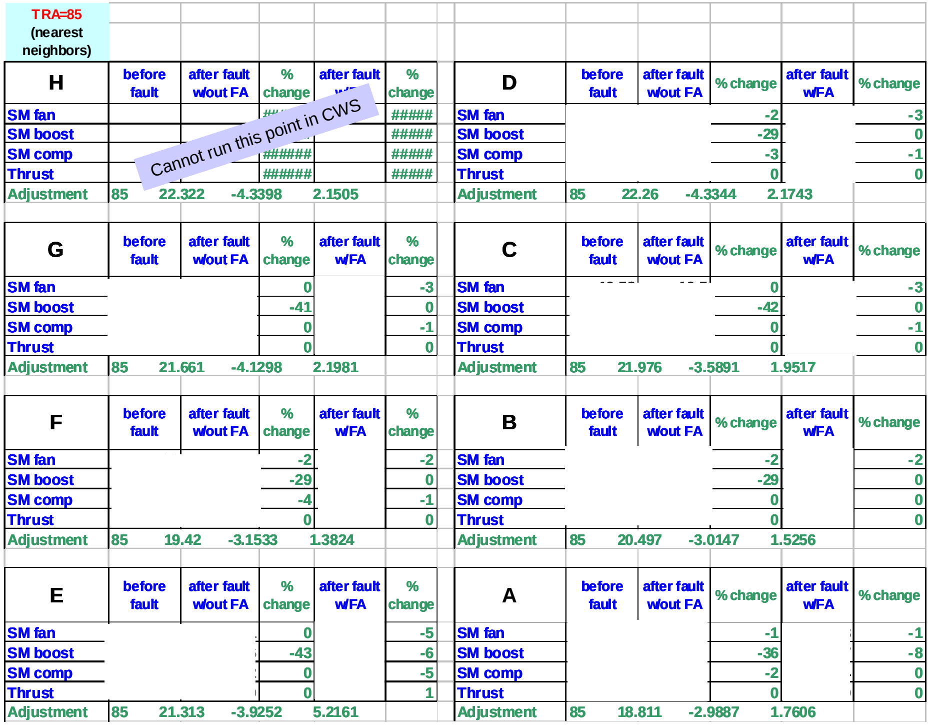
Cannot run this point in CWS

Cannot run this point in CWS

TABLE 20.—FA PERFORMANCE FOR TESTING POINTS FOR TRA = 60, HALF DETERIORATED ENGINE, LARGE HPC FAULT

TRA=60 (nearest neighbors)												
H	before fault	after fault w/out FA	% change	after fault w/FA	% change		D	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-1		SM fan			-2		-1
SM boost			-27		0		SM boost			-28		0
SM comp			-4		0		SM comp			-5		0
Thrust			0		-1		Thrust			0		0
Adjustment	60	18.741	0.19026	1.3038			Adjustment	60	18.632	0.17883	1.2864	
G	before fault	after fault w/out FA	% change	after fault w/FA	% change		C	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-3		SM fan			-2		-4
SM boost			-27		0		SM boost			-28		0
SM comp			-3		-2		SM comp			-3		-2
Thrust			0		0		Thrust			0		0
Adjustment	60	26.641	-0.047489	1			Adjustment	60	31.155	-0.16863	1.5495	
F	before fault	after fault w/out FA	% change	after fault w/FA	% change		B	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		-1		SM fan			-2		-1
SM boost			-25		0		SM boost			-27		0
SM comp			-5		-1		SM comp			-5		-1
Thrust			0		0		Thrust			0		0
Adjustment	60	22.805	-0.53023	1.3585			Adjustment	60	20.15	0.10112	1.1855	
E	before fault	after fault w/out FA	% change	after fault w/FA	% change		A	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-2		SM fan			-2		-1
SM boost			-27		0		SM boost			-24		0
SM comp			-3		0		SM comp			-5		-2
Thrust			0		0		Thrust			0		0
Adjustment	60	22.09	-1.3675	2.8464			Adjustment	60	21.254	0.26112	1	

TABLE 21.—FA PERFORMANCE FOR TESTING POINTS FOR TRA = 85, HALF DETERIORATED ENGINE, LARGE HPC FAULT



TRA=85 (nearest neighbors)												
H	before fault	after fault w/out FA	% change	after fault w/FA	% change		**D**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			[illegible]		#####		SM fan			-2		-3
SM boost			[illegible]		#####		SM boost			-29		0
SM comp			######		#####		SM comp			-3		-1
Thrust			######		#####		Thrust			0		0
Adjustment	85	22.322	-4.3398	2.1505			Adjustment	85	22.26	-4.3344	2.1743	
G	before fault	after fault w/out FA	% change	after fault w/FA	% change		**C**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		-3		SM fan			0		-3
SM boost			-41		0		SM boost			-42		0
SM comp			0		-1		SM comp			0		-1
Thrust			0		0		Thrust			0		0
Adjustment	85	21.661	-4.1298	2.1981			Adjustment	85	21.976	-3.5891	1.9517	
F	before fault	after fault w/out FA	% change	after fault w/FA	% change		**B**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-2		-2		SM fan			-2		-2
SM boost			-29		0		SM boost			-29		0
SM comp			-4		-1		SM comp			0		0
Thrust			0		0		Thrust			0		0
Adjustment	85	19.42	-3.1533	1.3824			Adjustment	85	20.497	-3.0147	1.5256	
E	before fault	after fault w/out FA	% change	after fault w/FA	% change		**A**	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		-5		SM fan			-1		-1
SM boost			-43		-6		SM boost			-36		-8
SM comp			0		-5		SM comp			-2		0
Thrust			0		1		Thrust			0		0
Adjustment	85	21.313	-3.9252	5.2161			Adjustment	85	18.811	-2.9887	1.7606	

Cannot run this point in CWS

7.4.2 Robustness of Fault Accommodation Approach

One important issue to be studied is the *robustness* of the whole FA approach with respect to engine parameter variations. Let us assume that engine variation due to deterioration dominates other causes of variation. Notice that for the model-based optimization problems solved to compute FA adjustments for design points we have been using half-deteriorated engine models, consistent with a fleet-average engine. A simple robustness analysis consists in determining the FA performance of these adjustments when used to accommodate faults for new engines and for full-deteriorated engines. Tables 23 and 22 show FA performance for testing points for new and full-deteriorated engines, respectively. At a first sight the FA results for a new engine do not seem to be good enough for the Booster SM. However, if we look closer, the Booster SMs recovered by FA for a new engine are, in all 7 cases, larger than the Booster SMs for fully deteriorated, un-faulted engines, and thus, we can claim that the FA is good enough. The bar charts in figure 52 capture this key aspect.

7.4.3 Fault Accommodation for Small and Medium HPC Faults

So far we have been considering only large HPC faults. Given the FA adjustments for large faults, the question is how do we compute the adjustments for *small* size and *medium* size faults.

We propose to downscale the adjustments obtained for large faults by 2/3 for FA of medium faults and by 1/3 for FA of small faults. This is consistent with the scale of small, medium and large faults being in the ratio 1:2:3. This approach is quite simple and proved to be very effective. As an example, table 24 shows that this approach is sufficient for medium faults for TRA = 85 (which seems to be the most demanding TRA). Similar results hold true for small faults and for other TRAs.

TABLE 22.—FA ROBUSTNESS FOR TESTING POINTS FOR TRA = 85, FULL-DETERIORATED ENGINE, LARGE HPC FAULT

TRA=85 FULLY DET ENG (nearest neighbors)

Cannot run this point in CWS

H	before fault	after fault w/out FA	% change	after fault w/FA	% change	D	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			[illegible]		#DIV/0!	SM fan			-1		-2
SM boost			[illegible]		#DIV/0!	SM boost			-21		0
SM comp			######		#DIV/0!	SM comp			-2		0
Thrust			######		#DIV/0!	Thrust			0		0
Adjustment						Adjustment	85	22.26	-4.3344	2	
G	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**	**C**	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**
SM fan			0		-2	SM fan			0		-3
SM boost			-31		0	SM boost			-31		0
SM comp			0		0	SM comp			0		-1
Thrust			0		0	Thrust			0		0
Adjustment	85	21.661	-4.1298	2		Adjustment	85	21.976	-3.5891	2	
F	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**	**B**	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**
SM fan			-1		-1	SM fan			-1		-1
SM boost			-20		0	SM boost			-22		0
SM comp			-4		-1	SM comp			-3		0
Thrust			0		0	Thrust			0		0
Adjustment	85	19.42	-3.1533	1		Adjustment	85	20.497	-3.0147	2	
E	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**	**A**	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**
SM fan			0		-5	SM fan			-1		-1
SM boost			-34		0	SM boost			-29		0
SM comp			0		-5	SM comp			-2		0
Thrust			0		1	Thrust			0		0
Adjustment	85	21.313	-3.9252	5		Adjustment	85	18.811	-2.9887	2	

TABLE 23.—FA ROBUSTNESS FOR TESTING POINTS FOR TRA = 85, NEW ENGINE, LARGE HPC FAULT

TRA=85 NEW ENG (nearest neighbors)

Cannot run this point in CWS

H	before fault	after fault w/out FA	% change	after fault w/FA	% change	D	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			[illegible]		#####	SM fan			-3		-3
SM boost			[illegible]		#####	SM boost			-38		-7
SM comp			######		#####	SM comp			-4		-1
Thrust			######		#####	Thrust			0		1
Adjustment						Adjustment	85	22.26	-4.3344	2.1743	
G	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**	**C**	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**
SM fan			0		-2	SM fan			0		-3
SM boost			-46		-10	SM boost			-47		-13
SM comp			0		0	SM comp			0		0
Thrust			0		0	Thrust			0		0
Adjustment	85	21.661	-4.1298	2		Adjustment	85	21.976	-3.5891	2	
F	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**	**B**	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**
SM fan			-3		-2	SM fan			-3		-2
SM boost			-36		-9	SM boost			-36		-10
SM comp			-5		-2	SM comp			-4		-2
Thrust			0		1	Thrust			0		0
Adjustment	85	19.42	-3.1533	1		Adjustment	85	20.497	-3.0147	2	
E	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**	**A**	**before fault**	**after fault w/out FA**	**% change**	**after fault w/FA**	**% change**
SM fan			0		-5	SM fan			-1		-1
SM boost			-46		-15	SM boost			-40		-16
SM comp			0		-1	SM comp			-1		0
Thrust			0		1	Thrust			0		0
Adjustment	85	21.313	-3.9252	5		Adjustment	85	18.811	-2.9887	2	

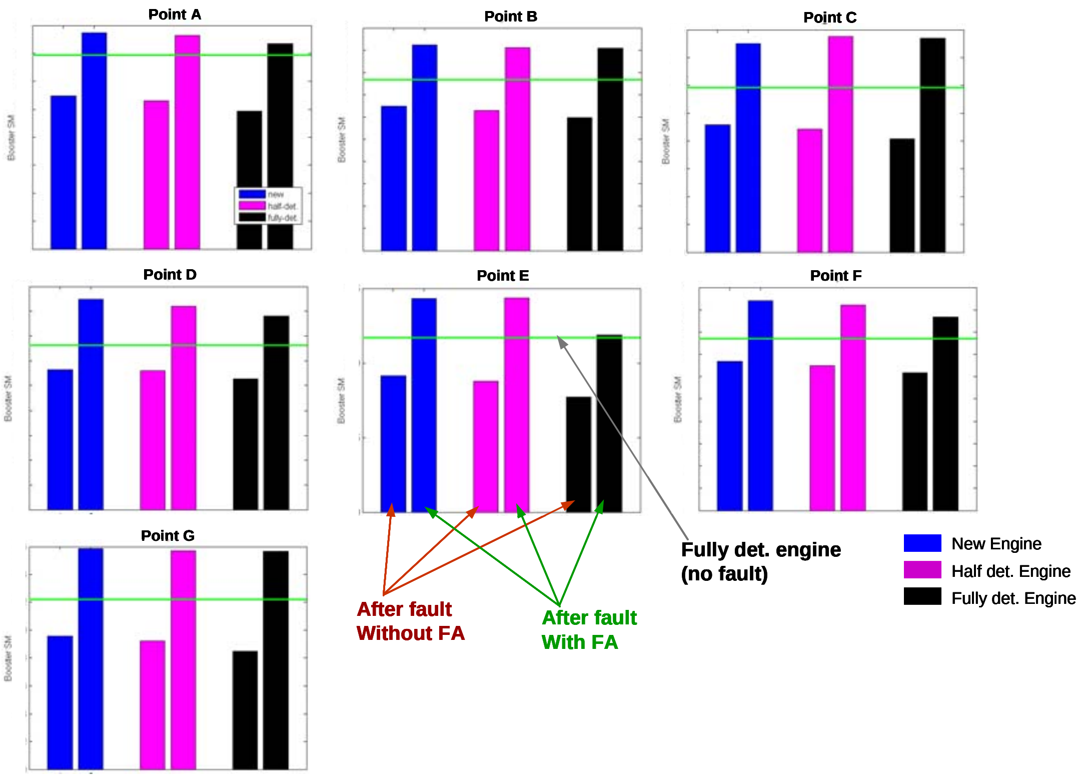
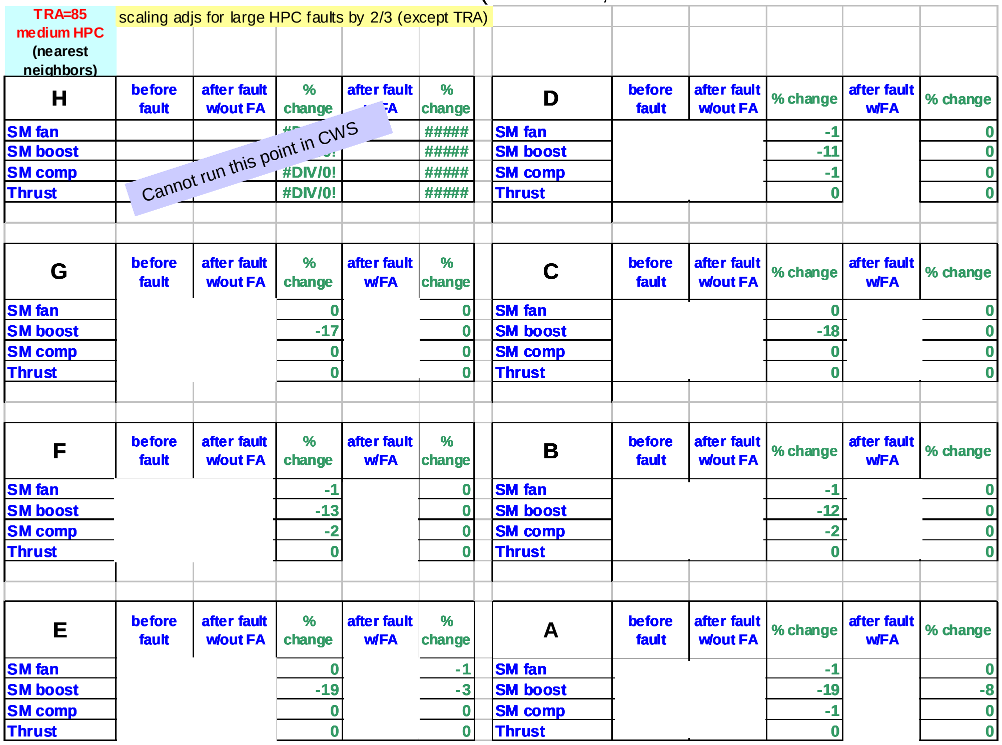


Figure 52.—Booster SM for engines at different deterioration levels with/without FA (TRA = 85, large HPC).

TABLE 24.—FA FOR MEDIUM HPC FAULTS USING ADJUSTMENTS FOR LARGE FAULTS DOWNSCALED BY 2/3 (TRA = 85, HALF-DETERIORATED ENGINE)



TRA=85 medium HPC (nearest neighbors)	scaling adjs for large HPC faults by 2/3 (except TRA)				
H	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			#DIV/0!		#####
SM boost			#DIV/0!		#####
SM comp			#DIV/0!		#####
Thrust			#DIV/0!		#####

Cannot run this point in CWS

D	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		0
SM boost			-11		0
SM comp			-1		0
Thrust			0		0

G	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		0
SM boost			-17		0
SM comp			0		0
Thrust			0		0

C	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		0
SM boost			-18		0
SM comp			0		0
Thrust			0		0

F	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		0
SM boost			-13		0
SM comp			-2		0
Thrust			0		0

B	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		0
SM boost			-12		0
SM comp			-2		0
Thrust			0		0

E	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			0		-1
SM boost			-19		-3
SM comp			0		0
Thrust			0		0

A	before fault	after fault w/out FA	% change	after fault w/FA	% change
SM fan			-1		0
SM boost			-19		-8
SM comp			-1		0
Thrust			0		0

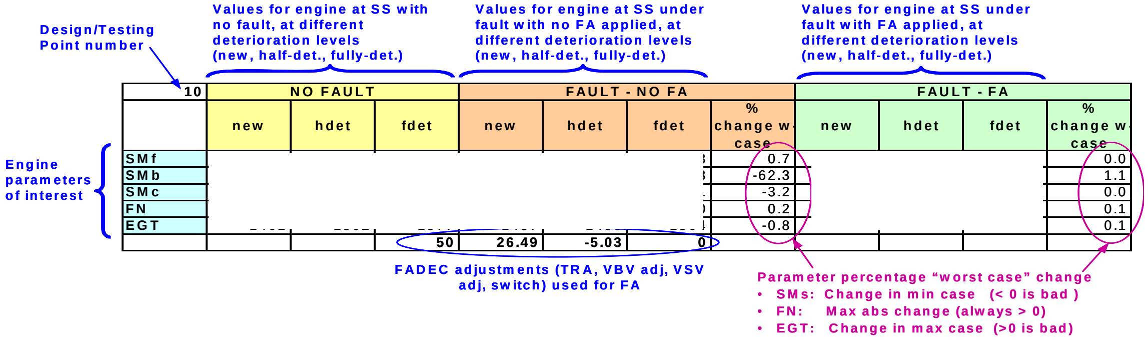
7.5 Fault Accommodation for HPT Faults

7.5.1 Fault Accommodation for Large HPT Faults

In this section, we present the results for large HPT fault accommodation using the general methodology described in section 7.3. Again, the optimum solutions were obtained off-line using model-based optimization at the design points shown in figure 49 and validated using online table look-ups at the testing points.

In the rest of this document, we will display the performance and robustness against deterioration of the developed FA approach for any given point in the flight envelope and fault type/magnitude using the “table format” shown in figure 53. The basic idea is to compare 3 fault situations (healthy engine, faulted engine with no FA applied, and faulted engine with FA applied) across different deterioration levels (new, half deteriorated, and fully deteriorated engines). Note that while the off-line model-based optimizations are done for half-deteriorated engine, it is applied online to any engine irrespective of deterioration level, which is not known. The columns marked by magenta ovals show worst-case percent change for both cases (without FA, with FA). In the case of the 3 stall margins, worst-case means changes in the minima over the 3 deterioration levels (e.g., 100*(4.803/12.73 -1) = –62.3%). In the case of FN, it means max absolute (i.e., disregarding the sign) percent change over all 3 deterioration levels, since ideally we don’t want the FA to lead to significant decrease or increase in net thrust. For EGT we look at changes in the max over the 3 deterioration levels. In short, negative numbers for percent changes in SMs imply loss in SM and are undesirable, while positive numbers for percent changes in EGT imply increase in EGT and is allowed if not violating the max EGT limit. On the other hand, any change in FN is undesirable, since we would like to recover un-faulted thrust if possible.

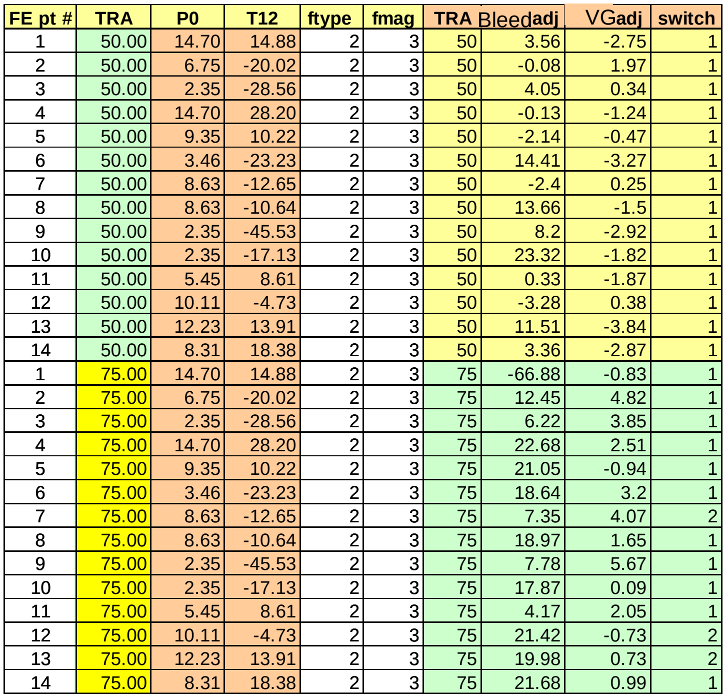
In this section, we focus on the HPT fault type and large fault magnitude. By solving the optimization problems explained before, we obtain the optimal adjustments for the design points. The optimal adjustments for TRA = 50 and for TRA = 75 are shown in figure 54. FA adjustments for other TRAs are obtained through interpolations. Notice that optimal TRA adjustments were very close to their nominal values of 50 or 75; therefore, for simplicity, we are not going to be actually using TRA adjustments for FA.



10	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.7				0.0
SMb							-62.3				1.1
SMc							-3.2				0.0
FN							0.2				0.1
EGT							-0.8				0.1
			50	26.49	-5.03	0					

Design/Testing Point number
Values for engine at SS with no fault, at different deterioration levels (new, half-det., fully-det.)
Values for engine at SS under fault with no FA applied, at different deterioration levels (new, half-det., fully-det.)
Values for engine at SS under fault with FA applied, at different deterioration levels (new, half-det., fully-det.)
Engine parameters of interest
FADEC adjustments (TRA, VBV adj, VSV adj, switch) used for FA
Parameter percentage “worst case” change
• SMs: Change in min case (< 0 is bad)
• FN: Max abs change (always > 0)
• EGT: Change in max case (>0 is bad)

Figure 53.—Table format used to display FA results.



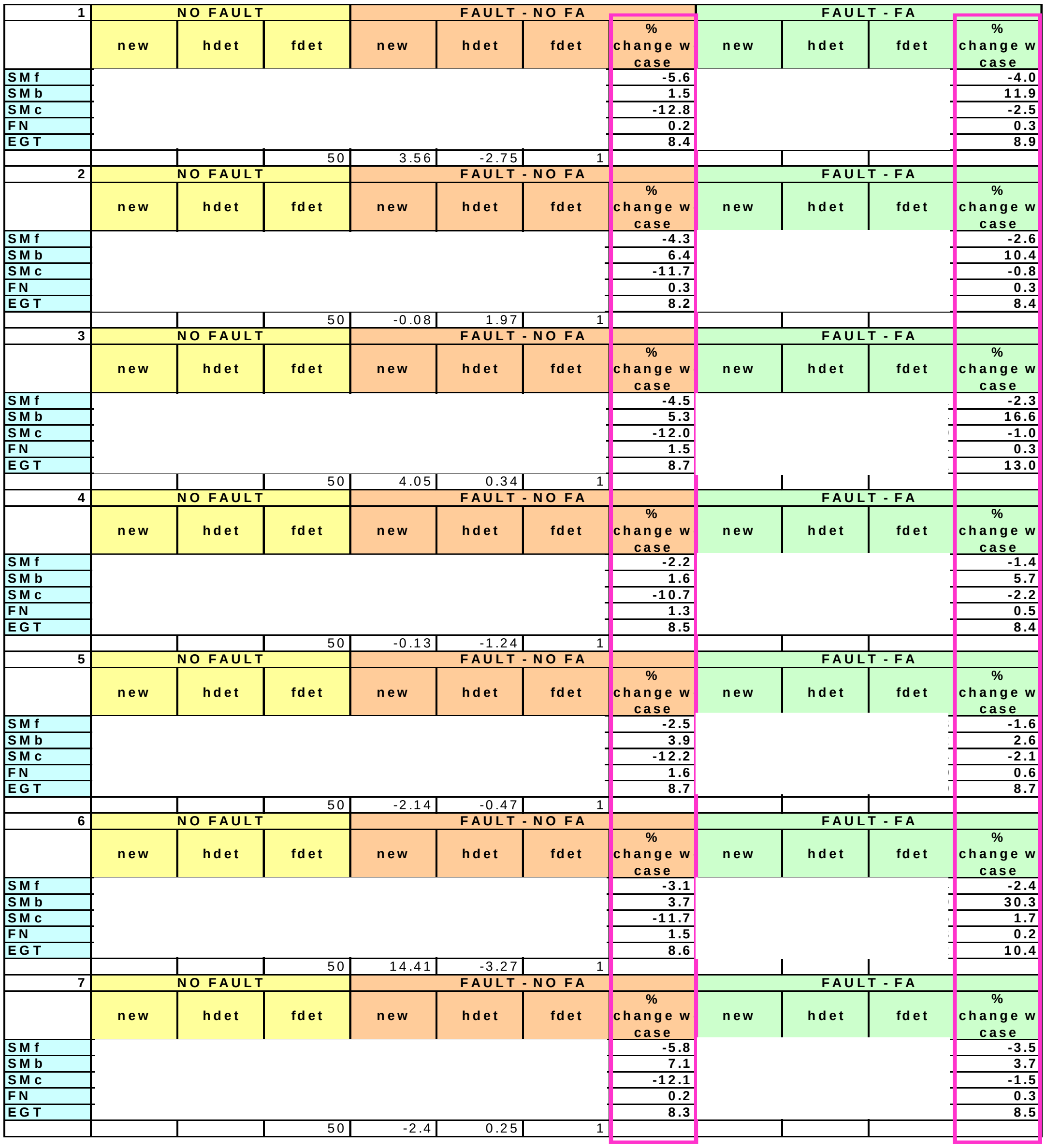
FE pt #	TRA	P0	T12	ftype	fmag	TRA	Bleedadj	VGadj	switch
1	50.00	14.70	14.88	2	3	50	3.56	-2.75	1
2	50.00	6.75	-20.02	2	3	50	-0.08	1.97	1
3	50.00	2.35	-28.56	2	3	50	4.05	0.34	1
4	50.00	14.70	28.20	2	3	50	-0.13	-1.24	1
5	50.00	9.35	10.22	2	3	50	-2.14	-0.47	1
6	50.00	3.46	-23.23	2	3	50	14.41	-3.27	1
7	50.00	8.63	-12.65	2	3	50	-2.4	0.25	1
8	50.00	8.63	-10.64	2	3	50	13.66	-1.5	1
9	50.00	2.35	-45.53	2	3	50	8.2	-2.92	1
10	50.00	2.35	-17.13	2	3	50	23.32	-1.82	1
11	50.00	5.45	8.61	2	3	50	0.33	-1.87	1
12	50.00	10.11	-4.73	2	3	50	-3.28	0.38	1
13	50.00	12.23	13.91	2	3	50	11.51	-3.84	1
14	50.00	8.31	18.38	2	3	50	3.36	-2.87	1
1	75.00	14.70	14.88	2	3	75	-66.88	-0.83	1
2	75.00	6.75	-20.02	2	3	75	12.45	4.82	1
3	75.00	2.35	-28.56	2	3	75	6.22	3.85	1
4	75.00	14.70	28.20	2	3	75	22.68	2.51	1
5	75.00	9.35	10.22	2	3	75	21.05	-0.94	1
6	75.00	3.46	-23.23	2	3	75	18.64	3.2	1
7	75.00	8.63	-12.65	2	3	75	7.35	4.07	2
8	75.00	8.63	-10.64	2	3	75	18.97	1.65	1
9	75.00	2.35	-45.53	2	3	75	7.78	5.67	1
10	75.00	2.35	-17.13	2	3	75	17.87	0.09	1
11	75.00	5.45	8.61	2	3	75	4.17	2.05	1
12	75.00	10.11	-4.73	2	3	75	21.42	-0.73	2
13	75.00	12.23	13.91	2	3	75	19.98	0.73	2
14	75.00	8.31	18.38	2	3	75	21.68	0.99	1

Figure 54.—Large HPT faults, optimal adjustments for design points.

Results for FA for some design points for TRA = 50 are shown in figure 55. We see that: HPT faults affect mainly Compressor SM (~12% loss), there is excellent pre-fault SM recovery in all design cases, the FA increases EGT by ~10%, and pre-fault FN is maintained.

Results for FA for some design points for TRA = 75 are shown in figure 56. We see that: (i) HPT faults affect mainly Booster SM (5-40% loss), (ii) in general there is very good recovery of pre-fault SMs, (iii) pre-fault FN is maintained, (iv) EGT increases, and in general is higher than in the un-faulted engine.

Results for FA for some testing points for TRA = 60 are shown in figure 57. In general, at this power level, HPT faults affect all SMs (~5-10% loss), we see good SM recovery in all 8 cases, the FA increases pre-fault EGT by ~10%, and pre-fault FN is maintained.

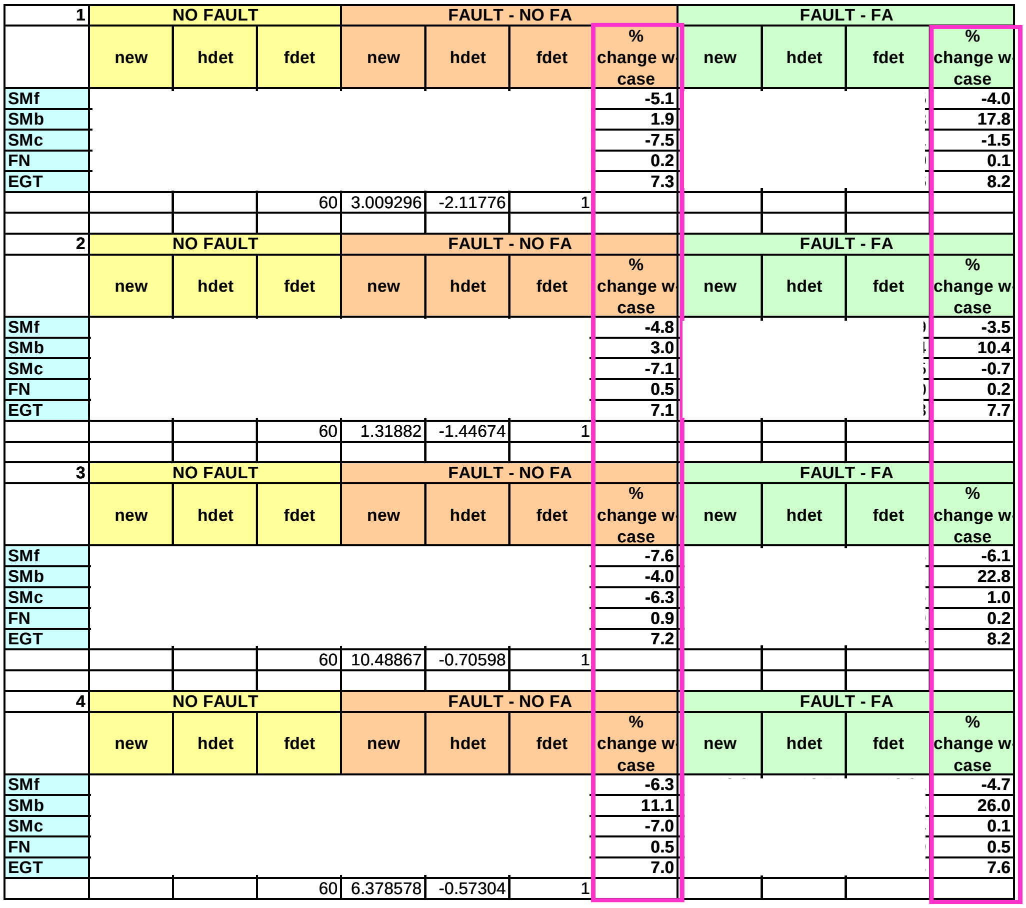


1	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-5.6				-4.0
SMb							1.5				11.9
SMc							-12.8				-2.5
FN							0.2				0.3
EGT							8.4				8.9
			50	3.56	-2.75	1					
2	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-4.3				-2.6
SMb							6.4				10.4
SMc							-11.7				-0.8
FN							0.3				0.3
EGT							8.2				8.4
			50	-0.08	1.97	1					
3	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-4.5				-2.3
SMb							5.3				16.6
SMc							-12.0				-1.0
FN							1.5				0.3
EGT							8.7				13.0
			50	4.05	0.34	1					
4	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-2.2				-1.4
SMb							1.6				5.7
SMc							-10.7				-2.2
FN							1.3				0.5
EGT							8.5				8.4
			50	-0.13	-1.24	1					
5	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-2.5				-1.6
SMb							3.9				2.6
SMc							-12.2				-2.1
FN							1.6				0.6
EGT							8.7				8.7
			50	-2.14	-0.47	1					
6	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-3.1				-2.4
SMb							3.7				30.3
SMc							-11.7				1.7
FN							1.5				0.2
EGT							8.6				10.4
			50	14.41	-3.27	1					
7	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-5.8				-3.5
SMb							7.1				3.7
SMc							-12.1				-1.5
FN							0.2				0.3
EGT							8.3				8.5
			50	-2.4	0.25	1					

Figure 55.—Large HPT faults, FA performance for Design Points 1 to 7 (TRA = 50).

8	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w case
SMf							-6.1				-2.3
SMb							-30.1				-12.1
SMc							2.4				1.4
FN							0.8				0.7
EGT							5.2				4.8
			75	18.97	1.65	1					
9	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w case
SMf							-9.9				-6.1
SMb							-12.9				21.3
SMc							3.2				-5.1
FN							1.4				1.0
EGT							7.3				7.3
			75	7.78	5.67	1					
10	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w case
SMf							-8.3				-7.3
SMb							-17.9				10.3
SMc							-3.7				1.7
FN							1.9				0.6
EGT							7.0				8.6
			75	17.87	0.09	1					
11	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet		new	hdet	fdet	% change w case
SMf							-7.3				-4.9
SMb							-12.0				5.4
SMc							-5.2				-0.7
FN							1.4				0.9
EGT							6.4				6.3
			75	4.17	2.05	1					
12	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w case
SMf							-5.1				-2.5
SMb							-36.0				18.1
SMc							2.4				2.8
FN							0.6				0.7
EGT							4.9				7.2
			75	21.42	-0.73	2					
13	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w case
SMf							-4.7				-4.0
SMb							-32.5				-5.0
SMc							-0.2				1.1
FN							0.9				0.4
EGT							5.4				4.9
			75	19.98	0.73	2					
14	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w case
SMf							-6.8				-7.2
SMb							-14.5				42.2
SMc							-5.4				-4.2
FN							1.2				1.6
EGT							6.1				7.5
			75	21.68	0.99	1					

Figure 56.—Large HPT faults, FA performance for Design Points 8 to 14 (TRA = 75).



1	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-5.1				-4.0
SMb							1.9				17.8
SMc							-7.5				-1.5
FN							0.2				0.1
EGT							7.3				8.2
			60	3.009296	-2.11776	1					
2	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-4.8				-3.5
SMb							3.0				10.4
SMc							-7.1				-0.7
FN							0.5				0.2
EGT							7.1				7.7
			60	1.31882	-1.44674	1					
3	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-7.6				-6.1
SMb							-4.0				22.8
SMc							-6.3				1.0
FN							0.9				0.2
EGT							7.2				8.2
			60	10.48867	-0.70598	1					
4	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-6.3				-4.7
SMb							11.1				26.0
SMc							-7.0				0.1
FN							0.5				0.5
EGT							7.0				7.6
			60	6.378578	-0.57304	1					

Figure 57.—Large HPT faults, test points 1 to 4 (TRA = 60).

7.5.2 Fault Accommodation for Small/Medium HPT Faults

We propose to scale the FA adjustments obtained for large HPT fault cases using scaling factors from 0 to 1, as shown in figure 58. Namely, the scaling is not only a function of the fault magnitude (as in the HPC case) but also a function of the TRA level. Notice that the figure should be interpreted as follows: (i) for TRA >=75, use adjustment scaling as shown with the magenta curve, (ii) for TRA <=50, use adjustment scaling as shown by the green curve, and (iii) for TRA between 50 and 75, interpolate between the green and magenta curves to get the right adjustment scaling.

As examples, figures 59 and 60 show FA results for some of the testing points for the case of *medium* HPT faults. We see that the recovery of the pre-fault metrics (SMs, FN) is very good in all cases. Similar results hold for the remaining testing points.

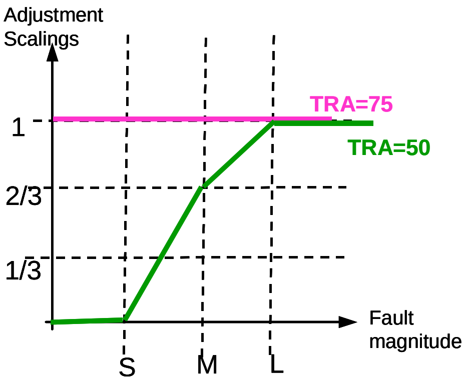
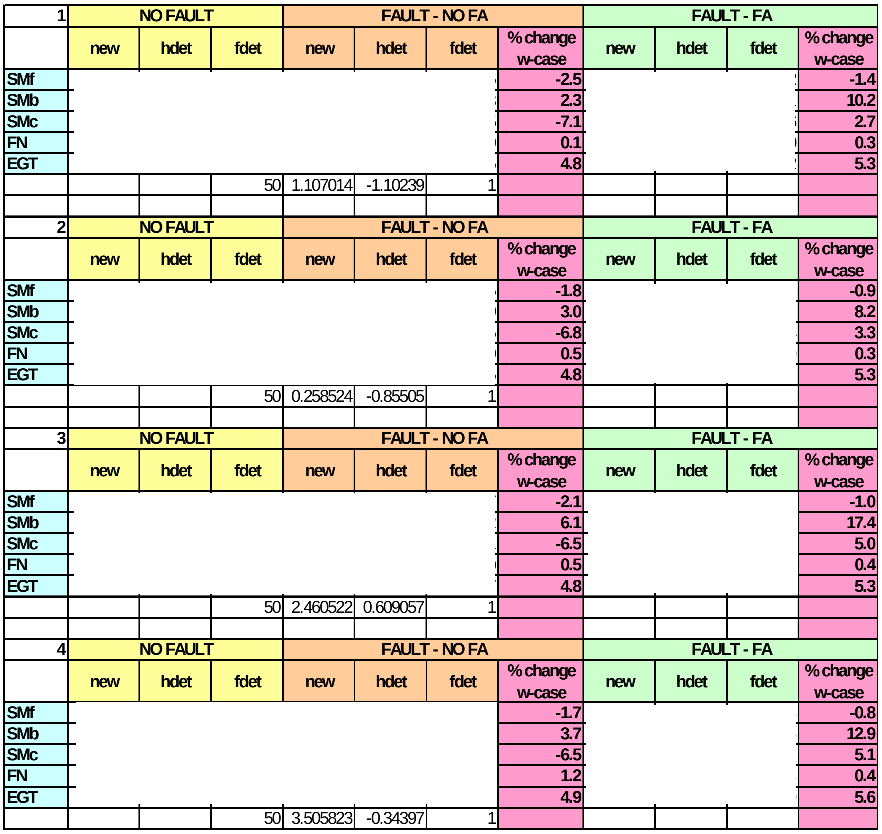
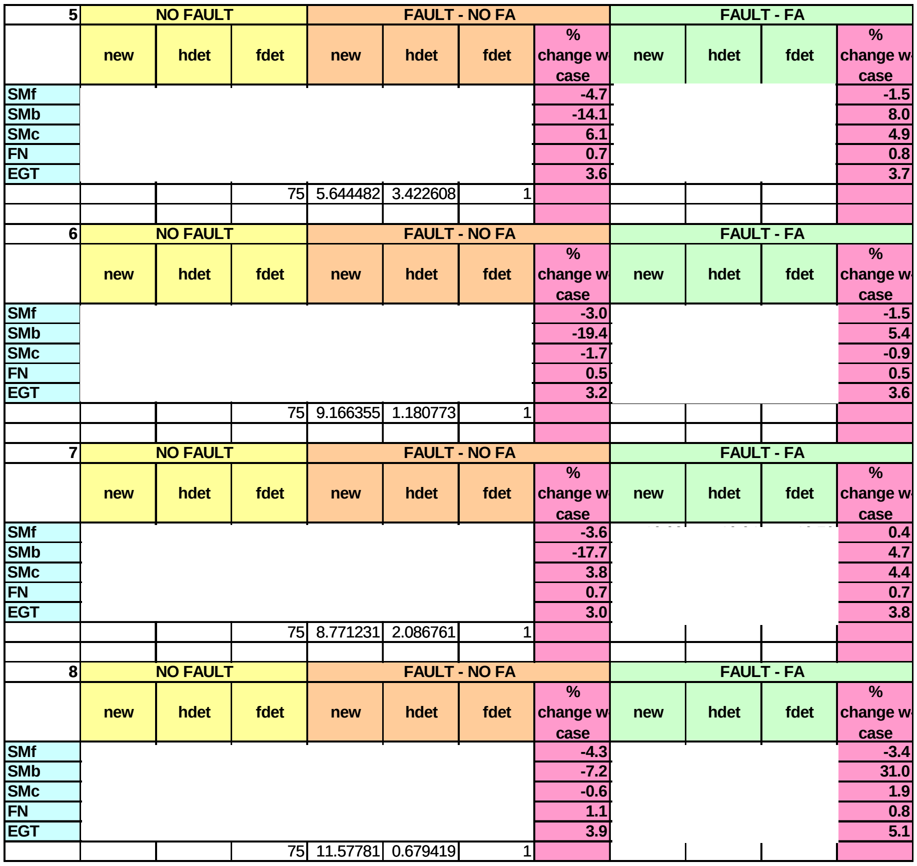


Figure 58.—Proposed scaling of the FA adjustments as a function of fault magnitude and TRA for HPT faults.



1	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-2.5				-1.4
SMb							2.3				10.2
SMc							-7.1				2.7
FN							0.1				0.3
EGT							4.8				5.3
			50	1.107014	-1.10239	1					
2	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-1.8				-0.9
SMb							3.0				8.2
SMc							-6.8				3.3
FN							0.5				0.3
EGT							4.8				5.3
			50	0.258524	-0.85505	1					
3	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-2.1				-1.0
SMb							6.1				17.4
SMc							-6.5				5.0
FN							0.5				0.4
EGT							4.8				5.3
			50	2.460522	0.609057	1					
4	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-1.7				-0.8
SMb							3.7				12.9
SMc							-6.5				5.1
FN							1.2				0.4
EGT							4.9				5.6
			50	3.505823	-0.34397	1					

Figure 59.—FA for Medium HPT faults, TRA = 50, testing points 1 to 4.



5	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-4.7				-1.5
SMb							-14.1				8.0
SMc							6.1				4.9
FN							0.7				0.8
EGT							3.6				3.7
			75	5.644482	3.422608	1					

6	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-3.0				-1.5
SMb							-19.4				5.4
SMc							-1.7				-0.9
FN							0.5				0.5
EGT							3.2				3.6
			75	9.166355	1.180773	1					

7	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-3.6				0.4
SMb							-17.7				4.7
SMc							3.8				4.4
FN							0.7				0.7
EGT							3.0				3.8
			75	8.771231	2.086761	1					

8	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							-4.3				-3.4
SMb							-7.2				31.0
SMc							-0.6				1.9
FN							1.1				0.8
EGT							3.9				5.1
			75	11.57781	0.679419	1					

Figure 60.—FA for Medium HPT faults, TRA = 75, testing points 5 to 8.

Figure 61 shows the impact of a small HPT fault on the performance metrics for TRA = 50 (no FA implemented). In all cases the impact is of 3% or less, and thus there is no need for FA (that is why the scaling factor is 0 in figure 58 for TRA = 50). In contraposition, figure 62 shows the impact of a small HPT fault on the performance metrics for TRA = 75 (no FA). We see that the impact can be substantial, and therefore FA is now desirable. Moreover, further studies had shown that for TRA = 75 we should not downscale the adjustments in order to have good FA.

In summary, a scaling dependent not only on fault magnitude, but also on TRA level; as shown in figure 58 works fine for small/medium HPT faults.

1	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-1.3
SMb							5.0
SMc							-3.3
FN							0.1
EGT							2.3
			50	0.553507	-0.55119	0	
2	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-1.0
SMb							4.0
SMc							-3.2
FN							0.3
EGT							2.4
			50	0.129262	-0.42753	0	
3	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-1.1
SMb							9.0
SMc							-3.0
FN							0.2
EGT							2.4
			50	1.230261	0.304528	0	
4	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-0.9
SMb							4.8
SMc							-2.9
FN							0.6
EGT							2.5
			50	1.752911	-0.17198	0	

Figure 61.—Small HPT faults, TRA = 50, testing points 1 to 4.

1	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-0.7
SMb							-9.7
SMc							0.9
FN							0.1
EGT							0.7
			75	2.029101	-0.02227	0	
2	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-0.9
SMb							-12.9
SMc							0.8
FN							0.1
EGT							1.1
			75	5.108018	0.090255	0	
3	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-1.8
SMb							-5.5
SMc							2.4
FN							0.3
EGT							1.4
			75	4.481961	1.396866	0	
4	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case
SMf							-2.1
SMb							-0.6
SMc							1.3
FN							0.5
EGT							1.9
			75	5.708696	0.402967	1	

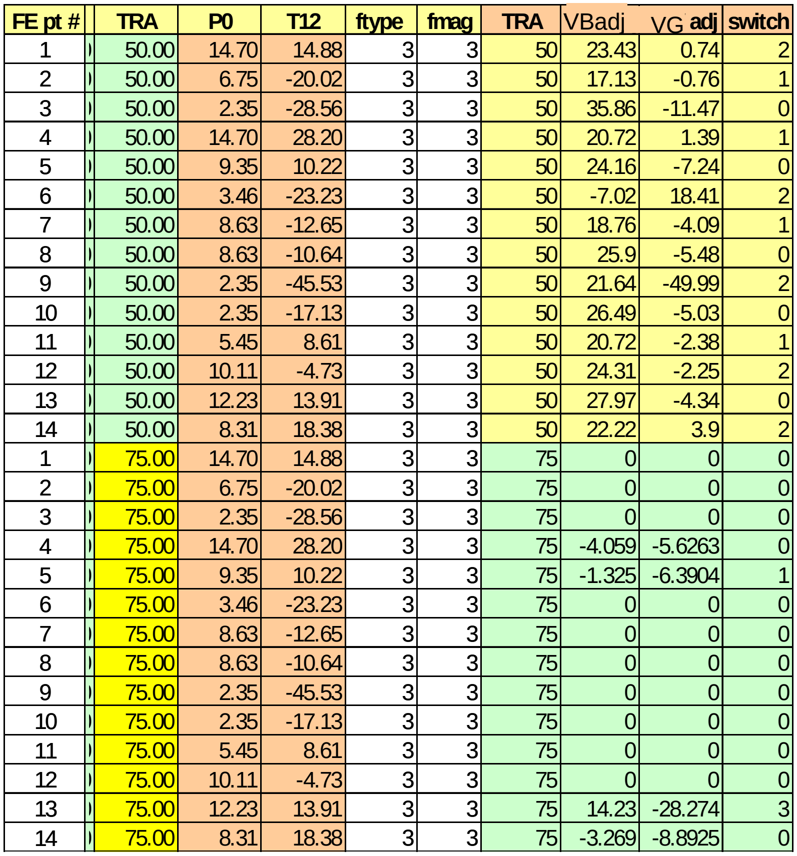
Figure 62.—Small HPT faults, TRA = 75, testing points 1 to 4.

7.6 Fault Accommodation for Variable Geometry (VG) Guide Vane Faults

7.6.1 Fault Accommodation for Large VG Faults

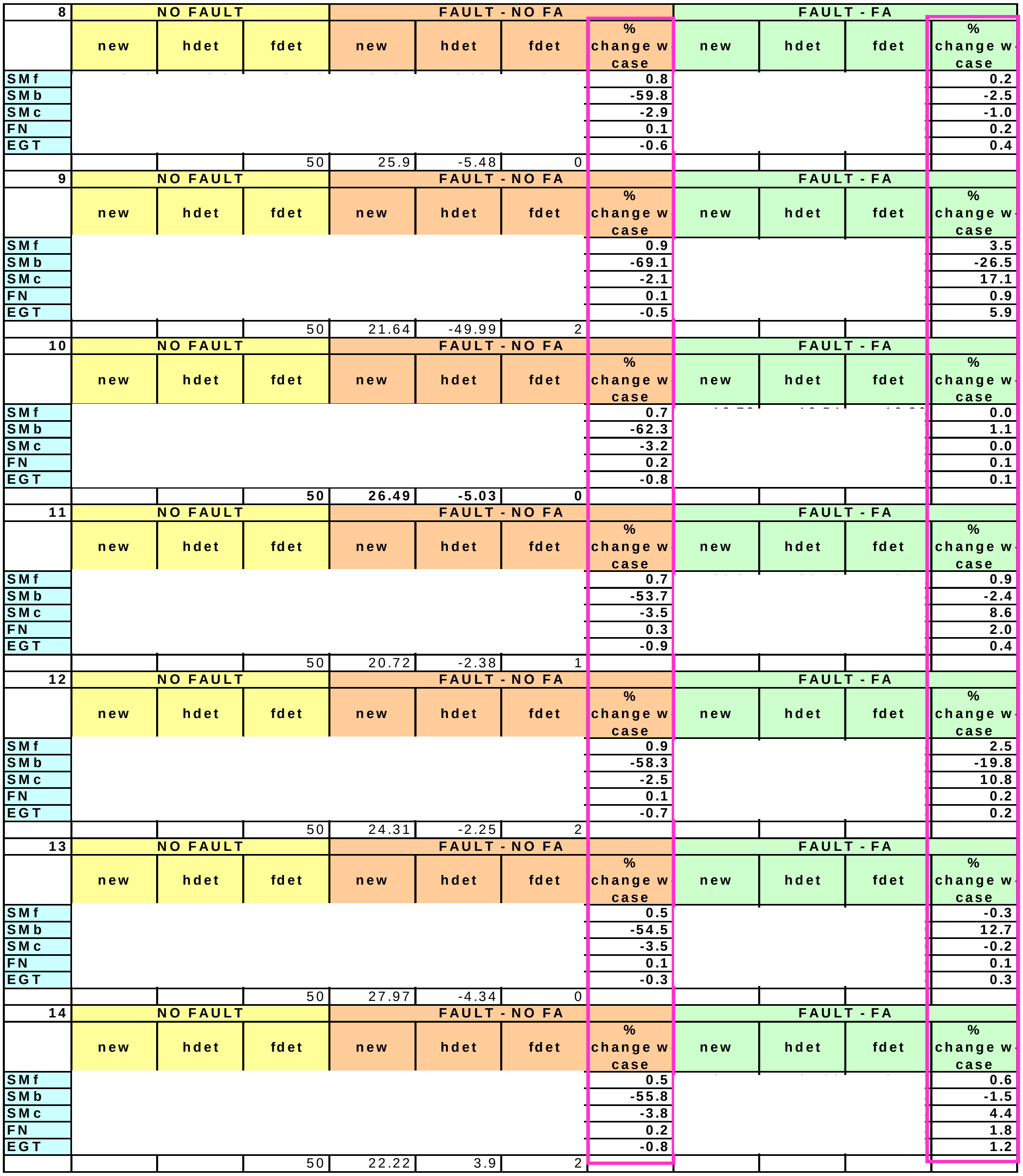
In this section, we focus on FA for variable geometry (VG) guide vane faults of large magnitude. Our initial thought for VG faults was that since this fault is really a bias in the VG actuator, it should perhaps be corrected by just modifying the VG adjustment, ideally enough to compensate for the bias. However, this is not the case, since the existing FADEC responds to the VG bias causing changes in all control actuators, thereby necessitating adjustments on all actuators. So we addressed the VG fault accommodation by optimizing all four adjustments mentioned before. By solving the optimization problems explained before, we obtained the optimal adjustments for the design points. The optimal adjustments for TRA = 50 and for TRA = 75 are shown in figure 63. FA adjustments for other TRAs are obtained through interpolations. We noticed that optimal TRA adjustments were very close to their nominal values of 50 or 75; therefore, for simplicity, we are not going to be actually using TRA adjustments for VG FA. Also notice, for many points for TRA = 75, all the adjustments are 0; i.e., for many design points we are not doing any accommodation at all. We have found that for many points with high TRAs, the impact of VG faults is not large, and on the other hand FA for SM recovery performs very poorly in terms of FN recovery. Therefore, we seem to be better off skipping the FA for those high TRA points.

Results for FA for some design points for TRA = 50 are shown in figure 64. We see that at this power level, VG faults affect Booster SM considerably (~60% loss), we achieve excellent SM recovery in most cases, and the FA increases EGT by less than 10% and maintains FN.



FE pt #		TRA	P0	T12	ftype	fmag	TRA	VBadj	VG adj	switch
1	)	50.00	14.70	14.88	3	3	50	23.43	0.74	2
2	)	50.00	6.75	-20.02	3	3	50	17.13	-0.76	1
3	)	50.00	2.35	-28.56	3	3	50	35.86	-11.47	0
4	)	50.00	14.70	28.20	3	3	50	20.72	1.39	1
5	)	50.00	9.35	10.22	3	3	50	24.16	-7.24	0
6	)	50.00	3.46	-23.23	3	3	50	-7.02	18.41	2
7	)	50.00	8.63	-12.65	3	3	50	18.76	-4.09	1
8	)	50.00	8.63	-10.64	3	3	50	25.9	-5.48	0
9	)	50.00	2.35	-45.53	3	3	50	21.64	-49.99	2
10	)	50.00	2.35	-17.13	3	3	50	26.49	-5.03	0
11	)	50.00	5.45	8.61	3	3	50	20.72	-2.38	1
12	)	50.00	10.11	-4.73	3	3	50	24.31	-2.25	2
13	)	50.00	12.23	13.91	3	3	50	27.97	-4.34	0
14	)	50.00	8.31	18.38	3	3	50	22.22	3.9	2
1	)	75.00	14.70	14.88	3	3	75	0	0	0
2	)	75.00	6.75	-20.02	3	3	75	0	0	0
3	)	75.00	2.35	-28.56	3	3	75	0	0	0
4	)	75.00	14.70	28.20	3	3	75	-4.059	-5.6263	0
5	)	75.00	9.35	10.22	3	3	75	-1.325	-6.3904	1
6	)	75.00	3.46	-23.23	3	3	75	0	0	0
7	)	75.00	8.63	-12.65	3	3	75	0	0	0
8	)	75.00	8.63	-10.64	3	3	75	0	0	0
9	)	75.00	2.35	-45.53	3	3	75	0	0	0
10	)	75.00	2.35	-17.13	3	3	75	0	0	0
11	)	75.00	5.45	8.61	3	3	75	0	0	0
12	)	75.00	10.11	-4.73	3	3	75	0	0	0
13	)	75.00	12.23	13.91	3	3	75	14.23	-28.274	3
14	)	75.00	8.31	18.38	3	3	75	-3.269	-8.8925	0

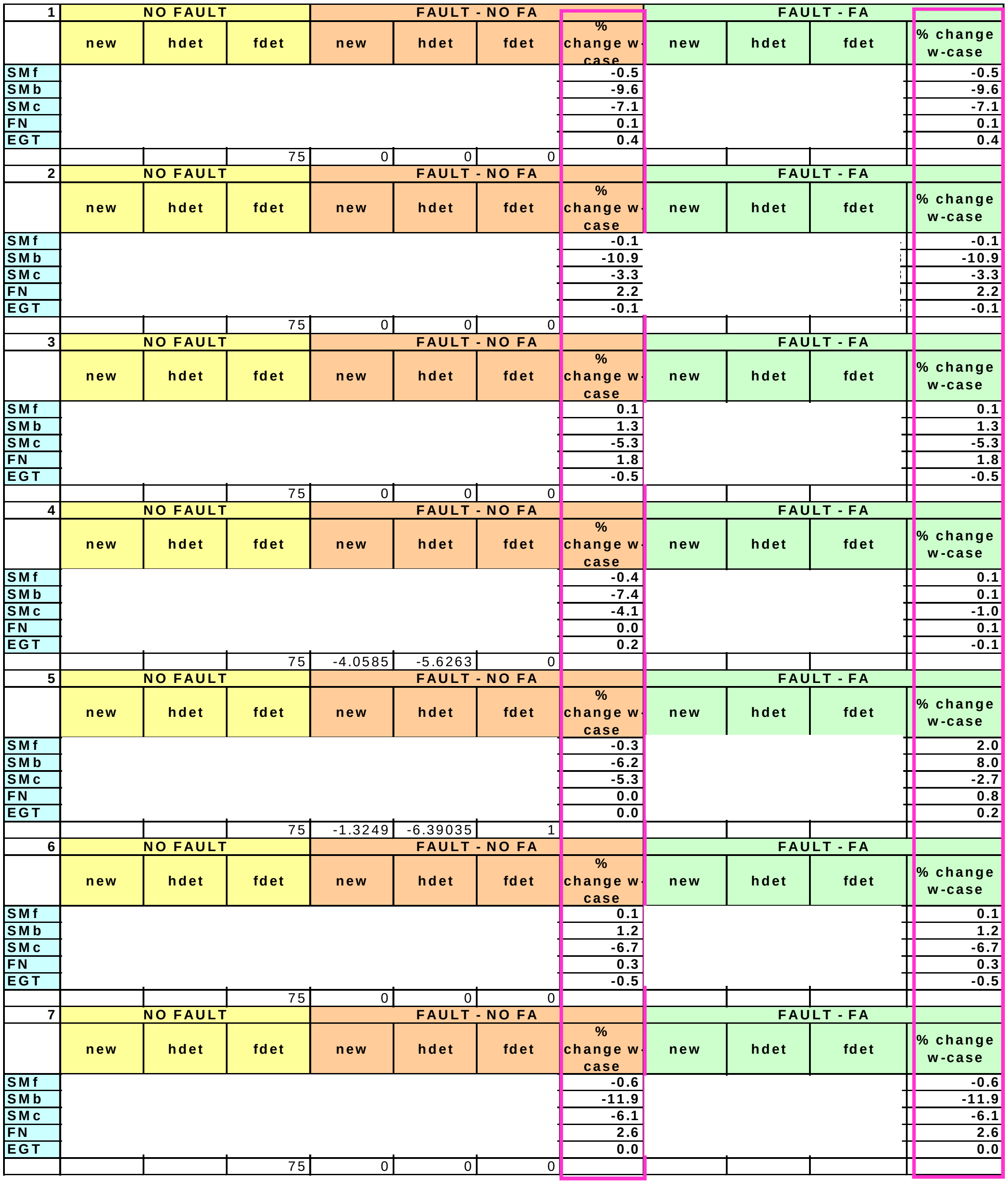
Figure 63.—Large VG fault, optimal adjustments for design points.



8	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.8				0.2
SMb							-59.8				-2.5
SMc							-2.9				-1.0
FN							0.1				0.2
EGT							-0.6				0.4
			50	25.9	-5.48	0					
9	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.9				3.5
SMb							-69.1				-26.5
SMc							-2.1				17.1
FN							0.1				0.9
EGT							-0.5				5.9
			50	21.64	-49.99	2					
10	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.7				0.0
SMb							-62.3				1.1
SMc							-3.2				0.0
FN							0.2				0.1
EGT							-0.8				0.1
			50	26.49	-5.03	0					
11	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.7				0.9
SMb							-53.7				-2.4
SMc							-3.5				8.6
FN							0.3				2.0
EGT							-0.9				0.4
			50	20.72	-2.38	1					
12	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.9				2.5
SMb							-58.3				-19.8
SMc							-2.5				10.8
FN							0.1				0.2
EGT							-0.7				0.2
			50	24.31	-2.25	2					
13	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.5				-0.3
SMb							-54.5				12.7
SMc							-3.5				-0.2
FN							0.1				0.1
EGT							-0.3				0.3
			50	27.97	-4.34	0					
14	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.5				0.6
SMb							-55.8				-1.5
SMc							-3.8				4.4
FN							0.2				1.8
EGT							-0.8				1.2
			50	22.22	3.9	2					

Figure 64.—Large VG fault, design points 8 to 14 (TRA = 50).

Results for FA for some design points for TRA = 75 are shown in figure 65. We observe the following: VG faults affect Booster SM and/or Compressor SM, in some cases SM loss is not important (e.g., cases # 3,6,9,10,11), SMs loss of 10% or more maybe important (e.g., cases # 2,7,8). However, in may cases, the optimization is not varying the adjustments from baseline

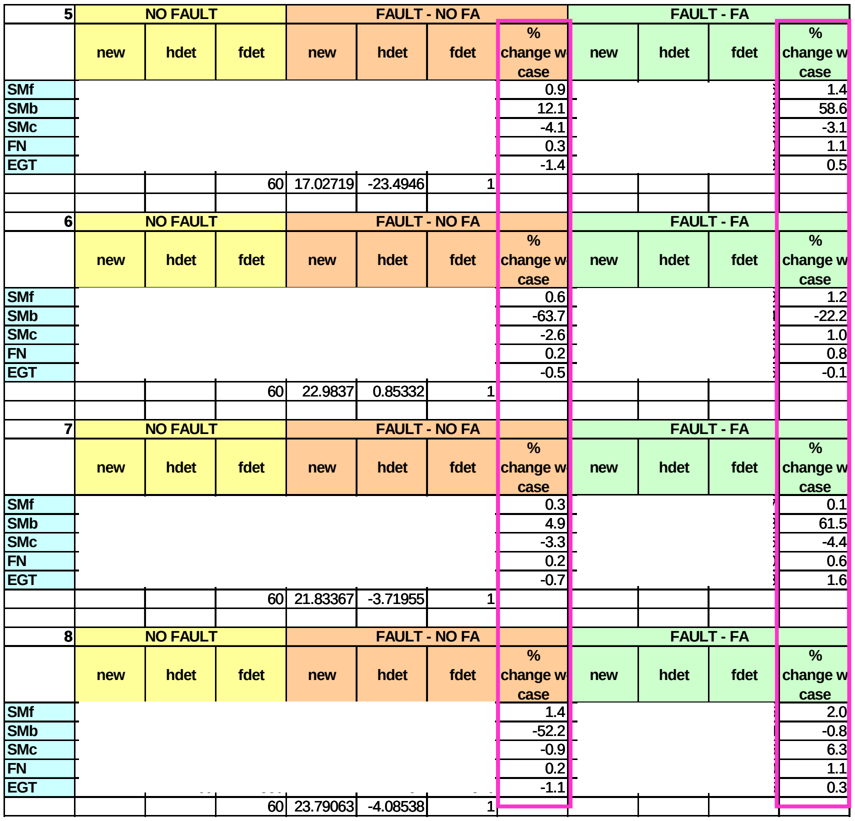


1	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-0.5				-0.5
SMb							-9.6				-9.6
SMc							-7.1				-7.1
FN							0.1				0.1
EGT							0.4				0.4
			75	0	0	0					
2	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-0.1				-0.1
SMb							-10.9				-10.9
SMc							-3.3				-3.3
FN							2.2				2.2
EGT							-0.1				-0.1
			75	0	0	0					
3	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.1				0.1
SMb							1.3				1.3
SMc							-5.3				-5.3
FN							1.8				1.8
EGT							-0.5				-0.5
			75	0	0	0					
4	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-0.4				0.1
SMb							-7.4				0.1
SMc							-4.1				-1.0
FN							0.0				0.1
EGT							0.2				-0.1
			75	-4.0585	-5.6263	0					
5	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-0.3				2.0
SMb							-6.2				8.0
SMc							-5.3				-2.7
FN							0.0				0.8
EGT							0.0				0.2
			75	-1.3249	-6.39035	1					
6	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.1				0.1
SMb							1.2				1.2
SMc							-6.7				-6.7
FN							0.3				0.3
EGT							-0.5				-0.5
			75	0	0	0					
7	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							-0.6				-0.6
SMb							-11.9				-11.9
SMc							-6.1				-6.1
FN							2.6				2.6
EGT							0.0				0.0
			75	0	0	0					

Figure 65.—Large VG fault, design points 1 to 7 (TRA = 75).

(nominal) values indicating possibly the presence of some constraints, e.g., VG saturated at fully open position, that inhibit any accommodation at this high power setting.

Results for FA for some testing points for intermediate power at TRA = 60 are shown in figure 66. We notice good recovery of pre-fault parameters.



5	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.9				1.4
SMb							12.1				58.6
SMc							-4.1				-3.1
FN							0.3				1.1
EGT							-1.4				0.5
			60	17.02719	-23.4946	1					
6	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.6				1.2
SMb							-63.7				-22.2
SMc							-2.6				1.0
FN							0.2				0.8
EGT							-0.5				-0.1
			60	22.9837	0.85332	1					
7	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							0.3				0.1
SMb							4.9				61.5
SMc							-3.3				-4.4
FN							0.2				0.6
EGT							-0.7				1.6
			60	21.83367	-3.71955	1					
8	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w case	new	hdet	fdet	% change w case
SMf							1.4				2.0
SMb							-52.2				-0.8
SMc							-0.9				6.3
FN							0.2				1.1
EGT							-1.1				0.3
			60	23.79063	-4.08538	1					

Figure 66.—Large VG fault, test points 1 to 4 (TRA = 60).

7.6.2 Fault Accommodation for Small/Medium VG Faults

7.6.3 Small/Medium VG faults

We propose to scale the FA adjustments obtained for large VG fault cases using scaling factors from 0 to 1 as shown in figure 67. Namely, the scaling is not only a function of the fault magnitude (as in the HPC case) but also a function of the TRA level. Notice that the figure should be interpreted as follows: (i) for TRA >=75, use adjustment scaling as shown with the magenta curve, (ii) for TRA <=50, use adjustment scaling as shown by the green curve, and (iii) for TRA between 50 and 75, interpolate between the green and magenta curves to get the right adjustment scaling.

Figure 68 shows the impact of a medium VG fault for TRA = 75 on SMs, FN and EGT, for some of the testing points. As we can see, there is no need for FA. The same holds for the remaining testing points.

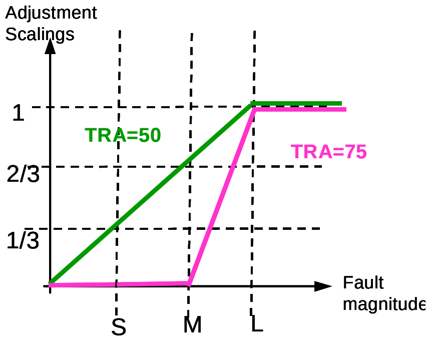
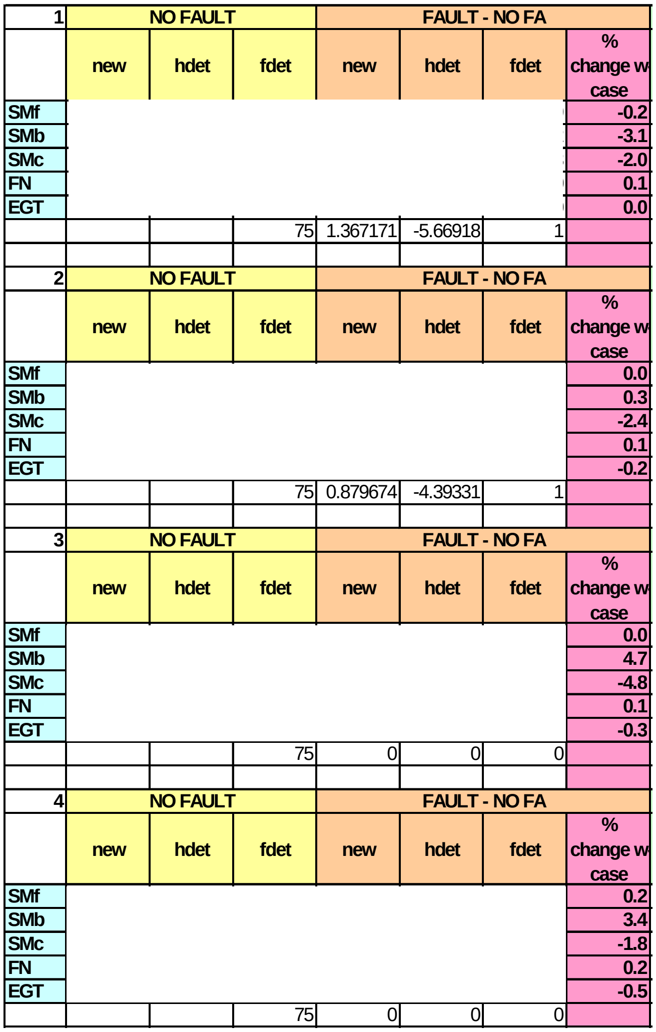


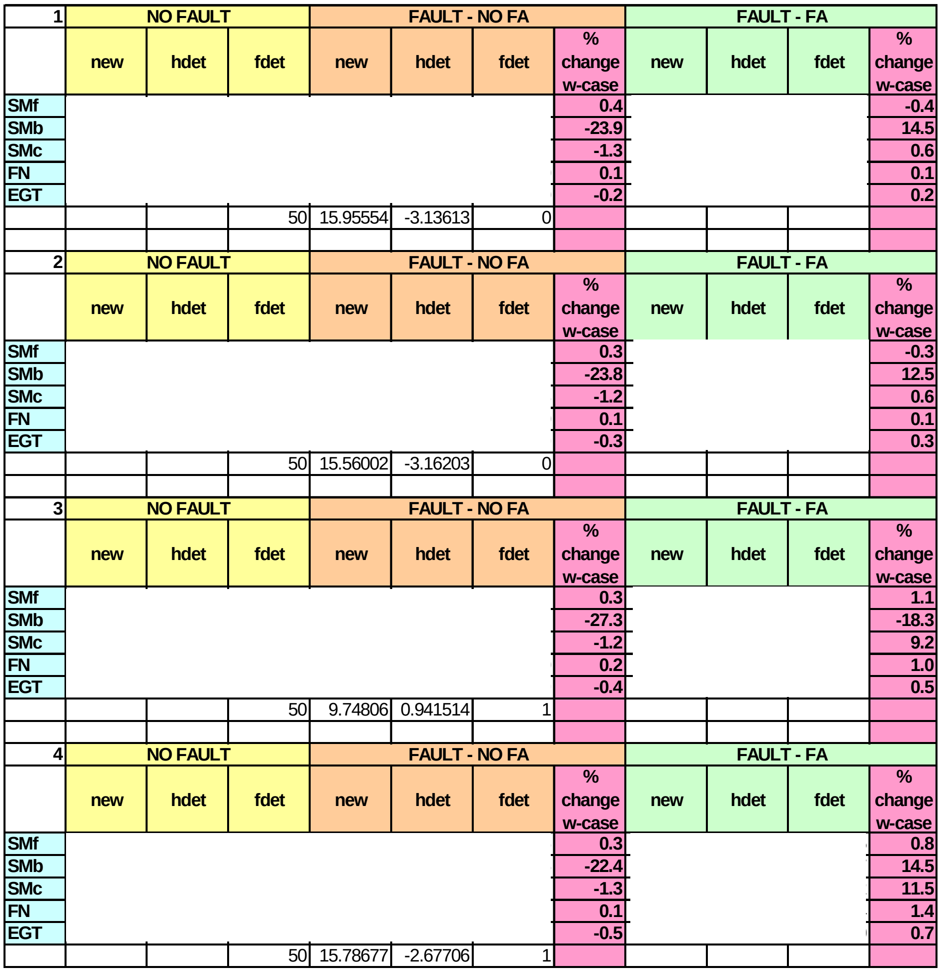
Figure 67.—Proposed scaling of the FA adjustments as a function of fault magnitude and TRA for VG faults.



1	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w case
SMf							-0.2
SMb							-3.1
SMc							-2.0
FN							0.1
EGT							0.0
			75	1.367171	-5.66918	1	
2	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w case
SMf							0.0
SMb							0.3
SMc							-2.4
FN							0.1
EGT							-0.2
			75	0.879674	-4.39331	1	
3	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w case
SMf							0.0
SMb							4.7
SMc							-4.8
FN							0.1
EGT							-0.3
			75	0	0	0	
4	NO FAULT			FAULT - NO FA			
	new	hdet	fdet	new	hdet	fdet	% change w case
SMf							0.2
SMb							3.4
SMc							-1.8
FN							0.2
EGT							-0.5
			75	0	0	0	

Figure 68.—Medium VG faults, TRA = 75, testing points 1 to 4. No need for FA.

Figure 69 shows that the impact of a medium VG fault for TRA = 50 is important. It also shows that our FA strategy works well. Figure 70 shows that our FA also works for small VG faults at TRA = 50.



1	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.4				-0.4
SMb							-23.9				14.5
SMc							-1.3				0.6
FN							0.1				0.1
EGT							-0.2				0.2
			50	15.95554	-3.13613	0					
2	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.3				-0.3
SMb							-23.8				12.5
SMc							-1.2				0.6
FN							0.1				0.1
EGT							-0.3				0.3
			50	15.56002	-3.16203	0					
3	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.3				1.1
SMb							-27.3				-18.3
SMc							-1.2				9.2
FN							0.2				1.0
EGT							-0.4				0.5
			50	9.74806	0.941514	1					
4	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.3				0.8
SMb							-22.4				14.5
SMc							-1.3				11.5
FN							0.1				1.4
EGT							-0.5				0.7
			50	15.78677	-2.67706	1					

Figure 69.—FA for Medium VG faults, TRA = 50, testing points 1 to 4.

1	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.1				-0.5
SMb							-5.7				17.9
SMc							-0.4				0.9
FN							0.0				0.1
EGT							-0.1				0.2
			50	7.97777	-1.56807	0					
2	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.1				-0.3
SMb							-5.7				15.7
SMc							-0.3				0.9
FN							0.1				0.1
EGT							-0.1				0.4
			50	7.780009	-1.58101	0					
3	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.1				-0.1
SMb							-8.2				6.0
SMc							-0.3				-0.5
FN							0.1				0.1
EGT							-0.1				-0.1
			50	4.87403	0.470757	0					
4	NO FAULT			FAULT - NO FA				FAULT - FA			
	new	hdet	fdet	new	hdet	fdet	% change w-case	new	hdet	fdet	% change w-case
SMf							0.1				-0.3
SMb							-7.0				14.1
SMc							-0.5				0.6
FN							0.1				0.2
EGT							-0.2				0.5
			50	7.893384	-1.33853	0					

Figure 70.—FA for Small VG fault, TRA = 50, testing points 1 to 4.

7.7 Fault Accommodation for PS3 Pressure Sensor Faults

In this section, we focus on the accommodation of a compressor exit pressure (PS3) sensor fault. Unlike the previous faults, this pressure sensor fault does not impact the engine performance or operability at steady-state conditions since the FADEC logic uses the pressure sensor only during accel/decel transients. Motivated by this, the accommodation strategy for this sensor fault is to use another EKF designed to estimate the correct (un-faulted) value of this pressure using the remaining eight sensors that are available and replace the faulty pressure sensor feedback with this estimated value in the FADEC logic.

For the pressure estimation, where the EKF serves as a “virtual sensor”, it is desirable to design the EKF for optimal pressure estimation that is robust to engine-to-engine variation and deterioration. Note that the embedded CLM used in the EKF is fixed for a half-deteriorated engine (mean fleet engine). Figure 71 shows estimation errors for sensor outputs and for pressure at steady-state conditions at SLS for new engine in subplots (a) and for fully deteriorated engine in subplots (b). The EKF estimation error for pressure is very small for new and deteriorated engines, significantly smaller than that obtained with the open-loop model. Notice that the base-line model we use for our EKF corresponds to a half-deteriorated engine to minimize the mean estimation bias at a fleet average (for this reason, the estimation errors for the open-loop model change signs when we go from a new to a fully deteriorated engine).

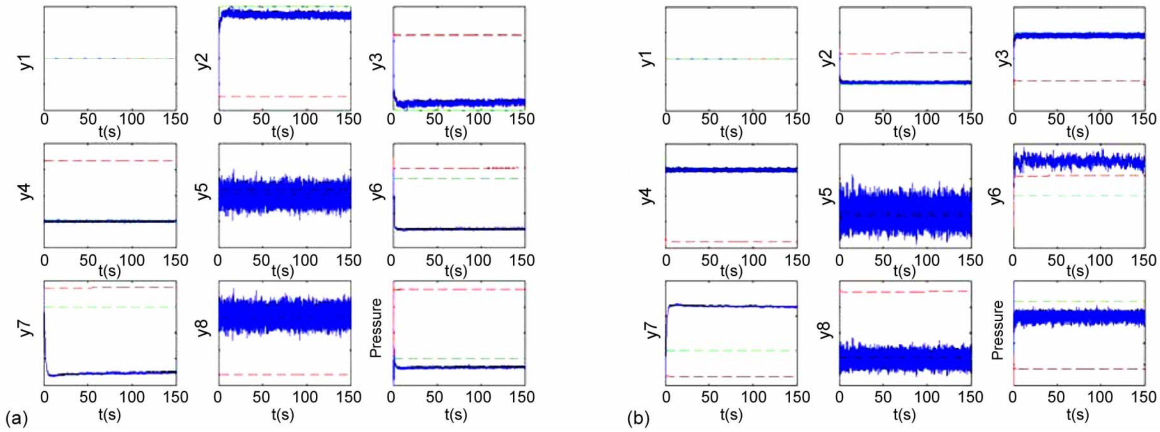


Figure 71.—Time responses for estimation errors in the 8 sensor outputs plus pressure, for new engine (a) and for fully deteriorated engine (b). Steady-state conditions: Sea-level static. Dashed red line: Open loop model (EKF gain = 0). Blue line: Optimally designed EKF.

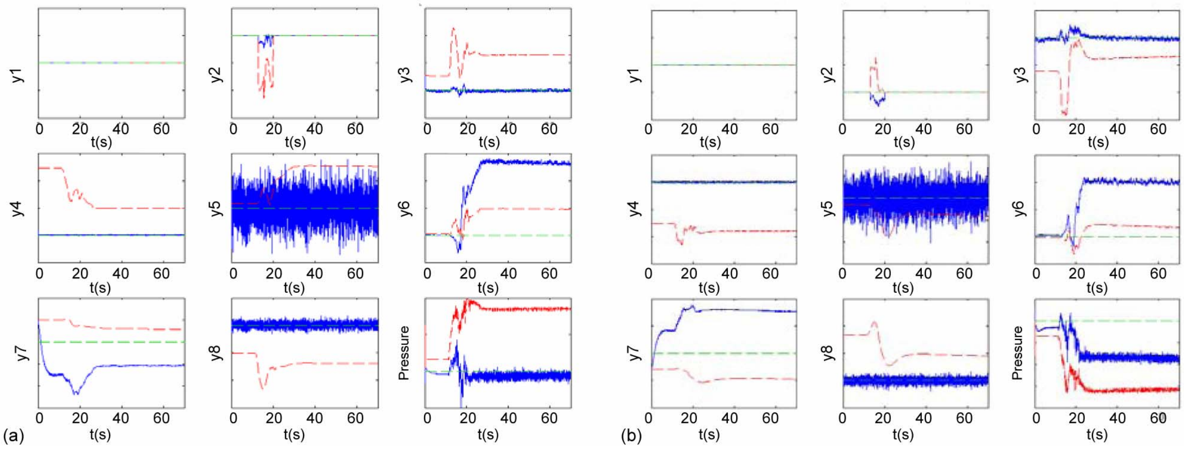


Figure 72.—Time responses for estimation errors in the 8 sensor outputs plus Pressure, for new engine (a) and for fully deteriorated engine (b). Transient Operation: Idle to takeoff (SLS) Dashed red line: Open loop model (EKF gain = 0). Blue line: Optimally designed EKF.

Figure 72 shows estimation errors versus time for an idle-to-takeoff transient, for new and deteriorated engines. Again, even during transients, the estimation for pressure using the optimally tuned EKF is pretty good and smaller than the open-loop estimation error.

The good performance of pressure estimation through the specially designed EKF using the remaining sensors during steady-state and transient conditions indicates the feasibility of accommodating PS3 pressure sensor faults through the proposed substitution of the faulty sensor feedback with the estimated value in the FADEC logic. This was verified in our FADEC simulator (FSIM) environment wherein the fault detection and accommodation algorithms were integrated and implemented in FADEC software and tested against transient simulations of the engine. These results will be discussed in detail in the next section, where we discuss the FSIM implementation and evaluation of fault detection and accommodation algorithms and show the results for PS3 pressure sensor fault accommodation during a transient in section 8.2.4.

8. Fault Detection and Accommodation Implementation in FSIM

The fault detection and accommodation algorithms mentioned in the previous sections were developed and tested in Matlab. For implementation of these algorithms in the FADEC, we had to convert the algorithms from Matlab to Beacon, which generates appropriate C-code through auto-code generation compatible with FADEC software specifications. The generated C code is compliant with FADEC software specifications and can be integrated with the existing FADEC code and tested in closed-loop with a CLM in a FADEC simulation (FSIM) environment. On the other hand, some functionalities (e.g., matrix operations) are not supported in Beacon, and they had to be hand-coded in C directly.

8.1 Implementation of Fault Detection (FD) and Fault Accommodation (FA) Algorithms in Beacon and C

A mix of Beacon 7 diagrams and hand-coded C was used to implement the fault detection and fusion algorithms in FSIM. The original Matlab implementation was broken into four distinct segments as illustrated in Figure 73. Each segment was created and tested in parallel to expedite the development process. Finally the individual segments and the overall detection and fusion algorithm were validated for a few test runs against the results obtained from the original Matlab implementation.

Figure 73 illustrates the file structure and calling flow within FSIM. The implementation is broken into:

1) Prefilter (filter raw EKF residuals)
2) Neural network (use neural network to find fault type, probability, and magnitude)
3) Multiple hypothesis testing (use hypothesis testing to find fault type, probability, and magnitude)
4) Fusion (fuse results from neural network and hypothesis testing to find fault type and magnitude)

Initially, it was thought that Beacon 7 was the best way to implement the FD (Fault Detection) algorithms mainly due to its widespread acceptance at GE Aircraft Engines. For this reason, all but four modules have been constructed using Beacon 7. The four routines to calculate fault magnitude in the neural network algorithm - HPCFLTMAG, HPTFLTMAG, VGFLTMAG, and PFLTMAG - were hand-coded in C primarily because Beacon 7 currently only supports single precision calculations, while double precision was necessary to get the desired accuracy of the results especially for the “tanh” function.

Unlike the fault detection algorithms, the EKF algorithm needs extensive matrix manipulation and matrix algebra routines, which are not currently available in Beacon 7. In light of this limitation of Beacon 7, we took the approach of directly coding the needed matrix routines, the CLM linearization and the EKF algorithm in C adhering to the necessary programming conventions and constraints for FADEC software.

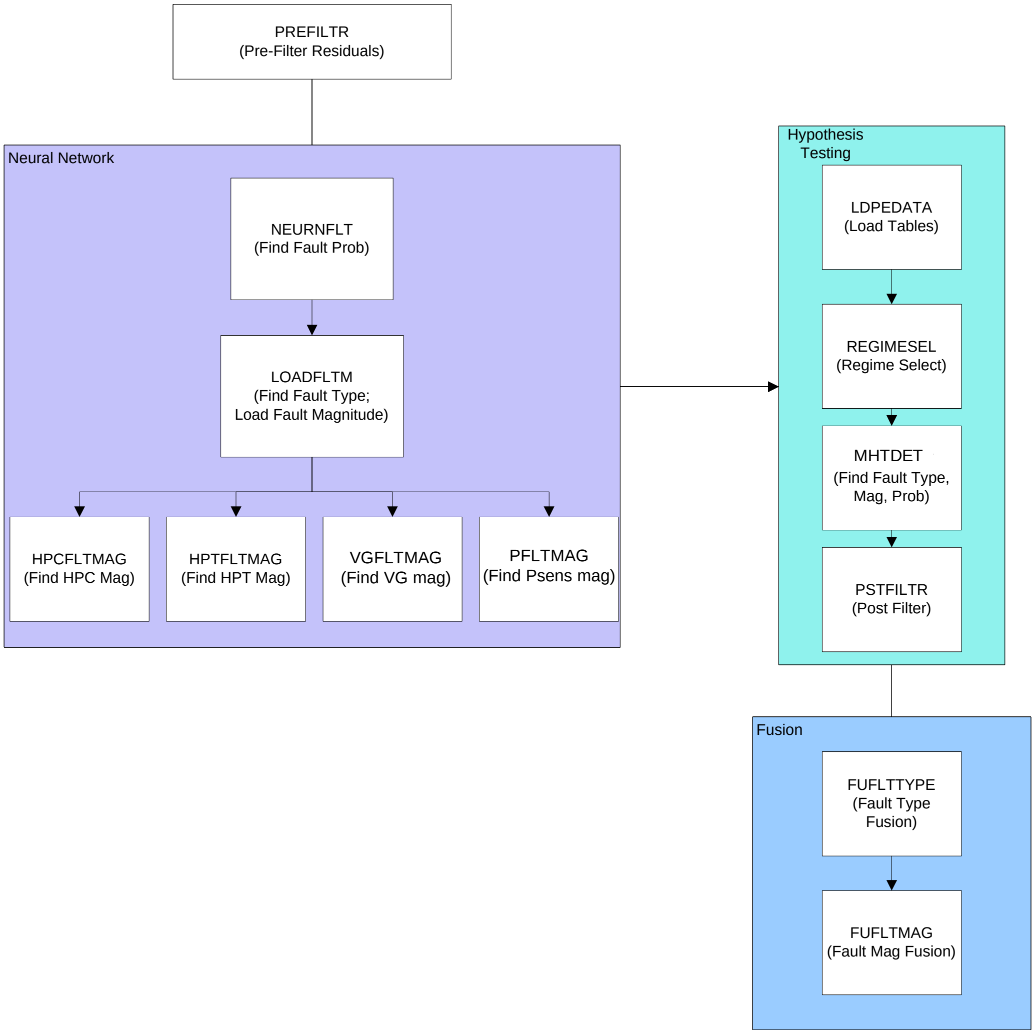


Figure 73.—Fault Detection File Flow in FSIM.

Figure 74 shows the various components that had to be coded in C for the EKF implementation. The C-code implementation was validated for test cases of no-fault/fault runs against Matlab results.

In addition to lack of matrix functions, some basic math functions were also not available in Beacon, e.g., tanh, exponential, which were coded in C using series expansion or look-up tables (the latter was the final implementation to maintain computational speed).

Finally, a finite-state-machine logic was implemented in C as the main subroutine for the coordinated execution of the EKF, fault detection and fault accommodation components in the FADEC software. Figure 75 shows the overall structure of the finite-state-machine logic implemented in C. In particular, since the FSIM and dry rig simulation runs have to be initiated from the engine startup (from no speed to flight idle), the state-machine initiates in a transient state, wherein the EKF and fault detection algorithms are disabled. Thereafter, the engine state is monitored using the embedded CLM in open loop (OL) until the dynamic states have reached a steady state in order to transition to the next state, i.e. “Steady State”. In this state, where the engine is operating at steady-state, the open-loop CLM is used to calculate the biases between

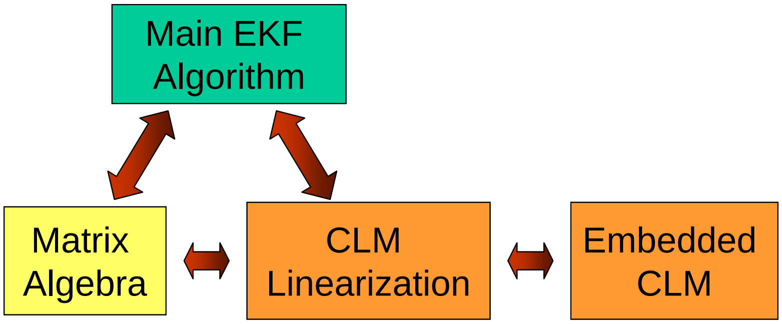


Figure 74.—Components of overall EKF implementation in C.

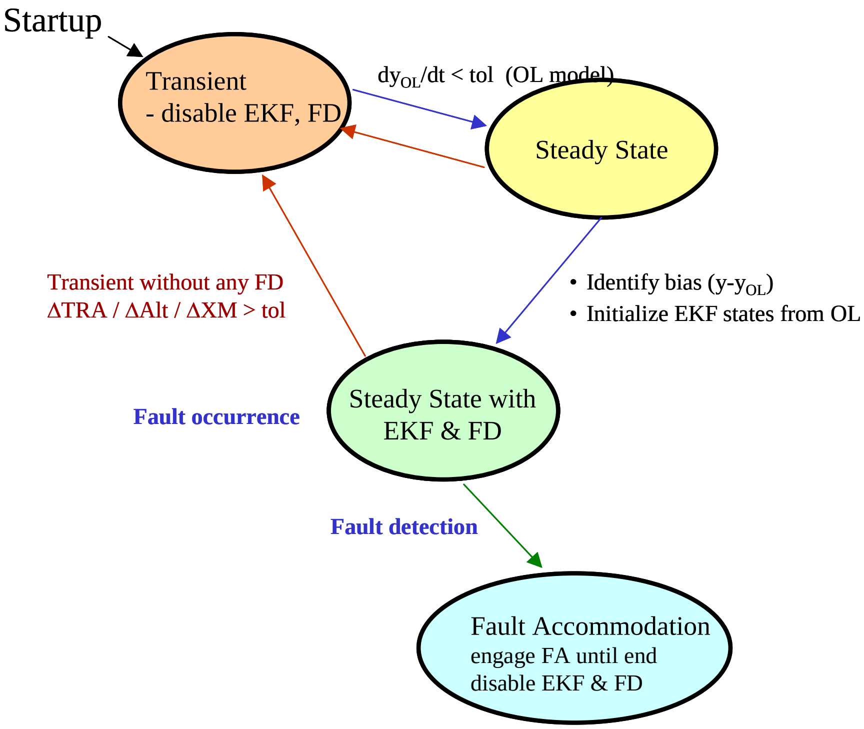


Figure 75.—Finite-state-machine logic for distinguishing steady-state and transient engine operation and enabling fault detection and accommodation algorithms.

the sensed engine outputs (true engine simulated with random engine-to-engine variation and deterioration level) and the corresponding output values predicted from the embedded CLM corresponding to a half-deteriorated engine. This bias accounts for the mismatch arising from the difference between the embedded model and the true engine. Also, the states of the OL CLM are used to initialize the states of the EKF. Once the sensor biases are calculated and the EKF states are initialized, the state-machine logic transitions to the next state, i.e., “Steady state with EKF and FD”, where the EKF and the fault detection algorithms are enabled for the detection of any fault. If a fault is detected, then the state-machine logic transitions to the final state: “Fault accommodation”. In this state, the EKF and fault detection algorithms are disabled since the fault is detected and latched and the corresponding fault accommodation logic is engaged for the rest of the flight. Also, if in the “Steady-state” and “Steady state with EKF and

FD" states, the TRA or flight conditions are changed abruptly leading to a pilot-induced transient behavior, then the state-machine logic will transition to the "transient" state and temporarily disable the EKF and fault detection algorithms. Such pilot-induced transients are distinguished from the ones caused by the occurrence of a fault by monitoring any rapid changes in TRA and/or flight envelope conditions.

8.2 FSIM Implementation and Modifications for Real-Time Implementation in FADEC

The implementation of the EKF, fault detection and fault accommodation algorithms in Beacon and C enabled generating FADEC compatible C-code and testing these algorithms in closed-loop simulations in a FADEC simulation environment (FSIM) in a closed-loop configuration depicted by figure 1. The CLM is used to simulate the true engine with desired level of engine-to-engine variation, engine deterioration and specific fault type and magnitude. The FSIM environment tests the actual FADEC software as it would be implemented in the FADEC hardware and is a critical step towards testing on FADEC hardware in a dry rig setup. The generated FADEC software can be thoroughly tested/debugged in this environment in preparation for dry rig implementations. However, the FSIM environment is not a real-time environment and runs on a HPUX platform, and does not restrict the use of computationally intensive algorithms. In particular, the implementation of the EKF algorithms used for the fault detection, as well as the EKF used for pressure estimation in the case of a PS3 pressure sensor fault were computationally expensive, especially due to the multiple calls to the CLM required to generate the linear model through numerical differentiation.

The above-mentioned issues predicated the need for assessing the computational requirements of the implemented MBFTC algorithms in FSIM and employing any necessary modifications to ensure real-time implementation on the actual FADEC hardware. In early 2005, we accomplished the real-time implementation of the algorithms in the Life Extending Control (LEC) program, another NASA RASER program, on the dry rig. In the LEC program, the CLM was simulated as an embedded engine model in the FADEC software. From that experience we learnt that we can afford 2 calls to the CLM per minor frame leaving some room for any additional calculations needed by the fault detection and accommodation algorithms.

For the EKF implementation, we need to linearize the nonlinear CLM, specifically with respect to 9 state variables being estimated and the 7algeberaic (SLAM) variables that are updated with 7 algebraic relations (SLAM). The initial code for the linearization used for this EKF implementation was implemented using central differencing, implying a total of 33 calls to the CLM to get a complete linear model update (1 call for nominal values of state derivatives and outputs and 2*(9+7) calls for central difference calculations). On top of these 33 calls to the CLM for linear model generation, the EKF implementation was designed for propagate and measurement update at the minor frame rate, adding 2 additional calls for state derivative and output evaluations, respectively. Moreover, the fault detection algorithm relies on simulating an open-loop half-deteriorated engine model to determine when we reach a steady state (see the state machine logic in figure 75) and thereafter calculate a bias correction in all 9 sensors to correct for the fact that the actual engine has a random engine-to-engine variation and deterioration. So, altogether, the fault detection algorithm involved 36 calls to the embedded CLM at the minor frame rate, which is clearly not feasible for implementation on the FADEC.

Motivated by this, we modified the fault detection algorithm such that the embedded CLM is never called more than 2 times in any minor frame, to leave some time for the rest of the calculations (matrix multiplications and inversion in the EKF implementation, fault detection and fusion algorithms). In particular, we modified the propagation and measurement update to occur at intervals of 4 minor frames, as well as the open-loop model integration at 4 minor frame intervals (synchronized with the EKF propagation calls). Also, we modified the linearization to use only a forward difference method, and distribute the calls for baseline values of state derivatives and outputs and perturbations of the 9 states and 7 algebraic (SLAM) variables over a cycle of 16 minor frames. Thus, the linear model is now updated every 16 minor frames, which is still fairly fast compared to the dynamics of the 9 states being estimated.

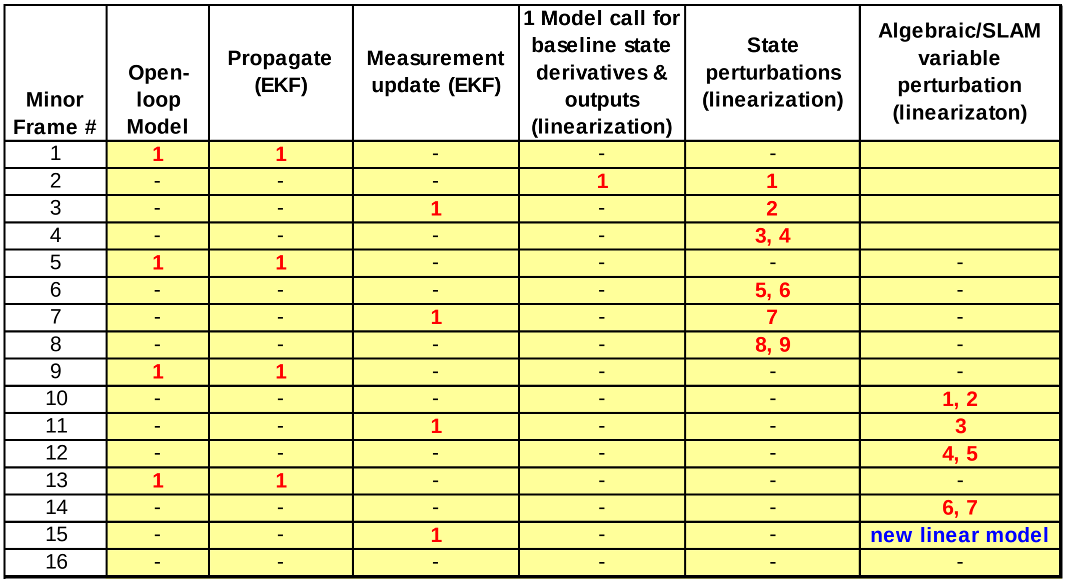
Figure 76 shows the detailed sequence of calls to the CLM for open-loop model simulation, linear model generation and the EKF, never exceeding more than 2 calls to the CLM in each minor frame and generating a new linear model over a cycle of 16 minor frames.

We have validated the above modifications to the fault detection algorithms in FSIM to ensure that the algorithm indeed runs much faster than before without any detrimental impact on the performance for fault detection and isolation.

Once a fault is detected and isolated, the state machine logic moves to the fault accommodation state and engages the requisite fault accommodation. The open-loop model call and the fault detection EKF are no longer needed. The fault accommodation for HPC, HPT and VG faults involve lookup tables, which is not very computationally expensive, so we don’t foresee any problem in real-time implementation for these on the FADEC. On the other hand, for the case of pressure sensor fault, the fault accommodation involves another EKF used to estimate the correct value of pressure using the remaining 8 sensors. In our initial implementation of this EKF in FSIM, a numerical differentiation of the CLM was used to generate the linear model with forward differencing for the 9 state variables and 7 algebraic (SLAM) variables. Thus, the linear model generation involved 17 (1 for baseline value of state derivatives and outputs and 16 for the perturbations of state and algebraic variables). Including the 2 additional calls for the propagation and measurement update in the EKF, there were altogether 19 calls to the CLM. Similar, to the EKF for fault detection, we modified this EKF for pressure estimation to distribute the calls to the CLM over a cycle of 18 minor frames. Figure 77 shows the detailed sequence of calls to the embedded CLM, never exceeding 2 calls in any minor frame. The propagation and measurement updates were modified to occur at intervals of 2 minor frames and a new linear model is generated every 18 minor frames.

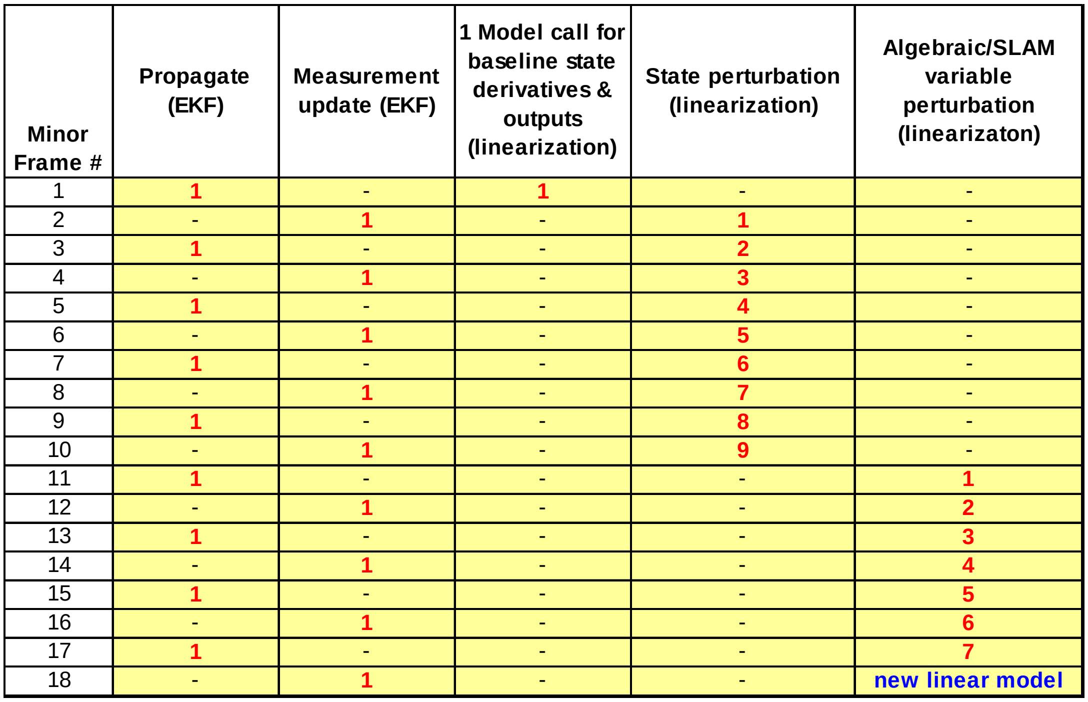
The above modifications for the pressure estimation EKF were also implemented and validated in FSIM to ensure that we get a good pressure estimation performance during transients as with the original implementation, while it runs significantly faster.

In addition to the above major changes in the CLM linearization and EKF algorithms for fault detection and accommodation, we also implemented some minor changes to speed up the computation. In particular, the NN algorithm involves the use of the *tanh* function. The initial implementation was achieved through calls to the *exp* function, which in turn employed a series calculation (Taylor series) until a desired accuracy was achieved. This was computationally too expensive. The *tanh* function was modified to use a lookup table, which is considerably faster. Similarly, the fusion algorithm requires the use of an exponential calculation (see eq. (12)). Initially, this was implemented through the use of *log* and *exp* functions, which in turn used Taylor series calculations and was, thus, very expensive. This function was also replaced with a look-up table, given that the average probability $p_{j,avg}$ is always in the range 0 to 1.



Minor Frame #	Open-loop Model	Propagate (EKF)	Measurement update (EKF)	1 Model call for baseline state derivatives & outputs (linearization)	State perturbations (linearization)	Algebraic/SLAM variable perturbation (linearizaton)
1	1	1	-	-	-	
2	-	-	-	1	1	
3	-	-	1	-	2	
4	-	-	-	-	3, 4	
5	1	1	-	-	-	-
6	-	-	-	-	5, 6	-
7	-	-	1	-	7	-
8	-	-	-	-	8, 9	-
9	1	1	-	-	-	-
10	-	-	-	-	-	1, 2
11	-	-	1	-	-	3
12	-	-	-	-	-	4, 5
13	1	1	-	-	-	-
14	-	-	-	-	-	6, 7
15	-	-	1	-	-	new linear model
16	-	-	-	-	-	-

Figure 76.—Sequence of calls to CLM for linearization and EKF for fault detection.



Minor Frame #	Propagate (EKF)	Measurement update (EKF)	1 Model call for baseline state derivatives & outputs (linearization)	State perturbation (linearization)	Algebraic/SLAM variable perturbation (linearizaton)
1	1	-	1	-	-
2	-	1	-	1	-
3	1	-	-	2	-
4	-	1	-	3	-
5	1	-	-	4	-
6	-	1	-	5	-
7	1	-	-	6	-
8	-	1	-	7	-
9	1	-	-	8	-
10	-	1	-	9	-
11	1	-	-	-	1
12	-	1	-	-	2
13	1	-	-	-	3
14	-	1	-	-	4
15	1	-	-	-	5
16	-	1	-	-	6
17	1	-	-	-	7
18	-	1	-	-	new linear model

Figure 77.—Sequence of calls to CLM for linearization and EKF for pressure estimation.

With the above modifications, it was verified that the FSIM simulations were accelerated considerably compared to the initial baseline version. The performance of this optimized FSIM implementation was validated against the baseline version to ensure against loss of performance. The next few sections provide sample illustrative runs for each of the fault type, using the final optimized FSIM implementation.

8.2.1 FSIM Implementation for HPC Fault

Figure 78 shows the performance of the real-time version of the FSIM implementation for a medium HPC fault. The system achieves a steady-state condition and then the medium HPC fault is injected at t = 215s, which leads to a significant loss in the booster and HPC stall margins. The fault detection algorithm detects the medium HPC fault at t = 225s and employs the corresponding fault accommodation. The fault accommodation provides good recovery of booster and HPC stall margins to pre-fault values, without any significant impact on thrust; there is very little impact on the fan stall margin due to this fault.

8.2.2 FSIM Implementation for HPT Fault

Figure 79 shows the performance of the real-time version of the FSIM implementation for a medium HPT fault. The system achieves a steady-state condition and then the medium HPT fault is injected at t = 215s, leading to loss in booster and HPC stall margins. The fault detection algorithm detects the medium HPT fault at t = 225s and employs the corresponding fault accommodation. The fault accommodation again leads to a good recovery of booster and HPC stall margins to pre-fault values or better, without any significant impact on thrust; there is very little impact on the fan stall margin due to this fault.

8.2.3 FSIM Implementation for VG Fault

Figure 80 shows the performance of the real-time version of the FSIM implementation for a large VG fault. The system achieves a steady-state condition and then the large VG fault is injected at t = 215s, which leads to a slight loss in HPC stall margin. The fault detection algorithm detects the large VG fault at t = 225s and employs the corresponding fault accommodation. The fault accommodation yields good recovery of the HPC stall margins to pre-fault value, without any significant impact on thrust; there is very little impact on the fan or booster stall margin due to this fault at this flight condition.

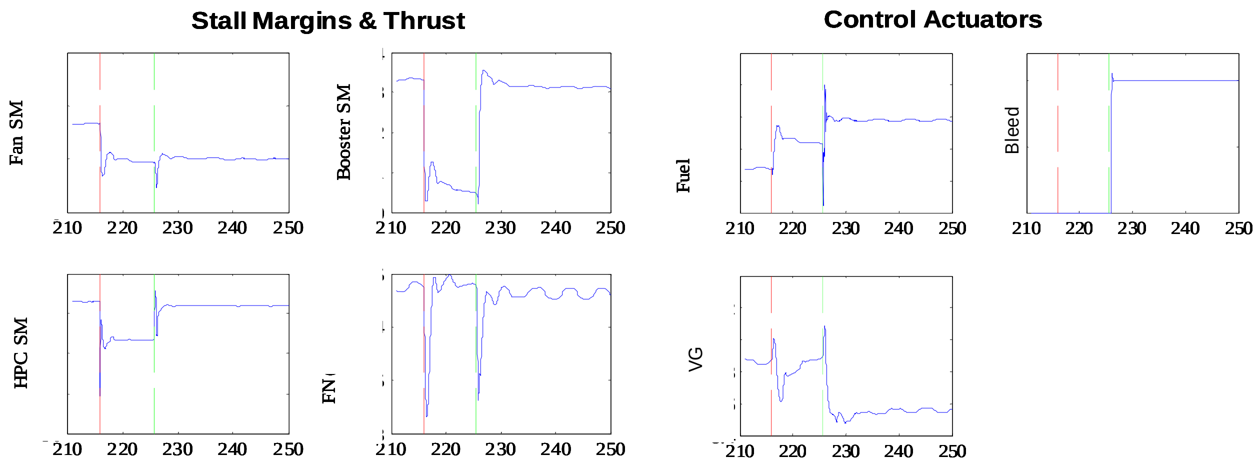


Figure 78.—Performance of real-time version of FSIM for a medium HPC fault.

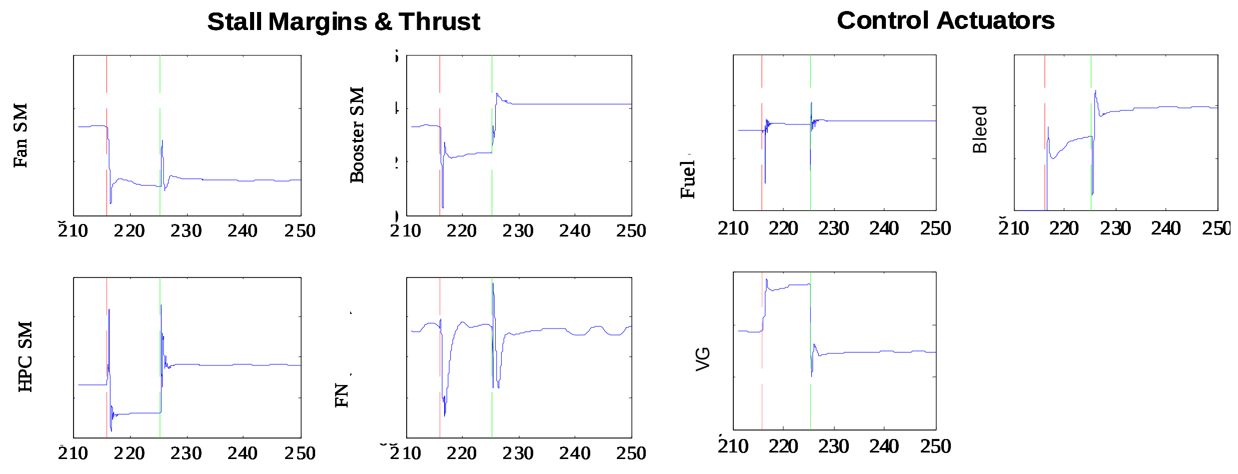


Figure 79.—Performance of real-time version of FSIM for a medium HPT fault.

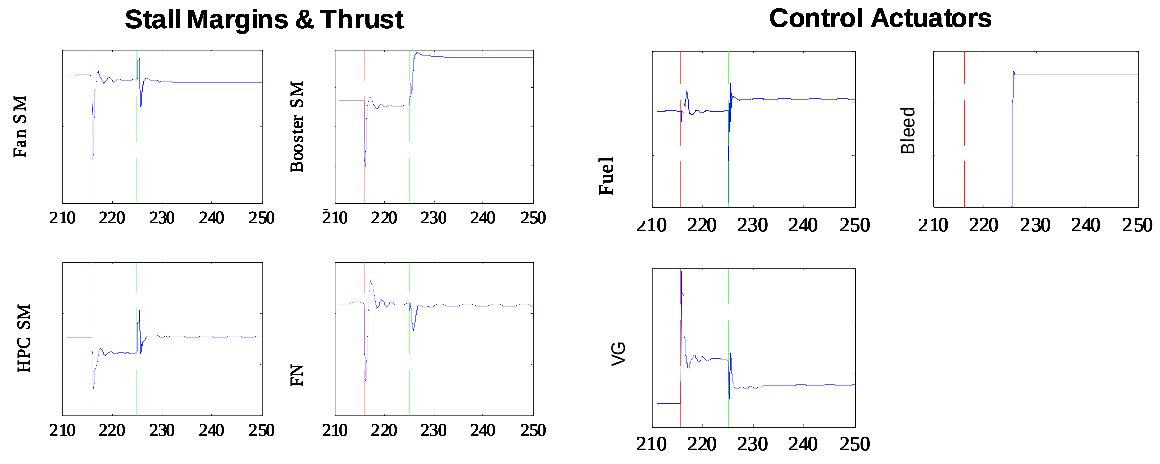


Figure 80.—Performance of real-time version of FSIM for large VG fault.

8.2.4 FSIM Implementation for Pressure Sensor Fault

Figure 81 shows the performance of the optimized real-time version of FSIM implementation for a large PS3 pressure sensor fault in the presence of TRA transients, with changes in TRA from 60 to 70, to 75 and finally to 85—as mentioned before, the pressure sensor has impact on the engine performance only during transient operation. In this simulation, the system is at steady state at TRA 60, and the pressure sensor fault is introduced. The results are very much the same as obtained by the initial (non-real-time) version of the logic. Clearly, while the faulty pressure sensor (in blue) is quite different during the transient sequence compared to the un-faulted (in red) sequence, the estimated pressure sequence (in green) obtained during the sensor fault accommodation is the same as the un-faulted sequence. The bottom plot shows the corresponding error between the faulted and estimated pressure values compared to the baseline, un-faulted sequence.

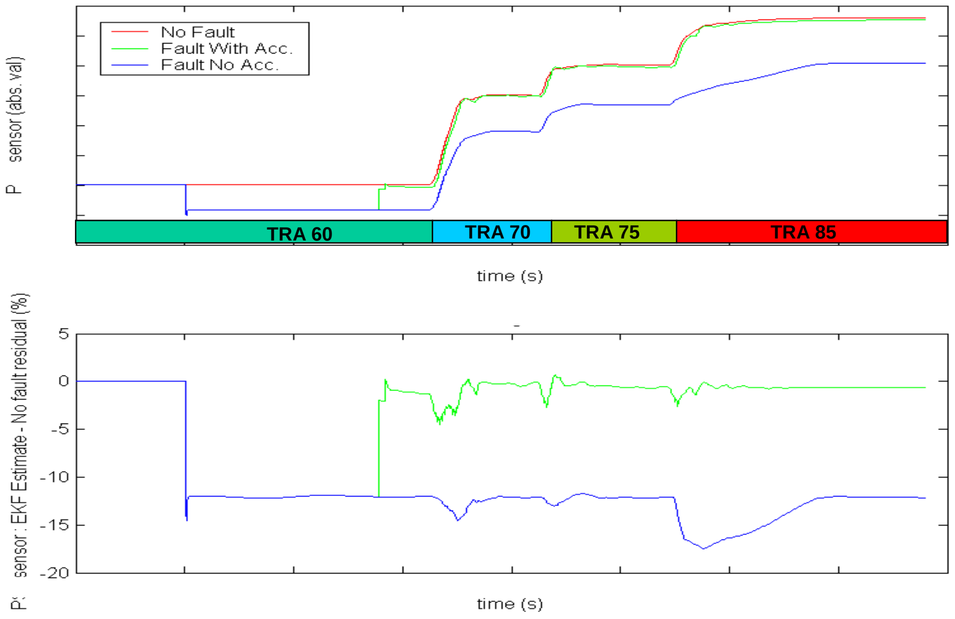
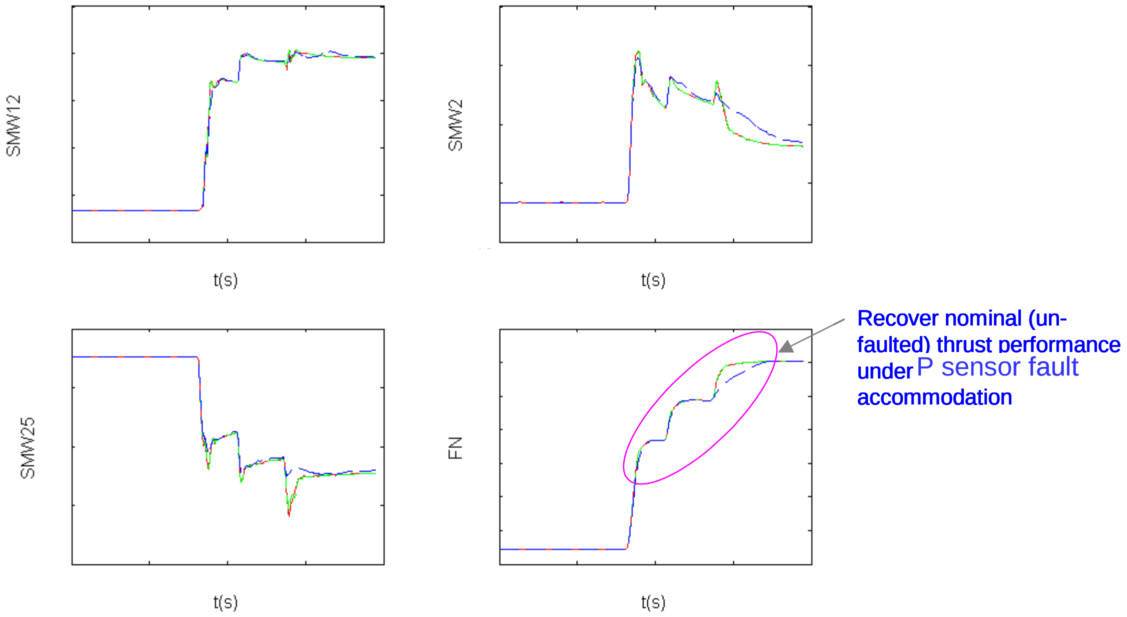


Figure 81.—Performance of real-time version of FSIM during a pressure sensor fault and TRA transients.



Recover nominal (un-faulted) thrust performance under P sensor fault accommodation

Figure 82.—Comparison of fan, booster and compressor stall margins and net thrust for un-faulted engine (red), faulted engine without accommodation, i.e., using faulted pressure feedback (blue) and faulted engine with accommodation, i.e., using estimated pressure feedback (green).

Figure 82 shows, the comparison of the stall margins in the fan, booster and HPC as well as the net thrust during this TRA sequence for the nominal un-faulted engine (red), faulted engine without any accommodation, i.e., using the faulty pressure feedback in FADEC (blue) and the faulted engine with accommodation, i.e., using the estimated pressure feedback in FADEC (green). Clearly, the sensor fault accommodation enables a near-perfect recovery of the un-faulted performance, while the faulted engine suffers significant deterioration in transient thrust response due to the faulty pressure feedback, especially during the TRA step from 75 to 85.

9. Conclusions

In this program, we have successfully developed and demonstrated the MBFTC technology for achieving the desired objectives of automated on-wing detection and accommodation of engine component faults. For this program, we focused on four key fault types from different engine control system components, i.e., sensor, actuator and gas-path turbo-machinery component. For these faults, we developed multiple model-based fault detection algorithms, which employ at their core the use of an Extended Kalman Filter (EKF). The EKF is in turn implemented using an embedded engine model, namely the component level model (CLM). The EKF allows accounting for system nonlinearities, and a wide variety of variations in an optimal manner, thereby facilitating the fault detection and isolation.

We tested the robustness and performance of the developed fault detection algorithms for all four fault types and different magnitudes, using extensive Monte Carlo simulations covering the entire flight envelope and in the presence of engine-to-engine variations, deterioration and sensor noise. A key step towards this was the optimized tuning of the EKF to minimize the impact of variation sources, while generating a structured residual pattern correlated with fault types and magnitudes. Two fault detection algorithms were implemented in parallel, namely (i) multiple hypothesis testing, and (ii) neural network, and a fusion algorithm was employed to exploit the complementary performance of these individual algorithms to achieve excellent overall fault detection performance.

The detection of fault type and magnitude allowed the use of automated fault accommodation through FADEC adjustments to take appropriate corrective control action and mitigate the adverse impact of the faults on the engine operability and performance. In general, the impact of these faults was predominantly seen in loss of booster and compressor stall margins. The accommodation strategy was designed to recover the lost stall margins while maintaining pre-fault thrust. To this end, off-line model-based optimization was performed for combinations of fault types and magnitudes at various points in the flight envelope to generate look-up tables for optimal FADEC adjustments, which were in turn interpolated online for optimal accommodation based on identified fault type and magnitude. The performance of the fault accommodation strategy was tested through simulations at various flight conditions, for new and deteriorated engines, and all four fault types and different fault magnitudes. On the other hand, for the pressure sensor fault, the fault accommodation strategy involved the use of a specially designed and tuned EKF that used the remaining sensors for optimal estimation of the true pressure value, and replacing the faulty pressure sensor feedback with the estimated value. The performance of this approach was verified during transient accelerations to demonstrate the recovery of un-faulted transient engine control performance.

The developed fault detection and accommodation algorithms were implemented and tested in a FADEC simulation (FSIM) environment, wherein the actual FADEC compatible software was tested in closed-loop simulations using the CLM to simulate the true engine with any variation and fault type/magnitude. This is a critical step in validating the FADEC software and implementing it on the actual FADEC hardware for dry rig testing. While we have been unable to implement and test the software in a dry rig due to problems with getting access to an appropriate commercial engine rig, we have addressed all likely issues pertaining to real-time implementation of the MBFTC algorithms in the real FADEC hardware. We are very confident that the final real-time version of the FSIM software will run in real-time on the actual FADEC.

In this program, we used a fixed embedded CLM in the EKF without any parameter adaptation to match the model to a particular engine with engine-to-engine variation and deterioration level. Rather, we followed the approach of tuning the EKF to be robust to these engine variations. A possible future extension is to incorporate a “tracking filter” to update the embedded engine model health parameters to match a specific engine, thereby allowing improved fault detection and isolation performance, especially during transient operation. Moreover, the tracking filter will provide an estimate of engine deterioration, thereby enabling fault accommodation tailored for the specific level of deterioration in the engine. Another obvious extension is to expand the set of faults considered to encompass all sensor, actuator, and turbo-machinery component faults.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)* 01-09-2008	**2. REPORT TYPE** Final Contractor Report	**3. DATES COVERED** *(From - To)* January 2002-September 2005

4. TITLE AND SUBTITLE
Model-Based Fault Tolerant Control

5a. CONTRACT NUMBER
NAS3-01135

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kumar, Aditya; Viassolo, Daniel

5d. PROJECT NUMBER

5e. TASK NUMBER
3

5f. WORK UNIT NUMBER
WBS-645846.02.07.03.03.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
GE Global Research
1 Research Circle
Niskayuna, New York 12309

8. PERFORMING ORGANIZATION REPORT NUMBER
E-16555

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORS ACRONYM(S)
NASA

11. SPONSORING/MONITORING REPORT NUMBER
NASA/CR-2008-215273

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category: 07
Available electronically at http://gltrs.grc.nasa.gov
This publication is available from the NASA Center for AeroSpace Information, 301-621-0390

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Model Based Fault Tolerant Control (MBFTC) task was conducted under the NASA Aviation Safety and Security Program. The goal of MBFTC is to develop and demonstrate real-time strategies to diagnose and accommodate anomalous aircraft engine events such as sensor faults, actuator faults, or turbine gas-path component damage that can lead to in-flight shutdowns, aborted take offs, asymmetric thrust/loss of thrust control, or engine surge/stall events. A suite of model-based fault detection algorithms were developed and evaluated. Based on the performance and maturity of the developed algorithms two approaches were selected for further analysis: (i) multiple-hypothesis testing, and (ii) neural networks; both used residuals from an Extended Kalman Filter to detect the occurrence of the selected faults. A simple fusion algorithm was implemented to combine the results from each algorithm to obtain an overall estimate of the identified fault type and magnitude. The identification of the fault type and magnitude enabled the use of an online fault accommodation strategy to correct for the adverse impact of these faults on engine operability thereby enabling continued engine operation in the presence of these faults. The performance of the fault detection and accommodation algorithm was extensively tested in a simulation environment.

15. SUBJECT TERMS
Aircraft engines; Systems health monitoring; Gas turbine engines; Flight safety

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email:help@sti.nasa.gov)
U	U	U	UU	98	**19b. TELEPHONE NUMBER** *(include area code)* 301-621-0390