NASA
Technical
Paper
2925

February 1990

Advanced Detection, Isolation, and Accommodation of Sensor Failures in Turbofan Engines

Real-Time Microcomputer Implementation

John C. DeLaat
and Walter C. Merrill

NASA

NASA Technical Paper 2925

1990

Advanced Detection, Isolation, and Accommodation of Sensor Failures in Turbofan Engines

Real-Time Microcomputer Implementation

John C. DeLaat
and Walter C. Merrill
Lewis Research Center
Cleveland, Ohio

NASA
National Aeronautics and Space Administration
Office of Management
Scientific and Technical Information Division

Contents

Summary

The objective of the Advanced Detection, Isolation, and Accommodation (ADIA) Program is to improve the overall demonstrated reliability of digital electronic control systems for turbine engines. For this purpose, an algorithm developed to detect, isolate, and accommodate sensor failures was combined with an existing multivariable control algorithm to give a complete control implementation with sensor analytical redundancy. The algorithm has been evaluated on a real-time engine simulation and demonstrated on a full-scale F100 turbofan engine; this was done by implementing it in real time on a microprocessor-based controls computer with state-of-the-art microprocessor hardware and software. The required computational power for the real-time implementation was achieved with parallel processing. High order language programming reduced the programming and maintenance costs of the implementation software.

This paper describes the real-time microprocessor implementation of the algorithm. Overviews of the multivariable control and ADIA algorithms are given. The test equipment used for evaluating and demonstrating the algorithm and the microprocessor hardware and software necessary for implementing it are described. The real-time implementation made possible the successful completion of the ADIA evaluation and demonstration. Conclusions and recommendations for the implementation strategy in future research programs are made.

Introduction

Over the past 35 yr, hydromechanical implementations of turbine engine control systems have matured into highly reliable units. However, with the trend towards increased engine complexity in order to meet ever-increasing engine performance requirements, the engine control has also become increasingly complex. Because of this trend toward complexity and the revolution in digital electronics, the control has evolved from a hydromechanical to a full authority digital electronic control (FADEC) implementation. These FADEC's must demonstrate levels of reliability as good or better than their hydromechanical predecessors.

Thus, in an effort to improve the overall reliability of the digital electronic control system, various redundancy management techniques have been applied to both the total control system and to individual components. Reference 1 shows that the engine sensors are the least reliable of the control system components. In fact, some type of sensor redundancy is required to achieve adequate control system reliability. One important type is analytical redundancy, wherein a mathematical model generates redundant information that can be compared to measured information to detect failures. Future engine systems with demanding mission reliability and flight safety requirements will require redundant information not only in the sensor subsystem but also in computers and actuator interfaces as well. Analytically redundant systems such as in the Advanced Detection, Isolation, and Accommodation (ADIA) Program utilize the full on-board computational capability to extract redundant information from dissimilar sensors. These systems not only will provide maximum system reliability with minimum hardware replication and computer interfaces but also will, in turn, offer weight and cost savings.

Considerable progress has been made in applying analytical redundancy to improve the reliability of the turbine engine control system. Reference 2 surveys these accomplishments and defines several technology needs. These needs include (1) the ability to detect small (soft) failures, (2) real-time implementation of algorithms capable of detecting soft failures, (3) a comparison of algorithm complexity versus performance, (4) a full-scale demonstration of a soft-failure detection capability, and (5) an evaluation of the pseudolinearized modeling approach. The ADIA program addresses all of these technology needs.

The ADIA program is organized into four phases: development, implementation, evaluation, and demonstration. In the development phase (refs. 3 and 4) the ADIA algorithm was designed by using advanced filtering and detection methodologies. In the implementation phase (ref. 5) this advanced algorithm was implemented in microprocessor-based hardware. A parallel computer architecture (three processors) allowed the algorithm to execute in a timeframe consistent with stable, real-time operation. In the evaluation phase (refs. 6 and 7) the advanced algorithm and its implementation were evaluated against a real-time hybrid simulation of the F100 engine. The objectives of this phase were to validate the algorithm's performance in real time and to establish a data base for the demonstration phase of the ADIA program. Recently, a full-scale F100 engine in the NASA Lewis Propulsion Systems Laboratory (PSL) was used to demonstrate the implemented algorithm's operation and performance over a substantial portion of the F100 engine's flight operating envelope (refs. 8 and 9). This report describes the details

of the algorithm's implementation for this engine demonstration. Detailed results of the engine demonstration are given in reference 9.

This report begins with a description of the test-bed system used in evaluating and demonstrating the ADIA algorithm. Next, the F100 multivariable control (MVC) and ADIA algorithms are described, as are the implementation hardware and software. Then specific implementation details and operational procedures for the engine demonstration are given. Finally, the results of the ADIA program implementation are summarized, and recommendations are given for future work.

Evaluation and Demonstration Test-Bed Configuration

Both the simulation evaluation and the engine demonstration of the ADIA algorithm were carried out on the test-bed configuration shown in figure 1. The test bed consisted of the F100 engine system or engine system simulation, the controls microcomputer system containing the MVC and the ADIA algorithms, the sensor failure simulator, and the F100 simplified engine simulator. Each of these components is described in the following sections.

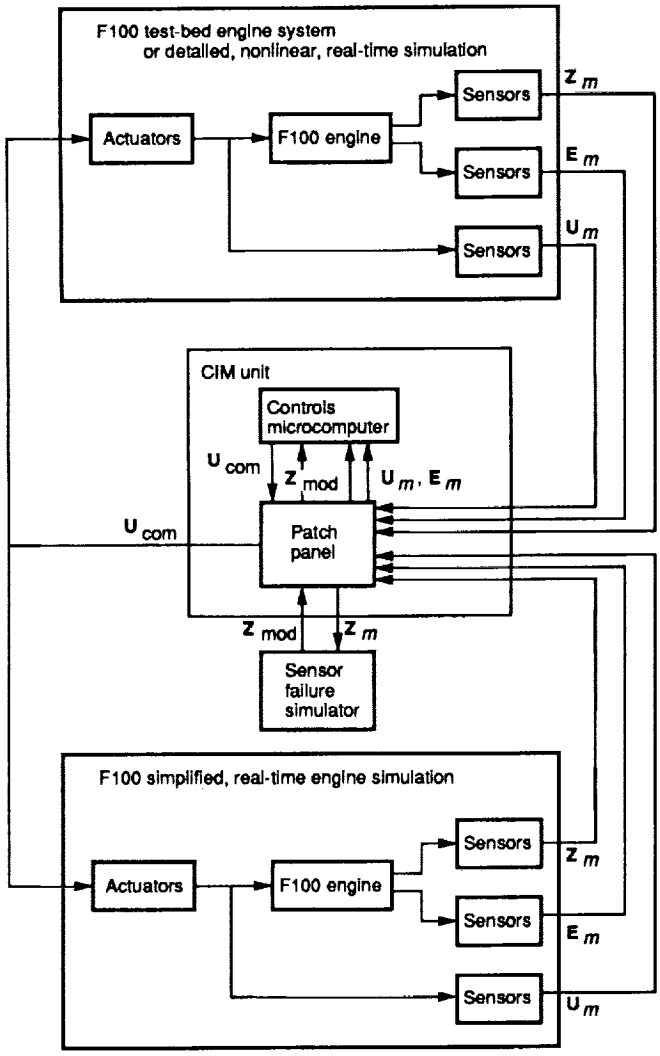


Figure 1.—Test-bed configuration.

F100 Engine System

The engine system consisted of the F100 turbofan engine, the actuators, and the sensors. The F100 engine is a high-performance, low-bypass-ratio, twin-spool turbofan engine. The engine has seven controlled inputs (four of which were used for this effort), five engine outputs, and four environmental variables. These variables are defined in table I. Strictly speaking, TT25 is an engine output variable; however, since TT25 is used only as a scheduling variable in the control (like TT2), it is considered an environmental variable and is not covered by the ADIA logic.

F100 Engine System Simulation

A real-time hybrid computer simulation of the F100 engine (ref. 10) was developed by NASA Lewis to support controls research programs (see fig. 2). The simulation has both wide-range steady-state and transient computing capabilities. This engine simulation was used for the preliminary control evaluation during the F100 multivariable control synthesis (MVCS) program (ref. 11) and for the algorithm evaluation phase of the ADIA program. In addition to the engine itself, the hybrid computers also simulate the engine actuators and sensors. Since the simulation is essentially the same as that used for the F100 MVCS program, data from that program were compared directly to data generated during the ADIA evaluation to validate normal mode control operation.

TABLE I.—ENGINE VARIABLES

Variable	Definition
Controlled engine inputs, $\mathbf{U}_{com}$ and $\mathbf{U}_m$	
WF	Main combuster fuel flow
AJ	Exhaust nozzle area
CIVV	Compressor inlet variable vanes
RCVV	Rear compressor variable vanes
Sensed engine outputs, $\mathbf{Z}_m$	
N1	Fan speed
N2	Compressor speed
PT4	Burner pressure
PT6	Exhaust nozzle pressure
FTIT	Fan turbine inlet temperature
Sensed environmental variables, $\mathbf{E}_m$	
P0	Ambient (static) pressure
PT2	Fan inlet (total) pressure
TT2	Fan inlet temperature
TT25	Compressor inlet temperature

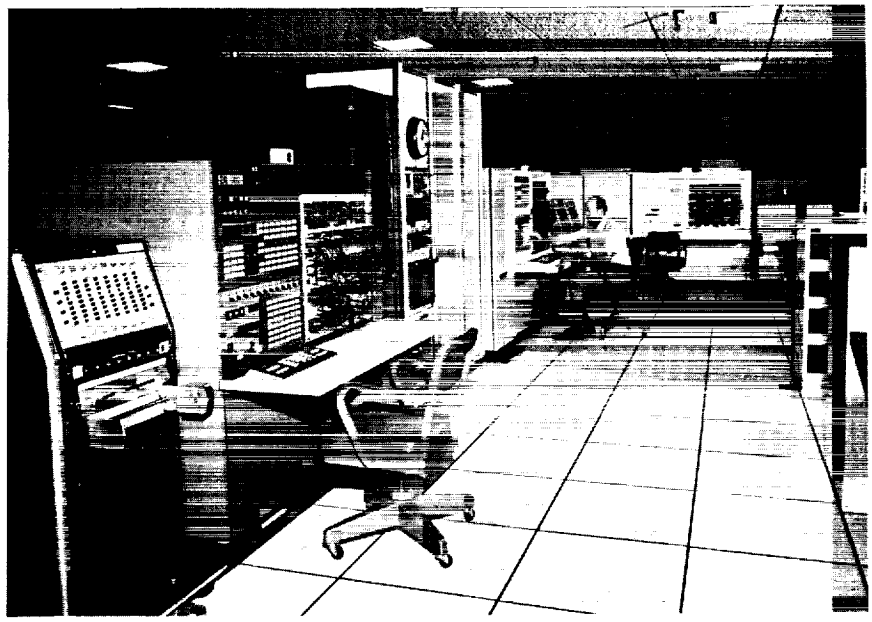


Figure 2.—Real-time hybrid simulation computers.

Controls Microcomputer System

The control, interface, and monitoring (CIM) unit (fig. 3) was designed and fabricated to provide an effective means of implementing control algorithms for research in real time using realistic hardware—that is, microcomputer hardware similar to that which would be used to build actual engine control systems. The CIM unit contains the microcomputer that implements the combined MVC–ADIA algorithm in real time; this microcomputer will be described in the section Microcomputer Implementation. The CIM unit also contains hardware and cabling to provide a flexible interface to and from the engine or engine simulation being controlled (fig. 4). This interface consists of cabling, a patching system, signal conditioning, and connectors. A monitoring system in the CIM unit allows the signals between the microcomputer and the controlled engine to be examined. This monitoring system consists of selection logic to determine which signal is to be examined and scaling logic to allow the signal to be viewed in either volts or engineering units. The interface and monitoring functions of the CIM unit are described in detail in reference 12.

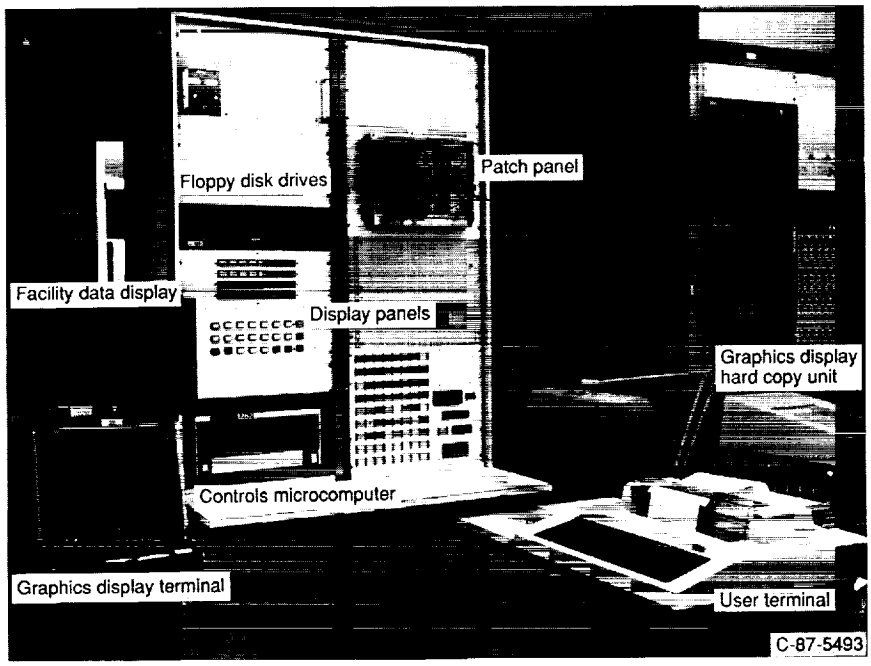


Figure 3.—Control, interface, and monitoring (CIM) unit.

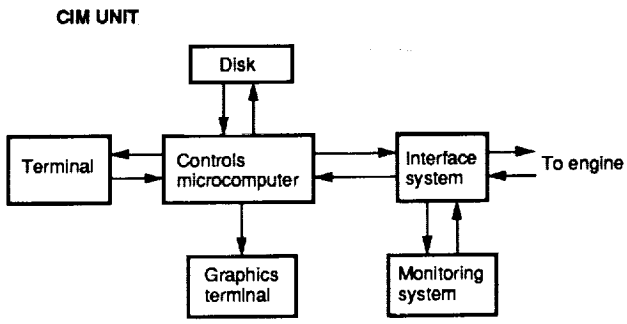


Figure 4.—Block diagram of CIM unit.

Sensor Failure Simulator

The sensor failure simulator (SFS) provides an efficient means of modifying engine sensor signals to simulate sensor failures. The SFS unit consists of a personal computer driving discrete analog hardware. The personal computer allows a menu-driven, top-down approach to failure scenario creation, retrieval, editing, and execution. A failure scenario consists of the sensor channel(s) to be failed, the failure mode(s) for each channel, and the time at which the failure occurs for each channel. The SFS can simulate any of four basic sensor failure modes: scale-factor change, bias, drift, and noise. These failure modes are implemented in analog electronic hardware that is controlled by the personal computer. The SFS allows complete and repeatable control over the failure size and the timing of failure injection. Details of the SFS are given in reference 13.

F100 Simplified Engine Simulator

The F100 simplified engine simulator is microprocessor-based and uses hardware and software similar to that used for the ADIA real-time implementation. During the engine demonstration the simulator was used to validate changes made to the controls microcomputer software; that is, all changes to the MVC or ADIA software were tested with the simplified engine simulator in order to guarantee the integrity of the software. The changes were then run with the actual engine. This procedure allowed checkout of software changes without compromising the safety of the engine. Another function of the engine simulator during engine testing was to simulate the engine actuators so that actuator failures could be detected. Details of how this was accomplished are in the Safety Procedures section of this report. Details of the simplified engine simulator design and implementation are described in reference 14. Figure 5 shows the SFS and the simplified engine simulator mounted in the PSL control room.

F100 Multivariable Control Algorithm

The control algorithm for this effort was developed during the F100 multivariable control synthesis (MVCS) program (ref. 15). In the MVCS program, linear quadratic regulator (LQR) theory was successfully used to design and implement a practical multivariable control (MVC) for a state-of-the-art turbofan engine. The MVC algorithm was designed by using continuous

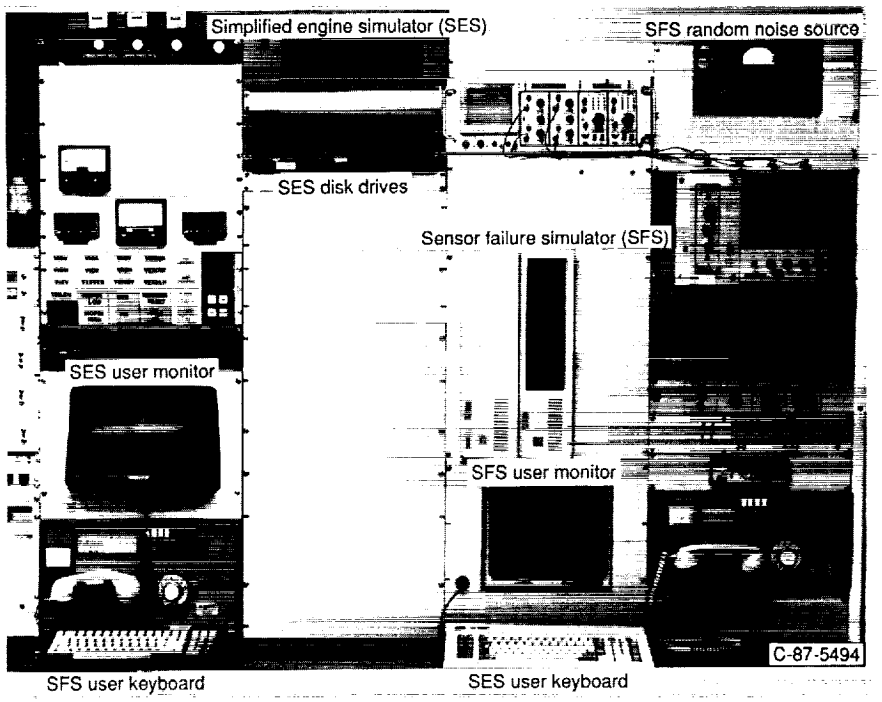


Figure 5.—Sensor failure simulator and simplified engine simulator.

system design techniques, because it was assumed that the update interval for the total control would be fast enough that discrete design techniques would not be necessary. Originally a high-speed minicomputer implemented the algorithm (ref. 16). More recently, the algorithm was implemented with a microcomputer. A simplified block diagram of the algorithm is shown in figure 6. The control mode is proportional plus integral with a feed-forward path to provide rapid response. The control and output variables used in the MVC are those shown in table I.

Seven modules make up the MVC algorithm: the reference point schedules (RPSCH), the transition control, the integral control, the linear quadratic regulator (LQR), the gain control, the engine protection logic, and the fan turbine inlet temperature (FTIT) estimator. Each of these modules is described in the following paragraphs.

The RPSCH use the pilot-input power lever angle PLA along with the flight conditions (ambient pressure P0, fan inlet pressure PT2, fan inlet temperature TT2, and compressor inlet temperature TT25) to calculate steady-state values throughout the entire engine operating range for the control variables $\mathbf{U}_s$, the output variables $\mathbf{Z}_s$, and the state variables $\mathbf{Y}_s$. In essence, the RPSCH contain an accurate, steady-state model of the engine.

The transition control operates on the RPSCH outputs to provide a transient model for the control to follow during changes in flight conditions or pilot commands. The transition control rate limits the outputs of the RPSCH to prevent excessive deviations, which could saturate the LQR.

The LQR provides the proportional action in the MVC algorithm by means of a set of proportional gains C_P calculated by the gain control. These gains act through all of the control variables to reduce deviations in the measured engine states $\mathbf{X}$ from their scheduled values computed by the transition control $\mathbf{X}_s$.

The integral control permits steady-state trimming of the engine operating point to satisfy performance requirements and engine limits. Using both integral gains C_I calculated by the gain control and the limit flags from the engine protection logic, the integral control eliminates any steady-state errors in the outputs caused by slight variations between the scheduled and actual engine parameters.

The gain control calculates gains for the LQR and the integral control. Because of the nonlinear nature of the engine, a single set of gains will not result in satisfactory performance at all engine operating conditions. The gains, therefore, are scheduled as functions of PT2, TT2, and transition control compressor speed $N2_s$. This allows the gains to reflect shifts in engine dynamics due to changes in flight conditions and/or engine power condition.

The engine protection logic limits the control variables to safe operating ranges throughout the entire engine operating envelope. In addition, whenever a limit is encountered on one of the control variables, a flag is set to limit that particular control's trim integrator in the integral control.

The FTIT estimator compensates for a slow FTIT sensor. During transient operation an estimated value of FTIT is used to predict whether or not overtemperature will occur. The integral control cuts back engine fuel flow, if required, to prevent the overtemperature. For the ADIA evaluation and demonstration, the MVC FTIT estimate was replaced by the ADIA FTIT estimate generated by the accommodation filter.

Additional information on the F100 MVC algorithm and its original minicomputer implementation can be found in references 16 and 17.

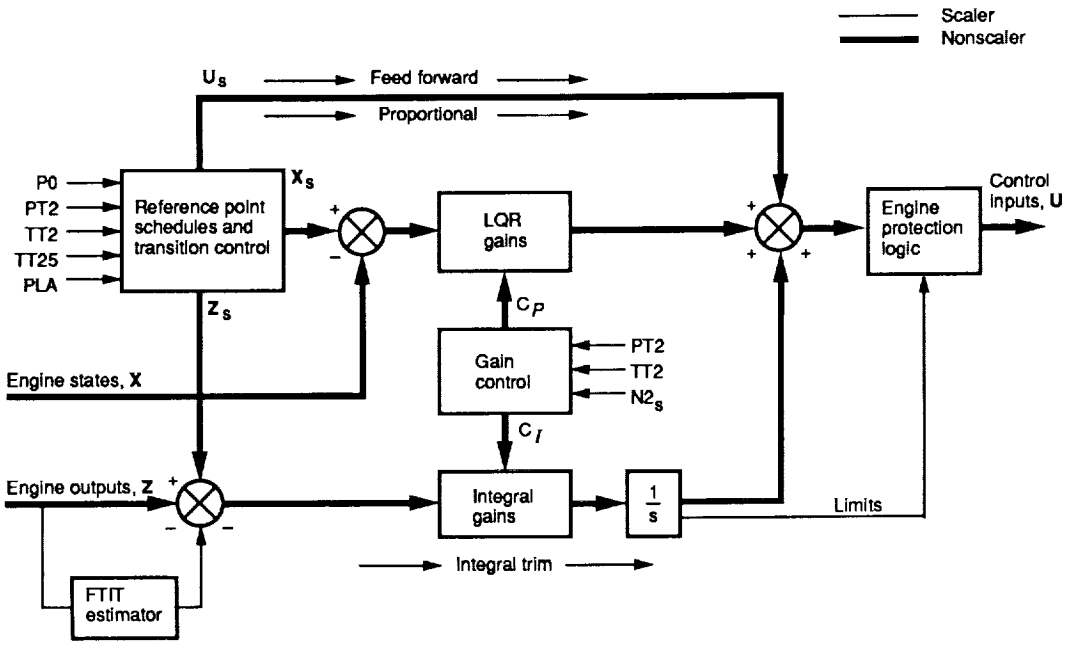


Figure 6.—Structure of F100 multivariable control.

Advanced Detection, Isolation, and Accommodation Algorithm

The ADIA algorithm to detect, isolate, and accommodate sensor failures in an F100 turbofan engine control system consists of three elements: (1) hard-failure detection and isolation logic, (2) soft-failure detection and isolation logic, and (3) an accommodation filter. Hard failures are defined as out-of-range or large bias errors that occur instantaneously in the sensed values. Soft failures are defined as small bias errors or drift errors that accumulate relatively slowly with time. The algorithm incorporates advanced filtering and detection logic, and is general enough to be applied to different engines or other types of control systems.

In the normal or unfailed mode of operation, the accommodation filter uses the full set of engine measurements to generate a set of optimal estimates of the measurements. These estimates $\hat{\mathbf{Z}}$ are used by the control law. If a sensor failure occurs, the detection logic notes the failure, and the isolation logic determines which sensor is faulty. This structural information is passed to the accommodation filter, which removes the faulty measurement from further consideration. The accommodation filter, however, continues to generate the full set of optimal estimates for the control. Thus the control mode does not have to restructure for any sensor failure.

As shown in figure 7, the ADIA algorithm inputs are the sensed engine output variables $\mathbf{Z}_m$, the engine environmental variables $\mathbf{E}_m$, and the sensed engine input variables $\mathbf{U}_m$. The outputs of the algorithm, that is, the estimates $\hat{\mathbf{Z}}(t)$ of the measured engine outputs $\mathbf{Z}_m(t)$, are inputs to the proportional part of the control. During normal mode operation, engine measurements are used by the integral control to ensure accurate steady-state operation. However, when a sensor failure is accommodated, the engine measurement is replaced with the corresponding accommodation filter estimate by reconfiguring the interface switch matrix.

Engine Model

The performance of the accommodation filter and the detection and isolation logic are strongly dependent on the model of the engine. The model used herein has a linear state-space structure. However, nonlinear engine characteristics are incorporated by representing the base points and the matrix elements within the linear state-space structure as nonlinear functions of various engine variables as follows:

$$\dot{\mathbf{X}} = F(\mathbf{X} - \mathbf{X}_b) + G(\mathbf{U} - \mathbf{U}_b) \tag{1}$$

$$\mathbf{Z} = H(\mathbf{X} - \mathbf{X}_b) + D(\mathbf{U} - \mathbf{U}_b) + \mathbf{Z}_b \tag{2}$$

Here the subscript b represents the base point, and $\mathbf{X}$ is the 4×1 model state vector, $\mathbf{U}$ the 4×1 control vector, and $\mathbf{Z}$ the 5×1 output vector. The F, G, H, and D matrices are the appropriately dimensioned system matrices. The system matrices and the model base points were determined at 109 operating points throughout the flight envelope. Three variables are sufficient to completely define a model operating point. Previous modeling efforts (ref. 3) used altitude, Mach number (MN), and power lever angle (PLA). However, in this study PLA, inlet pressure PT2, and inlet temperature TT2 were used. The latter set of variables is more appropriate for ensuring that all significant model dynamics are considered. Once

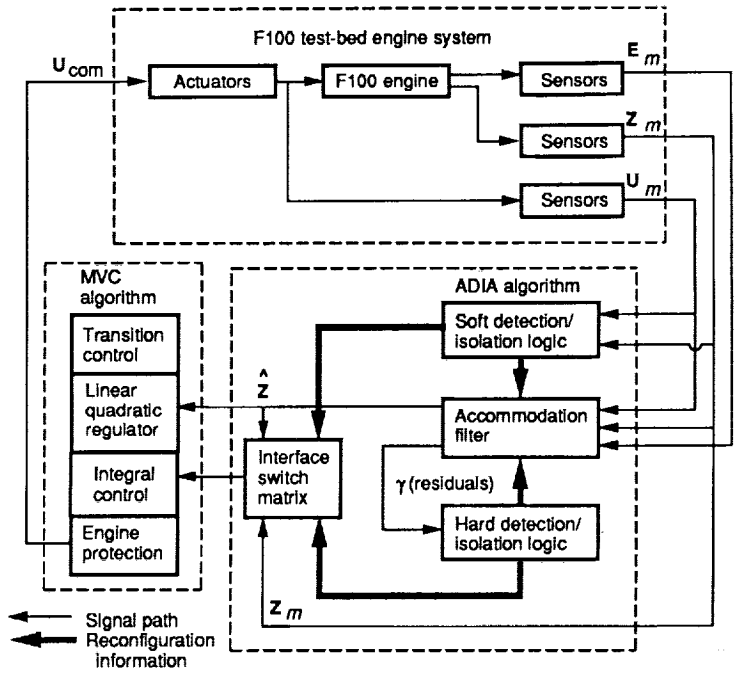


Figure 7.—Advanced detection, isolation, and accommodation (ADIA) block diagram.

system matrices are determined at all 109 operating points, the individual matrix elements are corrected by the engine inlet conditions $\mathbf{E}_m$ and scheduled as nonlinear functions of $\mathbf{Z}$. These functions are given in reference 4.

Accommodation Filter

The accommodation filter incorporates the engine model along with a Kalman gain update, to generate estimates of the engine outputs $\hat{\mathbf{Z}}$ and the engine states $\hat{\mathbf{X}}$ as follows:

$$\dot{\hat{\mathbf{X}}} = F(\hat{\mathbf{X}} - \mathbf{X}_b) + G(\mathbf{U}_m - \mathbf{U}_b) + K\gamma \tag{3}$$

$$\hat{\mathbf{Z}} = H(\hat{\mathbf{X}} - \mathbf{X}_b) + D(\mathbf{U}_m - \mathbf{U}_b) + \mathbf{Z}_b \tag{4}$$

$$\gamma = \mathbf{Z}_m - \hat{\mathbf{Z}} \tag{5}$$

where K is the Kalman gain matrix, γ is the residual vector, and the subscript m indicates the measured values.

Reconfiguration of the accommodation filter after the detection and isolation of a sensor failure is accomplished by forcing the appropriate residual element to zero. For example, if a failure has been isolated in the compressor speed sensor N2, the reconfiguration is effected by forcing $\gamma_2 = 0$. This is equivalent to setting sensed N2 equal to the estimate of N2 generated by the filter.

The accommodation filter was improved by adding integral action to improve steady-state accuracy of the FTIT estimate $\hat{\mathbf{Z}}_5$. Limiting the FTIT at high-power operation is an important engine control mode. However because the FTIT sensor is relatively slow, control action is based on the dynamically faster FTIT estimate. The FTIT limiting control has integral action; therefore a high degree of steady-state accuracy in the FTIT estimate is required to ensure satisfactory control. This accuracy is accomplished by augmenting the filter with the following additional state and output equations:

$$\dot{b} = K_6\gamma \tag{6}$$

$$\hat{\text{FTIT}} = \hat{\mathbf{Z}}_5 + b \tag{7}$$

where K_6 is a gain matrix, b is the temperature bias, and $\hat{\mathbf{Z}}_5$ is the unbiased temperature estimate. The addition of these dynamics, although improving FTIT estimation accuracy, results in a larger minimum detectable FTIT drift failure rate. The resulting filter structure, which includes the FTIT bias state, is the structure used in the accommodation filter and in all the hypothesis filters in the soft-failure detection and isolation logic.

Hard-Failure Detection and Isolation Logic

The hard-failure detection and isolation logic is straightforward. The residuals generated by the accommodation filter are used in the hard-failure detection logic. To detect and isolate hard failures, the absolute value of each component of the residual vector is compared to its own threshold. If the residual absolute value is greater than the threshold, a failure at the sensor corresponding to the residual element is detected and isolated. Threshold sizes are initially based on the standard deviation of the noise on the sensors. These standard deviation magnitudes are then increased to account for modeling errors in the accommodation filter. The hard-failure detection threshold values are twice the magnitude of the adjusted standard deviations, as can be seen in table II.

A failure is accommodated by reconfiguring the accommodation filter and all of the hypothesis filters in the soft-failure detection and isolation logic.

Soft-Failure Detection and Isolation Logic

The soft-failure detection logic consists of multiple hypothesis-based testing. Each hypothesis is implemented by using a Kalman filter. The soft-failure detection and isolation logic structure, shown in figure 8, consists of six hypothesis filters, one for normal mode operation and five for the failure modes (one for each engine output sensor). The structure for each hypothesis filter is identical to that of the accommodation filter. However, each hypothesis filter operates with a different set of measurements. For example, the first hypothesis filter H_1 uses all of the sensed engine outputs except the first, N1. The second uses all of the sensed outputs except the second, N2, and so on. Thus, each hypothesis filter generates a unique residual vector, γ_i. From this residual each hypothesis filter generates a statistic or likelihood based on a weighted sum of squared residuals (WSSR). Assuming Gaussian sensor noise, each sample of γ_i has a certain likelihood or probability

$$L_i = P_i(\gamma_i) = ke^{-\text{WSSR}_i} \tag{8}$$

where k is a constant and $\text{WSSR}_i = \gamma_i^T\Sigma^{-1}\gamma_i$ with $\Sigma = \text{diag}(\sigma_i^2)$. The σ_i are the standard deviations defined in table II.

TABLE II.—HARD-FAILURE DETECTION THRESHOLD MAGNITUDES

Sensor	Adjusted standard deviation, σ_i	Detection threshold
Speed, rpm		
N1 (fan) N2 (compressor)	300 400	600 800
Pressure, psi		
PT4 (burner) PT6 (exhaust nozzle)	30 5	60 10
Temperature, °R		
FTIT (fan turbine inlet)	250	500

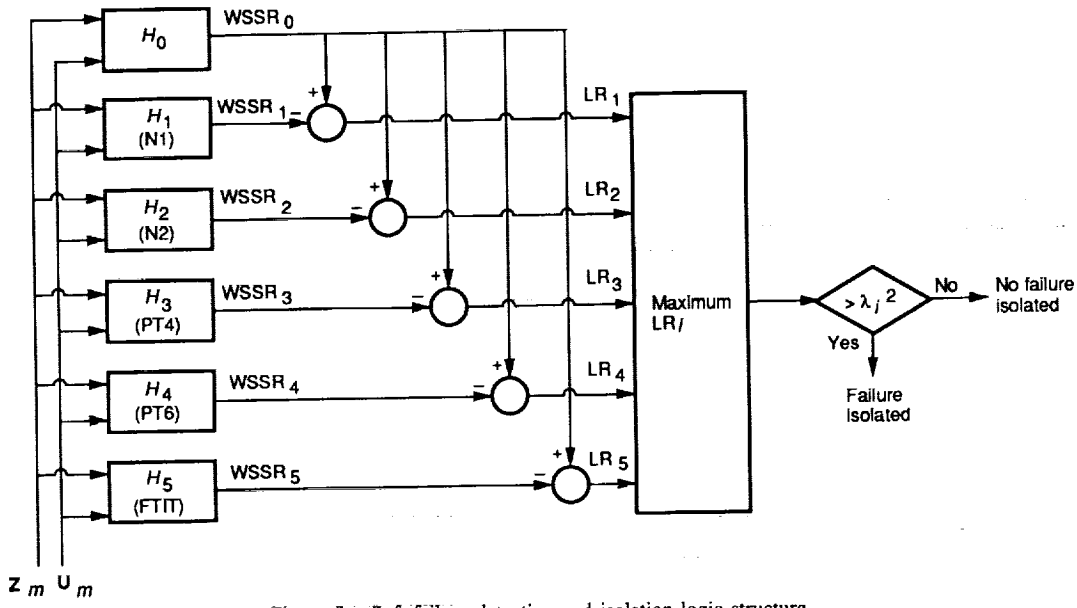


Figure 8.—Soft-failure detection and isolation logic structure.

These standard deviation values scale the residuals to unitless quantities that can be summed to form a WSSR. The WSSR statistic is smoothed to remove gross noise effects by a first-order lag with a time constant of 0.1 sec. When the log of the ratio of likelihoods LR is taken, then

$$\mathrm{LR}_i = \log\left(\frac{L_i}{L_0}\right) = \mathrm{WSSR}_0 - \mathrm{WSSR}_i \tag{9}$$

If the maximum log likelihood ratio exceeds the threshold, a failure is detected and isolated, and accommodation occurs. If a sensor failure has occurred in N1, for example, all of the hypothesis filters except H_1 will be corrupted by the faulty information. Thus each of the corresponding likelihoods will be small except for H_1. So the H_1 likelihood ratio will be the maximum that will be compared to the threshold to detect the failure.

Two steps are involved in accommodation. First, all seven of the filters (one accommodation and six hypothesis) are reconfigured to account for the detected failure mode; that is, the appropriate residual in each filter is forced to zero. Second, the states and estimates of all seven filters are updated to the correct values of the hypothesis filter that corresponds to the failed sensor.

Adaptive Threshold

Since the WSSR statistic is the sum of Gaussian variables squared, it has a chi-squared distribution. Initially, the soft-failure detection and isolation threshold was determined by standard statistical analysis of this distribution, thereby setting the confidence level of false alarms and missed detections. Next, the threshold was modified to account for modeling error. Initial evaluation studies showed that transient modeling error was dominant in determining the fixed threshold level; this threshold, in turn, was clearly too large for desirable steady-state operation. Consequently, an adaptive threshold was incorporated to improve steady-state detection and isolation while maintaining the algorithm's robustness to transient modeling error. The adaptive threshold is defined as follows:

$$\lambda_i = \lambda_{iSS}(\lambda_{EXP} + 1) \tag{10}$$

$$\tau\dot{\lambda}_{EXP} + \lambda_{EXP} = M_{tran} \tag{11}$$

This heuristically determined threshold consists of two parts. One part, λ_{iSS} the steady-state detection and isolation threshold, accounts for the steady-state, or low-frequency, modeling error. The second part, λ_{EXP}, accounts for the transient, or high-frequency modeling error. The adaptive threshold is triggered by an internal control system variable M_{tran}, which is indicative of transient operation. To minimize false alarms due to modeling during transients, the values of λ_{iSS}, τ, and M_{tran} were determined experimentally. When the engine experiences a transient, M_{tran} is set to 4.5; otherwise it is 0. The time constant $\tau = 2$ sec. The adaptive threshold expansion logic enabled λ_{iSS} to be reduced to 40 percent of its original value, which resulted in an 80 percent reduction in the detection and isolation threshold λ_i^2.

Microcomputer Implementation

The ADIA algorithm had been evaluated in nonreal time during algorithm development. This allowed debugging of the algorithm and a preliminary assessment of its capabilities. However, in order to evaluate the algorithm in detail and to

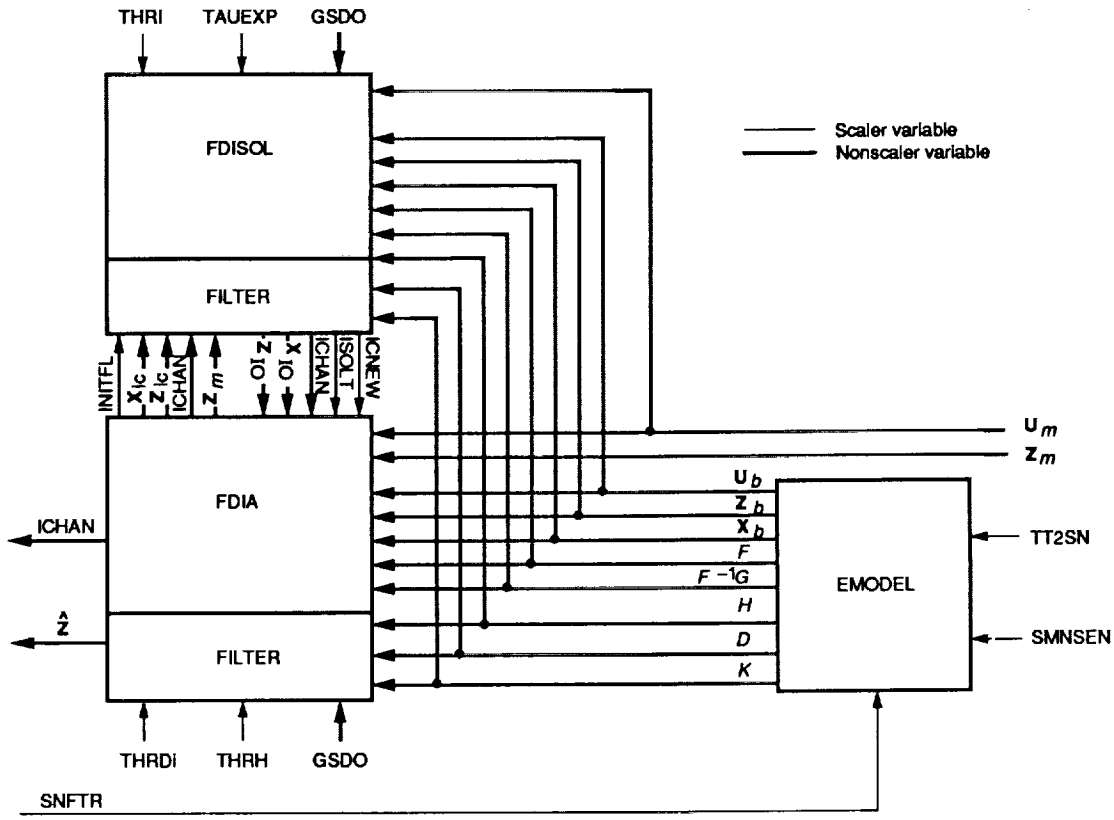


Figure 9.—Block diagram of ADIA software.

demonstrate it with a full-scale engine, implementation in real time was needed. One objective of the real-time implementation of the algorithm, then, was to respond to this need. The second objective was to use a realistic microcomputer, that is, hardware and software typical of that to be used in next generation turbofan engine controls.

The MVC–ADIA implementation has several distinct hardware and software features. The hardware for the controls computer is based on the Intel (Santa Clara, CA) 8086 processor architecture. Three CPU's operating in parallel allow the full MVC–ADIA algorithm to run in real time. The software is a combination of a previously developed 8086 implementation of the MVC algorithm (in its entirety) and the software to implement the ADIA algorithm. The ADIA software uses floating-point arithmetic and is coded almost entirely in the application language, FORTRAN. The FORTRAN subroutines have been optimized to execute in real time. Only the reference point schedules and table lookup routines within the ADIA algorithm remain in assembly language.

Algorithm Software

The ADIA algorithm is implemented by the four major software modules shown in figure 9: the engine model matrices calculation, EMODEL; the hard-failure detection and isolation logic and accommodation filter, FDIA; the soft-failure detection and isolation logic, FDISOL; and the filter calculation, FILTER, used by FDIA and FDISOL. Each of these modules is discussed in the following paragraphs.

The EMODEL software calculates the F, $F^{-1}G$, H, D, and K matrix elements as well as the **U**, **X**, and **Z** basepoints. The reason for computing $F^{-1}G$ instead of G is explained in the section describing FILTER. A block diagram of EMODEL is shown in figure 10. The inputs to the calculation are the transition control value of fan speed from the MVC (SNFTR), sensed fan inlet temperature (TT2SN), and sensed Mach number (SMNSEN). From these, the routine SNFMAP calculates a virtual power code PCV. The PCV is an indicator of the dynamic state of the engine and is equivalent to the PLA that would be required to cause the steady-state fan speed

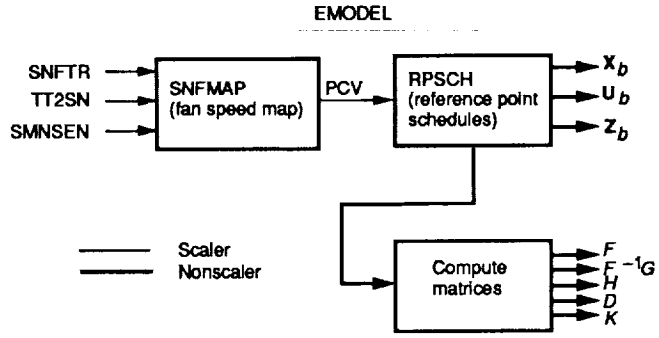


Figure 10.—Engine model calculation, EMODEL.

indicated by SNFTR, given TT2SN and SMNSEN. The PCV is an input to the reference point schedules (RPSCH), which are identical to those used in the F100 multivariable control except that PCV is substituted for PLA. The outputs of RPSCH are basepoint values for the ADIA filter states $\mathbf{X}_b$, engine outputs $\mathbf{Z}_b$, and the controlled engine inputs $\mathbf{U}_b$. All three sets of basepoints are necessary for the accommodation filter and hypothesis filter calculations. In addition, $\mathbf{Z}_b$ is used to compute the matrix elements for the F, $F^{-1}G$, H, D, and K matrices. Each of the matrix elements is either a constant or expressed as a polynomial of the $\mathbf{Z}$ basepoints $\mathbf{Z}_b$, of no greater than third order.

Figure 11 shows a block diagram of the FDIA module. The operation of this module, which contains the sensor heal logic, the hard-failure detection and isolation logic, and the accommodation filters, is described in the following paragraphs. The sensor heal logic allows sensors that have been declared bad because of spurious noise, or that have been failed during engine testing and then healed, to be brought back into consideration in computing the estimates of the engine outputs. The measured outputs $\mathbf{Z}_m$ of the engine are delayed so that the measurements from the previous update interval are used. The estimated sensor outputs from the last update interval, $\hat{\mathbf{Z}}_m$, are then subtracted from the measurements to get the residuals **GAMCH**. For each sensor channel that has been declared failed at some point in the past (i.e., its corresponding failure isolation flag **ICHAN** is set), the residual is compared to a heal threshold THRH multiplied by the standard deviation of that signal **GSDO** (see table II). If the residual is less than this product for 10 update intervals, the sensor is declared healed. The heal threshold is usually set to 10 percent of the hard-failure detection threshold. A sensor that is declared healed has its corresponding failure flag **ICHAN** reset to zero.

If a channel's failure flag is set, the corresponding residual, **GAM**, is set to zero; this guarantees that the channel will not be considered in the subsequent logic. In the hard-failure

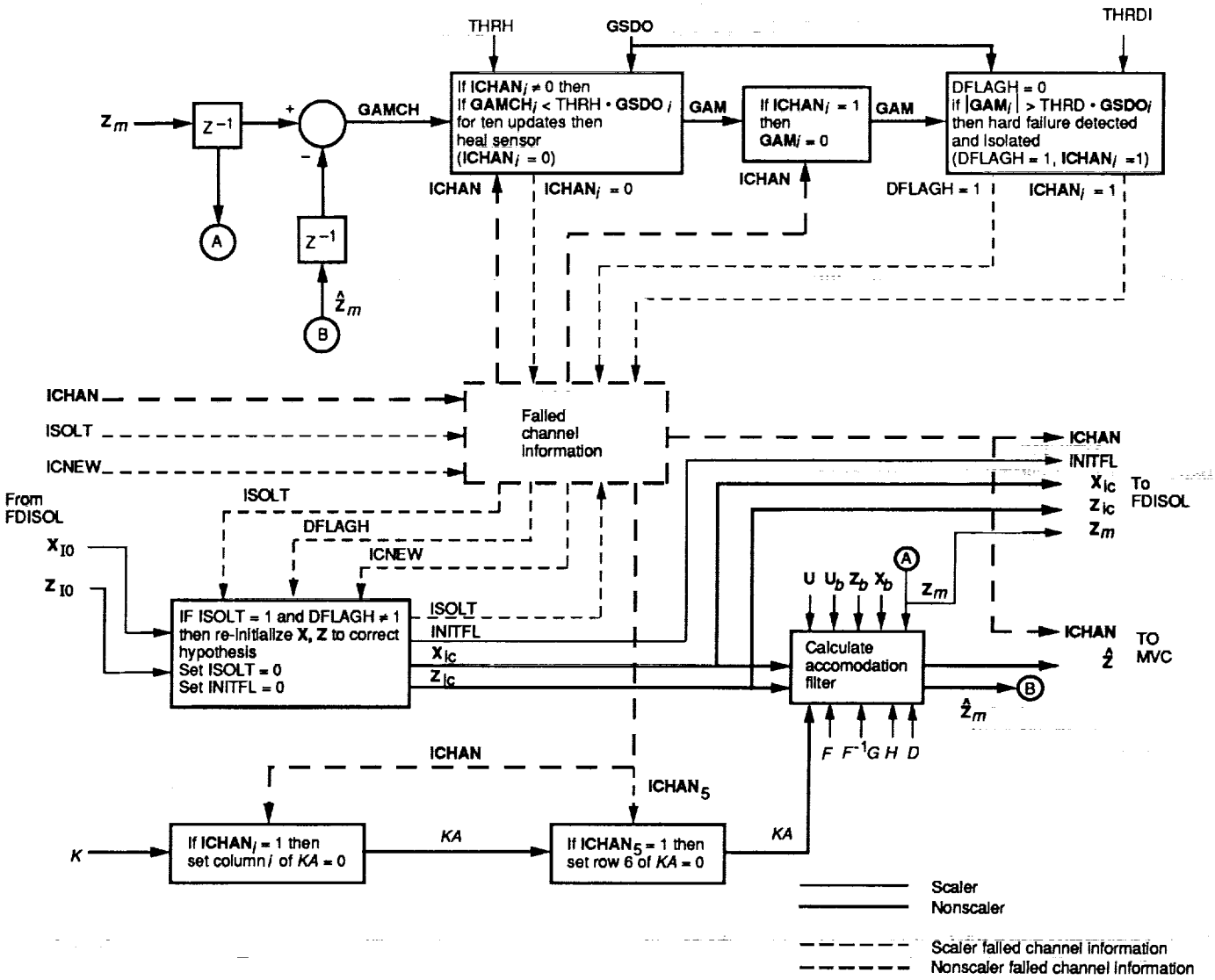


Figure 11.—Hard-failure detection and isolation logic and accommodation filter, FDIA.

detection and isolation logic, the residuals are compared to the product of the hard-failure detection and isolation threshold THRDI and the standard deviations **GSDO**. If the residuals are greater than this product, a hard failure is declared on the channel and the hard-failure detection flag DFLAGH is set along with the corresponding **ICHAN**. At this point all information concerning both the hard failures occurring in this update interval, and the soft failures occurring in the last update interval (flags **ICHAN**, ISOLT, and ICNEW to be explained later) are in the failed channel information.

The **ICHAN**'s are then examined, and for each **ICHAN** that is set (i.e., for each channel that has had a failure isolated), the corresponding column of the K-matrix is set to zero. This removes that measurement from the accommodation filter calculation. In addition, if $\mathbf{ICHAN}_5$ is set (i.e., an FTIT failure has occurred), then the entire sixth row of the K-matrix is set to zero. This removes the integrator state described earlier. If no hard failures have occurred in this update interval (i.e., DFLAGH = 0) but a soft-failure isolation has occurred (i.e., ISOLT = 1), the states and estimated outputs of the accommodation filter are reinitialized to the states and estimated outputs of the correct hypothesis filter $\mathbf{X}_{ic}$, $\mathbf{Z}_{ic}$, as indicated by the ICNEW flag from the soft-failure isolation logic. In addition, the soft-failure isolation initialization flag INITFL is set and passed to the soft-failure isolation logic, to cause all six of the hypothesis filters, as well, to be reinitialized to the correct hypothesis filter. Last, the accommodation filter is calculated by using the F, $F^{-1}G$, H, D, and modified K (KA) matrices along with the control, output, and state basepoints, $\mathbf{U}_b$, $\mathbf{Z}_b$, and $\mathbf{X}_b$ respectively—all of which were computed by EMODEL. The accommodation filter uses the delayed measurement $\mathbf{Z}_m$ to calculate the optimal estimates of the engine outputs $\hat{\mathbf{Z}}$ and the same estimates with sensor dynamics incorporated $\hat{\mathbf{Z}}_m$. The unlagged estimates $\hat{\mathbf{Z}}$ and the failure isolation flags **ICHAN** are passed to the MVC control so that the estimates can be used in the LQR and, if a failure has been isolated to a given channel, in the integral control.

The next major software module is the filter calculation FILTER, which calculates the accommodation filter and all six of the hypothesis filters. Figure 12 shows, in block diagram form, the filter calculation that corresponds to the filter equations given in the Accommodation Filter section earlier. However, the equations are actually computed slightly differently. Previously, the equations were given as

$$\dot{\hat{\mathbf{X}}} = F(\hat{\mathbf{X}} - \mathbf{X}_b) + G(\mathbf{U}_m - \mathbf{U}_b) + K\gamma \tag{3}$$

$$\hat{\mathbf{Z}} = H(\hat{\mathbf{X}} - \mathbf{X}_b) + D(\mathbf{U}_m - \mathbf{U}_b) + \mathbf{Z}_b \tag{4}$$

$$\gamma = \mathbf{Z}_m - \hat{\mathbf{Z}} \tag{5}$$

The filter is actually implemented as

$$\dot{\hat{\mathbf{X}}} = F[(\hat{\mathbf{X}} - \mathbf{X}_b) + F^{-1}G(\mathbf{U}_m - \mathbf{U}_b)] + K\gamma \tag{12}$$

$$\hat{\mathbf{Z}} = H(\hat{\mathbf{X}} - \mathbf{X}_b) + D(\mathbf{U}_m - \mathbf{U}_b) + \mathbf{Z}_b \tag{4}$$

$$\gamma = \mathbf{Z}_m - \hat{\mathbf{Z}} \tag{5}$$

An $(F^{-1}G)$-matrix in place of a G-matrix permits isolation of the steady-state and dynamic components of the states. (For further details see ref. 3.) Two "switches" that allow the filter to be initialized are included in this module; that is, the states and the estimates can be set to a specified initial condition.

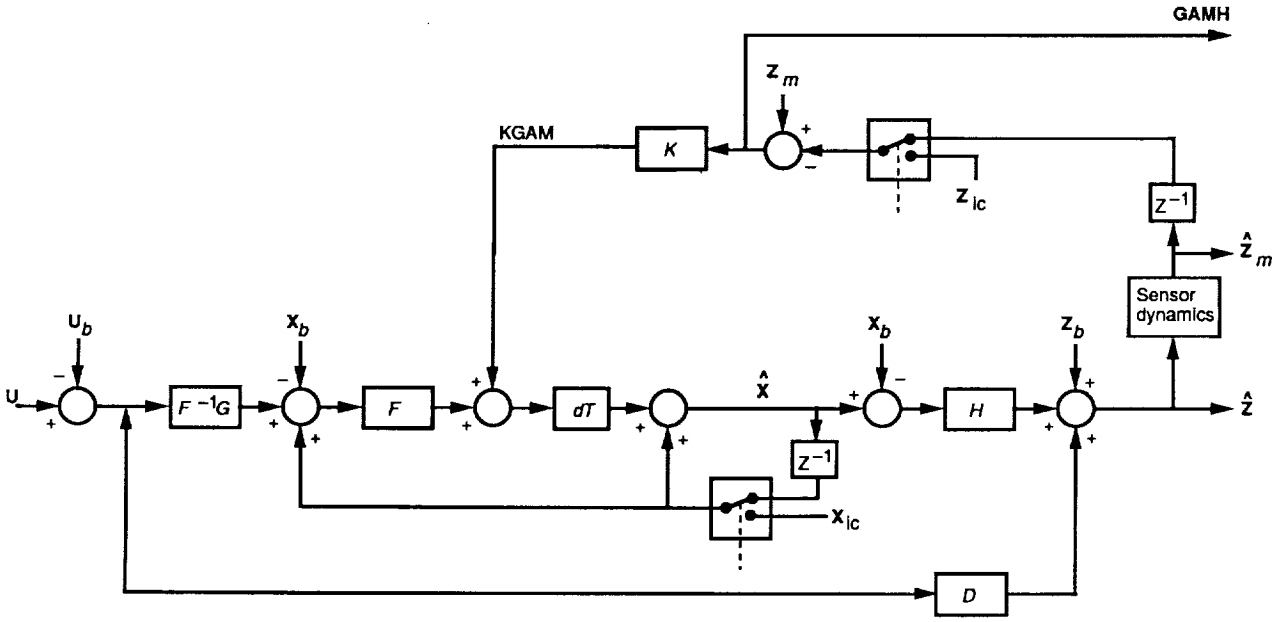


Figure 12.—Kalman filter calculation, FILTER.

Initialization is required at startup and when a soft failure is isolated, whereupon the states and estimates need to be set to the correct hypothesis filter. The optimal estimates of the engine outputs $\hat{\mathbf{Z}}$ are shown (fig. 12) as one of the three outputs of the filter calculation. These optimal estimates are then filtered to simulate the sensor dynamics. For the two speeds and two pressures, N1, N2, PT4, and PT6, the latter filters are simple first-order lags. For FTIT, the filter is a combination of two first-order lags, one fast and one slow, which emulates the FTIT thermocouple. The estimates $\hat{\mathbf{Z}}_m$ of the sensor outputs are used in the FDIA sensor heal and hard-failure-detection logics. The last outputs of the filter calculation, the computed residuals GAMH, are used in the soft-failure isolation logic.

The soft-failure detection and isolation logic, FDISOL, is shown in figure 13. As explained earlier, there are six isolation or hypothesis filters. Each of these has a unique set of states, estimates, and residuals and a unique K-matrix. The K-matrix for the hypothesis zero filter is identical to that computed by EMODEL. However for each of the remaining hypothesis filters, the column of the K-matrix corresponding to the filter number is set to zero; that is, for hypothesis filter 1, column 1 is set to zero, for hypothesis filter 2, column 2 is set to zero, and so forth. These modified K-matrices, $K_{I,1}$ through $K_{I,5}$, are used in their respective hypothesis filter calculations. If the initialization flag INITFL is set (i.e., the accommodation filter has accommodated a new soft failure), then the states and estimates of each hypothesis filter are set to the states and estimates of the correct hypothesis filter. In addition, all the hypothesis filter outputs (HI_{00} to HI_5) are reset to zero, and the soft-failure isolation flag ISOLT and INITFL are reset.

For each hypothesis filter in which a failure has been isolated to a given channel (i.e., that channel's corresponding **ICHAN** is set), the measurement for that channel is set to the filter estimate. Since the residual on that channel is $\gamma = \mathbf{Z}_m - \hat{\mathbf{Z}}$, setting $\mathbf{Z}_m$ to the filter estimate sets the residual corresponding to a failed channel to zero in each hypothesis filter; this has the effect of removing that channel from the hypothesis-generation procedure. The six hypothesis filters are then calculated by using the FILTER calculation described previously. The output of interest from the filter is the residual vector. The six residual vectors $\mathbf{GAMH}_0$ to $\mathbf{GAMH}_5$ are used along with the standard deviations **GSDO** to compute six WSSR's (as discussed earlier). The WSSR's are smoothed by using a simple first-order lag with a time constant of TAUEXP = 0.1 sec. The five smoothed failure-case WSSR's,

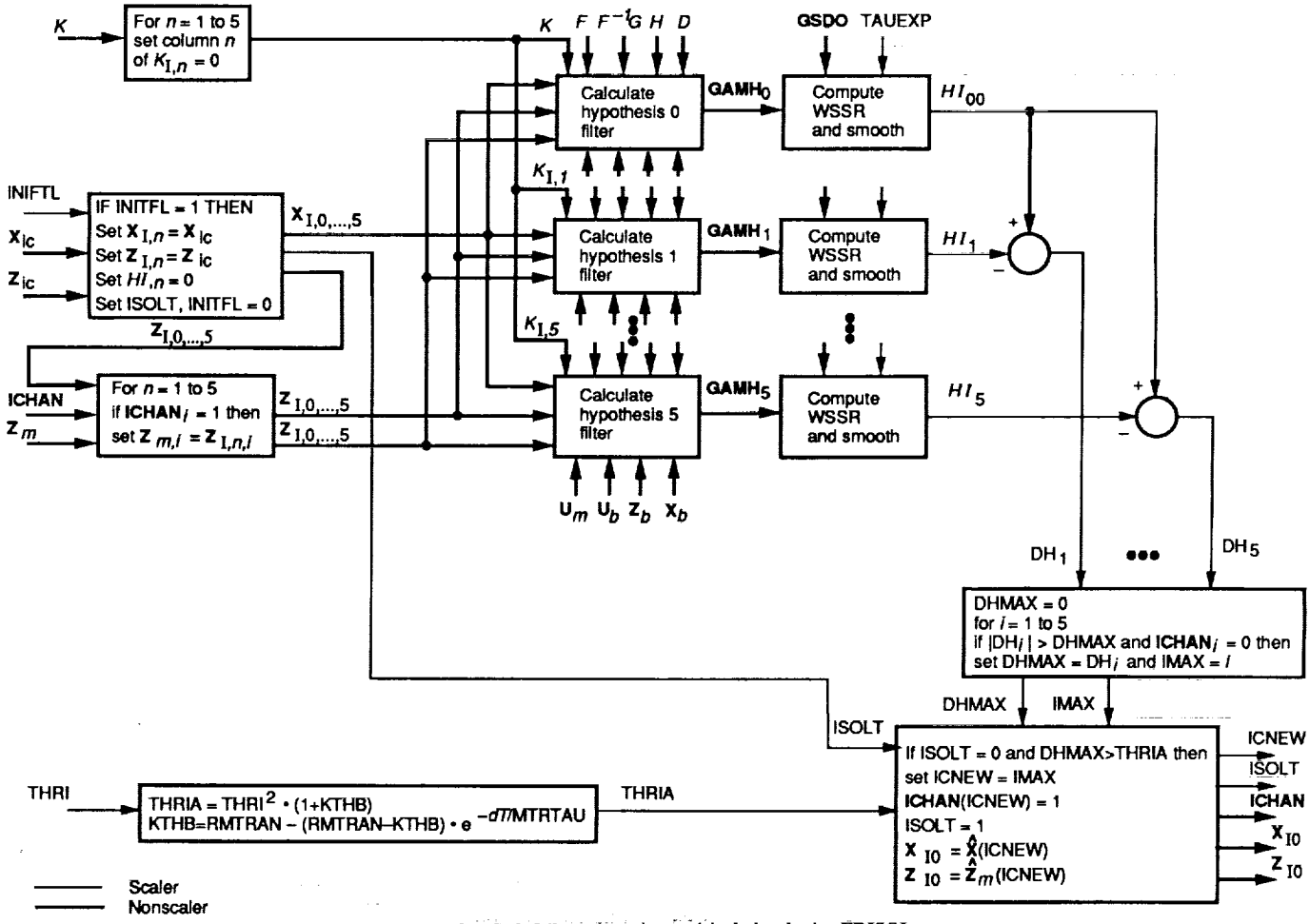


Figure 13.—Soft-failure detection and isolation logic, FDISOL.

HI_1 to HI_5, are subtracted from the unfailed one, HI_{00}, to give the likelihood ratio for each failure case DH_1 to DH_5, where DH_1 corresponds to the likelihood of an N1 failure, DH_2 corresponds to the likelihood of an N2 failure, and so on.

For channels that have not had a previous failure isolated (their **ICHAN**'s are not set), the maximum likelihood ratio is found; DHMAX then contains the maximum likelihood ratio, and IMAX contains the channel number corresponding to that maximum likelihood ratio. The adaptive soft-failure isolation threshold THRIA is computed as discussed earlier. If ISOLT = 0 (i.e., all previous soft failures have been accommodated), then DHMAX is compared to THRIA, and if DHMAX is greater, a soft-failure is detected and isolated. In such a case, ICNEW is set to the channel number of the failed channel, the corresponding failure isolation flag **ICHAN** is set, and the soft-failure isolation flag ISOLT is set. These three variables along with the state and sensor estimates of the correct hypothesis are then passed to FDIA to accommodate the failure.

Controls Microcomputer Hardware and Software Design

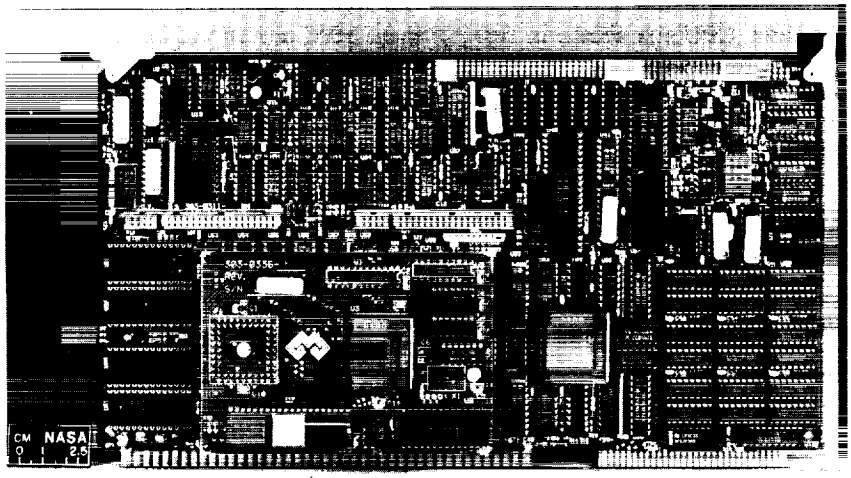
Implementing the MVC–ADIA algorithm required integrating the ADIA algorithm with the existing microcomputer implementation of the F100 MVC algorithm. The update interval of the microprocesso-based MVC implementation was 22 msec. The F100 engine system dynamics required that the combined MVC–ADIA algorithm update interval be 40 msec or less.

The microcomputer implementation of the MVC algorithm had been developed by porting the minicomputer implementation of the MVC algorithm that was used for the F100 MVC program to an Intel 8086 microprocessor-based controls microcomputer. The ADIA algorithm was then merged with this MVC implementation to give a full microcomputer implementation of the control algorithm with sensor analytical redundancy. The controls microcomputer, although still based on the Intel 8086 microprocessor architecture, used multiple processors operating in parallel to be within the update interval of 40 msec that was necessary for stable engine operation.

Initially, only the normal-mode accommodation filter and the hard-failure detection and isolation logic of the ADIA algorithm were added to the MVC algorithm. For this initial configuration a second 8086-based CPU, running in parallel, was added to the CPU that was used solely to implement the MVC. Data were transferred between the Intel 86/30 single-board computers through dual-ported memory, and synchronization between CPU's was achieved through interrupts. The software for the combined MVC–ADIA algorithm was alloted so that the ADIA software ran on the second CPU while the MVC algorithm remained intact on the first CPU. This straightforward way to partition the algorithm allowed the parallel-processing mechanism to be evaluated. The soft-failure detection and isolation logic was to be added to the second CPU at a later date.

In the original design of the ADIA algorithm, the soft-failure isolation logic would start only after a soft failure was detected by the soft-failure detection logic. The soft-failure detection logic was subsequently added to the second CPU, but because the soft-failure isolation logic required a significant amount of processing time, a third CPU was added to implement it in parallel with the soft-failure detection logic. Data were transferred and synchronized in the same manner as with the two-CPU implementation. Recently, the three 8086-based CPU's were replaced with three 80186-based CPU's. These are Monolithic Systems (Englewood, CO) MSC 8186 single-board computers (see fig. 14). The features of these single-board computers are shown in table III.

The new CPU's are software-compatible with the old CPU's but are considerably faster. Each of the new CPU's provides approximately 0.7 million instructions per second (MIPS), so the three processors combined provide on the order of 2 MIPS. The three CPU's are contained in an 18-slot Multibus chassis that also contains a floppy disk controller, a graphics interface, and both analog and digital input/output boards. (The hardware



C-86-2889

Figure 14.—Monolithic Systems 8186 single board computer.

TABLE III.—MSC 8186 FEATURES

[Hardware and software compatible with Intel iSBC 86/30.]

(a) General features

Dynamic RAM, dual-ported, 150 nsec zero-wait-state, kB	128
EPROM, kB	256
Programmable timers/counters	5
Levels of vectored interrupt control	13
Compatible serial interface	RS-232
Compatible bus interface	Multibus (IEEE 796)

(b) Individual processor features

	8-MHz 80186 microprocessor[a]	8-MHz 8087 numerics coprocessor[b]
Registers, 16-bit, number		
General purpose	8	----------
Segment	4	----------
Status and control	2	----------
Register stack, 80-bit, number	----------	8
Arithmetic	Signed, fixed point	Signed, floating point
Processing time, μsec		
Addition	1.25	15.6
Multiplication	4.5	21
Division	7.6	28.7

[a]Has integrated peripherals including three timers, two DMA controllers, and a programmable interrupt controller.

[b]Compatible with IEEE floating point standard 754.

configuration is shown in fig. 15.) The disk controller allows programs to be loaded from disk into each CPU's memory and permits research data to be saved to disk. The graphics interface allows research data to be displayed on a graphics terminal. The analog input/output boards provide an interface to the engine or engine simulation. The digital input/output boards not only provide an interface to switches on the CIM unit front panel but also allow discrete controller inputs and outputs.

The relative timing for the three CPU's is shown in figure 16. The arrows in the figure represent interrupts. The first event to occur is a timer interrupt to CPU 1; this indicates the beginning of an update interval. The first CPU then samples all the algorithm inputs through the analog-to-digital (A/D) converters and all the mode switches through the digital input boards. The measurements required for the ADIA part of the algorithm on CPU's 2 and 3 are then converted by CPU 1 to floating-point numbers and transferred to CPU 2. An interrupt is sent from CPU 1 to CPU 2 to indicate that the algorithm inputs are now available. Next, CPU 1 computes the parts of the MVC algorithm that are not dependent on the outputs of the ADIA algorithm—the reference point schedules, the transition control, and the gain control. Concurrently, CPU 2 uses the algorithm inputs to compute the engine model matrices and basepoints and the Kalman gain matrix. This information is passed to CPU 3 for use in the soft-failure isolation logic. Then CPU 2 sends an interrupt to CPU 3 to indicate the information is available. The soft-failure isolation logic computes throughout the remainder of the current update interval and into the next. Any soft-isolation information that results is transferred from CPU 3 to CPU 2, which performs the hard-failure detection and accommodation filter computations, using the isolation information if needed. An interrupt is sent from CPU 2 to CPU 1 indicating the ADIA calculations are complete. Next, CPU 1 reads the resulting ADIA outputs (i.e., information about any sensors that have failed and the computed estimates of the engine outputs) from CPU 2 and then finishes the MVC calculations. The controlled variables are sent to the engine actuators through the digital-to-analog (D/A) converters. In the last step, CPU 2 calculates altitude and Mach number from the engine environmental variables, for use during the next update interval.

Data Acquisition Software

The microcontroller interactive data system (MINDS) is used for data acquisition (ref. 18). This software runs on CPU 1 during the time when the CPU is not executing the MVC algorithm (fig. 16). The MINDS package has both steady-state and transient data-taking capabilities and can access any variable in the MVC or ADIA algorithms. Variables are defined by name, memory location, and for integer variables, their scale factors.

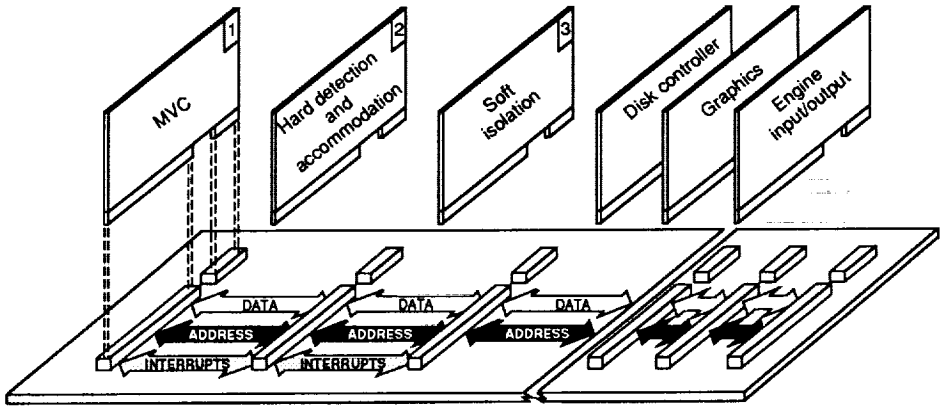
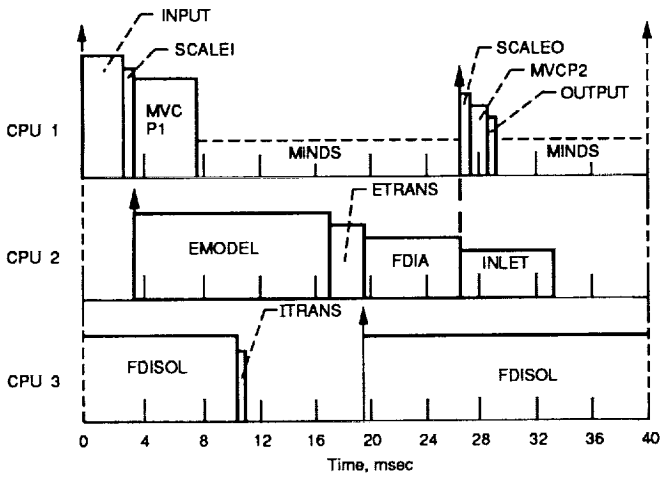


Figure 15.—Hardware implementation of ADIA.



CPU 1		
	INPUT	A/D conversion and scaling of engine measurements
	SCALEI	Conversion of measurements to floating point and transfer to CPU 2
	MVCP1	MVC part 1: reference point schedule, transition control, and gain schedule calculations
	MINDS	Interactive data system (spare time calculation)
	SCALEO	Transfer of estimates and reconfiguration information from CPU 2 and conversion to fixed point
	MVCP2	MVC part 2: proportional control, integral control, and engine protection logic
	OUTPUT	Unscaling and D/A conversion of engine inputs
CPU 2		
	EMODEL	Engine model matrix and base point calculation
	ETRANS	Transfer of EMODEL information to CPU 3
	FDIA	Accommodation filter and hard detection and isolation logic calculations
	INLET	Mach number and altitude calculation
CPU 3		
	FDISOL	Hypothesis filters and soft detection and isolation logic calculations
	ITRANS	Transfer of soft isolation information from CPU 3

Figure 16.—Advanced detection, isolation, and accommodation timing for 8-MHz MSC 8186. Arrows represent interrupts.

Large amounts of steady-state data can be obtained by grouping the variable names into tables and requesting a printout of a given table. All the variables within a table are sampled and printed on the CIM unit user console. Transient data are collected by defining tables of variables to be sampled over time. The variable definitions, as well as the steady-state and transient table definitions, are saved to disk so that they may be recalled each time MINDS is run. The transient data sampling interval and the total sampling time are defined at run time. The steady-state and/or transient data taken can be uplinked to a mainframe computer for off-line processing. In addition, the software has been enhanced to allow plotting of transient data on-line, immediately following a transient data sample, while the controls microcomputer continues to operate. The on-line transient data display of internal MVC and ADIA variables was an indispensable tool in the algorithm evaluation process and in resolving operational difficulties during the engine test.

Implementation Languages

As previously discussed, different parts of the combined MVC–ADIA algorithm are divided among three CPU's. The MVC is implemented in fixed-point assembly language on CPU 1. When the MVC was originally implemented on a microcomputer (3 yr prior to the ADIA implementation), assembly language programming with fixed-point arithmetic was necessary to achieve real-time execution of the algorithm. With the development of efficient floating-point coprocessing hardware—in this case the Intel 8087—came the capability of implementing real-time controls in floating-point arithmetic. The advantages of programming in floating-point arithmetic and an application language such as FORTRAN, rather than in fixed-point assembly language as was used for the MVC, include increased software reliability and reduced software development and maintenance costs. Thus, most of the ADIA algorithm running on CPU's 2 and 3 is programmed in floating-point arithmetic and FORTRAN.

The primary disadvantage to using an application language is that it generally produces a less efficient object code than the equivalent functions programmed in assembly language. Execution efficiency is critical for real-time control systems such as the MVC–ADIA. Programmed entirely in FORTRAN and as originally coded, the ADIA algorithm took more than an order of magnitude longer than the maximum 40-msec update interval. To hasten execution, table lookup routines, which are written to take advantage of the 8087 architecture (ref. 19) and are executed frequently in the ADIA algorithm, were implemented in assembly language, as were the hardware interface routines, which have no FORTRAN equivalent. The EMODEL schedules for computing the filter basepoints are functionally identical to the reference point schedules in the MVC, thus using the MVC assembly language schedules saves additional computing time. So that the remainder of the algorithm could remain in FORTRAN, the source code was optimized to make it run more efficiently (ref. 20). As shown in figure 16, the entire MVC–ADIA algorithm now executes in less than the maximum 40 msec.

Memory Requirements

The memory requirements for each of the three CPU's are shown in figure 17. Each CPU has, in addition to its share of

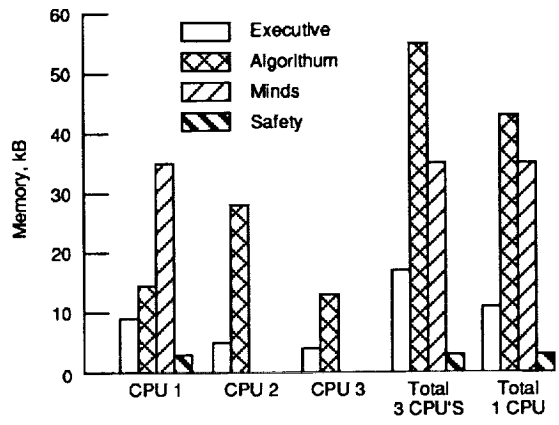


Figure 17.—Memory requirements of MVC-ADIA.

the MVC–ADIA algorithm, an executive routine that maintains correct real-time operation of the total algorithm. The amounts of memory required for the algorithm and for the executive routine are shown for each CPU. Also the memory requirements for MINDS on CPU 1 and for the safety software (which is described in a later section of this report and which runs only on CPU 1) are shown. In all cases the code and the constants occupy about 75 percent, and the data and the variables about 25 percent, of the total memory required. Figure 17 also shows the total number of kilobytes (kB) of memory required for all three CPU's combined for each of the following: the executive routines (16.9 kB), the total algorithm (54 kB), MINDS (34.4 kB),and the safety software (2.5 kB). A state-of-the-art 32-bit microprocessor is probably capable of real-time execution of the MVC and ADIA algorithms on a single CPU. If so, figure 17 shows what the total memory requirement would be for such an MVC–ADIA implementation. Combining the algorithms would eliminate some redundant code in the executive routines and in that part of the algorithm which was replicated because of being distributed across multiple CPU's. About 5 kB of executive and 15 kB of algorithm are replicated in the multiple CPU implementation; thus a single CPU implementation would take about 20 kB less memory.

Software Run-Time Environment

The programs for each of the CPU's are downloaded into memory by using a commercially available disk operating system, CP/M–86. The operating system allows research data to be saved to disk and provides terminal interfaces to both the CIM unit user console and graphics display.

Software Development Environment

Software development for the MVC–ADIA code was done on an Intel Series III Microprocessor Development System. Each routine was either assembled or compiled on this system. The software for each of the CPU's was linked and located in order to generate the executable object modules. The executable modules were transferred to the CIM unit by converting them to ASCII files and copying them to a CP/M-86 formatted file on a floppy disk. This floppy disk was then transferred to the CIM unit.

On the CIM unit, a CP/M-86 executable file was generated from the Intel executable object modules for each of the three CPU's by using the CP/M-86 utility GENCMD. The routines for CPU's 2 and 3 were then loaded from disk into RAM on boards 2 and 3 by using the operating system. Lastly, the software for CPU 1 was loaded from disk and executed. The result was the multiprocessor operation described earlier in this report.

In order to ensure software version control, any changes to be made to the software were first documented in the software listings, and then the module to be changed was identified. As each change was made, a record of it was kept in a software configuration control document. Finally, a new version number for each of the individual modules changed was incorporated into a new master version list. The software configuration control document, the master version list, and a narrative of the changes were given to the test conductor for incorporation into the engine-test permanent records.

Engine Demonstration-Specific Hardware and Software

This section summarizes the CIM unit safety and operational procedures devised for the ADIA engine demonstration in the PSL. A primary concern of the research staff was the safety of the engine while it was operating under research control (MVC–ADIA). Any event that could compromise the safety of the engine had to be detected, and appropriate action taken. Procedures to safely start the research control and to transition smoothly and safely from backup or bill-of-material control (BOM) to research control and back again were defined for operation of the CIM unit during the engine test. Failure modes within the CIM unit were identified, and safety procedures to avoid compromising the safety of the engine were defined.

Operating Procedures

To allow for startup and a smooth transition from backup control to research control and back again, the controls microcomputer contains four operational modes: (1) startup, (2) initialization, (3) run, and (4) abort.

The executive software in the controls microcomputer maintains real-time operation of the algorithm software and contains the logic to switch between these operational modes. Each of the modes is described herein.

Startup mode.—During a typical run, the CIM unit was powered up and placed in the startup mode. In this mode, the full MVC–ADIA algorithm was run with synthesized inputs generated internally by software. This mode allowed any obvious faults within the computer to be identified before an attempt was made to run it with actual sensed values. With the CIM unit in startup mode, the engine was started, and the altitude cell was adjusted to obtain the correct values of P0, PT2, and TT2 for the initial flight point.

Initialization mode.—If no problems were detected in the startup mode with the engine at flight idle under BOM control, the CIM unit was put in the initialization mode. In this mode the control was run using actual sensed information from the engine. All five integrator outputs within the MVC algorithm integral control were fixed at zero. The MVC outputs were compared with the sensed actuator feedbacks. If all of the outputs were within a specific tolerance of the feedback signal, the MVC integral control output initial conditions were set so that the MVC actuator commands were exactly the same as the corresponding sensed actuator positions; this ensured a smooth transfer from BOM to research control. If no safety problems were detected, the control was put into the run mode.

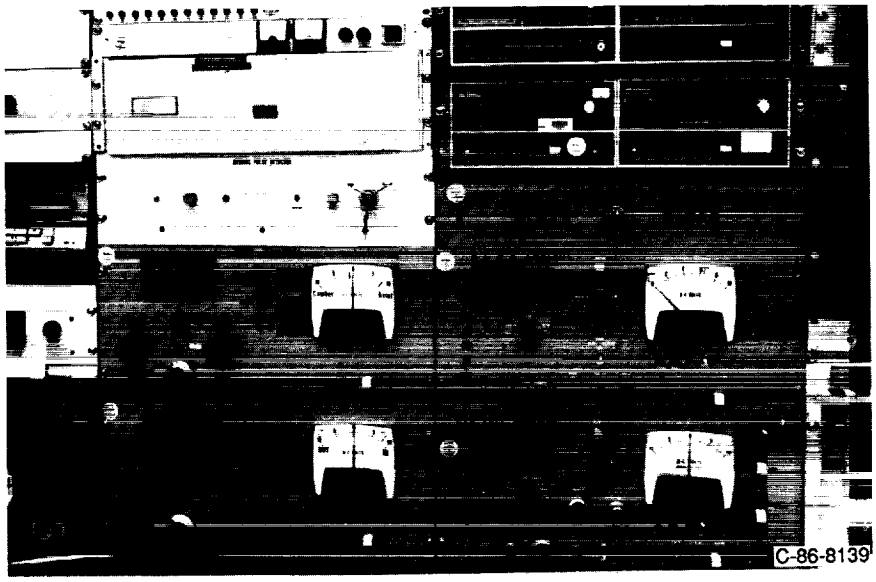


Figure 18.—Actuator panels and missing-pulse detector.

Run mode.—In the run mode, a permit light on the CIM unit front panel indicated that actuator panels could be switched from the BOM control to the research control. These panels, shown in figure 18, provided the interface between the CIM unit and the engine actuators and provided the switching between research and BOM controls. (Additional details on the panels and research actuation system can be found in ref. 17.) To transfer to research control, the actuator panel switches (one for each actuator) were manually engaged, one at a time, by the panel operator. This caused the actuator commands to be derived from the MVC–ADIA logic instead of the backup control. Since the MVC–ADIA actuator commands were set equal to the actuator feedbacks during the initialization mode, this transfer of control was very smooth. Once transfer to research control was complete, the MVC integrators were released, the CIM unit safety software was engaged, and full control of the engine was assumed by the MVC–ADIA.

Abort mode.—If the CIM unit safety software (to be discussed next) detected a problem, the control switched from the run mode to the abort mode. In the abort mode, the control software was frozen, thereby causing the PSL safety logic to command the actuator panels to switch the engine to the backup control. In addition, freezing the software allowed inspection of the control inputs, internal variables, and outputs at the moment of problem detection, in order to help determine the cause of the abort. An abort could also be initiated by the engine operator or actuator panel operator and detected by the CIM unit through a sense line. At shutdown, the panel operator issued an abort to switch control of the engine to backup, where it remained throughout the shutdown procedure.

Safety Procedures

The normal PSL safety systems were used to ensure safe engine operation. These included maximum speed and maximum temperature protection, chop-to-idle or off logic, and master fuel shutoff logic. However, certain failures, which compromise engine safety, can occur during operation of the engine on research control. When these failures were detected, as a safety measure, control reverted to the backup until the cause of the problem could be identified. The following failure modes cause adverse behavior of the MVC–ADIA control:

(1) Failure on input of sensors, sensor lines, and/or CIM unit A/D converters
(2) Failure on output of CIM unit D/A converters, output lines, and/or actuators
(3) Failure of controls microcomputer hardware
(4) Failure of controls microcomputer software

Safety systems were designed to accommodate each of these failure modes. The failure modes and the action taken to accommodate them are summarized in table IV. Once a sensor, actuator system, or output failure has been detected, the controls microcomputer is automatically frozen. This causes a reversion to backup control for any anticipated failure modes and thus minimizes risk to the engine. Specific safety procedures for sensor system failures, actuator system failure, and controls computer hardware and software failures are presented in the following paragraphs.

Sensor and actuator system failures.—For the preliminary engine runs in which the MVC was run without the ADIA sensor failure logic, sensor failure checks determined the health of all the sensors. In addition, actuator failure checks were used to check the integrity of the CIM unit outputs, output lines, actuator panels, and actuator hardware. The following paragraphs describe the sensor and actuator failure checks.

Input failure logic: The purpose of the input failure logic is to determine the validity of a particular sensor signal. This is done by using a subset of four possible checks on each signal that is fed to the control. The first check is a minimum-

TABLE IV.–CIM UNIT SAFETY SYSTEMS

Failure component	Detection method	Action taken
Sensor systems		
Engine sensors (without ADIA)	Sensor check software	Reverts to backup
Engine sensors (with ADIA)	ADIA algorithm	ADIA failure accommodation
Environmental sensors	Sensor check software	Reverts to backup
A/D converters	Sensor check software	Reverts to backup
Cabling	Sensor check software	Reverts to backup
CIM unit controls computer		
Hardware	Abort logic (watchdog timer on each CPU)	Reverts to backup
Software	Abort logic	Reverts to backup
	Output rate checks	Reverts to backup
	Overflow checks	Reverts to backup
Research actuation system		
Research actuators	Simulate actuators on simplified engine simulator	Reverts to backup
Actuator feedback sensors D/A converters Cabling	Error check on difference between simulated and actual actuator feedbacks	Reverts to backup

maximum check:

$$S_{\min} \leq S_n \leq S_{\max}$$

where S_n is the current value of a sensed signal, $S_{\min}$ is the minimum value of a sensed signal, and $S_{\max}$ is the maximum value of a sensed signal. This check allows the detection of a gross, hard, out-of-range failure that would be erroneous anywhere within the engine's flight envelope.

The second check is a rate check:

$$| S_n - S_{n-1} | \leq e$$

where S_{n-1} is the value of the sensed signal for the last update interval and e is the error tolerance within which the difference must lie.

By comparing the present and past values of the signal, an erratically responding sensor can be detected, and a hard failure anticipated.

The third check is a percent-of-point deviation check:

$$S - (N\text{-percent } S) \leq S_n \leq S + (N\text{-percent } S)$$

where $S \pm (N\text{-percent } S)$ is the nominal sensor value plus or minus a given percentage. This third check is basically a minimum-maximum check with the error bounds defined as being plus or minus a given percentage of the original sensed value. This check is effective only on variables that are not expected to change at a given flight condition. However, since the error bounds are considerably narrower than for the minimum-maximum check, a hard-failure or slow-drift condition can be detected much more quickly.

The fourth check is a reference-point deviation check:

$$| S_n - S_{\text{ref},n-1} | \leq e$$

where $S_{\text{ref},n-1}$ is the modeled value of the sensor from the previous update interval. This check uses the control's reference point schedules and transition control to produce steady-state and transient signal models that can be compared with the actual sensed value. This error check detects not only hard failures and failures caused by erratically responding sensors but also drift or sensors whose dynamic responses may be incorrect.

The actuator failure check is based on the difference between a sensed actuator feedback value A and a simulated one A_{ES} as follows:

$$| A - A_{\text{ES}} | \leq e$$

The modeled values are computed in the F100 simplified engine simulator from simulations of the fuel flow, exhaust nozzle, RCVV, and CIVV actuators. This check allows detection of errors in the forward loop, such as D/A converter failures, transmission line failures, actuator panel failures, and actuator hardware failures, before they are detected indirectly through a sensor and do inadvertent damage to the engine.

Discussion of sensor and actuator failure checks: Table V shows all of the sensed signals, including actuator feedbacks, grouped according to the types of failure checks applicable to each, along with their corresponding error tolerances and

TABLE V.–SENSOR AND ACTUATOR FAILURE CHECKS

[Values are percentage of full scale.]

Sensor	Minimum-maximum		Delta	Percent-of-point deviation	Reference-point deviation		Actuator model	
	Minimum	Maximum			Steady state	Transient	Steady state	Transient
PT2	1.30	75.00	-----	±20.00	-----	-----	-----	-----
TT2	−2.60	43.00	-----	±2.00	-----	-----	-----	-----
P0	.25	70.00	-----	±20.00	-----	-----	-----	-----
N1	20.00	77.00	-----	-------	6.00	13.30	-----	-----
N2	57.00	93.00	-----	-------	5.30	7.30	-----	-----
PT6	1.00	65.00	-----	-------	15.00	25.00	-----	-----
PT4	3.30	67.00	-----	-------	8.20	14.60	-----	-----
WF	.60	82.00	-----	-------	9.00	21.00	9.00	12.00
FTIT	17.00	60.00	8.30	-------	-----	-----	-----	-----
TT25	4.50	55.00	12.00	-------	-----	-----	-----	-----
PLA	6.60	66.60	14.60	-------	-----	-----	-----	-----
RCVV	0	100	-----	-------	-----	-----	18.00	27.00
CIVV	0	100	-----	-------	-----	-----	24.00	36.00
AJ	0	100	-----	-------	-----	-----	4.20	9.80

limit values. Note that a minimum-maximum failure check is performed on each signal to provide an initial failure screen for a hard failure. In addition, group-specific checks are made to provide additional coverage.

The percent-of-point deviation checks are made on the test cell conditions (i.e., PT2, TT2, and P0) since they will vary only a small percentage at any given flight condition. Clearly, if this type of system were being used in a flight environment, independent checks of this type (i.e., very tight minimum-maximum limits) could be made by using Mach number and altitude information from the airplane's central air-data computer.

The reference-point deviation checks are performed on N1, N2, PT6, WF, and PT4. These signals are all modeled by the control's reference-point schedules and transition logic to provide open-loop trajectories for the LQR. Therefore, in essence, the magnitude of LQR state deviations is being examined at all times to determine if a sensor failure exists.

The rate checks are used to anticipate failure of sensors whose signals have no modeled values and no attributes that would allow limitation of their valid range. Failure of any one of these signals may be catastrophic. In the event of a PLA failure, the engine would be unable to hold a reference point. Failure of FTIT could result in an overtemperature, especially if the engine was run at military power. A TT25 failure could result in an engine stall due to compressor geometry mis-scheduling. However, since both FTIT and TT25 are thermocouples, they would most likely fail in a hard-failure mode, which is easy to detect.

The actuator model checks are performed on the WF, AJ, CIVV, and RCVV feedback signals.

For all of the failure checks discussed, a signal must fail four consecutive times to be declared bad. Each signal has 10-Hz analog filters to provide extra protection against spurious noise that might inadvertently abort the control.

The error tolerance and limit values for all of the failure checks that are summarized in table V were obtained from the MVCS engine tests. They were originally derived from the MVCS hybrid simulation evaluation results. The reference-point deviation checks and the actuator model checks have two sets of error tolerances: one for steady state and one for transient. This is desirable since most of the test time is spent at steady state, and the dynamic models are not as good as the ones for steady state. Thus, to provide rapid detection of most failures, tight error tolerances are used in steady state. These error tolerances are increased during a transient to prevent detection of false failures.

For the later runs of the engine, the ADIA logic was engaged and the sensor failure checks on N1, N2, PT4, PT6, and FTIT were performed entirely with the ADIA algorithm. As a result, the sensor failure checks from the MVCS program were no longer needed for these channels; however, all other sensor and actuator failure checks remained in effect.

Controls computer hardware failures.—The controls microcomputer in the CIM unit contains three CPU's. Since all three are used for computation of the MVC-ADIA algorithm, a hardware failure of any one of these CPU's would adversely affect operation of the research control. For this reason, monitoring the health of each of the CPU's is desirable. Each CPU generates a signal indicating that it is operating correctly. These signals are sent from the CIM unit as modified square waves or pulse trains so that a simple monostable circuit in the PSL safety systems can detect if the signal is absent. An abort logic circuit, or missing-pulse detector (fig. 18) was designed, built, and placed in the PSL control room for this

purpose. Thus a hardware failure or power loss on any one of the CPU's can be detected. This information is OR'd with the engine operator's abort signal and passed to the permit input on the actuator panels; this procedure allows reversion to backup to be initiated by either a CIM unit failure or by the operator. In addition, the OR'd output of the abort logic is monitored by the CIM unit so that the MVC–ADIA is frozen by any abort situation.

Controls computer software failures.—The system just described will also detect gross errors in the MVC–ADIA software if those errors cause one or more of the CPU's to stop generating their health signal. However, if the CPU's continue to function, checks are needed to ensure that the CIM unit is computing the MVC–ADIA algorithm correctly and is producing correct outputs. The controls computer software screens for software execution errors, including integer and floating-point overflows and divide by zero. Any of these execution errors will cause reversion to backup control. The computer software also does output checks to prevent erratic control outputs from being sent to the actuators.

Output processing and failure checking scales the control outputs to make them compatible with the inputs to the research actuators, and it performs the following check on WF, AJ, CIVV and RCVV:

$$| O_n - O_{n-1} | \leq e$$

where O_n is the current value of control output; O_{n-1} is the past value of control output; and e is the error tolerance within which the difference must lie.

This check allows one last test of the control's health by ascertaining that the outputs are not behaving erratically. Erratic behavior could be caused by an undetected overflow in an arithmetic or shift operation or, possibly, by an actual hard failure of the computer arithmetic unit. The error tolerances for the output check were derived by analyzing data from the algorithm evaluation on the hybrid simulation. The fuel flow and nozzle area control outputs may fail this check only once to be considered a failure. A failure at this point would indicate a possible catastrophic computer problem, so reverting to backup control should be done as quickly as possible. Because of the control's filtering action, noise on the input sensors should not cause noise on the fuel flow and nozzle area control outputs. However, for the RCVV and CIVV outputs, noise is a problem owing to the pass-through nature of the schedules for these outputs. For instance, the noise on the inputs to the RCVV schedule (TT25 and N2) could cause noise on the RCVV control output. Thus the RCVV and CIVV control outputs must fail this test three times in order to be considered a failure.

Engine Test Facility Signal Interface

When the algorithm was evaluated with the F100 hybrid simulation, all inputs to and outputs from the CIM unit were represented by linear, ±10–V signals. The inputs were read in by A/D converters within the controls microcomputer; likewise all outputs, including the CPU health signals discussed previously, were generated by the D/A converters within the controls microcomputer. For the engine demonstration, however, the real characteristics of the measurement devices and actuators had to be accounted for in the controls microcomputer. These devices included thermocouples, pressure transducers, slide-wire transducers, electro-hydraulic actuators, and the like. Interface electronics to minimize the impact of these varied devices on the CIM unit controls microcomputer were provided by the PSL operations personnel. However, software that would accommodate nonlinearities and scale factors different from those used with the hybrid simulation was required.

Results and Discussion

The real-time microcomputer implementation of the combined MVC–ADIA algorithm performed extremely well. Research results of the evaluation and demonstration of the ADIA algorithm with a real-time hybrid simulation and with an actual F100 engine are given in references 7 to 9. These results, which include steady-state and transient data for the F100 MVC combined with the ADIA algorithm for the no-failure case and for a variety of failure scenarios, show excellent algorithm performance.

Several features of particular interest demonstrate the feasibility of implementing the ADIA algorithm in a production engine control.

First, the algorithm is programmed almost entirely in a high order language (HOL). All previous research control applications and most current engine controls are programmed in assembly language in order to attain real-time operation.

The commonly held belief that assembly language programming (although it is deemed necessary in some real-time applications) is less reliable than HOL programming seemed to be supported during the simulation evaluation and engine demonstration of the ADIA algorithm. During this 3-yr period, the software was updated a number of times because of its evolution from the initial evaluation version to the final demonstration version. Even with this extensive software experience base, several latent faults in the software emerged during the PSL demonstration. Note that the MVC code on CPU 1 was the most mature part of the ADIA code, having been developed almost 2 yr prior to the ADIA implementation. In addition, it was a direct translation from the code used for the MVCS engine demonstration; yet all of the software faults were in the assembly language programs in the MVC running on CPU 1. Furthermore, all were in sections of the code that were manipulating scaled integer numbers. Using a HOL, then, should be accompanied by the use of floating-point arithmetic. This is accommodated in hardware with most state-of-the-art microprocessors.

Several comments are also in order about using parallel processing in the implementation of the ADIA. When this

decision was being made, two factors were considered important: first, using off-the-shelf microprocessors to realistically emulate next generation engine control computers, and second, maintaining the original structure of the ADIA algorithm so as to minimize the impact on the ADIA algorithm software. (As mentioned in the implementation languages section, the ADIA algorithm required considerable code optimization in order to be implemented in real time. However, the structure of the algorithm remained identical to the original version delivered to NASA Lewis at the end of algorithm development.) Some alternatives to using parallel processors were considered, including updating the model matrices every few update intervals and freezing control outputs while isolating and accommodating failures. However, parallel processing allowed the algorithm to be used as originally formulated and to be fully updated each control update interval.

An additional benefit, and a direct result of the increased computing power provided by adding a third parallel processor, was continuous execution of the soft-failure isolation logic; this eliminated the need for the original soft-failure detection logic. That is, soft-failure detection and isolation were both performed by the same logic, thereby actually simplifying the ADIA algorithm. Finally, the way the algorithm was partitioned onto multiple processors accentuated the simple interface between the MVC control algorithm and the ADIA sensor failure logic, and in turn, showed the generic nature of the ADIA algorithm.

Concluding Remarks

The use of parallel processing and high order language programming not only has demonstrated the value of these technologies for sophisticated control applications but also has allowed the research implementation of a combined control and sensor failure algorithm in a cost-effective manner.

Research results of the simulation evaluation and engine demonstration of the advanced detection, isolation, and accommodation (ADIA) algorithm show that the real-time implementation worked very well. Indeed, with the actual F100 engine the algorithm performed almost exactly as predicted by the real-time simulation evaluation. The fact that the ADIA algorithm performed as predicted, and in a timeframe, memory size, and with hardware and software that realistically emulates future engine control systems, leads to the conclusion that it not only works well but also is practical and feasible for engine control systems.

As turbofan engine control system complexity continues to increase to provide improved engine system performance, the software cost (already a major part of the control system cost) will dominate total system cost. Consequently, sophisticated hardware, and more importantly improved software engineering techniques, will be required. Toward the latter goal, the assembly language executive routine in the ADIA program should be replaced with a high-level-language, real-time executive routine to take advantage of real-time operating systems and/or real-time constructs found in high-level languages such as Ada. With respect to hardware, since the speed of computers has been increasing at a rapid pace (and is expected to continue to do so), current 32-bit microprocessors, most notably the Intel 80386, could perform the entire ADIA algorithm on a single CPU rather than on three CPU's. This single-CPU implementation would decrease the hardware cost of the ADIA implementation even further. Clearly, combining advanced microprocessors with structured software design and implementation techniques will enable the use of analytical redundancy in future complex aerospace control systems.

Lewis Research Center
National Aeronautics and Space Administration
Cleveland, Ohio, July 28, 1989

Appendix—Symbols and Abbreviations

A	sensed value of actuator feedback
A_{ES}	simulated value of actuator feedback
A/D	analog to digital
ADIA	advanced detection, isolation, and accommodation
AJ	exhaust nozzle area
BOM	bill-of-material control
b	temperature bias
C_I	integral gain
C_P	proportional gain
CIM	control, interface, and monitoring
CIVV	compressor inlet variable vanes
CPU	central processing unit
D	system feedthrough matrix
D/A	digital to analog
DFLAGH	hard-failure detection flag
DH	likelihood ratio for each failure case
DHMAX	maximum likelihood ratio
E	environmental variables
EMODEL	engine model matrices calculation
e	error tolerance
F	system matrix
FADEC	full authority digital electronic control
FDIA	hard-failure detection and isolation logic and accommodation filter
FDISOL	soft-failure detection and isolation logic
FILTER	Kalman filter calculation
FORTRAN	application-oriented programming language
FTIT	fan turbine inlet temperature
$F^{-1}G$	steady-state gain matrix
F100	turbofan engine
GAM	residual vector
GAMCH	residual vector used for sensor heal
GAMH	hypothesis residual vector
GENCMD	command file generator
GSDO	sensor adjusted standard deviation
H	system output matrix, hypothesis filter
HI	hypothesis or smoothed weighted sum of squared residuals
HOL	high order language
ICHAN	corresponding failure isolation flag
ICNEW	channel number of the most recent failed channel
IMAX	channel number with the maximum likelihood ratio
INITFL	soft-failure isolation initialization flag
ISOLT	soft-failure isolation flag
K	Kalman gain matrix
K_6	FTIT integral gain matrix
KA	modified K-matrix
KI	hypothesis or isolation K-matrix
KTHB	adaptive threshold bias value
L	likelihood of residual vector
LQR	linear quadratic regulator
LR	likelihood ratio
M_{tran}	control system variable indicating transient operation
MINDS	microcontroller interactive data system
MIPS	million instructions per second
MTRTAU	M_{tran} time constant
MVC	multivariable control
MVCS	multivariable control synthesis program
N1	fan speed
N2	compressor speed
O	value of control output
P	probability
PCV	virtual power code
PLA	power level angle
PSL	Propulsion Systems Laboratory
PT2	fan inlet (total) pressure
PT4	burner pressure
PT6	exhaust nozzle pressure
P0	ambient (static) pressure
RAM	random access memory
RCVV	rear compressor variable vanes
RPSCH	reference point schedules
S	nominal value of sensed signal
SFS	sensor failure simulator
SMNSEN	sensed mach number
SNFMAP	virtual power code calculation
SNFTR	transition control value of fan speed
dT	update interval

TAUEXP	first-order lag time constant
THRDI	hard-failure detection and isolation threshold
THRH	heal threshold
THRI	steady-state soft-failure isolation threshold
THRIA	adaptive soft-failure isolation threshold
TT2	fan inlet temperature
TT25	compressor inlet temperature
TT2SN	sensed fan inlet temperature
U	controlled engine inputs
WF	main combustor fuel flow
WSSR	weighted sum of squared residuals
X	engine state vector
Y	control state variables
Z	engine outputs
γ	residual vector
λ	adaptive threshold
Σ	diagonal matrix of adjusted standard deviations
σ	adjusted standard deviation of sensor noise
τ	threshold time constant

Subscripts:

b	base point or steady-state operation
com	command
EXP	transient
I	isolation
IO	isolation output
i	ith value of vector
ic	correct isolation hypothesis filter
m	measured
max	maximum value
min	minimum value
mod	modified
n	current value
n—1	past value
ref	modeled value
S	scheduled
SS	steady-state
0/00	normal mode
1	N1 output
2	N2 output
3	PT4 output
4	PT6 output
5	FTIT output

Superscripts:

^	estimated
.	time derivative
T	transpose

References

1. Baker, L.E.; Warner, D.E.; and Disparte, C.P.: Design of Fault Tolerant Electronic Engine Controls. AIAA Paper 81-1496, July 1981.
2. Merrill, W.C.: Sensor Failure Detection for Jet Engines Using Analytical Redundancy. Journal of Guidance Control Dynamics, vol. 8, no. 6, Nov.-Dec. 1985, pp. 673-682.
3. Beattie, E.C., et al.: Sensor Failure Detection System—For the F100 Turbofan Engine. (PWA-5736-17 Pratt and Whitney Aircraft Group; NASA Contract NAS3-22481) NASA CR-165515, 1981.
4. Beattie, E.C., et al.: Sensor Failure Detection for Jet Engines. (PWA-5891-18, Pratt and Whitney Aircraft Group; NASA Contract NAS3-23282) NASA CR-168190, 1983.
5. DeLaat, J.C.; and Merrill, W.C.: A Real-Time Implementation of an Advanced Sensor Failure Detection, Isolation, and Accommodation Algorithm. AIAA Paper 84-0569, Jan. 1984 (NASA TM-83553).
6. Merrill, W.C.; and DeLaat, J.C.: A Real-Time Simulation Evaluation of an Advanced Detection, Isolation, and Accommodation Algorithm for Sensor Failures in Turbine Engines. NASA TM-87289, 1986.
7. Merrill, W.C.; DeLaat, J.C.; and Bruton, W.M.: Advanced Detection, Isolation, and Accommodation of Sensor Failures—Real-Time Evaluation. NASA TP-2740, 1987.
8. Merrill, W.C., et al.: Full-Scale Engine Demonstration of an Advanced Sensor Failure Detection, Isolation, and Accommodation Algorithm—Preliminary Results. AIAA Paper 87-2259, Aug. 1987 (NASA TM-89880).
9. Merrill, W.C., et al.: Advanced Detection, Isolation, and Accommodation of Sensor Failure in Turbofan Engines—Engine Demonstration Results. NASA TP-2836, 1988.
10. Szuch, J.R.; and Seldner, K.: Real-Time Simulation of F100-PW-100 Turbofan Engine Using the Hybrid Computer. NASA TM X-3261, 1975.
11. Szuch, J.R., et al.: F100 Multivariable Control Synthesis Program—Evaluation of A Multivariable Control Using a Real-Time Engine Simulation. NASA TP-1056, 1977.
12. DeLaat, J.C.; and Soeder, J.F.: Design of a Microprocessor-Based Control, Interface, and Monitoring (CIM) Unit for Turbine Engine Controls Research. NASA TM-83433, 1983.
13. Melcher, K.J., et al.: A Sensor Failure Simulator for Control System Reliability Studies. NASA TM-87271, 1986.
14. Litt, J.S.; DeLaat, J.C.; and Merrill, W.C.: A Microprocessor-Based Real-Time Simulator of a Turbofan Engine. NASA TM-100889, AVSCOM-TR-88-C-011, 1988.
15. DeHoff, R.L., et al.: F100 Multivariable Control Synthesis Program, Vol. I. Development of F100 Control Systems. AFAPL-TR-77-35, Vol. I, 1977. (Avail. NTIS, AD-A052420).
16. Soeder, J.F.: F100 Multivariable Control Synthesis Program—Computer Implementation of the F100 Multivariable Control Algorithm. NASA TP-2231, 1983.
17. Lehtinen, B., et al.: F100 Multivariable Control Synthesis Program—Results of Engine Altitude Tests. NASA TM S-83367, 1983.
18. Soeder, J.F.: MINDS—A Microcomputer Interactive Data System for 8086-Based Controllers. NASA TP-2378, 1985.
19. Mackin, M.A.; and Soeder, J.F.: Floating-Point Function Generation Routines for 16-Bit Microcomputers. NASA TM-83783, 1984.
20. DeLaat, J.C.: A Real-Time FORTRAN Implementation of a Sensor Failure Detection, Isolation and Accommodation Algorithm. Proceedings of the 1984 American Control Conference, Vol. 1, IEEE, 1984, pp. 572-573.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TP-2925		

4. Title and Subtitle

Advanced Detection, Isolation, and Accommodation of Sensor Failures in Turbofan Engines—Real-Time Microcomputer Implementation

5. Report Date

February 1990

6. Performing Organization Code

7. Author(s)

John C. DeLaat and Walter C. Merrill

8. Performing Organization Report No.

E-4391

10. Work Unit No.

505-62-01

9. Performing Organization Name and Address

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135-3191

11. Contract or Grant No.

13. Type of Report and Period Covered

Technical Paper

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, D.C. 20546-0001

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The objective of the Advanced Detection, Isolation, and Accommodation Program is to improve the overall demonstrated reliability of digital electronic control systems for turbine engines. For this purpose, an algorithm has been developed which detects, isolates, and accommodates sensor failures by using analytical redundancy. The performance of this algorithm has been evaluated on a real-time engine simulation and has been demonstrated on a full-scale F100 turbofan engine. This report describes the real-time implementation of the algorithm. The implementation used state-of-the-art microprocessor hardware and software, including parallel processing and high order language programming.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Sensors; Failures; Detection; Redundancy; Fault tolerance; Computer; Implementation; Feedback controls; Isolation; Accommodation; Engine test	Unclassified – Unlimited Subject Category 08

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No of pages	22. Price*
Unclassified	Unclassified	32	A03

NASA FORM 1626 OCT 86

*For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1990