NASA Technical Memorandum 102327

A Real Time Microcomputer Implementation of Sensor Failure Detection for Turbofan Engines

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

August 1989

NASA

A REAL TIME MICROCOMPUTER IMPLEMENTATION OF SENSOR FAILURE DETECTION FOR TURBOFAN ENGINES

John C. DeLaat and Walter C. Merrill
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

ABSTRACT

An algorithm has been developed which detects, isolates, and accommodates sensor failures using analytical redundancy. The performance of this algorithm has been demonstrated on a full-scale F100 turbofan engine. The algorithm has been implemented in real-time on a microprocessor-based controls computer which includes parallel processing and high order language programming. Parallel processing was used to achieve the required computational power for the real-time implementation. High order language programming was used in order to reduce the programming and maintenance costs of the algorithm implementation software. The sensor failure algorithm was combined with an existing multivariable control algorithm to give a complete control implementation with sensor analytical redundancy. This paper describes the real-time microprocessor implementation of the algorithm which resulted in the successful completion of the algorithm engine demonstration.

E-5029

INTRODUCTION

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. Although hydromechanical implementations of turbine engine control systems have matured into highly reliable units, there is a trend towards increased engine complexity in order to meet ever increasing engine performance requirements. Consequently, the engine control has become increasingly complex. Because of this complexity

trend and the revolution in digital electronics, the control has evolved from a hydromechanical to a full authority digital electronic control (FADEC) implementation. These FADEC type controls must demonstrate the same or improved levels of reliability as their hydromechanical predecessors.

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. Studies have shown that the least reliable of the control system components are the engine sensors.[1] In fact some type of sensor redundancy will be required to achieve adequate control system reliability. One important type is analytical redundancy which uses a mathematical model to generate redundant information that can be compared to measured information to detect failures. Analytically redundant systems can have cost and weight savings over systems that use hardware redundancy.

Considerable progress has been made in the application of analytical redundancy to improve turbine engine control system reliability. Reference 2 surveys these accomplishments and defines several technology needs. These needs include (1) the ability to detect soft failures over a wide range of operating conditions, (2) real-time implementations of algorithms capable of detecting soft failures, (3) a comparison of algorithm complexity versus performance, and (4) a full scale demonstration of a soft failure detection capability. The ADIA program addresses all of these technology needs.

The ADIA program is organized into four phases: development,[3,4] implementation,[5] evaluation,[6] and demonstration. In the development phase the ADIA algorithm was designed using advanced filtering and detection methodologies. In the implementation phase this advanced algorithm was implemented in microprocessor based hardware. A parallel computer architecture (three processors) was used to allow the algorithm to execute in a time frame

consistent with stable, real-time operation. In the evaluation phase the advanced algorithm and its implementation were evaluated against a real-time hybrid simulation of the F100 engine. Most recently, the algorithm and its implementation have been demonstrated with a full scale F100 engine in the NASA Lewis Propulsion Systems Laboratory. The objective of this demonstration phase was to engine test the performance of the ADIA algorithm on realistic hardware over a substantial portion of the F100 engine flight operating envelope. This paper describes the implementation of the algorithm used to accomplish this engine demonstration. Detailed results of the engine demonstration are given in Refs. 7 and 8.

This paper contains a description of the test bed system used in the engine demonstration, descriptions of the F100 multivariable control algorithm and the ADIA algorithm, and a description of the implementation hardware and software used for the engine demonstration. The implementation aspects of the ADIA program are summarized, with recommendations given for future implementation work.

DEMONSTRATION TEST BED

The engine demonstration of the ADIA algorithm was carried out on the test bed shown in Fig. 1. The test bed consists of the F100 Engine System, the Controls Microcomputer System containing the multivariable control and the ADIA algorithm, the Sensor Failure Simulator, and the F100 Simplified Engine Simulator.

The Engine System consists 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 four controlled inputs, five engine outputs, and four environmental variables. These variables are defined as follows:

Controlled Engine Inputs, U_{com} and U_m		Sensed Engine Outputs, Z_m		Sensed Environmental Variables, E_m	
WF	Main combustor fuel flow	N1	Fan speed	P0	Ambient (static) pressure
AJ	Exhaust nozzle area	N2	Compressor speed	PT2	Fan inlet (total) pressure
CIVV	Compressor inlet variable vanes	PT4	Burner pressure	TT2	Fan inlet temperature
RCVV	Rear compressor variable vanes	PT6	Exhaust nozzle pressure	TT25	Compressor inlet temperature
		FTIT	Fan turbine inlet temperature		

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.

The Control, Interface, and Monitoring (CIM) Unit[9] (Fig. 2) contains the microcomputer used to implement the algorithm in real-time. The CIM Unit 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. In addition to the controls microcomputer, the CIM Unit also contains hardware to provide a flexible interface to and from the engine. 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.

The Sensor Failure Simulator[10] (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 allows complete and repeatable control over the failure size and the timing of failure injection.

The F100 Simplified Engine Simulator[11] is used to validate changes made to the controls microcomputer software. The simulator is microprocessor-based

and uses hardware and software similar to that used for the ADIA real-time implementation. During the engine demonstration, all changes to the algorithm software were tested with the Simplified Engine Simulator in order to validate changes to the software without compromising the safety of the engine. The engine simulator was also used during engine testing to simulate the engine actuators, so that actuator failures could be detected.

F100 MULTIVARIABLE CONTROL

The multivariable control (MVC) system shown in Fig. 3 is essentially a model following, proportional-plus-integral control. The MVC control[12] was previously demonstrated in an altitude test of an F100 engine.[13] The components of the control are the reference point schedules and transition control, the proportional control logic, the integral control logic, and the engine protection logic. The reference point schedules generate a desired engine operating point in response to the pilot's thrust command (PLA) and sensed engine environment. The transition control generates rate limited command trajectories for smooth transition between steady-state operating points. The proportional and integral control logic minimize transient and steady-state deviations from the commanded trajectories. The engine protection logic limits the size of the commanded engine inputs. The normal control mode in the MVC logic uses fuel flow to set engine fan speed and nozzle area to set nozzle pressure (engine pressure ratio). However, at those conditions where limiting is required, fuel flow can be used to limit the maximum FTIT, the maximum PT4, or the minimum PT4.

ADIA Algorithm Description

The ADIA algorithm detects, isolates, and accommodates sensor failures in an F100 turbofan engine control system. The algorithm incorporates advanced filtering and detection logic and is general enough to be applied to different engines or other types of control systems. The algorithm detects two classes

of sensor failures, hard and soft. 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 increase relatively slowly with time. The ADIA algorithm (Fig. 3) consists of four elements: (1) hard sensor failure detection and isolation logic; (2) soft sensor failure detection and isolation logic; (3) an accommodation filter; and (4) the interface switch matrix.

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 ($\hat{Z}(t)$) are used by the control law. When a sensor failure occurs, the detection logic determines that a failure has occurred. The isolation logic then determines which sensor is faulty. This structural information is passed to the accommodation filter. The accommodation filter then 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. The ADIA algorithm inputs as shown in Fig. 3 are the sensed engine output variables, $Z_m(t)$, the sensed engine environmental variables, $E_m(t)$, and the sensed engine input variables, $U_m(t)$. The outputs of the algorithm, the estimates, $\hat{Z}(t)$, of the measured engine outputs, $Z_m(t)$, are used as input to the proportional part of the control. During normal mode operation, engine measurements are used in the integral control to ensure accurate steady-state operation. When a sensor failure is accommodated, the measurement in the integral control is replaced with the corresponding accommodation filter estimate by reconfiguring the interface switch matrix.

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

$$\frac{d\hat{X}}{dt} = F(\hat{X} - X_b) + G(U_m - U_b) + K\varepsilon \tag{1}$$

$$\hat{Z} = H(\hat{X} - X_b) + D(U_m - U_b) + Z_b \tag{2}$$

$$\varepsilon = Z_m - \hat{Z} \tag{3}$$

Here the subscript b represents the base point (steady-state engine operating point) and X is the 4 by 1 model state vector, U_m is the 4 by 1 sensed control vector, and Z_m is the 5 by 1 sensed output vector. The matrix K is the Kalman gain matrix and ε is the residual vector. The F, G, H, and D matrices are the appropriately dimensioned system matrices. Their individual matrix elements along with those of K are corrected by the engine inlet conditions E_m and scheduled as nonlinear functions of Z_b.[4] An improvement that was added to the accommodation filter was the incorporation of integral action to improve steady-state accuracy of the FTIT estimate $\hat{Z}_5$. One important engine control mode is the limiting of FTIT at high power operation. Because the FTIT sensor is relatively slow, control action is based upon the dynamically faster FTIT estimate. Because the FTIT limiting control has integral action, 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

$$\frac{db}{dt} = K_6\varepsilon \tag{4}$$

$$\hat{FTIT} = \hat{Z}_5 + b \tag{5}$$

where K_6 is a gain matrix, b is the temperature bias, and $\hat{Z}_5$ is the unbiased temperature estimate. The addition of these dynamics, while improving

FTIT estimation accuracy, results in a larger minimum detectable FTIT drift failure rate. This filter structure, which includes the FTIT bias state, is the structure used in the accommodation filter and all the hypothesis filters used in the soft detection and isolation logic.

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 compressor speed sensor failure (N2) has been isolated, the effect of reconfiguration is to force $\varepsilon_2 = 0$. This is equivalent to setting sensed N2 equal to the estimate of N2 generated by the filter. The residuals generated by the accommodation filter are used in the hard failure detection logic.

Hard failure detection and isolation logic. - The hard sensor failure detection and isolation logic is straightforward. To accomplish hard failure detection and isolation 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, then a failure is detected and isolated for the sensor corresponding to the residual element. Threshold sizes are initially determined from 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 detection threshold values are twice the magnitude of these adjusted standard deviations. These magnitudes are summarized in Table 1.

Soft failure detection and isolation logic. - The soft failure detection logic (Fig. 4) consists of multiple-hypothesis-based testing. Each hypothesis is implemented using a Kalman filter. A total of six hypothesis filters are shown, one for normal mode operation (H_0) and five for the failure modes (one for each engine output sensor, H_1 to H_5). The structure for each hypothesis filter is identical to the accommodation filter. However, each hypothesis

filter uses 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, ε_i. From this residual each hypothesis filter generates a statistic or likelihood based upon a weighted sum of squared residuals (WSSR). Assuming Gaussian sensor noise, each sample of ε_i has a certain likelihood or probability

$$L_i = p_i(\varepsilon_i) = ke^{-WSSR_i} \tag{6}$$

where k is a constant and $WSSR_i = \varepsilon_i^T \Sigma^{-1} \varepsilon_i$ where T is matrix transposition and $\Sigma = \text{diag}(\sigma_i^2)$.

The σ_i are the adjusted standard deviations defined in Table 1. These standard deviation values scale the residuals to dimensionless 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. The log of the ratio of each hypothesis likelihood to the normal mode likelihood is calculated. Mathematically, when the log of the ratio of likelihoods is taken, then

$$LR_i = \log\left(\frac{L_i}{L_0}\right) = WSSR_0 - WSSR_i \tag{7}$$

If the maximum log likelihood ratio exceeds the soft failure detection and isolation threshold, then 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 LR_1. Thus, LR_1 will be the maximum and it will be compared to the threshold to detect the failure.

Adaptive threshold. - Initially, the soft failure detection/isolation threshold was determined by standard statistical analysis of the residuals to set the confidence level of false alarms and missed detections. The threshold was then modified to account for modeling error. It was soon apparent from initial evaluation studies that transient modeling error was dominant in determining the fixed threshold level. It was also clear that this threshold was too large for desirable steady-state operation. Thus, an adaptive threshold was incorporated to make the algorithm more robust to transient modeling error while maintaining steady-state performance.

The adaptive threshold defined as

$$\Gamma_i = \Gamma_{iSS}\left(\Gamma_{EXP} + 1\right) \tag{8}$$

$$\tau \frac{d\Gamma}{dt} EXP + \Gamma_{EXP} = M_{tran} \tag{9}$$

was heuristically determined and consists of two parts. One part, Γ_{iSS}, is the steady-state detection/isolation threshold which accounts for steady-state, or low frequency modeling error. The second part, Γ_{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. The values of Γ_{iSS}, τ, and M_{tran} were found by experimentation to minimize false alarms during transients. When the engine experiences a transient, M_{tran} is set to 4.5, otherwise it is 0. The threshold time constant $\tau = 2$ sec. The adaptive threshold expansion logic enabled Γ_{iSS} to be reduced to 40 percent of its original value which results in an 80 percent reduction in the detection/isolation threshold Γ_i^2. The adaptive threshold logic is illustrated in Fig. 5 for a PLA pulse transient.

Failure accommodation. - For accommodation two separate steps are taken. First, all seven of the filters (the accommodation filter and the six hypothesis filters) are reconfigured (the appropriate residual in each filter

is forced to zero) to account for the detected failure mode. Second, if a soft failure was detected, the states and estimates of all seven filters are updated to the values of the hypothesis filter which corresponds to the failed sensor.

MICROCOMPUTER IMPLEMENTATION

The objective of the real-time implementation of the algorithm was to allow a demonstration of the algorithm with a full scale engine using 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. Three CPU's are used, operating in parallel. The software which implements the ADIA algorithm uses floating-point arithmetic. In addition, the ADIA software is coded almost entirely in the application language, FORTRAN. The FORTRAN subroutines have been optimized to execute in real time. Only the schedules used to generate the engine model basepoints and table look-up routines within the ADIA algorithm are coded in assembly language.

Controls Microcomputer Hardware/Software Design

Implementing the MVC-ADIA algorithm required integrating the ADIA algorithm with the existing microcomputer implementation of the F100 multivariable control (MVC). The update interval of the microprocessor-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 resulting control microcomputer, based on the Intel 80186 microprocessor architecture, used multiple processors operating in parallel to satisfy the update interval requirement.

The software for the combined MVC-ADIA algorithm was partitioned so that the ADIA software ran on a second CPU while the MVC algorithm ran on the first. Because the soft-failure isolation logic required a significant amount

of processing time, a third CPU was added to implement the soft isolation logic in parallel. Data were transferred between CPU's through dual-ported memory, and synchronization between CPU's was achieved through interrupts. The features of the Monolithic Systems MSC 8186 single-board computers used are shown in Table 2.

Each of these CPU's provides approximately 0.7 million instructions per second (MIPS), such that the three processors combined provide on the order of 2 MIPS. These three CPU's are contained in an 18 slot Multibus chassis. The resulting hardware configuration is shown in Fig. 6. In addition to the CPU's, the chassis also contains a floppy disk controller, a graphics interface, and both analog and digital input/output boards.

The relative timing for the three CPU's is shown in Fig. 7. The arrows in the figure represent interrupts. The first event to occur is an update interval timer interrupt to CPU 1. CPU 1 then samples all the algorithm inputs. The measurements required for the algorithm on CPU's 2 and 3 are then converted by CPU 1 to floating point numbers and transferred to CPU 2 with an interrupt to indicate that the algorithm inputs are now available. CPU 1 then computes the parts of the MVC algorithm that are not dependent on the outputs of the ADIA algorithm, that is, the reference point schedules and transition control. Concurrently, CPU 2 uses the algorithm inputs to compute the engine model matrices and basepoints, and the Kalman gain matrix. This information is then passed to CPU 3, with an interrupt to indicate the information is available, for use in the soft failure isolation logic. The soft failure isolation logic computes through the remainder of the current update interval and into the next. Any soft isolation information that results is then transferred from CPU 3 to CPU 2. CPU 2 performs the hard failure detection and accommodation filter computations using the isolation information if needed. An interrupt is then sent from CPU 2 to CPU 1 indicating the ADIA

calculations are completed. CPU 1 reads the resulting ADIA outputs, finishes the MVC calculations, and sends the controlled variables to the engine actuators. Lastly, CPU 2 calculates altitude and Mach number from the engine environmental variables for use during the next update interval.

The Microcontroller Interactive Data System (MINDS),[14] used for data acquisition, runs in the spare time on CPU 1 (Fig. 7). The MINDS package has both steady-state and transient data-taking capabilities and can access any variable in the MVC or ADIA algorithm for display or plotting.

Implementation Languages

The MVC is implemented in fixed-point assembly language on CPU 1. When the MVC was originally implemented on a microcomputer 3 years prior to the ADIA implementation, assembly language programming using 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. Thus most of the ADIA algorithm running on CPU's 2 and 3 is programmed in floating-point arithmetic and the application-oriented language FORTRAN. FORTRAN was chosen because the ADIA as developed was coded in FORTRAN.

The advantages of using floating-point arithmetic and an application language such as FORTRAN rather than programming in fixed-point assembly language as was used for the MVC include increased software reliability and reduced software development and maintenance costs. The primary disadvantage to using an application language is that it generally produces less efficient object code than the equivalent functions programmed in assembly language. Entirely in FORTRAN and as originally coded, the ADIA algorithm took over an order of magnitude longer than the required 40 msec update interval. To speed execution, table lookup routines,[15] which are written to take advantage of the

8087 architecture and are executed frequently in the ADIA algorithm, are implemented in assembly language. The hardware interface routines, which have no FORTRAN equivalent, are also implemented in assembly language. The schedules used to compute the accommodation filter basepoints are identical functionally to the reference point schedules in the MVC, so the MVC assembly language schedules were used to save additional compute time. To allow the remainder of the algorithm to remain in FORTRAN, the source code has been optimized to make it run more efficiently.[16] As shown in Fig. 7, the entire MVC-ADIA algorithm now executes in less than the required 40 msec.

Memory Requirements

The memory required for each of the three CPU's is shown in Fig. 8. Each CPU has, in addition to its share of the MVC-ADIA algorithm, an executive routine that maintains correct real-time operation of the total algorithm. The memory required for the algorithm and for the executive are shown for each CPU. The memory required for MINDS is shown for CPU 1. It was necessary to ensure that while the engine was operating under research control (MVC-ADIA), any event which compromised the safety of the engine be detected and an appropriate action taken. Failure modes within the controls microcomputer were identified and safety procedures defined which avoided compromising the safety of the engine. The memory required for the safety software, which runs only on CPU 1 is shown. In all cases the code and the constants were about 75 percent, and the data and the variables about 25 percent, of the total memory required. Figure 8 shows the total memory required for all executives (16.9 kbytes), the total algorithm (54 kbytes), MINDS (34.4 kbytes), and the safety software (2.5 kbytes) for all three CPU's combined. It is likely that a state-of-art 32-bit microprocessor is capable of real-time execution of the MVC and ADIA algorithms on a single CPU. Thus, Fig. 8 shows what the total memory required for the MVC-ADIA implementation would be if it could indeed be combined onto a

single CPU. This would eliminate some redundant code in the executives and in the algorithm which was replicated due to being distributed across multiple CPU's. As shown, about 5 kbytes of executive and 15 kbytes of algorithm are replicated in the multiple CPU implementation. Thus a single CPU implementation would take about 20 kbytes less memory.

RESULTS AND DISCUSSION

The real-time microcomputer implementation of the combined MVC-ADIA algorithm performed extremely well. Sensor failure detection and accommodation were demonstrated at eleven different operating points which included subsonic and supersonic conditions and medium and high power operation of the engine. The minimum detectable failure magnitudes represent excellent algorithm performance and compare favorably to values predicted by simulation. Accommodation performance was excellent. Transient engine operation over the full power range with single sensors failed and accommodated was successfully demonstrated. Open loop engine operation (all feedback sensors failed and accommodated) over at least 75 percent of the power range was also demonstrated at two different operating conditions.[8]

There were several features of the implementation which are of particular interest and which demonstrate the feasibility of implementing the ADIA algorithm in a production engine control. The implementation uses almost entirely High Order Language (HOL) programming. All previous research control applications and most current engine controls use assembly language programming in order to attain real-time operation. Using an HOL greatly reduces software costs both in the programming and maintenance of the software due to increased software reliability. The experience gained during the ADIA evaluation and demonstration supports this conclusion. The ADIA algorithm was evaluated with a real-time simulation of the F100 engine over a period of 2 years. During that time there were a number of software changes made due to

the evolution of the software from the initial evaluation version to the final demonstration version. Even with this extensive software experience base there were several latent faults in the software that emerged during the PSL demonstration. All of the faults were in the assembly language programs in the MVC running on CPU 1. This reinforces the commonly held belief that assembly language programming, although deemed necessary in some real-time applications, is a major contributor to software unreliability when compared to HOL's. Furthermore, these errors all had to do with scaled integer arithmetic. 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 the parallel processing implementation used for the ADIA. When the decision was made to use parallel processing, several factors were considered. First, it was important to use off-the-shelf microprocessors to realistically emulate next generation engine control computers. Second, it was desired to maintain the original structure of the ADIA algorithm so as to minimize the number of changes to the ADIA algorithm software. As mentioned earlier in this report, 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 as delivered to NASA Lewis Research Center at the end of algorithm development. Alternatives to using parallel processors were considered. These included only updating the model matrices every several update intervals and freezing control outputs while isolating and accommodating failures. However, the use of the parallel processors allowed the algorithm to be used as originally formulated and allowed the algorithm to be fully updated each control update interval. 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. This in turn points out the generic nature of the ADIA algorithm.

CONCLUDING REMARKS

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

Research results of the evaluation and demonstration of the Advanced Detection, Isolation, and Accommodation (ADIA) algorithm with a real-time hybrid simulation and with an actual F100 engine are given in Refs. 6 to 8. These results include steady state and transient data for the F100 Multivariable Control combined with the ADIA algorithm for the no failure case and for a variety of failure scenarios. These results show that the real-time implementation performed very well, and that the performance of the algorithm with the actual F100 engine was almost identical to that predicted by the real-time simulation evaluation. The combination of the fact that the ADIA performed as predicted and in a time frame, memory size, and with hardware and software that realistically emulates future engine control systems leads to the conclusion that the ADIA algorithm not only works well, but is practical and feasible for engine control systems.

As turbofan engine control system complexity continues to increase to provide improved performance of engine systems, the software cost, already a major part of the control system costs, will dominate total system cost. It is therefore important to note that sophisticated hardware, and more importantly improved software engineering techniques, will be required. The assembly language executive used for the ADIA program should be replaced by a high level language real-time executive. This executive may make use of real-time operating systems and/or real-time constructs found in high level

languages such as Ada. In addition, the speed of computers has been increasing at a rapid pace and is expected to continue to do so. There are now 32-bit microprocessors that 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. Combining advanced microprocessors with structured software design and implementation techniques will enable the use of analytical redundancy in future, complex aerospace control systems.

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. J. Guidance Control Dynamics, vol. 8, no. 6, Nov.-Dec. 1985, pp. 673-682. (NASA TM-83695, 1985.)
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. NASA TM-83553, 1984.
6. Merrill, W.C.; DeLaat, J.C.; and Bruton, W.M.: Advanced Detection, Isolation, and Accommodation os Sensor Failures -- Real-Time Evaluation. Journal of Guidance, Control, and Dynamics, vol. 11, no. 6, Nov.-Dec. 1988, pp. 517-526.

7. DeLaat, J.C.; and Merrill, W.C.: Advanced Detection, Isolation, and Accommodation of Sensor Failures in Turbofan Engines - Real Time Microcomputer Implementation. NASA TP, to be published.

8. Merrill, W.C., et al.: Full-Scale Engine Demonstration of an Advanced Sensor Failure Detection, Isolation, and Accommodation Algorithm: Preliminary Results. NASA TM-89880, 1987.

9. 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.

10. Melcher, K.J., et al.: A Sensor Failure Simulator for Control System Reliability Studies. NASA TM-87271, 1986.

11. Litt, J.S.; DeLaat, J.C.; and Merrill, W.C.: A Microprocessor-Based Real-Time Simulator of a Turbofan Engine. NASA TM-100889, 1988.

12. Soeder, J.F.: F-100 Multivariable Control Synthesis Program. Computer Implementation of the F-100 Multivariable Control Algorithm. NASA TP-2231, 1983.

13. Lehtinen, B., et al.: F-100 Multivariable Control Synthesis Program-Results Engine Altitude Tests. NASA TM-S-83367, 1983.

14. Soeder, J.F.: MINDS: A Microcomputer Interactive Data System for 8086-Based Controllers. NASA TP-2378, 1985.

15. Mackin, M.A.; and Soeder, J.F.: Floating-Point Function Generation Routines for 16-Bit Microcomputers. NASA TM-83783, 1984.

16. 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.

TABLE 1. - HARD DETECTION

THRESHOLD MAGNITUDES

Sensor	Adjusted standard deviation	Detection threshold
N1	300 rpm	600 rpm
N2	400 rpm	800 rpm
PT4	30 psi	60 psi
PT6	5 psi	10 psi
FTIT	250 °R	500 °R

TABLE 2. - MSC 8186 FEATURES

8 MHz 80186 microprocessor
  Eight general purpose 16-bit registers
  Four 16-bit segment registers
  Two 16-bit status and control registers
  Integrated peripherals including three timers, two DMA controllers and a programmable interrupt controller
  Signed, fixed-point arithmetic (add 1.25 μsec, multiply 4.5 μsec, divide 7.6 μsec)
8 MHz 8087 numerics coprocessor
  Eight deep 80-bit register stack
  Compatible with IEEE floating point standard 754
  Signed, floating point arithmetic (add 15.6 μsec, multiply 21 μsec, divide 28.7 μsec)
128 kbytes of dual-ported, 150 nsec zero-wait-state dynamic RAM
256 kbytes of EPROM
Five total programmable timer/counters
13 Levels of vectored interrupt control
RS-232 compatible serial interface
IEEE 796/Multibus compatible bus interface

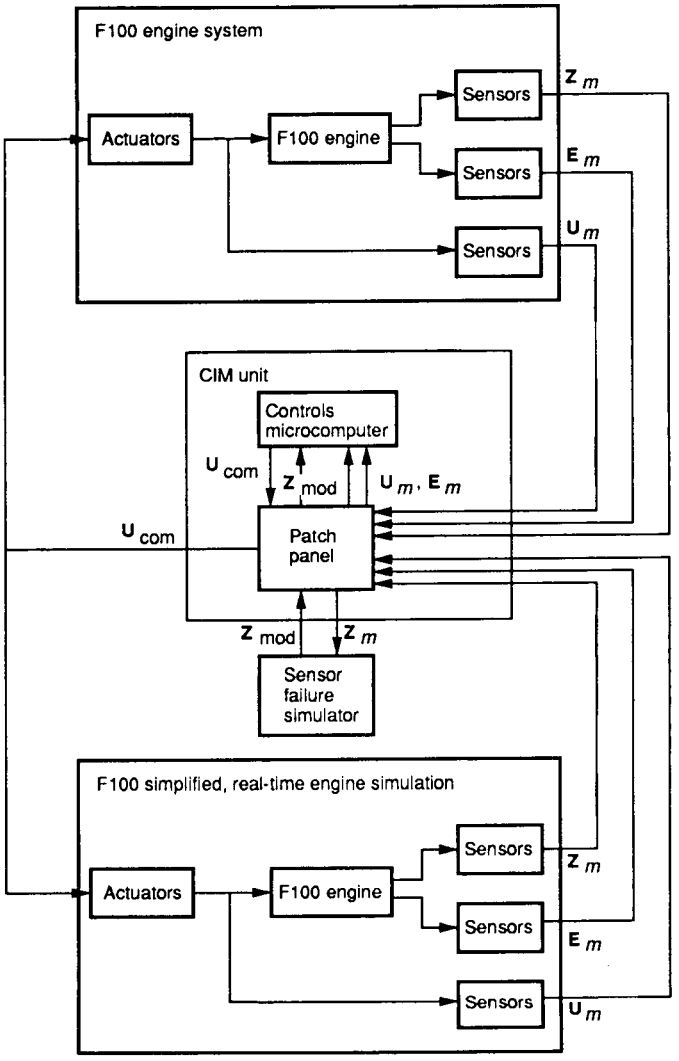


Figure 1. - Test bed configuration.

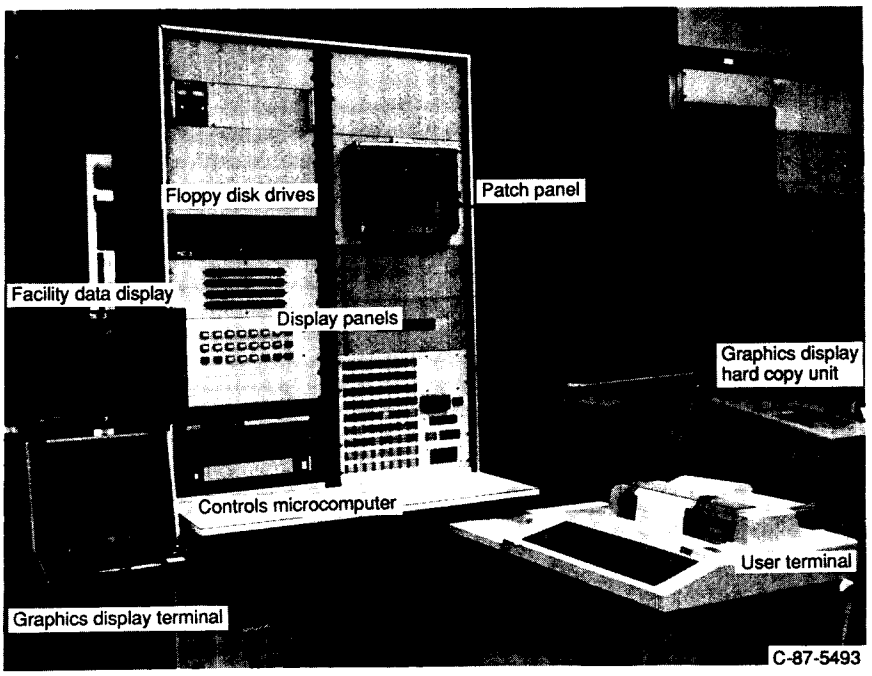


Figure 2. - Control, interface, and monitoring unit.

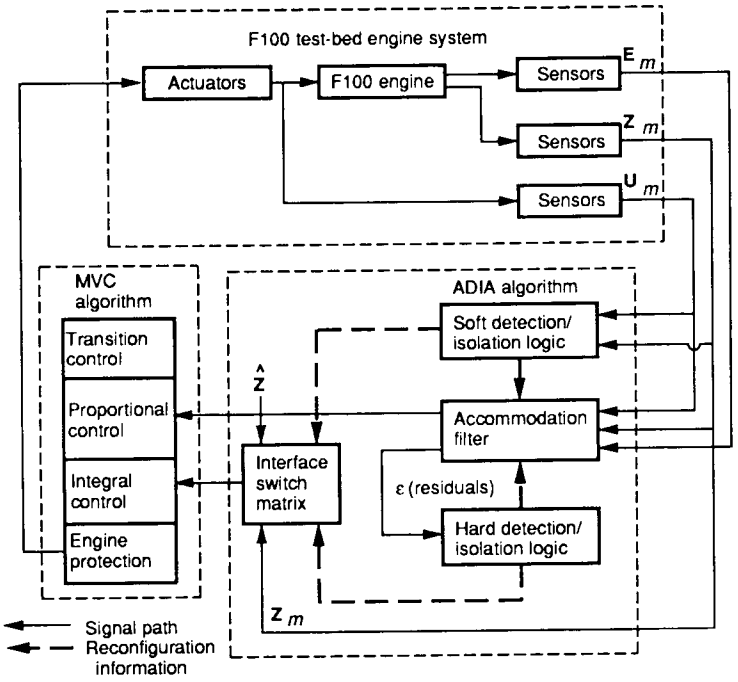


Figure 3. - ADIA block diagram.

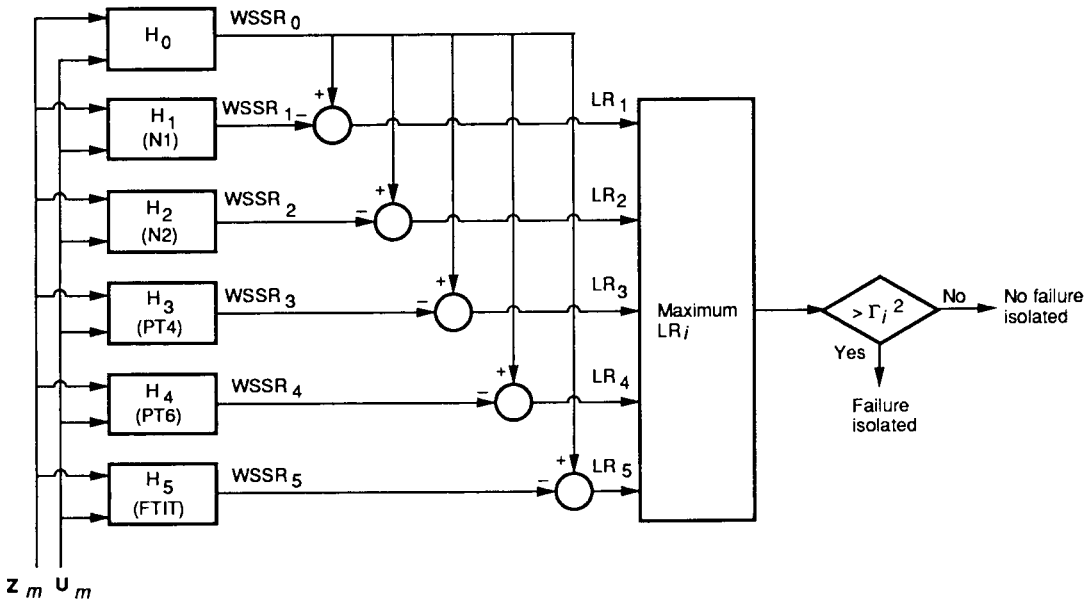


Figure 4. - Soft failure detection/isolation logic structure.

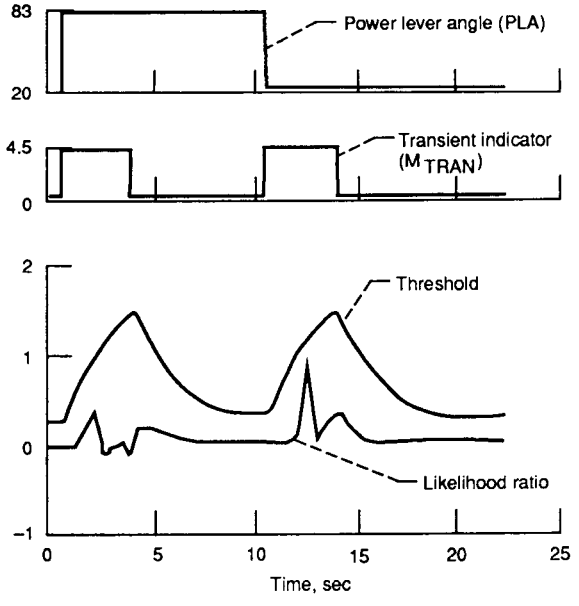


Figure 5. - Soft-failure detection adaptive threshold.

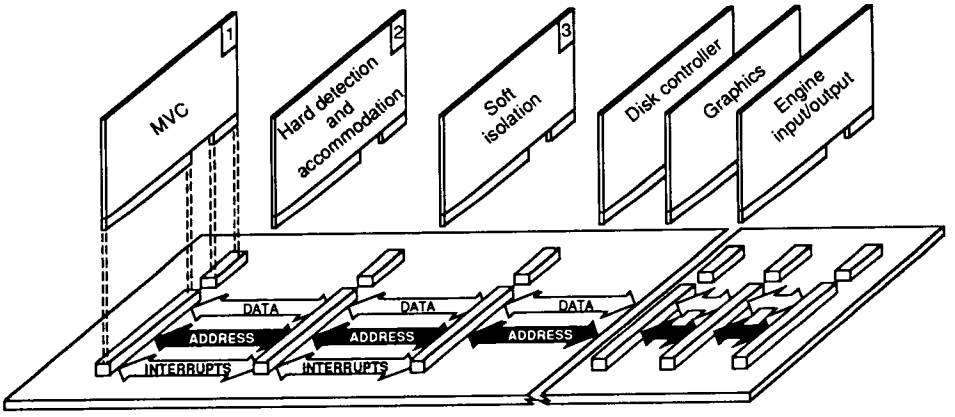
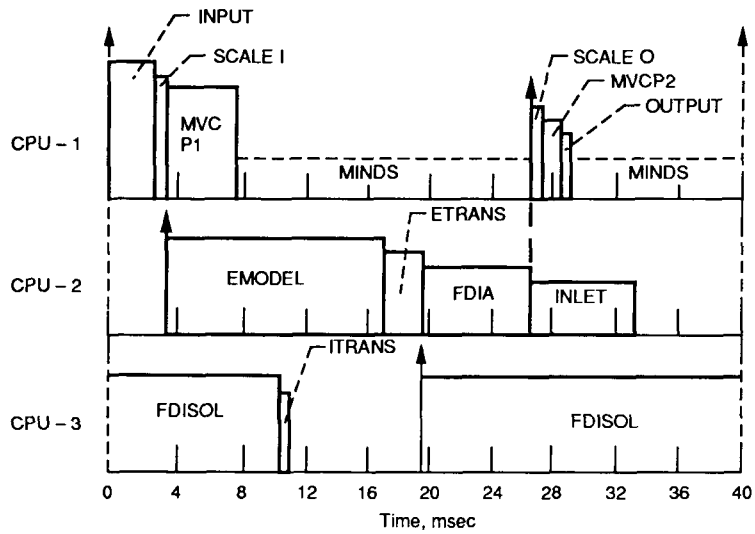


Figure 6. - ADIA inplementation.



CPU - 1		
	INPUT	A/D CONVERSION AND SCALING OF ENGINE MEASUREMENTS
	SCALE I	CONVERSION OF MEASUREMENTS TO FLOATING POINT AND TRANSFER TO CPU-2
	MVCP1	MVC PART 1: REFERENCE POINT SCHEDULE AND TRANSISTION CONTROL
	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 7. - ADIA timing for 8-MHz MSC 8186.

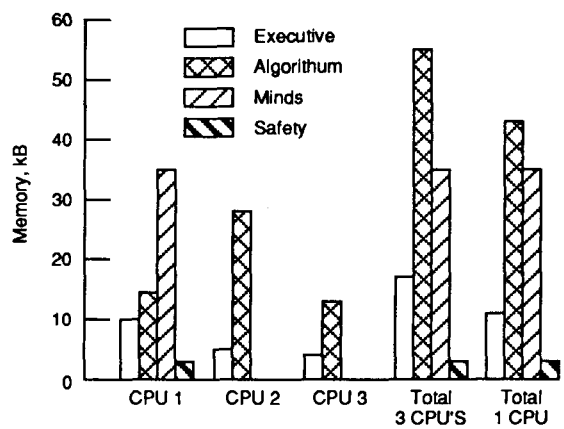


Figure 8. - MVC-ADIA memory requirements.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-102327		

4. Title and Subtitle	5. Report Date
A Real Time Microcomputer Implementation of Sensor Failure Detection for Turbofan Engines	August 1989
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
John C. DeLaat and Walter C. Merrill	E-5029
	10. Work Unit No. 505-62-01

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	
	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546-0001	

15. Supplementary Notes

16. Abstract

An algorithm has been developed which detects, isolates, and accommodates sensor failures using analytical redundancy. The performance of this algorithm has been demonstrated on a full-scale F100 turbofan engine. The algorithm has been implemented in real-time on a microprocessor-based controls computer which includes parallel processing and high order language programming. Parallel processing was used to achieve the required computational power for the real-time implementation. High order language programming was used in order to reduce the programming and maintenance costs of the algorithm implementation software. The sensor failure algorithm was combined with an existing multivariable control algorithm to give a complete control implementation with sensor analytical redundancy. This paper describes the real-time microprocessor implementation of the algorithm which resulted in the successful completion of the algorithm engine demonstration.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Sensors; Failures; Detection; Microprocessor; Real time; Fault tolerance; Implementation; Feedback controls	Unclassified – Unlimited Subject Category 60

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No of pages	22. Price*
Unclassified	Unclassified	26	A03

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