An Aid for Mechanization of Flight Control Systems on Micro-computers

T. V. Rama Murthy
National Aerospace Laboratories, Bangalore-560 017
and
V. Seshadri
Indian Institute of Technology, Madras-600 036

ABSTRACT

This paper deals with the development of an automated aid to translate the block diagram of flight control system (FCS) to assembly level code. By defining a suitable syntax and by building a matrix of inputs and outputs of the blocks, it becomes easy to translate the block diagram. It is also shown how the process of fault detection can be automated. The results obtained through the automated aid have been validated by ORACL library using the block diagram of Cstar controller of F8 aircraft.

2. DEVELOPMENTS IN ACTIVE CONTROL TECHNOLOGY

Fly-by-wire, electrical signaling and control configured vehicles (CCV) have been the major milestones in the development of active control technology (ACT). The applications of ACT have been mainly in the following areas: (i) artificial stability, (ii) ride and handling quality improvements, (iii) load alleviation, (iv) flutter speed enhancement, (v) centre of gravity control, (vi) envelope limiting, (vii) fatigue reduction and (viii) pilot relief-autopilot functions.

NOMENCLATURE

$q, N_c, \delta_c$ aircraft states
$\delta_e$ elevator command
$S$ signature camberra metric
$S_k$ geometric moving average of $S$
$T_F$ execution time of control law
$A_R, B_R, C_R, D_R$ state space specifications of regulator
$A_p, B_p, C_p, D_p$ state space specifications of plant

INTRODUCTION

Although attempts have been made to translate the block diagrams of flight control systems to assembly level code, it appears that issues of detection of faults like sensor faults, hardware/software faults and self-test are not directly addressed as a part of computer-aided tools. An earlier work on the translator has been improved by (i) implementing and verifying fault-detection algorithms, (ii) validating the tool using ORACL library, and (iii) debugging aid has been made easier by employing Turbopascal $>=5.0$. 

Received 14 September 1992, revised
computation including gain schedule and logic (about 35%) and (iii) fault-tolerance including redundancy management, self-test, etc. (about 40 per cent)\textsuperscript{2}.

The executive software deals with the overall supervision of the system. The complete software cycle called 1 major cycle consists of \( n \) minor cycles, each minor cycle is typically 10 or 20 ms to meet the real-time requirements. The executive initiates the minor cycle operations from the table of computations. It also communicates with the pilot for auto-pilot and other outerloop options and for status reporting.

There are many benefits due to CCV implementation. For example, in F-8 aircraft, control laws have been designed for improved handling qualities, envelope limiting, gust alleviation, capability to fly with reduced static stability in lateral axis improved dutch roll damping, turn coordination\textsuperscript{3}, etc. The outerloop modes are: attitude hold, altitude hold, mach hold; auto-pilot functions like approach/land, go around and take off\textsuperscript{4}. The executive/utility sets up the computation table depending on options selected by the pilot. Often, the controller has to get gain settings based on function of mach number and/or dynamic pressure or mach/altitude grid. Hence gain scheduling task will also find entry in minor cycle computation table. All tasks which need processing rates of 50, 25, 12.5, 6.25 Hz will be initiated once in each, alternate, once in 4 and once in 8 minor cycles respectively assuming minor cycle of 20 ms.

2.2 Fault Detection Measures

Redundancy management essentially deals with failure monitoring, fault isolation and voting mechanism. Reconfiguration of a failed element may take place by analytic redundancy technique or ignored by continuation with remaining redundant units. The following form part of pre-flight test\textsuperscript{5}: limit cycle tests, structural resonance tests, frequency response tests, electromagnetic interference tests (including lightning tests).

The self-test and in-flight monitoring capability required depends entirely on its criticality and assumes major importance and the various techniques that are used are given elsewhere\textsuperscript{6}.

Sensor monitoring is done through: (i) comparison monitoring of feedback sensor signals, (ii) software limiting of outerloop sensor signals and (iii) monitoring of critical gains in air-data computation. Servo monitoring is performed by comparing actual servo output against digital models. Input/output circuitry check is done by feeding known constants to A/D converter and checking the code. Similarly, a known digital code is fed to DAC and its analog value checked. CPU monitoring is performed by self-test. All known instructions and control crucial to flight safety are checked. Dynamic computation monitoring is performed by an external independent analog element in some cases. Watchdog timer checks whether the computations are completed within the specified time limits. Memory monitoring is performed by checking, (i) all words accessed from memory for parity, (ii) periodic checksums of critical instructions, constants and scratch pad locations and (iii) program flow through critical instructions.

3. NEED FOR AUTOMATED AID

It is reported that assembly level programs have been used for implementing the flight control programs of modern class of fighter aircraft like F/A 18, F-8 and JA-37. Real-time considerations have forced the designers to write assembly level programs to translate flight control laws with the attendant poor productivity and inflexibility in incorporating changes in control laws. Often, hard debugging effort is required to make the assembly level programs. The designer has to keep track of interconnection of blocks to obtain the exact word length of variables. Some errors due to shorter word length of variables, wrong entries, inadequate delay operation of variables in filters may go unnoticed or be corrected after many run-time attempts.

Fault-detection measures like sensor failures, timing faults and value faults are important in FCS software design. It is desirable to automate these processes. Since the assembly level program is generated manually, it is time consuming to design self-test of the instruction set of the processor.

To meet these requirements, the automated aid has been designed and it works in the environment of IBM PC/MSDOS, MASM assembler, LINK and Turbopascal \( \geq 5.0 \). The automated aid generates reliable codes at optimum assembly level with FCS specifications being expressed in easy-to-use high level syntax. Any hardware/software fault usually manifests in terms of value fault and/or timing fault.
It is known that important instructions are exercised as a part of self-test routine in in-flight monitoring. It will be very easy to select the set of basic instructions from that used by the automated aid. The code-segment part is generated using only the predefined set of macros. From the macro-bodies, a set of basic instructions which can express the complete macro-set is selected. A program is then exercised which makes use of these basic instructions. By comparing the results obtained with the expected values, it is possible to flag any error that occurs while running the self-test.

4. PARSER

4.1 Syntax Design

Typical FCS block diagrams have been studied to arrive at a suitable syntax for representing the blocks. Each block is defined in terms of a set or subset of input(s), output, constants, control variables, etc. The following syntactic entities are defined to specify the various blocks:

\[ <\text{ALPHA}> ::= A/B/C/J/Z \quad <\text{NUMS}> ::= 0/1/2/3/9 \]
\[ <\text{ALPHANUM}> ::= \text{ALPHA}/\text{NUMS} \]
\[ <\text{SYMBOL}> ::= <\text{ALPHA}> f_0^7 [\text{ALPHANUM}] \]

The operator \( f_0 \) specifies that all syntax items enclosed by square brackets are to be repeated 0 or \( m \) number of times \((m \geq 5)\). Syntax to handle the numerical constants are defined as follows:

\[ <\text{REALNUM}> ::= <\text{INTEGER}>/\text{FRACTION} >\]
\[ <\text{FRACTION}> ::= <\text{INTEGER}> \]
\[ <\text{INTEGER}> ::= <\text{NUMS}> \]
\[ f_0^7 [\text{NUMS}] \]
\[ <\text{REALNUMS}> ::= <\text{REALNUM} >\]
\[ f_0^7 [, <\text{REALNUM}] >\]
\[ <\text{SYMBOLS}> ::= <\text{SYMBOL} >\]
\[ f_0^{14} [, <\text{SYMBOL} >] \]

One can take advantage of limited number of blocks that are required and select certain alphabets to denote the blocks. For example, an amplifier may be defined by a three-character symbol as shown below:

\[ <\text{AMPLIFIER}> ::= A<\text{S}/\text{ALPHANUM}> <\text{ALPHA} = \text{NUM} >, \quad \text{the \ character \ S denotes a summing block, where gains are \pm 1. Other blocks are defined in a similar way.} \]
\[ <\text{GAIN}> ::= G<\text{CU}/\text{ALPHANUM}> <\text{ALPHANUM}>, \quad \text{the \ character \ C indicates a controllable gain-block where output is expressed in terms of variables \( Y, X \) as \( (Y/X) \); \( U \) indicates an adaptive gain-block where gain is obtained through interpolation.} \]
\[ <\text{FILTER}> ::= F<\text{U}/\text{ALPHANUM} > <\text{ALPHANUM}>, \quad \text{where \( U \) indicates an adaptive filter block where the filter coefficients are obtained through interpolation.} \]
\[ <\text{SWITCH}> ::= S<\text{ALPHANUM} > <\text{ALPHANUM}>, \quad \text{this block is similar to a single-pole, multi-throw switch.} \]
\[ <\text{QUANTIZE}> ::= Q<\text{X}/\text{L}> <\text{ALPHANUM} > <\text{ALPHANUM}>, \quad \text{where \( O \) indicates an observer block, (see Sec 6 also).} \]

Now, an amplifier may be completely specified as:

\[ <\text{AMP}> ::= <\text{AMPLIFIER} ><\text{SYMBOL} > <\text{REALNUM}> <\text{SYMBOL} > \]

Dialogue procedures have been developed so that the specifications are entered in the required sequence. The sequence of specification entry is determined from the block selected by the user and hence the syntax check can be made based on the definitions given in the preceding paragraphs. Procedures to handle SYMBOL($S$), REALNUM($S$), BLOCK have been designed as a part of parsing algorithms, which also check and warn the entry of duplicate block-names and output symbols.

4.2 Adaptive Block

The control law is generally obtained by using optimization techniques at typical flight conditions spanning the flight envelope. As a result, at any other flight condition, the scheduling calls for single or double
interpolations. The interpolation formulae\(^8\) are used to estimate the parameters, (gain values or filter coefficients) for any given value of the grid variable(s).

There are two assembly procedures INTRPL1 and INTRPL2 for estimating the interpolated values based on single or double grids. The scheduled values \(Y_1, \ldots, Y_i\) are assumed to be stored in ROM in short-real format at discrete values \(X_L, X_L + X_{ND}, X_L + 2X_{ND} \ldots X_L + iX_{ND} \ldots\) of selected parameter like dynamic pressure or mach-altitude grid pair. In adaptive gain or filter blocks, the gain value or the filter coefficients are to be interpolated as a function of grid variable(s) \(X(\text{or } X, Y)\).

5.3 Constants in Data-segment

Constants like filter coefficients, filter order, amplifier gains, etc. will have to be defined in the data-segment. The block-name being unique is used by prefixing it to these constants so that duplication errors are avoided. The filter coefficients \(a_i, b_i\) of filter \(F00\) are thus defined as \(F00\_NUMR\_COEF\) and \(F00\_DENMR\_COEF\). This technique is followed while defining other constants.

A procedure in the translator declares all the variables in the data-segment with the correct size based on the contents of the array \(\text{INPUTS and Filtord}\). These details enable the calculation of gain value or filter coefficients based on any given value of \(X\) (or \(X, Y\)). The automated aid handles the scheduling of blocks by defining the syntax for these blocks in terms of the above entities.

5. CODE GENERATION

The code generator generates 8086/8087 processor-based (i) data-segment where constants and variables with correct size and type are declared and (ii) code-segment where the code for the functional part of each block is generated.

5.1 Filter Discretization

As the controller is implemented in digital domain, the filter blocks use continuous to discrete-domain transformation. Pre-warped Tustin transformation of \(s\) to \(Z\) domain has been used due to its valid cascaded property and widespread usage in FCS\(^5,9\). A procedure in the automated aid package obtains digital filter coefficients \(a_i, b_i\) from \(s\)-plane transfer function, viz. \(H(s) = \sum_{i=0}^{m} A_i s^i / \sum_{j=0}^{n} B_j s^j\) and \(m > n \text{ in general and } A_i, B_i\) are real. The Tustin transformation \(s \leftrightarrow 2(Z-1)/(KT(Z+1))\) where \(T = \text{sampling interval and } K = \text{Pre-warping correction factor},\) gives output \(Y(nT) = \sum_{i=0}^{m} a_i X((n-i)T - \sum_{j=1}^{m} b_j Y((n-j)T).\)

5.2 Algorithms for Obtaining the Size of the Variable

As each block is parsed, the entities — the inputs, outputs and control variables — are entered into an aggregate array \(\text{INPUTS}\) by invoking a procedure \(\text{INSERT}\). \(\text{INSERT}\), in turn, invokes a function \(\text{OKIN}\) passing each entity as an argument. If \(\text{OKIN}\) finds the entity in the array \(\text{INPUTS}\), then it returns false; else it returns true. If \(\text{OKIN}\) returns a true, then \(\text{INSERT}\) updates the array \(\text{INPUTS}\) with the entity.

An array \(\text{Filtord}\) of integer contains the length of the entity at the index corresponding to the same entity in the array \(\text{INPUTS}\). Whenever \(\text{OKIN}\) returns a false, it also gives the index of the array where the entity is stored. The length of the variable gets updated, if the order of the current filter to which it is connected is more. By this procedure, it is possible to keep track of the route of the variable so that its maximum length can be determined exactly.

5.4 Code-segment Generation

In code-segment, the macro-calls are generated in the order corresponding to each block specified in the block diagram. The inputs, outputs of filters are updated before the termination of the translated assembly program. The macros are all predefined and the macro-assembler\(^10\) takes care of differing lengths of the passed arguments by using \(\text{EXITM}\) directive. Table 1 lists the macros necessary to generate the codes for the functional part of each block. The automated aid generates the macro-call statement with the appropriate arguments collected during the parsing stage. The F8 Cstar block diagram, sketched in Fig. 1 and Fig. 2(a), shows its specification file \(\text{EXP.DAT}\). Now, the automated aid is invoked by calling its command file \(\text{AA}\) in DOS environment. \(\text{AA}\) generates the code-segment part in \(\text{AFCS.ASM}\) and data part with initialization in \(\text{AFCS.DSG}\). The data part and the
A short-period approximation of longitudinal dynamics of F-8 aircraft has been taken as the plant for the design of the observer. The signature Camberra metric\textsuperscript{12} is filtered by a geometric moving average filter.

If the system and observer states are $X$ and $X$, then the difference between the outputs of the system and observer will be $C(X-X)$, so that the observer is given by

$$\hat{x} = A\hat{x}(t) + BU(t) + L[Y(t) - C\hat{x}(t)]$$

$$= (A-LC)\hat{x}(t) + BU(t) + LY(t).$$

Where $X = [q N z e]^{T}$

$$L = \begin{bmatrix} 98.33 & -584400 & 146.1 \end{bmatrix}^{T}$$

By using ORACLS\textsuperscript{13} and data given by Eqns 2 to 4, $N_{f}(s)$ is given by

$$N_{f}(s) = H(s)\delta_{c}(s) + G(s)q(s)$$

where

$$H(s) = \frac{1362[s^{2} + 98.9725 - 70881]}{(s + 112.5)(s + 457095) + 64730}$$

$$G(s) = \frac{-595500(s + 13.333)(s + 8.716)}{(s^{2} + 112.5s^{2} + 457095s + 64730)}$$

6.2 $N_{f}$ Sensor Failure Detection

For detecting $N_{f}$ sensor failure, the signature Camberra metric. $S = (N_{f} - N_{f}\text{sense})/(N_{f} + N_{f}\text{sense})$ is used. A fixed weight $W$ is given to current observation of $S$ and complementary weight $(1-W)$ is given to geometric average of all the subsequent observations. Then geometric average at $k$th sampling instant is given by $S_{k}^{*} = W S_{k} + (1-W) S_{k-1}^{*}$.
The code-segment AFCS.ASM generated by the automated-aid
The geometric moving average process is considered equivalent to operation of filter where the output is given by

\[ y(n) = Wx(n) + (1-W)y(n-1) \]

The transfer function of the equivalent analog filter is \((s + (2/T))/((s(2W)/W)(2/T))\). The observer forms a subclass of the decision block specified as DO<ALPHANUM>. The full syntax of the observer block is so defined to enable extraction of control input, normal sensor signal, back-up signal, output signal, limits for switching to analytical value, and time after which the observer is to be used. The time factor is to ward off the false alarm which may be present either during the initial asymptotic region or whenever a new model of the plant is inserted due to change in the flight condition. This parameter may not be required if interpolation of the observer can be done at an acceptable rate.

6.3 Timing-fault, Value-fault and Self-test

The automated aid, apart from translating the FCS block diagram, enables detection of two important types of faults as a part of in-flight monitoring, viz. value faults and timing faults. This is done through a set of pre-defined macros. It is possible to obtain a minimal instruction set by which one can express the complete set of predefined macros. This way of organizing the translator enables easy design of processor self-test.

6.3.1 Timing-fault Detection

If the execution time lies between a maximum of \(t_{\text{max}}\) and a minimum of \(t_{\text{min}}\) and the current sample of the execution time of the given control law is \(T_t\) then if \((t_{\text{min}} < T_t < t_{\text{max}})\) then timing fault is false; else timing fault is true.
Determination of \( t_{\text{min}} \) and \( t_{\text{max}} \) has to be carried out based on macros defined for various blocks. Since the macrocall contains relevant information, it will be easy to obtain the execution time by the translator itself. Table 2 gives the list of macros that have been used and the execution time in terms of number of clock cycles of 8086/8087 pair. The translator obtains the total execution time in terms of number of clock cycles. The execution time found out could set a watch dog timer.

Verification of timing-fault detection has been done by an interrupt service routine which makes use of timer ticks available on IBM PC.

Table 2. List of MACROS and their execution time

<table>
<thead>
<tr>
<th>MACRO</th>
<th>EXECUTION TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>17 + (21 + 150 \cdot \text{TOT}) \cdot \text{ORDER}</td>
</tr>
<tr>
<td>UPDATE</td>
<td>144 \cdot \text{N}</td>
</tr>
<tr>
<td>AMPL</td>
<td>268 + (\text{NUM} - 1) \cdot 266</td>
</tr>
<tr>
<td>SUMPN</td>
<td>109 + \text{S} \cdot 139</td>
</tr>
<tr>
<td>GAIN</td>
<td>268</td>
</tr>
<tr>
<td>GAINC</td>
<td>494</td>
</tr>
<tr>
<td>OBS</td>
<td>667</td>
</tr>
<tr>
<td>CMP</td>
<td>658</td>
</tr>
<tr>
<td>FILTER (ORDER = 1)</td>
<td>884 \cdot \text{T}</td>
</tr>
<tr>
<td>(ORDER = 2)</td>
<td>1410 \cdot \text{T}</td>
</tr>
<tr>
<td>(ORDER &lt; 6)</td>
<td>113 + 290 \cdot (2 \cdot \text{ORDER} + 1)</td>
</tr>
<tr>
<td>SWITCH</td>
<td>153 + 436 \cdot \text{I}</td>
</tr>
<tr>
<td>SCH1 (1 GRID)</td>
<td>3186</td>
</tr>
<tr>
<td>GAIN BLOCK</td>
<td>3798 + 2237 \cdot \text{ORDER}</td>
</tr>
<tr>
<td>FILTER BLOCK</td>
<td>6963</td>
</tr>
<tr>
<td>SCH2 (2 GRID)</td>
<td>7002 + 4750 \cdot \text{ORDER}</td>
</tr>
<tr>
<td>GAIN BLOCK</td>
<td></td>
</tr>
<tr>
<td>FILTER BLOCK</td>
<td></td>
</tr>
</tbody>
</table>

\( \text{TOT} \): Numbers of inputs and outputs connected to filter blocks of a given order greater than 2; \( \text{I} \): number of throws to the switch block; \( \text{N} \): 1 or 2 (filter blocks of order less than 3; \( \text{NUM} \): number of inputs to an amplifier; \( \text{S} \): number of inputs to a summing block.

6.3.3 Instruction Set Self-test

The instruction set self-test is often monitored in-flight\(^6\). It will be possible to obtain the basic set (Fig. 3) which has been used for defining the macros. An efficient test-program containing the basic instructions is then executed to generate certain test results and are checked against the expected values. Any deviations can be used to change the flag conditions.

7. VALIDATION OF THE TRANSLATOR

The model following Cstar controller sketched in Fig. 1 forms the FCS block diagram to be used for validation. Simulation based on the discrete state-space model of Cstar controller of short period dynamics of F–8 aircraft is compared with that generated by the automated aid at different sampling rates. As a part of the validation, the \( N_z \) sensor fault-detection is introduced and the performance of the closed loop with reconstructed \( N_z \) is compared with that of discrete simulation. Timing-fault and value-fault detection algorithms are next tested.

7.1 Reduction to Standard Form

The problem of discretizing the continuous time state equations \( \dot{X} = AX + BU \) and \( Y = CX \) consists of finding a discrete-time model \( X(k + 1)T = MX(k)T + NU(k)T \) and \( Y(k)T = CX(k)T \), where \( X(k)T \) and \( X(k + 1)T \) represent state vectors at sampling instants \( kT \) and \( (k + 1)T \). The solutions of \( M \) and \( N \) are well known and are given by \( M = \exp(At) \) and \( N = \int_0^T \exp(Av) \cdot dv \cdot B \). Figures 4 and 5 represent the standard forms of the regulator-plant cascade. The regulator and plant portions have subscripts \( r \) and \( p \) respectively. The following are the notations: input and output vectors \( U_r, Y_r \); feedback error signals \( U_r, U_p \), state vectors \( X_r \) and \( X_p \); regulator is specified by \( A_r, B_r, C_r \) and \( D_r \) and plant specified by \( A_p, B_p, C_p \) and \( D_p \).

The F–8 Cstar controller block diagram has to be reduced to the state-space form

\[
\begin{bmatrix}
\dot{X}_r \\
\dot{X}_p
\end{bmatrix} = A \begin{bmatrix} X_r \\ X_p \end{bmatrix} + B U_p
\]

It can be shown that\(^{14}\)
The following are the basic instructions that have been used for designing macros. \( \text{memb}, \text{memw}, \text{memr} \) refer to memory byte location, memory word location and short-real memory location respectively. \( \text{imm} \) refers to immediate value as one of the operands.

\[
\begin{align*}
\text{ADD SI,imm} & \quad \text{ADD BX,imm} & \quad \text{ADD SI,memw} \\
\text{ADD memw,AX} & \quad \text{ADD AX,memw} & \quad \text{ADD memw,imm} \\
\text{procedure} & \quad \text{memw,imm} & \quad \text{CMP memb,imm} & \quad \text{CMP SI,imm} \\
\text{CX} & \quad \text{DEC memw} \\
\text{FADD memr} & \quad \text{FADDP ST(1),ST} & \quad \text{FCOMP memr} \\
\text{FCOMP memr[SI]} & \quad \text{FDIV memr} & \quad \text{FILD memw} \\
\text{FISTP memw} & \quad \text{FLD mem} & \quad \text{FLD mem[4]} \\
\text{FLD mem[SI][4]} & \quad \text{FLD mem[SI][ROM]} & \quad \text{FLDZ} \\
\text{FMUL memr} & \quad \text{FMUL memr[4]} & \quad \text{FMUL memr[SI]} \\
\text{FMULP ST(1),ST} & \quad \text{FST memr} & \quad \text{FSTP memr[4]} \\
\text{FSTP memr} & \quad \text{FSTP memr[SI]} & \quad \text{FSTP memr[BX]} \\
\text{FSTSW memw} & \quad \text{FSUBP ST(1),ST} & \quad \text{FSUB memw} \\
\text{AX} & \quad \text{JBE short label} & \quad \text{JE short label} & \quad \text{JMP short label} \\
\text{JMP far ptr label} & \quad \text{JNE label} & \quad \text{LOOP label} \\
\text{MOV AL,imm} & \quad \text{MOV AX,SEG ESEG} & \quad \text{MOV AX,memw} \\
\text{MOV BX,imm} & \quad \text{MOV CL,imm} & \quad \text{MOV CX,imm} \\
\text{MOV DS,AX} & \quad \text{MOV ES,AX} & \quad \text{MOV SI,imm} \\
\text{MOV SI,memw} & \quad \text{MOV SI,Ax} & \quad \text{MOV SI,CX} \\
\text{MOV memb,imm} & \quad \text{MOV memw,imm} & \quad \text{MOV memw,SI} \\
\text{MUL memw} & \quad \text{RET} \\
\text{AX,CL} & \quad \text{SHL SI,imm} & \quad \text{SUB SI,imm}
\end{align*}
\]

Figure 3. The set of instructions of 8086/8087 processors that are used in the macros.

\[
A = \left[ \frac{(A_p + B_p K_p)}{O} \right] = \left[ \frac{B_p C_r}{A_r} \right]
\]

\[
B = [B_p D_r : B_r]^T
\]

\[
X = [q N_2 \delta_e N_1 N_1 1 1 q 1 p]^T \text{where } p = \delta_e(s)/(K_1 s + K_2)
\]

\[
X = [z z z z z] \text{and } c(s)/(0.3493 s^3 + 6.538 s^2 + 45.56 s + 55) = U(s)/(0.41767 s^5 + 6.562 s^4 + 36.26 s^3 + 55) = Z
\]

Equation (6) represents the standard form and one can obtain its discrete time model. ORACL-based programs\(^\text{14}\) have been developed which give the discrete

Figure 4. Regulator and plant cascade.

\[127\]
Figure 6. Block diagram of F-8 Cstar controller with sensor failure detection and reconfiguration.

Figure 7. The specification file corresponding to the block diagram sketched in Fig. 6 block DFO performs fault-detection and reconfiguration. It can be noticed that the Cambera metric is used as the signature. The filter F-11 implements the geometric moving average of the signature discussed earlier. The results of the

time model of the Cstar controller and the responses of $q, N_f$ for a step input at various sampling rates. These are stored for comparison with the Tustin equivalent. Figures 6 and 7 show the block diagram with sensor
sensor failure and reconfiguration at the sampling rate of 50 Hz is shown in Fig. 8 and it has been compared to the healthy sensor case.

Figure 8. $N_\theta$ response with sensor failure at 2 s.

8. CONCLUSIONS

The automated aid (i) can handle most of the blocks that exist in FCS, (ii) does not need sophisticated work station and can work on IBM PC or INTEL microprocessor development systems and (iii) has potentiality for use in autonomous systems and can aid in generation of N-version program for improved reliability.

ACKNOWLEDGEMENTS

The authors would like to thank the Director, NAL and Head, ALD for permission to publish this paper. The authors would also like to thank Dr A Krishnan, Dr A Pedar, Mr Shyam Chetty and Mr A Shartharam for helpful discussions.

REFERENCES