Development of a Helicopter Emergency Maneuvers Trainer

by

Bruce Charles Haycock

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Institute for Aerospace Studies
University of Toronto

0-612-62891-4

Development of a Helicopter Emergency Maneuvers Trainer

Degree of Master of Applied Science, 2001

Bruce Charles Haycock

Institute for Aerospace Studies

University of Toronto

Abstract

This thesis describes the development of a generic helicopter code to be used as a helicopter emergency maneuvers trainer for piloted flight simulations. The helicopter employed in the current simulation is a Bell 205, with a two-bladed teetering rotor and skid-type landing gear.

In order to create a model with clearly defined modules and full documentation, the physical model was implemented using the commercial software package MATRIXx, a graphical based program capable of simulating dynamic systems and able to generate C-code for use in a real-time simulation.

The simulation model has been implemented and run in real-time on the UTIAS Flight Research Simulator. Piloted simulations have demonstrated a wide flight envelope including typical operating regimes as well as basic emergency maneuvers.

Acknowledgements

I would like to thank my supervisor, Prof L. D. Reid for his guidance and support throughout the project. I also gratefully acknowledge the assistance of Mr. Wolfgang Graf for his assistance with the simulator hardware, as well as Andrew Pierce for his assistance in the early stages of the project, and the Natural Sciences and Engineering Research Council for financial support.

Contents

List of Figures

1. Introduction

This project was undertaken to fill a void in the available resources for helicopter training. There is currently a very comprehensive and well-established training syllabus for helicopters using actual aircraft, as well as some limited use of flight simulators to augment the actual flying time. However, due to the complex nature of helicopters, the simulation model used is simplified in order to allow the simulation to run in real time. As a result, the flying characteristics are realistic within the context of the simplifications, but cannot be expected to realistically present all possible flight conditions.

The motivation for this thesis was to develop a flight simulation model that was general enough to be used across the entire flight envelope, including a variety of emergency maneuvers such as autorotations and tail rotor failures. This model would be developed using a MATRIXx software package rather than straight programming using Fortran or C code. The simulation can then be used to test the effectiveness of a simulator in training helicopter emergency maneuvers. For this simulation, the main rotor was modeled as a two-bladed teetering rotor and the landing gear consists of a pair of skids. In this configuration, it is possible to simulate helicopters such as the Bell 205 and Bell 206.

The emergency maneuvers to be included in the simulation flight envelope are a highly important element in the training of a helicopter pilot. In the event of an engine failure, the helicopter must be autorotated to carry out a landing. An autorotation in a helicopter is akin to a gliding approach to landing in a powered fixed-wing aircraft. In an autorotation, upwards flow through the rotor drives the main rotor, allowing it to rotate, providing lift and thus reducing the rate of descent. In this scenario, the inner portion of a rotor blade is stalled, the middle portion produces a blade force component in the direction of rotation driving the rotor, and the outer portion produces the majority of the lift. Once close to the ground, kinetic energy stored in the rotating rotor is used to slow the helicopter and reduce the rate of descent, thus cushioning the landing (see Figure 1.1).

Emergency procedures involving loss of power or loss of a tail rotor present a unique set of problems that make the use of simulators highly desirable. There are all the typical advantages of flight simulators, such as full control over the surrounding environment and reduced cost due to the high operating cost of helicopters from fuel and maintenance costs, but in addition autorotations are relatively risky and must be practiced with caution. If the flare to landing is performed incorrectly, a hard landing is often the result. Once the kinetic energy stored in the blade has been used to reduce the rate of descent, even with an operational engine, it requires some time to regain rotor momentum. If the flare is performed incorrectly and touchdown is not accomplished at the desired time, there may not be enough energy in the rotor to prevent a hard touchdown. Looking through the NTSB online database from the United States turns up 19 accident reports as a result of practice autorotations in 2000 alone. As such, the use of simulators is much safer, eliminating the possibility of damaging expensive equipment due to hard landings in an incorrectly performed autorotation to landing.

This simulation of a Helicopter Emergency Maneuvers Trainer (HEMT) was developed at UTIAS under contract to the Defense and Civil Institute of Environmental Medicine (DCIEM). The contract report "Flight Dynamics for Helicopter Emergency Maneuvers Trainer", DCIEM Contract W7711-005923/001/TOR (Reference 3.1) contains a complete description of the physical model used in the simulation as well as documentation of the entire MATRIXx model. Although this thesis contains portions of the same material (notably Section 3) for completeness, the contract report should be referred to for the complete model, while this thesis contains material not found in the contract report pertaining to testing and development work.

The model is implemented on the six-degree-of-freedom full motion simulator located at the University of Toronto Institute for Aerospace Studies shown in Figure 1.2. A binocular three-dimensional visual scene is presented to the pilot using a fiber optic helmet mounted display, which is described in Reference 1.1, and control forces are provided through the use of a McFadden Control Loader.

1.1 References

1.1 Reid, L.D., Sattler, D.E., Graf, W.O., Dufort, P.A., and Zielinski, A.W., 1998, "Cockpit Technology Simulation Development Study for Enhanced/Synthetic Vision System", UTIAS Report No. 355.

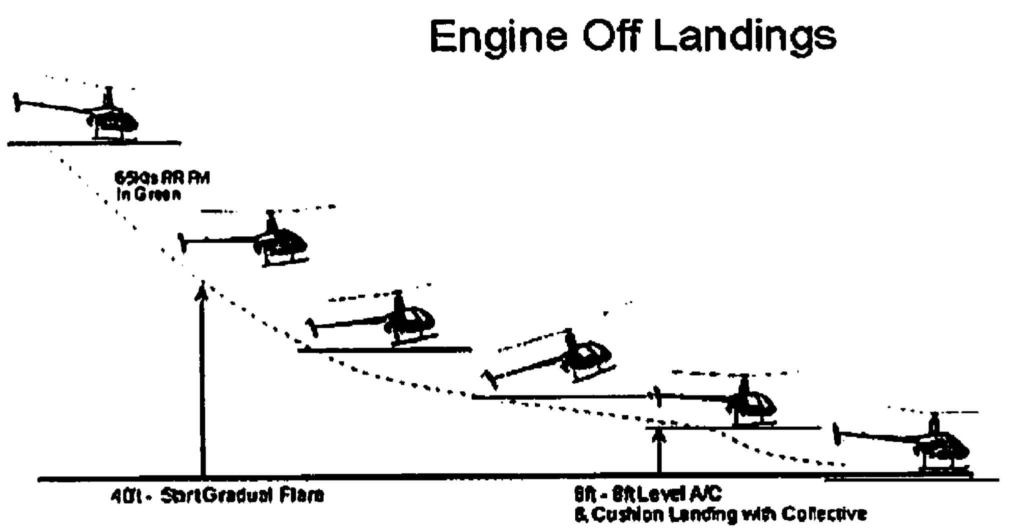


Figure 1.1. Autorotative Landings

Figure 1.2 UTIAS Flight Simulator

2. MATRIXx

As mentioned above, it was decided to develop a full new simulation implemented in the MATRIXx environment rather than modify existing code for this thesis. This allows for the creation of a model with clearly defined modules and complete documentation. In this way, it was also possible to have full control over the simulation with the freedom to change any portion if desired, plus the familiarity with the entire system to be able to make these changes. As well, the end user contracting the project required a code it could have full control over without limitations.

Although MATRIXx has been used in the development of many simulations, it has not been used for a helicopter simulation, so this allowed an opportunity to explore if it was possible and identify any limitations. In addition, many companies in the industry use MATRIXx to develop new simulator codes. As such, its use follows current industry practice.

The simulation has been developed in a MATRIXx Version 6.1 software environment installed on an SGI Indigo 2 workstation. This product has been created by Integrated Systems Inc. (see Reference 2.1) and is suited to real-time flight simulator applications. Three elements of the MATRIXx Product Family have been employed:

(i) Xmath

(ii) SystemBuild

(iii) Autocode

Xmath software provides a system analysis and visualization environment. It contains over 700 predefined functions and commands, interactive color graphics and a programmable graphical user interface. SystemBuild visual modeling and simulation software allows interactive model verification, testing and modification. The Xmath interpreter supports a multiprocess architecture that consists of three executable programs: Xmath, SystemBuild and the Simulator. It uses Interprocess Communication to coordinate activities and transfer data. Autocode is an automatic code generator for SystemBuild models. It outputs compilable C or Ada code. The Autocode output can be compiled to produce the real-time flight simulation.

The flight simulation model is created as a series of interconnected blocks displayed on block diagram pages. In the MATRIXx environment Super Blocks are special blocks made up of Blocks or other Super Blocks. Blocks are used to specify the operations that constitute the system model. The block diagrams in Appendix B illustrate their use. The first page of the sequence gives the top level Super Blocks. The contents of each of these Super Blocks are given in sequence on the following pages. The name of each Super Block appears at the top of its block and this name appears at the top of the block diagram page giving its contents. Blocks containing the label *Block Script* are the only ones that require further description outside the pages of block diagrams. The operations carried out in a Block Script block are described in block script code sheets following the block diagram pages. Each such block is identified by the name at the top of the block and this name also appears at the top of the page containing its block script code. This is the computer code which Xmath supports. If a block is neither a Super Block nor a Block Script block, then its function is defined by its physical shape and labels.

Inputs to a block come from the outputs of other blocks. This is indicated by the interconnections between the blocks in the block diagrams. This transfer can also be achieved by processing one block's output variable by a Write to Block and then using a Read from Block to pick up the variable and feed it into the input of another block. In the case of fixed parameters and numerical constants, they can be input to a Block Script block through *parameter* and *environment* lists in the block script code.

All blocks except Block Script blocks can employ matrices as inputs and outputs. In the case of Script Block blocks the matrices must be converted into one-dimensional arrays (or vectorized) and this form is used for the input and output processes. An example of this can be seen in the block script code L_BL1_B in Appendix C.

In the diagram of Block 1.2.1 (Blade Transformation Matrices) in Appendix B the Block Script block L_BL1_B Matrix is shown with a matrix input and a matrix output. However, when the block script code for this block is implemented it can be seen that the input and output are vectorized versions of the matrices (indicated by the letter V following their names). These arrays are sequences of 9 numbers with the first 3 representing the first row of the matrix, the second 3 representing the second row of the matrix and the last 3 representing the third row of the matrix. In order to carry out matrix manipulations within the block script code, the input is first converted back into a matrix by using two nested *do loops*. The final step before leaving the block script code is to vectorize the output matrix, creating L_BL1_B_V(M).

Further details concerning the MATRIXx blocks and other features are contained in Section 3.

2.1 Algebraic Loops

In MATRIXx an algebraic loop occurs when the output of a block is fed back as an input. For people used to programming in languages such as FORTRAN or C, problems associated with such loops come as a surprise. In these languages the program automatically assigns a previous value to such a fed back variable and no problem arises. However MATRIXx does not do this automatically, and will give an error message if you construct such an algebraic loop. Several techniques to circumvent such algebraic loops were tested for the present application, including the use of MATRIXx State variables, as discussed in Section 4. Only one method proved to be universally successful in removing the loop without introducing or allowing other unwanted changes to occur in the program. This method replaces feedback signal paths in the MATRIXx block diagrams with a *Write to* Block followed by a *Read from* Block. This acts as a feedback loop with a single step delay applied to the fed back variable, as required to break the algebraic loop. The disadvantage of this is that the feedback path does not appear explicitly as a line on the block diagram. However no suitable alternative could be found.

A related rule in MATRIXx is that the same name cannot be assigned to both an input and an output from the same Block Script block.

2.2 References

2.1 —— 1996, "MATRIXx Product Family; Getting Started (UNIX).

3. Helicopter Model

The helicopter model is described in its entirety in Reference 3.1. Included below is a partial description, including the overall layout and a few sections of interest. The main rotor and landing gear modules are included in full, as this is where a large amount of work was required and will be discussed later in this thesis.

The model is split into sections representing the various components of the helicopter, such as the main rotor, tail rotor, and landing gear. In addition, there are sections for the startup calculations, interface with the real-time simulator, atmospheric calculations, and a flight control system. This flight control system contains two unique modes, the first being a bare-airframe system, where pilot controls are fed directly to the simulation, and the second is a Translational Rate Command (TRC) flight controller. In this TRC mode, the pilot's inputs govern various system rates, with the controller providing the suitable control inputs to initiate and maintain that rate.

The top-level Super Block of the model is shown in Figure 3.1. This shows the various subsystems mentioned above and the interconnections among them. In addition, all external inputs and outputs from the model are shown. Along with the basic pilot control inputs, a number of logic flags are available to the simulator operator, represented by the inputs IFCS, ICLUTCH, IFUEL, ITAIL, and KASE. IFCS turns the TRC flight controller on or off, KASE represents different sling load cases, including pick-up, carrying, and dropping of the load, and the remaining inputs control failure modes for emergency maneuvers training. ICLUTCH disengages the engine from the rotor, and can be used to simulate sudden loss of all engine power. IFUEL stops the flow of fuel to the engine, this also simulates loss of engine power, but power is lost through a more gradual decay rather than a sudden drop, and engine rpm remains coupled to rotor rotation. ITAIL is used to simulate sudden loss of tail rotor thrust. Tail rotor forces and moments acting on the body are set to zero, as is the torque applied to the engine and the angular momentum contribution.

The following two sections describe the physical models used for the Main Rotor and Landing Gear. These two sections (3.1 and 3.2) are then further subdivided into three parts. The first subsection describes the physical model with all relevant equations. The second contains notes pertaining to the MATRIXx implementation of the model, pointing out those features that may not be self-explanatory and unusual blocks employed. The actual MATRIXx Super Blocks and Block Script block code are included in the appendices at the end of this thesis. The third subsection lists all notation used in the model, with both the symbols from the physical equations and the corresponding computer name. In addition, Appendix A contains background information such as notation, a description of computer variable names, and numerical techniques.

3.1 Main Rotor

3.1.1 Physical Model

A blade element model based on Reference 3.2 is used to represent the main rotor. Each blade is divided up into elements and these elements are treated as independent two-dimensional airfoils. In the present simulation a two blade teetering rotor is used. The overall forces and moments produced by the main rotor are found by summing up the contributions from all the elements. The main rotor is treated as an independent dynamic system with its angular velocity determined by the Power Train Module. Its forces and moments are transmitted to the helicopter body via the hub and rotor mast. A teetering rotor has two angular degrees-of-freedom; flapping and rotation about the mast (see Reference 3.4).

3.1.1.1 Rotor Blade Geometry

In order to apply the blade element theory it is first necessary to specify the geometric properties of each blade and element. The generalized blade is presented in Figure 3.1.1. Each blade is composed of NL elements. The boundaries between blade elements are selected such that each element sweeps out an annulus of area $\pi\delta$ as it completes one revolution about the mast. The mid-point of the j-th element is defined by the radius $r_m(j)$ and it is selected so that half the area of the j-th annulus lies outside the circle swept out by the mid-point. From the above it follows that based on areas (where j = 1 for the element closest to the hub and j = NL for the element at the rotor tip):

$$\pi r_m^2(j) = \pi r_m^2(j-1) + \pi\delta$$

or

$$r_m^2(j) = r_m^2(j-1) + \delta \tag{3.1.1}$$

and from Figure 3.1.1

$$\pi R^2 = \pi r_o^2 + \pi NL\delta \tag{3.1.2}$$

where R is the main rotor radius. From (3.1.2)

$$\delta = (R^2 - r_o^2) / NL \tag{3.1.3}$$

and from Figure 3.1.1

$$\pi r_m^2(1) = \pi r_o^2 + \pi\delta / 2$$

or

$$r_m^2(1) = r_o^2 + \delta / 2 \tag{3.1.4}$$

In the above (3.1.3) is used to specify δ, (3.1.4) is used to find $r_m(1)$ and (3.1.1) is used to find $r_m(j)$ for $j \neq 1$. From Figure 3.1.1 the inner radius $r_{IN}(j)$ and outer radius $r_{OUT}(j)$ of each element can be found based on annular areas, to be

$$r_{IN}^2(j) = r_m^2(j) - \delta / 2 \tag{3.1.5}$$

$$r_{OUT}^2(j) = r_m^2(j) + \delta / 2 \tag{3.1.6}$$

The mean chord and area of each element can be found using Figure 3.1.1. Let the inner (towards the hub) and outer (towards the tip) chords of an element be $c_{IN}(j)$ and $c_{OUT}(j)$ respectively. Each element is a trapezoid and thus the area of the j-th element is given by

$$S(j) = \left(c_{IN}(j) + c_{OUT}(j)\right)\left(r_{OUT}(j) - r_{IN}(j)\right) / 2 \tag{3.1.7}$$

and define the mean chord as

$$\tilde{c}(j) = \left(c_{IN}(j) + c_{OUT}(j)\right) / 2 \tag{3.1.8}$$

Thus

$$S(j) = \tilde{c}(j)\left(r_{OUT}(j) - r_{IN}(j)\right) \tag{3.1.9}$$

From Figure 3.1.1 the chord c at any radius r is given by

$$c = C_R - \frac{(r-r_o)}{(R-r_o)}(C_R - C_T) \tag{3.1.10}$$

From (3.1.8) and (3.1.10)

$$\tilde{c}(j) = C_R - \frac{(C_R - C_T)}{2(R-r_o)}\left(r_{IN}(j) + r_{OUT}(j) - 2r_o\right) \tag{3.1.11}$$

Thus (3.1.11) is used to find $\tilde{c}(j)$ and then (3.1.9) is used to find $S(j)$.

3.1.1.2 Rotor Blade Kinematics

The main rotor is located at the top of the rotor mast. The origin of reference frame F_S is located at the center of the hub of the main rotor at the top of the mast as shown in Figure 3.1.2 with its z-axis along the mast. A body-fixed reference frame F_B is located with its origin at the CG of the helicopter body (i.e., the helicopter less the main rotor system). The origin of F_S is located by the vector **rS** from the origin of F_B (see Figure 3.1.3). It is assumed that the mast is tilted relative to F_B such that the Euler angle which carries F_B into F_S is given by

$$\underline{E}_S = \begin{bmatrix} 0 & \theta & 0 \end{bmatrix}_S^T \tag{3.1.12}$$

The y-axis of F_S is parallel to that of F_B.

A rotating reference frame F_R is defined as shown in Figure 3.1.4. F_R has its origin at the origin of F_S and their z-axes are coincident. The y-axis of F_R is aligned with Blade 1 of the rotor and follows it in rotation (actually the position of Blade 1 when its flapping angle is zero). The azimuthal angle ψ is shown in Figure 3.1.4 following the standard helicopter convention.

Both ψ and main rotor angular velocity relative to the helicopter body (Ω) are positive for counterclockwise rotation of the rotor when viewed from above. This is the assumed direction of rotation in this simulation. Ω is the time derivative of ψ. The Euler angle which carries F_S into F_R is given by

$$\underline{E}_R = \begin{bmatrix} 0 & 0 & (\pi/2 - \psi) \end{bmatrix}^T \tag{3.1.13}$$

An additional reference frame F_{IN} is specified in Figure 3.1.5. Its origin is located at the origin of F_S (the hub) with the z-axes of F_{IN} and F_S are coincident. The x-axis of F_{IN} is in the direction of the projection of the hub's instantaneous airspeed vector ($\underline{US}_S$) on the x-y plane of F_S. The angle β_W is as shown in the figure where

$$\beta_W = \tan^{-1}(US_{yS} / US_{xS}) \tag{3.1.14}$$

$\underline{US}_S$ can be found in Section 3.1.1.4. The Euler angle which carries F_S into F_{IN} is given by

$$\underline{E}_{IN} = \begin{bmatrix} 0 & 0 & \beta_W \end{bmatrix}^T \tag{3.1.15}$$

3.1.1.3 Pitt/Peters Dynamic Inflow Model

In the process of developing thrust, the main rotor induces an additional inflow through the rotor disc. This inflow velocity V_{IN} is positive when it flows down from above and through the disc. It is normally positive. V_{IN} is assumed to be parallel to the main rotor mast. Figure 3.1.6 from Reference 3.3 shows typical flow patterns through a rotor. In general the inflow is not uniform over the rotor disc and it responds dynamically to changes in the thrust and moments being generated by the main rotor. The Pitt/Peters Dynamic Inflow Model captures these features and it is described in References 3.3 and 3.5. In this model the inflow velocity over the main rotor disc is given by (at location r, ψ_{IN}):

$$V_{IN} = V_o + \frac{r}{R}(V_1 \sin\psi_{IN} + V_2 \cos\psi_{IN}) \tag{3.1.16}$$

The inflow azimuthal angle ψ_{IN} is associated with F_{IN} as shown in Figure 3.1.5. The radii r and R are shown in Figure 3.1.1. V_0, V_1 and V_2 come from the solution of the Pitt/Peters differential equations

$$\underline{VPP} = [V_0 \quad V_1 \quad V_2]^T \tag{3.1.17}$$

$$\underline{M}\left[\frac{\dot{\underline{VPP}}}{\Omega^2 R}\right] + \underline{L}^{-1}\left[\frac{\underline{VPP}}{\Omega R}\right] = [C_{TA} \quad C_{LA} \quad C_{MA}]^T \tag{3.1.18}$$

where:

Ω	is the main rotor angular velocity about the mast, relative to F_B
C_{TA}	is the main rotor thrust coefficient given by

$$C_{TA} = T_A / \rho\pi R^2 (\Omega R)^2 \tag{3.1.19}$$

and T_A is the main rotor thrust (positive along the negative z-axis of F_{IN}; see Figure 3.1.5)

C_{LA}	is the main rotor aerodynamic rolling moment coefficient given by

$$C_{LA} = L_A / \rho\pi R^2 (\Omega R)^2 R \tag{3.1.20}$$

and L_A is the main rotor aerodynamic rolling moment (positive about the x-axis of F_{IN})

C_{MA}	is the main rotor aerodynamic pitching moment coefficient given by

$$C_{MA} = M_A / \rho\pi R^2 (\Omega R)^2 R \tag{3.1.21}$$

and M_A is the main rotor aerodynamic pitching moment (positive about the y-axis of F_{IN})

The aerodynamic force and moments T_A, L_A and M_A are computed in Section 3.1.1.8.

Equations (3.1.18) to (3.1.21) are rearranged for solution as

$$\underline{\dot{VPP}} = \underline{M}^{-1}\left(\left[T_A \quad L_A/R \quad M_A/R \right]^T / \rho\pi R^3 - (\Omega R \underline{L}^{-1}) \underline{VPP} / R \right) \tag{3.1.22}$$

In (3.1.22) the following holds

$$\underline{M}^{-1} = \begin{bmatrix} 75\pi/128 & 0 & 0 \\ 0 & -45\pi/16 & 0 \\ 0 & 0 & -45\pi/16 \end{bmatrix} \tag{3.1.23}$$

and define the matrix

$$LINV(L, M) = \Omega R \underline{L}^{-1} \tag{3.1.24}$$

Based on Reference 3.4 the following holds

$$LINV(1,1) = \left(4V_T \cos\chi_{pp}\right) / D \tag{3.1.25}$$

$$LINV(2,2) = -0.25 V_m K_2 \tag{3.1.26}$$

$$LINV(3,3) = 2 LINV(2,2) / D \tag{3.1.27}$$

$$LINV(1,3) = \left(V_T K_0 \sin\chi_{pp}\right) / D \tag{3.1.28}$$

$$LINV(3,1) = V_m LINV(1,3) / V_T \tag{3.1.29}$$

$$LINV(1,2) = LINV(2,1) = LINV(2,3) = LINV(3,2) = 0 \tag{3.1.30}$$

In order to find the variables on the right-hand side of (3.1.25) to (3.1.29) equations (3.1.31) to (3.1.43) are employed.

$$\mu_{pp2} = US_{xS}^2 + US_{yS}^2 \tag{3.1.31}$$

$$\mu_{pp} = \mu_{pp2}^{1/2} \tag{3.1.32}$$

$$v = T_A / \left(2\rho\pi R^2 \left(\mu_{pp2} + (v - VS_{zS} + WSA_{zS})^2\right)^{1/2}\right) \tag{3.1.33}$$

Equation (3.1.33) must be solved for v. This is accomplished in the MATRIXx code of Section 3.1.3 using the Relaxation Method described in Appendix A.1.

In (3.1.33) $\underline{VS}_S$ is the inertial velocity of the hub (the origin of F_S) and it is developed in Section 3.1.1.4. $\underline{\text{WSA}}_S$ in (3.1.33) is the atmospheric wind at the hub.

$$\lambda_{pp} = v - VS_{zS} + WSA_{zS} \tag{3.1.34}$$

$$\chi_{pp} = \tan^{-1}\left(\mu_{pp} / \lambda_{pp}\right) \tag{3.1.35}$$

$$V_T = \left(\mu_{pp2} + \lambda_{pp}^2\right)^{1/2} \tag{3.1.36}$$

$$V_m = \left(\mu_{pp2} + \lambda_{pp}(\lambda_{pp} + v)\right) / V_T \tag{3.1.37}$$

$$K_o = 15\pi / 64 \tag{3.1.38}$$

$$K_1 = K_o^2 \qquad (3.1.39)$$

$$K_2 = 1 + \cos \chi_{pp} \tag{3.1.40}$$

$$K_3 = 1 - \cos\chi_{pp} \tag{3.1.41}$$

$$D = 2\cos\chi_{pp} + K_1 K_3 \tag{3.1.42}$$

In (3.1.33) v is the uniform inflow predicted by simple momentum theory. λ_{pp} is ($-US_{3S}$) based on v and χ_{pp} is the main rotor wake skew angle based on λ_{pp}. χ_{pp} approaches zero in the hover.

3.1.1.4 *Velocities at the Main Rotor Hub*

In order to develop the dynamics and aerodynamics associated with the main rotor it is necessary to obtain expressions for the inertial velocities and airspeeds over the rotor disc. The airspeed of the hub (the origin of F_S) is **US** and in F_S components it is given by

$$\underline{US}_S = \underline{VS}_S - \underline{WS}_S \tag{3.1.43}$$

where $\underline{VS}_S$ is the inertial velocity of the hub and $\underline{WS}_S$ is the wind speed at the hub. In the present development $\underline{WS}_S$ is composed of atmospheric effects given by **WSA** and the inflow V_{IN} from Section 3.1.1.3. Thus, using (3.1.16) and noting that $r = 0$ at the hub, it follows that

$$\underline{WS}_S = \underline{WSA}_S + \begin{bmatrix} 0 & 0 & K_{GE}V_o \end{bmatrix}^T \tag{3.1.44}$$

In (3.1.44) the uniform inflow V_0 has been reduced due to ground effect (see Section 3.1.1.5) by the scaling factor K_{GE} which depends on main rotor height above ground level. The inertial velocity is found by using (A.6,4) of Reference 3.6

$$\underline{VS}_S = \underline{L}_{SB}\underline{VS}_B \tag{3.1.45}$$

$$\underline{VS}_B = \underline{VB}_B + \underline{\tilde{\omega B}}_B\,\underline{rS}_B \tag{3.1.46}$$

$\underline{VB}_B$ and $\underline{\omega B}_B$ come from the flight equations. From (A.2.4) and (3.1.12) it follows that

$$\underline{L}_{SB} = \begin{bmatrix} \cos\theta_S & 0 & -\sin\theta_S \\ 0 & 1 & 0 \\ \sin\theta_S & 0 & \cos\theta_s \end{bmatrix} \tag{3.1.47}$$

If **WSA** is given as components in the Earth-fixed frame F_E then

$$\underline{WSA}_S = \underline{L}_{SE}\underline{WSA}_E \tag{3.1.48}$$

where

$$\underline{L}_{SE} = \underline{L}_{SB}\underline{L}_{BE} \tag{3.1.49}$$

and $\underline{L}_{BE}$ is available from the Flight Equations.

3.1.1.5 *Ground Effect on Main Rotor*

The effect of the ground on the aerodynamics of the main rotor is predicted by a simple model presented in Reference 3.2. A scaling factor due to ground effect, K_{GE}, is applied to the uniform inflow V_0 produced in Section 3.1.1.3 (as done in (3.1.44)). The variation in K_{GE} with height is based on the height of the main rotor hub above the local ground plane. Using the **rG** of Figure 3.2.1 to specify the location of the local ground plane (assumed to be a horizontal plane) Figure 3.1.3 shows the hub location and the location of the ground plane. The height of the hub above ground level (H_{AGL}) is then given by

$$\begin{aligned} H_{AGL} &= rG_{zE} - rB_{zE} - rS_{zE} \\ &= h_{CG} - rS_{zE} \end{aligned} \tag{3.1.50}$$

where
$$\underline{rS}_E = \underline{L}_{EB}\underline{rS}_B \tag{3.1.51}$$

and h_{CG}, the height of the CG of the helicopter body above local ground level, is calculated in the Flight Equations module. K_{GE} is then given by:

$$K_{GE} = 1 \tag{3.1.52}$$

when $H_{AGL} > 5R$

and

$$K_{GE} = (1 - 0.115(R / H_{AGL})^2 US_{zS} / U_{TOT})^{-2/3} \qquad (3.1.53)$$

when $H_{AGL} \leq 5R$

In (3.1.53)

$$U_{TOT} = |\underline{US}_S| \qquad (3.1.54)$$

It should be noted in (3.1.53) that as US_{zS} approaches zero K_{GE} approaches unity and the ground effect disappears. US_{zS} approaches zero as the wake from the rotor becomes parallel to the plane of the rotor disc.

3.1.1.6 Blade Element Angle of Attack and Mach Number

The aerodynamic forces acting on each blade element depend on the local angle of attack and Mach number. Therefore the latter quantities must be evaluated at the location of each element at all time steps. Start by finding the airspeed of the *j*-th element of the *i*-th blade.

Let F_{Bti} be a blade-fixed reference frame for the *i*-th blade as shown in Figure 3.1.7. The origin of F_{Bti} is located at the hub (the origin of F_S and F_R) and its *y*-axis lies along the blade towards the tip. The *x*-axis of F_{Bt1} is coincident with the *x*-axis of F_R. The *x*-axis of F_{Bt2} is coincident with the negative *x*-axis of F_R. In a teetering rotor the blade motion relative to F_R is a simple flapping about the *x*-axis with the flapping angle β_i (see Figure 3.1.7) defined to be positive when the blade tip is above the *xy*-plane of F_R. Thus the Euler angles which carry F_R into F_{Bti} are given by

$$\underline{E}_{Bt1} = [-\beta_1 \quad 0 \quad 0]^T \qquad (3.1.55)$$

and

$$\underline{E}_{Bt2} = [-\beta_2 \quad 0 \quad \pi]^T \qquad (3.1.56)$$

In the above, β_i comes from Section 3.1.1.11. From (3.1.136), (3.1.137), (3.1.146) and

$$\underline{L}_{B\ell iS} = \underline{L}_{B\ell iR}\underline{L}_{RS} \tag{3.1.57}$$

it follows that

$$\underline{L}_{B\ell 1S} = \begin{bmatrix} \sin\psi & \cos\psi & 0 \\ -\cos\beta_1\cos\psi & \cos\beta_1\sin\psi & -\sin\beta_1 \\ -\sin\beta_1\cos\psi & \sin\beta_1\sin\psi & \cos\beta_1 \end{bmatrix} \tag{3.1.58}$$

and

$$\underline{L}_{B\ell 2S} = \begin{bmatrix} -\sin\psi & -\cos\psi & 0 \\ \cos\beta_2\cos\psi & -\cos\beta_2\sin\psi & -\sin\beta_2 \\ \sin\beta_2\cos\psi & -\sin\beta_2\sin\psi & \cos\beta_2 \end{bmatrix} \tag{3.1.59}$$

The local angle of attack and Mach number are based on airspeed components at the mid-point of each blade element expressed in $F_{B\ell i}$. For the j-th element of the i-th blade the airspeed is

$$\underline{UB\ell ij}_{B\ell i} = \underline{VB\ell ij}_{B\ell i} - \underline{WB\ell ij}_{B\ell i} \tag{3.1.60}$$

where **VBℓij** is the inertial velocity of the mid-point and **WBℓij** is the wind speed at the location of the mid-point.

The inertial velocity $\underline{VB\ell ij}_{B\ell i}$ in (3.1.60) is found by applying (A.6,4) of Reference 3.6 (using the fact that the origins of $F_{B\ell i}$ and F_S are at the same location).

$$\underline{VB\ell ij}_{B\ell i} = \underline{VS}_{B\ell i} + \underline{\widetilde{\omega B\ell i}}_{B\ell i}\,\underline{rmj} \tag{3.1.61}$$

where

$$\underline{VS}_{B\ell i} = \underline{L}_{B\ell iS}\underline{VS}_S \qquad (3.1.62)$$

$$\underline{rmj} = \begin{bmatrix} 0 & r_m(j) & 0 \end{bmatrix}^T \quad (3.1.63)$$

$\underline{\omega B\ell i}_{B\ell i}$ is the angular velocity of $F_{B\ell i}$ relative to F_E, expressed in $F_{B\ell i}$

$\underline{\omega B\ell i}_{B\ell i}$ can be found from the following. The angular velocity of $F_{B\ell i}$ relative to F_E is the sum of the angular velocity of $F_{B\ell i}$ relative to F_R ($\omega\, i$) and the angular velocity of F_R relative to F_E (ω**R**). From Figure 3.1.7 it can be seen that the angular motion of $F_{B\ell i}$ relative to F_R is flapping about the x-axis of $F_{B\ell i}$. Thus

$$\underline{\omega i}_{B\ell i} = \begin{bmatrix} -\dot{\beta}_i & 0 & 0 \end{bmatrix}^T \tag{3.1.64}$$

where $\dot{\beta}_i$ comes from Section 3.1.1.11. Since Ω is the angular velocity (about the negative z-axis of F_S) of F_R relative to F_B and ω**B** is the angular velocity of F_B relative to F_E, it follows that

$$\underline{\omega R}_S = \underline{\omega B}_S + [\,0 \quad 0 \quad -\Omega\,]^T \tag{3.1.65}$$

where

$$\underline{\omega B}_S = \underline{L}_{SB}\,\underline{\omega B}_B \tag{3.1.66}$$

and $\underline{\omega B}_B$ (the inertial angular velocity of the helicopter body) is given in the Flight Equations module. Thus

$$\underline{\omega B\ell i}_{B\ell i} = \underline{\omega R}_{B\ell i} + \begin{bmatrix} -\dot{\beta}_i & 0 & 0 \end{bmatrix}^T \tag{3.1.67}$$

where

$$\underline{\omega R}_{B\ell i} = \underline{L}_{B\ell iS}\,\underline{\omega R}_S \tag{3.1.68}$$

From (3.1.65) and (3.1.68)

$$\underline{\omega R}_{B\ell i} = \underline{L}_{B\ell iS}\left(\underline{\omega B}_S + [\,0 \quad 0 \quad -\Omega\,]^T\right) \tag{3.1.69}$$

WBℓij in (3.1.60) is made up of atmospheric effects **WBℓAij** and the main rotor inflow.

$$\underline{WB\ell ij}_{B\ell i} = \underline{WB\ell Aij}_{B\ell i} + \underline{L}_{B\ell iS}\underline{VINGEij}_{S} \tag{3.1.70}$$

In (3.1.70) $\underline{VINGEij}_S$ is the inflow vector based on (3.1.16) with V_0 attenuated by the ground effect parameter K_{GE} from Section 3.1.1.5. Let

$$VIN_{GE} = K_{GE}V_o + \frac{r}{R}(V_1 \sin\psi_{IN} + V_2 \cos\psi_{IN}) \tag{3.1.71}$$

Since the inflow is parallel to the z-axis of F_S it follows that

$$\underline{VINGEij}_S = [0 \quad 0 \quad VINGEij\,]^T \tag{3.1.72}$$

where $VINGEij$ is the value of (3.1.71) when

$$r = r_m(j) \tag{3.1.73}$$

and (from Figure 3.1.5)

$$\psi_{IN} = \psi + \beta_W \tag{3.1.74}$$

for $i = 1$

and

$$\psi_{IN} = \psi + \beta_W + \pi \tag{3.1.75}$$

for $i = 2$

If the atmospheric wind is expressed in F_E components then in (3.1.70)

$$\underline{WB\ell Aij}_{B\ell i} = \underline{L}_{B\ell iE}\underline{WB\ell Aij}_{E} \tag{3.1.76}$$

where

$$\underline{L}_{B\ell iE} = \underline{L}_{B\ell iS}\underline{L}_{SB}\underline{L}_{BE} \tag{3.1.77}$$

$$= \underline{L}_{B\ell iB}\underline{L}_{BE}$$

using (3.1.47), (3.1.58) and (3.1.59), with $\underline{L}_{BE}$ given in Section 3.8 of Reference 3.1.

The lift and drag on each blade element is based on the application of simple sweep theory (see Reference 3.4). In this theory the spanwise component of airspeed is ignored (i.e., set to zero). The resulting flow diagram is shown in Figure 3.1.8. The angle of attack of the $F_{B\ell i}$ x-axis for the j-th element is given by

$$\alpha ij_x = \tan^{-1}\left(UB\ell ij_{zB\ell i} / UB\ell ij_{xB\ell i}\right) \tag{3.1.78}$$

The zero-lift line ($z\ell\ell$) of this blade element is located relative to the $F_{B\ell i}$ x-axis by the pitch angle θij_b. (When $\theta ij_b = -\alpha ij_x$ the lift on the element is zero.) The contributors to θij_b are described in Section 3.1.1.7. From Figure 3.1.8 it is seen that the angle of attack of the zero-lift line, αij_{2D} is

$$\alpha ij_{2D} = \alpha ij_x + \theta ij_b \tag{3.1.79}$$

The local Mach number is also based on ignoring the spanwise component of airspeed and is given by

$$Mij_{2D} = U2ij^{1/2} / U_{SOUND} \tag{3.1.80}$$

where

$$U2ij = UB\ell ij^2_{xB\ell i} + UB\ell ij^2_{zB\ell i} \tag{3.1.81}$$

3.1.1.7 Blade Pitch

The local blade pitch angle θij_b is made up of control inputs from the pilot and transmitted by the swashplate, and the geometric structure of the blade (we will ignore aeroelastic twisting in the present simulation). Thus we write

$$\theta ij_b = \theta i_c + \theta j_{twist} + \theta_{zll} \tag{3.1.82}$$

where θi_c is the control input for the i-th blade given by

$$\theta 1_c = A_0 - A_1 \cos(\psi + \Delta) - B_1 \sin(\psi + \Delta) \tag{3.1.83}$$

$$\theta 2_c = A_0 + A_1 \cos(\psi + \Delta) + B_1 \sin(\psi + \Delta) \tag{3.1.84}$$

In the above

θ_{zll}	is the pitch offset at the root of the main rotor
θj_{twist}	is the twist in the blade relative to the root, as measured at the j-th element
A_0	is the collective input
A_1	is the lateral cyclic input; a positive value for A_1 produces a positive (right) rolling torque
B_1	is the longitudinal cyclic input; a positive value for B_1 produces a negative (nose down) pitching torque
Δ	is a phase shift intended to reduce response cross-coupling to control inputs

The definitions of A_1 and B_1 reflect the fact that due to the dynamics of the main rotor, control inputs that produce an aerodynamic rolling torque on the rotor disc tend to produce pitching of the thrust vector. The need for Δ indicates that this phasing between input and output is not exactly orthogonal. A negative θj_{twist} is normally employed in order to reduce the lift distribution towards the blade tip and this creates a more uniform inflow and results in improved rotor aerodynamic efficiency.

3.1.1.8 Main Rotor Aerodynamic Forces and Moments

The lift and drag on each blade element is found by using αij_{2D} and Mij_{2D} from Section 3.1.1.6 and a lookup table containing two-dimensional sectional C_L and C_D data for the blade airfoil employed, expressed as a function of α and M, where α is the angle of attack of the zero lift line. Due to the extreme range in angle of attack experienced by a rotor blade, the α range must span $-\pi \leq \alpha \leq \pi$. The orientation of the element's lift (Lij) and drag (Dij) is shown in Figure 3.1.8. These arodynamic forces can be resolved into $F_{B\ell i}$ components to produce the following

$$FAij_{xB\ell i} = Lij \sin \alpha ij_x - Dij \cos \alpha ij_x \tag{3.1.85}$$

$$FAij_{zB\ell i} = -Lij \cos \alpha ij_x - Dij \sin \alpha ij_x \tag{3.1.86}$$

$$FAij_{yB\ell i} = 0 \tag{3.1.87}$$

As suggested in Reference 3.2, the lift towards the tip of the rotor blade will be reduced in general due to three-dimensional flow effects on the real helicopter. This is modeled by the tip loss factor *TLF* which multiplies *Lij* and varies with radius *r* along the blade. *TLF* only differs from unity at radii very near the tip in practice. In the present simulation *TLF* is given by the following.

For blade elements with (see (3.1.6))

$$r_{OUT}(j) \leq PTLR \tag{3.1.88}$$

$$\text{set} \quad TLF(j) = 1 \tag{3.1.89}$$

For blade elements with (see (3.1.5))

$$r_{IN}(j) \geq PTLR \tag{3.1.90}$$

$$\text{set}\quad TLF(j) = 0 \tag{3.1.91}$$

For blade elements which do not satisfy (3.1.88) or (3.1.90)

$$TLF(j) = \left(PTLR - r_{IN}(j)\right)/\left(r_{OUT}(j) - r_{IN}(j)\right) \tag{3.1.92}$$

where in the above

$$PTLR = R \times PTL \tag{3.1.93}$$

and PTL is a parameter specified for the simulated helicopter.

The Lij and Dij are then found from

$$Lij = TLF(j)\left(\frac{1}{2}\rho U2ij \times S(j)\right)Cij_L \tag{3.1.94}$$

$$Dij = \left(\frac{1}{2}\rho U2ij \times S(j)\right)Cij_D \tag{3.1.95}$$

where Cij_L and Cij_D come from the lookup table for $\alpha\, ij_{2D}$ and Mij_{2D}.

The total aerodynamic forces and moments for each rotor blade are found by adding up the contributions from each blade element. Thus the total blade aerodynamic force for the i-th blade is given by

$$\underline{FAT}_{B\ell i} = \sum_{j=1}^{NL} \underline{FAij}_{B\ell i} \tag{3.1.96}$$

The total blade aerodynamic moment for the i-th blade, about the hub, is given by

$$\underline{GAT}_{B\ell i} = \sum_{j=1}^{NL} \underline{GAij}_{B\ell i} \tag{3.1.97}$$

where

$$GAij_{xB\ell i} = r_m(j) FAij_{zB\ell i} \tag{3.1.98}$$

$$GAij_{yB\ell i} = 0 \tag{3.1.99}$$

$$GAij_{zB\ell i} = -r_m(j)FAij_{xB\ell i} \tag{3.1.100}$$

The total aerodynamic forces and moments for the complete main rotor are found by adding up the contributions from both blades. Thus the main rotor aerodynamic force expressed in F_S components is

$$\underline{FA}_S = \underline{\underline{L}}_{SB\ell 1}\underline{FAT}_{B\ell 1} + \underline{\underline{L}}_{SB\ell 2}\underline{FAT}_{B\ell 2} \tag{3.1.101}$$

and the main rotor aerodynamic moment about the hub, expressed in F_S components is

$$\underline{GA}_S = \underline{\underline{L}}_{SB\ell 1}\underline{GAT}_{B\ell 1} + \underline{\underline{L}}_{SB\ell 2}\underline{GAT}_{B\ell 2} \tag{3.1.102}$$

The aerodynamic forces and moments required in Section 3.1.1.3 must be expressed in F_{IN} components. Thus

$$T_A = -FA_{zS} \tag{3.1.103}$$

(since the z-axes of F_S and F_{IN} coincide) and using Figure 3.1.5

$$L_A = GA_{xS}\cos\beta_W + GA_{yS}\sin\beta_W \tag{3.1.104}$$

$$M_A = GA_{yS}\cos\beta_W - GA_{xS}\sin\beta_W \tag{3.1.105}$$

3.1.1.9 Main Rotor Hub Forces

Consider Figure 3.1.9 showing all the external forces acting on the i-th rotor blade. $\mathbf{Fi_G}$ is due to gravity, $\mathbf{Fi_A}$ is the aerodynamic force and $\mathbf{Fi_H}$ is due to the rotor mast. $\mathbf{Fi_{INT}}$ is an internal structural force on the blade due to the other rotor blade since both blades in the teetering rotor are assumed to be part of a single rigid structure. By using Newton's Third Law, **-Fi$_H$** is the force applied by the *i*-th rotor blade to the top of the rotor mast at the hub location. Now apply Newton's Second Law to the *i*-th rotor blade

$$\mathbf{Fi_H} + \mathbf{Fi_A} + \mathbf{Fi_{INT}} + \mathbf{Fi_G} = m_{B\ell}\ \mathbf{aCGi} \tag{3.1.106}$$

where $m_{B\ell}$ is the mass of a single blade and **aCGi** is the inertial acceleration of the *i*-th blade's CG. Now solve (3.1.106) for $\underline{FH}_{B\ell i}$, the components of **Fi$_H$** expressed in $F_{B\ell i}$

$$\underline{FH}_{B\ell i} = m_{B\ell}\underline{aCG}_{B\ell i} - \underline{FAT}_{B\ell i} - \underline{FG}_{B\ell i} - \underline{FINT}_{B\ell i} \tag{3.1.107}$$

where

$$\underline{FG}_{B\ell i} = m_{B\ell}\underline{L}_{B\ell iE}\,\underline{g}_E \tag{3.1.108}$$

and $\underline{L}_{B\ell iE}$ is given by (3.1.77). The expression for $\underline{aCG}_{B\ell i}$ can be found as follows:

Apply (A.6,7) from Reference 3.6 to obtain

$$\underline{aCG}_{B\ell i} = \underline{aS}_{B\ell i} + (\dot{\underline{\tilde{\omega B\ell i}}}_{B\ell i} + \underline{\tilde{\omega B\ell i}}_{B\ell i}\,\underline{\tilde{\omega B\ell i}}_{B\ell i})\underline{rCG}_{B\ell i} \tag{3.1.109}$$

$$\underline{aS}_{B\ell i} = \underline{L}_{B\ell iB}\,\underline{aS}_B \tag{3.1.110}$$

(see (3.1.77) for $\underline{L}_{B\ell iB}$)

$$\underline{aS}_B = \underline{aB}_B + (\dot{\underline{\tilde{\omega B}}}_B + \underline{\tilde{\omega B}}_B\,\underline{\tilde{\omega B}}_B)\underline{rS}_B \tag{3.1.111}$$

$$\underline{aB}_B = \underline{\dot{VB}}_B + \underline{\tilde{\omega B}}_B\,\underline{VB}_B \tag{3.1.112}$$

where $\underline{aB}_B$ comes from the Flight Equations and

$$\underline{rCG}_{B\ell i} = [0 \quad rCG \quad 0]^T \tag{3.1.113}$$

(with rCG a given parameter specifying the location of the blade's CG) where $\underline{\omega B\ell i}_{B\ell i}$ is the inertial angular velocity of the frame $F_{B\ell i}$ given by (3.1.67) and (3.1.69). Now differentiate (3.1.67) to give

$$\underline{\dot{\omega B}\ell i}_{B\ell i} = \underline{\dot{\omega R}}_{B\ell i} + [-\ddot{\beta}_i \quad 0 \quad 0]^T \tag{3.1.114}$$

and differentiate (3.1.69) to give

$$\underline{\dot{\omega R}}_{B\ell i} = \underline{L}_{B\ell iS}(\underline{\dot{\omega B}}_S + [0 \quad 0 \quad -\dot{\Omega}]^T)$$

$$+ \underline{\dot{L}}_{B\ell iS}(\underline{\omega B}_S + [0 \quad 0 \quad -\Omega]^T) \tag{3.1.115}$$

In the above $\dot{\Omega}$ comes from the Power Train. $\underline{\dot{L}}_{B\ell iS}$ can be found by differentiating (3.1.58) and (3.1.59) and noting that

$$\dot{\psi} = \Omega \tag{3.1.116}$$

and from Section 3.1.1.11

$$\dot{\beta}_1 = \dot{\beta} \tag{3.1.117}$$

$$\dot{\beta}_2 = -\dot{\beta} \tag{3.1.118}$$

$\underline{\omega B}_S$ and $\underline{\dot{\omega B}}_S$ come from (recalling that $\underline{L}_{SB}$ is a constant)

$$\underline{\omega B}_S = \underline{L}_{SB}\underline{\omega B}_B \qquad (3.1.119)$$

$$\underline{\dot{\omega B}}_S = \underline{L}_{SB}\underline{\dot{\omega B}}_B \qquad (3.1.120)$$

From differentiating (3.1.58)

$$\underline{\dot{L}}_{B\ell 1S} = \begin{bmatrix} \Omega\cos\psi & -\Omega\sin\psi & 0 \\ \dot{\beta}\sin\beta_1\cos\psi & -\dot{\beta}\sin\beta_1\sin\psi & -\dot{\beta}\cos\beta_1 \\ +\Omega\cos\beta_1\sin\psi & +\Omega\cos\beta_1\cos\psi & \\ -\dot{\beta}\cos\beta_1\cos\psi & \dot{\beta}\cos\beta_1\sin\psi & -\dot{\beta}\sin\beta_1 \\ +\Omega\sin\beta_1\sin\psi & +\Omega\sin\beta_1\cos\psi & \end{bmatrix} \qquad (3.1.121)$$

and from differentiating (3.1.59)

$$\underline{\dot{L}}_{B\ell 2S} = \begin{bmatrix} -\Omega\cos\psi & \Omega\sin\psi & 0 \\ \dot{\beta}\sin\beta_2\cos\psi & -\dot{\beta}\sin\beta_2\sin\psi & \dot{\beta}\cos\beta_2 \\ -\Omega\cos\beta_2\sin\psi & -\Omega\cos\beta_2\cos\psi & \\ -\dot{\beta}\cos\beta_2\cos\psi & \dot{\beta}\cos\beta_2\sin\psi & \dot{\beta}\sin\beta_2 \\ -\Omega\sin\beta_2\sin\psi & -\Omega\sin\beta_2\cos\psi & \end{bmatrix} \qquad (3.1.122)$$

from both blades

$$\underline{fR}_B = -\sum_{i=1}^{2} \underline{L}_{BB\ell i}\underline{FH}_{B\ell i} \qquad (3.1.123)$$

Now from Newton's Third Law

$$\mathbf{F1}_{\mathbf{INT}} = -\mathbf{F2}_{\mathbf{INT}} \qquad (3.1.124)$$

thus $\mathbf{Fi}_{\mathbf{INT}}$ drops out of (3.1.123) and we are left with (combining (3.1.107) to (3.1.124))

$$\underline{fR}_B = -\sum_{i=1}^{2} \underline{L}_{BB\ell i}\,\underline{FHI}_{B\ell i}$$

$$= \sum_{i=1}^{2} \underline{L}_{BB\ell i}\left(\underline{FG}_{B\ell i} - \underline{aS}_{B\ell i} + \underline{FAT}_{B\ell i} - m_{B\ell}\left(\dot{\tilde{\underline{\omega B\ell i}}}_{B\ell i} + \tilde{\underline{\omega B\ell i}}_{B\ell i}\,\tilde{\underline{\omega B\ell i}}_{B\ell i}\right)\underline{rCG}_{B\ell i}\right) \quad (3.1.125)$$

where $\underline{FHI}_{B\ell i}$ is defined by (3.1.125).

3.1.1.10 Main Rotor Hub Moments

Consider Figure 3.1.10 showing all the external moments acting on the *i*-th rotor blade (acting about the origin of $F_{B\ell i}$). $\mathbf{Mi_G}$ is due to gravity, $\mathbf{Mi_A}$ is the aerodynamic moment and $\mathbf{Mi'_H}$ is due to the rotor mast. $\mathbf{Mi_{INT}}$ is an internal structural moment on the blade due to the other rotor blade since both blades in the teetering rotor are assumed to be part of a single rigid structure. By using Newton's Third Law, - $\mathbf{Mi'_H}$ is the moment applied by the *i*-th rotor blade about the top of the rotor mast at the hub location. Note that we are neglecting the moments related to the swashplate inputs to blade pitch and the pitch dynamics of the blade, as being much smaller than the other contributions.

Now we will apply the Extended Euler Equations using the origin of $F_{B\ell i}$ as the reference point for moments and the blade's inertia matrix. These equations result when the moment equations are derived (such as in Reference 3.6) about a reference point other than the body's CG. When the CG is the reference point then equations like (3.8.8) result. In the more general case where the CG is not the reference point (but assuming a rigid body), the Extended Euler Equations for the *i*-th blade referenced to the origin of $F_{B\ell i}$ are (see Reference 3.4)

$$[\mathbf{Mi'_H} + \mathbf{Mi_A} + \mathbf{Mi_{INT}} + \mathbf{Mi_G}]_{B\ell i} = \underline{I}_{B\ell}\,\dot{\underline{\omega B\ell i}}_{B\ell i} + \tilde{\underline{\omega B\ell i}}_{B\ell i}\,\underline{I}_{B\ell}\,\underline{\omega B\ell i}_{B\ell i} + \tilde{\underline{rCG}}_{B\ell i}\,\underline{aS}_{B\ell i} m_{B\ell}$$

(3.1.126)

where $\underline{I}_{B\ell}$ is the inertia matrix of a single blade expressed in $F_{B\ell i}$

$\underline{rCG}_{B\ell i}$	is the location of the i-th blade's CG expressed in $F_{B\ell i}$
$\underline{aS}_{B\ell i}$	is the inertial acceleration of the hub (and hence the origin of $F_{B\ell i}$) expressed in $F_{B\ell i}$

Now solve (3.1.126) for $\underline{MH}'_{B\ell i}$, the components of $\mathbf{Mi}'_{H}$ expressed in $F_{B\ell i}$ are

$$\underline{MH}'_{B\ell i} = \underline{MH}_{B\ell i} - \underline{MINT}_{B\ell i} \tag{3.1.127}$$

where

$$\underline{MH}_{B\ell i} = \underline{I}_{B\ell}\, \dot{\underline{\omega B\ell i}}_{B\ell i} + \widetilde{\underline{\omega B\ell i}}_{B\ell i}\, \underline{I}_{B\ell}\, \underline{\omega B\ell i}_{B\ell i} + \widetilde{\underline{rCG}}_{B\ell i}\, \underline{aS}_{B\ell i}\, m_{B\ell} - \underline{GAT}_{B\ell i} - \underline{MG}_{B\ell i} \tag{3.1.128}$$

where

$$\underline{MG}_{B\ell i} = m_{B\ell}\, \widetilde{\underline{rCG}}_{B\ell i}\, \underline{L}_{B\ell iE}\, \underline{g}_{E} \tag{3.1.129}$$

In (3.1.128) the blade inertia matrix $\underline{I}_{B\ell}$ is expressed relative to the origin of $F_{B\ell i}$. In evaluating it the assumption is made that the blade is a thin plate lying in the xy-plane of $F_{B\ell i}$. Hence we ignore the effect of blade pitch and twist. It is assumed that $F_{B\ell i}$ are principal axes for the blade and thus $\underline{I}_{B\ell}$ is diagonal with

$$\underline{I}_{B\ell} = \begin{bmatrix} I_{xB\ell} & 0 & 0 \\ 0 & I_{yB\ell} & 0 \\ 0 & 0 & I_{zB\ell} \end{bmatrix} \tag{3.1.130}$$

Now from (3.1.114), (3.1.128) and (3.1.130) express $\underline{MH}_{B\ell i}$ as

$$\underline{MH}_{B\ell i} = [-I_{xB\ell}\, \ddot{\beta}\, i \quad 0 \quad 0]^{T} + \underline{TEMPi} \tag{3.1.131}$$

where

$$\underline{TEMPi} = \underline{I}_{B\ell}\ \underline{\dot{\omega R}}_{B\ell i} + \underline{\widetilde{\omega B\ell i}}_{B\ell i}\ \underline{I}_{B\ell}\ \underline{\omega B\ell i}_{B\ell i} + \underline{\widetilde{rCG}}_{B\ell i}\ \underline{aS}_{B\ell i}\ m_{B\ell} - \underline{GAT}_{B\ell i} - \underline{MG}_{B\ell i}$$

(3.1.132)

The total moment applied by the main rotor about the top of the mast is the sum of the contributions from both blades. In F_R components (using (3.1.127)) this is

$$\underline{GS}_R = -\sum_{i=1}^{2} \underline{L}_{RB\ell i}\underline{MH}'_{B\ell i} \tag{3.1.133}$$

Now from Newton's Third Law

$$\mathbf{M1}_{\mathbf{INT}} = \mathbf{-M2}_{\mathbf{INT}} \tag{3.1.134}$$

Thus $\mathbf{Mi}_{\mathbf{INT}}$ drops out of (3.1.133) and we are left with (using (3.1.127) and (3.1.128))

$$\underline{GS}_R = -\sum_{i=1}^{2} \underline{L}_{RB\ell i}\ \underline{MH}_{B\ell i} \tag{3.1.135}$$

and from (A.2.5), (3.1.55) and (3.1.56)

$$\underline{L}_{RB\ell 1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta 1 & \sin\beta 1 \\ 0 & -\sin\beta 1 & \cos\beta 1 \end{bmatrix} \tag{3.1.136}$$

$$\underline{L}_{RB\ell 2} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\cos\beta 2 & -\sin\beta 2 \\ 0 & -\sin\beta 2 & \cos\beta 2 \end{bmatrix} \tag{3.1.137}$$

From (1.1.135) to (1.1.137)

$$GS_{xR} = -(MH_{xBf1} - MH_{xBf2}) \qquad (3.1.138)$$

From the design of the teetering rotor it follows that (see Figure 3.1.11)

$$GS_{xR} = 0 \qquad (3.1.139)$$

and from (1.1.138) and (1.1.139)

$$MH_{xBf1} - MH_{xBf2} = 0 \qquad (3.1.140)$$

Now combine (3.1.117), (3.1.118), (3.1.131) with (3.1.140) to obtain

$$\ddot{\beta} = (TEMP1_x - TEMP2_x)/2I_{xBf} \qquad (3.1.141)$$

and (3.1.141) is used in Section 3.1.1.11 to develop the blade flapping equations. It also follows that

$$\underline{GS}_R = \begin{bmatrix} 0 & GS_{yR} & GS_{zR} \end{bmatrix}^T \qquad (3.1.142)$$

From (3.1.131) and (3.1.135) to (3.1.137)

$$\begin{bmatrix} GS_{yR} \\ GS_{zR} \end{bmatrix} = -\begin{bmatrix} \cos\beta_1 & \sin\beta_1 \\ -\sin\beta_1 & \cos\beta_1 \end{bmatrix} \begin{bmatrix} TEMP1_y \\ TEMP1_z \end{bmatrix} - \begin{bmatrix} -\cos\beta_2 & -\sin\beta_2 \\ -\sin\beta_2 & \cos\beta_2 \end{bmatrix} \begin{bmatrix} TEMP2_y \\ TEMP2_z \end{bmatrix} \qquad (3.1.143)$$

In order to find the components of the applied moment in F_B, use

$$\underline{GS}_B = \underline{L}_{BR}\, \underline{GS}_R \qquad (3.1.144)$$

where

$$\underline{L}_{BR} = \underline{L}_{BS}\, \underline{L}_{SR}$$

$$= \underline{L}_{SB}^{T}\,\underline{L}_{SR} \tag{3.1.145}$$

Now from (A.2.5) and (3.1.13)

$$\underline{L}_{SR} = \begin{bmatrix} \sin\psi & -\cos\psi & 0 \\ \cos\psi & \sin\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1.146}$$

and from (3.1.47), (3.1.145) and (3.1.146)

$$\underline{L}_{BR} = \begin{bmatrix} \sin\psi\cos\theta_S & -\cos\psi\cos\theta_S & \sin\theta_S \\ \cos\psi & \sin\psi & 0 \\ -\sin\psi\sin\theta_S & \cos\psi\sin\theta_S & \cos\theta_S \end{bmatrix} \tag{3.1.147}$$

Thus from (1.1.142), (1.1.144) and (1.1.147)

$$\underline{GS}_B = \begin{bmatrix} -\cos\psi\cos\theta_S & \sin\theta_S \\ \sin\psi & 0 \\ \cos\psi\sin\theta_S & \cos\theta_S \end{bmatrix} \begin{bmatrix} GS_{yR} \\ GS_{zR} \end{bmatrix} \tag{3.1.148}$$

3.1.1.11 Blade Flapping Equations

Figures 3.1.7 and 3.1.12 show the flapping angles β_i for a teetering rotor. The angle β_o represents a geometrically set coning angle that is preset in the rotor structure. β is the teetering parameter and it is seen that

$$\beta_1 = \beta_0 + \beta \tag{3.1.149}$$

$$\beta_2 = \beta_0 - \beta \tag{3.1.150}$$

The differential equation describing the flapping motion of the rotor is given by (3.1.141) which is repeated below

$$\ddot{\beta} = (TEMP1_{\tau} - TEMP2_{\tau}) / 2I_{\tau B f} \qquad (3.1.151)$$

It is seen from Figures 3.1.11 and 3.1.12 that the rotor is considered to be a rigid structure consisting of two blades. We neglect any structural deformation which may be present.

It should be noted that <u>TEMPi</u> in the right-hand side of (3.1.151) contains β_i and $\dot{\beta}_i$. This equation is solved by numerical integration in the simulation.

3.1.1.12 Moments About the Helicopter Body CG Due to the Main Rotor

The main rotor applies moments about the helicopter body CG due to $\underline{GS}_B$ (the moment applied about the top of the mast) and due to $\underline{fR}_B$ (the force applied to the top of the mast). Thus the total moment from these two effects is given by

$$\underline{GR}_B = \underline{GS}_B + \tilde{\underline{rS}}_B \, \underline{fR}_B \qquad (3.1.152)$$

3.1.1.13 Main Rotor Wake Angle

The main rotor wake angle χ is the angle between US_S and the z-axis of F_S. It is in the range 0 to π. χ is used in later sections when determining the interference effect of the main rotor wake on other parts of the helicopter. χ is found from

$$\chi = \tan^{-1}(\mu_{pp}/(-US_{zS})) \qquad (3.1.153)$$

and we take

$$\chi = 0 (3.1.154)$$

when $\underline{US}_S = 0$.

3.1.1.14 Wind Model

It should be noted that as explained in Atmospheric Effects of Reference 3.1, the mean wind at the helicopter body CG is used for the main rotor as well. Therefore if turbulence is absent it follows that

$$\underline{WSA_E} = \underline{WA_E} \tag{3.1.155}$$

3.1.1.15 Integrating the Blade Flapping Equations

It was found during the testing described in Section 4.3 that solving the blade flapping equations was very sensitive to the integration scheme employed. The most stable solution technique found (allowing the lowest update rates) was a variation of the Fourier Prediction Technique described in Reference 3.2. This method was selected for use and is described below.

It is assumed that the solution being sought is periodic in nature. This is consistent with blade flapping responses. The solution is assumed to have the form

$$\ddot{x}(t) = A\cos(\omega t) \tag{3.1.156}$$

and thus we can take

$$\dot{x}(t) = \frac{A}{\omega}\sin(\omega t) \tag{3.1.157}$$

$$x(t) = -\frac{A}{\omega^2}\cos(\omega t) \tag{3.1.158}$$

Let ΔT be the time step used in solving the flapping equations. The solution at time $(t + \Delta T)$ from (3.1.157) and (3.1.158) is thus

$$\dot{x}(t+\Delta T) = \frac{A}{\omega}\sin(\omega t + \omega\Delta T) \tag{3.1.159}$$

$$x(t+\Delta T) = -\frac{A}{\omega^2}\cos(\omega t + \Delta T) \qquad (3.1.160)$$

Difference equations for use in integrating to find $x(t + \Delta T)$ can be found from the above. Let

$$x_n \equiv x(t) \qquad (3.1.161)$$

$$x_{n+1} \equiv x(t+\Delta T) \qquad (3.1.162)$$

Expanding (3.1.159) leads to

$$\dot{x}_{n+1} = \frac{A}{\omega}\{\sin(\omega t)\cos(\omega\Delta T) + \cos(\omega t)\sin(\omega\Delta T)\} \qquad (3.1.163)$$

From (3.1.156), (3.1.157) and (3.1.163)

$$\dot{x}_{n+1} = \dot{x}_n\cos(\omega\Delta T) + \ddot{x}_n\sin(\omega\Delta T)/\omega \qquad (3.1.164)$$

Expanding (3.1.160) leads to

$$x_{n+1} = -\frac{A}{\omega^2}\cos(\omega t)\cos(\omega\Delta T) + \frac{A}{\omega^2}\sin(\omega t)\sin(\omega\Delta T) \qquad (3.1.165)$$

From (3.1.156) to (3.1.158) and (3.1.165)

$$\begin{aligned} x_{n+1} &= 0 + x_{n+1} \\ &= \left(x_n + \frac{A}{\omega^2}\cos(\omega t)\right) + x_{n+1} \\ &= \left(x_n + \ddot{x}_n/\omega^2\right) - \ddot{x}_n\cos(\omega\Delta T)/\omega^2 + \dot{x}_n\sin(\omega\Delta T)/\omega \end{aligned}$$

$$= x_n + \dot{x}_n \sin(\omega \Delta T)/\omega + \ddot{x}_n \left(\frac{1-\cos(\omega \Delta T)}{\omega^2} \right) \tag{3.1.166}$$

(3.1.164) and (3.1.166) are a set of difference equations which can be used to perform a double integration of $\ddot{x}(t)$. In the limit as ΔT approaches zero this method reduces to an Euler integration method. In testing the method it was found that when $x = \beta$ the best results were achieved for

$$\omega = 2\Omega \tag{3.1.167}$$

3.1.1.16 Filtering Main Rotor Forces and Moments

As indicated in Reference 3.2, it is necessary to pass the outputs $\underline{fR}_B$, $\underline{GR}_B$ and $\underline{GS}_R$ from the Main Rotor through a low-pass filter before inputing them to the Flight Equations. This is because the blade element model produces unrealistically sharp peaks in these variables at the two-per-revolution frequency and the low-pass filter suppresses them to produce a more realistic response. The filter has the transfer function

$$\frac{\omega_{LP}}{s + \omega_{LP}} \tag{3.1.168}$$

where ω_{LP} is the break frequency. The filtered force outputs are $\underline{fRF}_B$, and the filtered moment outputs are $\underline{GRF}_B$ and $\underline{GSF}_R$.

3.1.2 MATRIXx

The block diagrams and block script code for the Main Rotor module are found in Appendix B and C, respectively. Tilda and Transpose Blocks are used at several locations as required.

As can be seen from the block diagrams, the present module has several levels of blocks within blocks. To help in tracing paths through these blocks, a Table of Contents for the pages of block diagrams is provided. The title of each page is given along with a numerical page identifier (appearing at the top of the block diagram page). The identifier may be construed as a section or subsection number in a written report. For example, the block of subsection 1.4.1.2 is the second block for which a separate block diagram is provided, on the page represented by subsection 1.4.1. Since the same block form may be used at several locations in the Main Rotor module, in order to eliminate duplicate block diagrams, the form of a repeated block can be found by going to the page subsection number in brackets following the block title in the Table of Contents. It should be noted that all the subsection numbers associated with a block diagram page are given at the top of the page. A Table of Contents is also provided for the pages of block script following the block diagrams. These are listed in alphabetical order.

In the diagram of Block 1.1.2 (Rotor Angles) the block PSI 0 to 2PI is used to keep the rotor azimuthal angle PSI in the range 0 to 2π as it rotates. In Block 1.2.1 the MATRIXx Transpose Block (A^T) has been used to carry out the required matrix transpose operations. Throughout this module use has been made of Write to Blocks followed by Read from Blocks in order to avoid algebraic loops. For example see the diagram of Block 1.4.1.2 (Blade Element Lift and Drag).

Many of the block diagrams have pairs of vertical lines running from top to bottom. See for example the diagram of Block 1. (Main Rotor). These represent sequencers and they are used to force MATRIXx to carry out operations in the desired order. All operations to the left of a division line will be carried out before those to the right.

In the diagram of Block 1.4.1.5 (Pitt Peters Inflow) initial conditions are placed on the Integrators by summing VPPic(K) with the VPP(K) outputs.

In the diagram of Block 1.4.1.2 (Blade Element Lift and Drag) a "while" loop is used to step through the calculations for the NL elements of each of the 2 main rotor blades. The element number *elementN* specifies the element under consideration and it is incremented by one on each pass through the loop. When it reaches 2NL the Break logic variable is set to unity and the loop is stopped. In the present simulation the variable array sizes are such that the maximum allowable value for NL is 10.

In the diagram of Block 1.4.1.2 (Blade Element Lift and Drag) the calculation of CL and CD is handled differently for ALPHA2D inside and outside the range ±0.56 radians. As described in Reference 3.1, detailed data influenced by Mach number exist inside the range, while less detailed data applies outside. If ALPHA2D is inside the range ±0.56 radians then the logic variable SMALLALPHA is set to unity, otherwise it is set to zero. It should be noted that ALPHA2D is converted from radians to degrees by multiplying it by R2D before going to the linear interpolation blocks. This is because the data found for the lift and drag coefficients (see Reference 3.1) was expressed with α in degrees. Thus the Small Alpha range becomes ±32 degrees. When combined with the NOT Logic Block and the Condition Blocks, the appropriate data analysis is employed. Recall that if the upper input to a Condition Block is unity (zero) then the block is implemented (not implemented). The Logic Switch Block following the CL and CD calculations selects the Small Alpha results as output when SMALLALPHA (the upper input) is unity and the Large Alpha results as output when SMALLALPHA is zero. The Small Alpha determinations are given in Block 1.4.1.2.1 (CL CD Small Alpha). They are found as a bilinear interpolation over ALPHA2D and MACH2D, of the data reported Reference 3.1. The Large Alpha determinations are given in Block 1.4.1.2.2 (CL CD Large Alpha). They are found as a linear interpolation over the magnitude of ALPHA2D, of the data reported in Reference 3.1.

The low-pass filters of (3.1.168) are implemented in the diagram of Block 1.5 (Filter FR GR). Table 3.1.1 lists the parameter data required to run this module.

3.1.3 Main Rotor Notation

Units	Variable	Computer Name	Description
m/s^2	$\underline{aB}_B$	AB_B(K)	inertial acceleration of helicopter body CG, expressed in F_B
m/s^2	$\underline{aCG}_{B\ell i}$	ACG_BL(K,i)	inertial acceleration of the CG of the *i*-th blade, expressed in $F_{B\ell i}$
m/s^2	$\underline{aS}_{B\ell i}$	AS_BL(K,i)	inertial acceleration of the hub expressed in $F_{B\ell i}$
rad	A_0	AA0	collective input to the main rotor blade pitch
rad	A_1	AA1	amplitude of lateral cyclic input to the main rotor blade pitch
m/s^2	**ACGi**	-	inertial acceleration of the CG of the *i*-th blade
rad	-	AZ	$(\psi + \Delta)$
rad	-	A1C	lateral cyclic input to blade pitch
rad	B_1	BB1	amplitude of longitudinal cyclic input to the main rotor blade pitch
rad	-	B1S	longitudinal cyclic input to blade pitch
m	$\bar{c}(j)$	CM(j)	mean chord of the *j*-th blade element
m	$c_{IN}(j)$	-	inner chord of the *j*-th blade element
m	$c_{OUT}(j)$	-	outer chord of the *j*-th blade element
	Cij_D	CD(i,j)	drag coefficient for the *j*-th element of the *i*-th blade
	-	CDN	a single working value of CD(i,j)

Units	Variable	Computer Name	Description
	C_{ijL}	CL(i,j)	lift coefficient for the *j*-th element of the *i*-th blade
	-	CLN	a single working value of CL(i,j)
	C_{LA}	-	main rotor aerodynamic rolling moment coefficient
	C_{MA}	-	main rotor aerodynamic pitching moment coefficient
m	C_R	CR	root chord of the blade
m	C_T	CT	tip chord of the blade
	C_{TA}	-	main rotor thrust coefficient
	D	D	Pitt/Peters model variable
rad	-	DPHI	R2xTSAMP
N	D_{ij}	DRAG(i,j)	drag acting on the *j*-th element of the *i*-th blade
	-	DUM(K,i)	intermediate variable in calculating *TEMP(K,i)*
	-	DUMMY_i(K,2)	intermediate variable in calculating ACG_BL(K,i)
rad	$\underline{E}_{B\ell 1}$	-	Euler angles $[-\beta_1 \quad 0 \quad 0]^T$ which carry F_R into $F_{B\ell 1}$
rad	$\underline{E}_{B\ell 2}$	-	Euler angles $[-\beta_2 \quad 0 \quad \pi]^T$ which carry F_R into $F_{B\ell 2}$
rad	$\underline{E}_{IN}$	-	Euler angles $[0 \quad 0 \quad \beta_W]^T$ which carry F_S into F_{IN}
rad	$\underline{E}_R$	-	Euler angles $[0 \quad 0 \quad (\pi/2 - \psi)]^T$ which carry F_S into F_R

Units	Variable	Computer Name	Description
rad	$\underline{E}_S$	-	Euler angles $[0 \quad \theta \quad 0]_S^T$ which carry F_B into F_S
N	$\underline{fR}_B$	FR_B(K)	force applied to helicopter body by the main rotor, expressed in F_B
N	$\underline{fRF}_B$	FRF_B(K)	low-pass filtered version of $\underline{fR}_B$
N	$FAij_{xBti}$	FAX_BL(i,j)	x-component of aerodynamic force acting on the j-th element of the i-th blade, expressed in F_{Bti}
N	$FAij_{yBti}$	-	y-component of aerodynamic force acting on the j-th element of the i-th blade, expressed in F_{Bti}
N	$FAij_{zBti}$	FAZ_BL(i,j)	z-component of aerodynamic force acting on the j-th element of the i-th blade, expressed in F_{Bti}
N	$\underline{FA}_S$	FA_S(K)	main rotor aerodynamic force expressed in F_S
N	$\underline{FAT}_{Bti}$	FAT_BL(K,i)	aerodynamic force on the i-th blade, expressed in F_{Bti}
N	-	FATX_BL(i)	x-component of the aerodynamic force on the i-th blade, expressed in F_{Bti}
N	-	FATZ_BL(i)	z-component of the aerodynamic force on the i-th blade, expressed in F_{Bti}
N	$\underline{FG}_{Bti}$	FG_BL(K,i)	gravitational force applied to the i-th blade, expressed in F_{Bti}
N	$\underline{FH}_{Bti}$	-	force applied to the i-th blade by the top of the rotor mast, expressed in F_{Bti}
N	$\underline{FHI}_{Bti}$	FHI_BL(K,i)	value of $\underline{FH}_{Bti}$ when $\underline{FINT}_{Bti}$ is set to zero

Units	Variable	Computer Name	Description
N	$\mathbf{Fi}_A$	-	aerodynamic force applied to the *i*-th blade
N	$\mathbf{Fi}_G$	-	gravitational force applied to the *i*-th blade
N	$\mathbf{Fi}_H$	-	force applied to the *i*-th blade by the top of the rotor mast
N	$\mathbf{Fi}_{INT}$	-	internal force applied to the *i*-th blade
N	$\underline{FINT}_{Bti}$	-	internal force applied to the *i*-th blade, expressed in F_{Bti}
m/s^2	$\underline{g}_E$	-	acceleration due to gravity expressed in F_E
N.m	$\underline{GAij}_{Bti}$	-	aerodynamic moment about the hub due to the *j*-th element of the *i*-th blade, expressed in F_{Bti}
N.m	$\underline{GA}_S$	GA_S(K)	main rotor aerodynamic moment about the hub, expressed in F_S
N.m	$\underline{GAT}_{Bti}$	GAT_BL(K,i)	aerodynamic moment about the hub due to the *i*-th blade, expressed in F_{Bti}
N.m	-	GATX_BL(i)	*x*-component of the aerodynamic moment about the hub due to the *i*-th blade, expressed in F_{Bti}
N.m	-	GATZ_BL(i)	*z*-component of the aerodynamic moment about the hub due to the *i*-th blade, expressed in F_{Bti}
N.m	$\underline{GR}_B$	GR_B(K)	moment applied to the helicopter body about its CG by the main rotor, expressed in F_B
N.m	$\underline{GRF}_B$	GRF_B(K)	low-pass filtered version of $\underline{GR}_B$

Units	Variable	Computer Name	Description
N.m	$\underline{GS}_R$	GS_R(K)	moment applied to the helicopter body about the top of the rotor mast by the main rotor, expressed in F_R
N.m	$\underline{GSF}_R$	GSF_R(K)	low-pass filtered version of $\underline{GS}_R$
m	h_{CG}	HCG	height of the CG of the helicopter body above local ground level
m	H_{AGL}	HAGL	height of the main rotor hub above the local ground plane
kg.m^2	$\underline{I}_{B\ell}$	IBL(K,M)	main rotor blade inertia matrix expressed in $F_{B\ell i}$
kg.m^2	-	IBLx,IBLy,IBLz	diagonal elements of $\underline{I}_{B\ell}$
	j	elementN	index representing the j-th element when $j = elementN$
	K_{GE}	KGE	ground effect scale factor
	K_0,K_1,K_2,K_3	K0, K1, K2, K3	Pitt/Peters model parameters
	$\underline{L}^{-1}$	LINV(L,M)	matrix of variables in Pitt/Peters model
N.m	L_A	LA	main rotor aerodynamic rolling moment about the x_{IN}-axis
	$\underline{L}_{BE}$	L_B_E(L,M)	rotation matrix from F_E to F_B
	$\underline{L}_{BR}$	-	rotation matrix from F_R to F_B
	$\underline{L}_{B\ell iB}$	L_BLi_B(L,M)	rotation matrix from F_B to $F_{B\ell i}$
	$\underline{L}_{B\ell iE}$	L_BLi_E(L,M)	rotation matrix from F_E to $F_{B\ell i}$
	$\underline{L}_{B\ell iR}$	L_BLi_R(L,M)	rotation matrix from F_R to $F_{B\ell i}$
	$\underline{L}_{B\ell iS}$	L_BLi_S(L,M)	rotation matrix from F_S to $F_{B\ell i}$

Units	Variable	Computer Name	Description
N	Lij	LIFT(i,j)	lift acting on the j-th element of the i-th blade
	$\underline{L}_{RS}$	L_R_S(L,M)	rotation matrix from F_S to F_R
	$\underline{L}_{SB}$	L_S_B(L,M)	rotation matrix from F_B to F_S
	$\underline{L}_{SE}$	L_S_E(L,M)	rotation matrix from F_E to F_S
kg	m_{Bl}	MBLADE	mass of a single main rotor blade
	$\underline{M}^{-1}$	MINV(L,M)	matrix of parameters in Pitt/Peters model
N.m	M_A	MA	main rotor aerodynamic pitching moment about the y_{IN}-axis
N.m	$\underline{MG}_{B\ell i}$	-	gravitational moment applied to the i-th blade about the origin of $F_{B\ell i}$, expressed in $F_{B\ell i}$
N.m	$\underline{MH}_{B\ell i}$	-	the value of $\underline{MH}'_{B\ell i}$ when $\underline{MINT}_{B\ell i}$ is set to zero
N.m	$\underline{MH}'_{B\ell i}$	-	moment applied to the i-th blade about the origin of $F_{B\ell i}$, by the top of the rotor mast, expressed in $F_{B\ell i}$
N.m	$\mathbf{Mi}_A$	-	aerodynamic moment applied to the i-th blade about the origin of $F_{B\ell i}$
N.m	$\mathbf{Mi}_G$	-	gravitational moment applied to the i-th blade about the origin of $F_{B\ell i}$
N.m	$\mathbf{Mi}'_H$	-	moment applied to the i-th blade about the origin of $F_{B\ell i}$, by the top of the rotor mast
N.m	$\mathbf{Mi}_{INT}$	-	internal moment applied to the i-th blade about the origin of $F_{B\ell i}$

Units	Variable	Computer Name	Description
	Mij_{2D}	MACH2D(i,j)	two-dimensional Mach number of the *j*-th element of the *i*-th blade
	-	MACH2DN	a single working value of MACH2D(i,j)
N.m	$\underline{MINT}_{Bfi}$	-	internal moment applied to the *i*-th blade about the origin of F_{Bfi}, expressed in F_{Bfi}
	NL	NL	number of blade elements per blade
	-	NU_TEMP	intermediate variable
	PTL	PTL	blade tip loss parameter
	$PTLR$	PTLR	blade tip loss ratio
m	r	-	radial distance from hub
m	$r_{IN}(j)$	RIN(j)	inner radius of the *j*-th blade element
m	$r_m(j)$	RM(j)	nominal mid-point radius of the *j*-th blade element (based on swept area)
m	$r_{OUT}(j)$	ROUT(j)	outer radius of the *j*-th blade element
m	r_0	R0	radial distance of the root of the blade from the center of the hub
m	$\underline{rB}_E$	RB_E(K)	location of the helicopter body CG relative to F_E
m	rCG	RCG	blade radius at the blade CG
m	$\underline{rCG}_{Bfi}$	-	location of blade CG relative to F_{Bfi}
m	$\underline{rG}_E$	RG_E(K)	location of local ground plane relative to F_E
m	$\underline{rmj}$	-	location of mid-point of the *j*-th blade element relative to F_{Bti}

Units	Variable	Computer Name	Description
m	$\underline{rS}_B$	RS_B(K)	location of the main rotor hub relative to the origin of F_B
m	R	R	main rotor radius
rad/s	-	R2	2ROMEGA
	-	R2D	$180/\pi$
m^2	$S(j)$	S(J)	area of the j-th blade element
s	t	-	time
N	T_A	TA	main rotor aerodynamic thrust in F_{IN}
N.m	$\underline{TEMPi}$	TEMP(K,i)	intermediate variable in calculating $\underline{MH}_{B\ell i}$
	$TLF(j)$	TLF(j)	blade tip loss factor, applied to the j-th element
rad	$TOTTWIST$	TOTTWIST	total blade twist, root to tip
m/s	U_{SOUND}	USOUND	local speed of sound
m/s	U_{TOT}	UTOT	$\lvert\underline{US}_S\rvert$
m/s	$\underline{UB\ell ij}_{B\ell i}$	UBLi_BLi(K,j)	airspeed of the j-th element of the i-th blade, expressed in $F_{B\ell i}$
m/s	$\underline{US}_S$	US_S(K)	airspeed of the main rotor hub expressed in F_S
m/s	-	US_S_1	US_S(1)
m^2/s^2	$U2ij$	U2(i,j)	$UB\ell ij^2_{xB\ell i} + UB\ell ij^2_{zB\ell i}$
m/s	V_{IN}	-	inflow from Pitt/Peters model without ground effect
m/s	V_m	VM	Pitt/Peters model variable

Units	Variable	Computer Name	Description
m/s	V_T	VT	Pitt/Peters model variable
m/s	V_0	V0	uniform inflow component from Pitt/ Peters model without ground effect
m/s	V_1	V1	velocity parameter in Pitt/Peters model
m/s	V_2	V2	velocity parameter in Pitt/Peters model
m/s	$\underline{VB}_B$	VB_B(K)	inertial velocity of the helicopter body CG expressed in F_B
m/s	$\underline{VB\ell ij}_{B\ell i}$	VBLi_BLi(K,j)	inertial velocity of the j-th element of the i-th blade, expressed in $F_{B\ell i}$
m/s	VIN_{GE}	-	inflow from Pitt/Peters model including ground effect
m/s	$VINGE_{ij}$	VINGE(i,j)	inflow from Pitt/Peters model including ground effect, at the j-th element of the i-th blade
m/s	$\underline{VINGEij}_S$	-	inflow vector from Pitt/Peters model including ground effect, at the j-th element of the i-th blade, expressed in F_S
m/s	-	VINBLi(K,j)	inflow vector from Pitt/Peters model including ground effect, at the j-th element of the i-th blade, expressed in $F_{B\ell i}$
m/s	$\underline{VPP}$	VPP(K)	$[V_0 \quad V_1 \quad V_2]^T$
m/s	$\underline{VS}_S$	VS_S(K)	inertial velocity of the main rotor hub expressed in F_S
N	-	WBLADE	weight of a single main rotor blade
m/s	$\underline{WA}_E$	WA_E(K)	atmospheric mean wind at the helicopter body CG expressed in F_E

Units	Variable	Computer Name	Description
m/s	$\underline{WB\ell Aij}_{B\ell i}$	WBLiA_BLi(K,j)	the part of $\underline{WB\ell ij}_{B\ell i}$ due to atmospheric effects, expressed in $F_{B\ell i}$
m/s	$\underline{WB\ell ij}_{B\ell i}$	WBLi_BLi(K,j)	wind speed at the j-th element of the i-th blade, expressed in $F_{B\ell i}$
m/s	$\underline{WS}_S$	WS_S(K)	wind speed at the main rotor hub expressed in F_S
m/s	$\underline{WSA}_S$	WSA_S(K)	the part of $\underline{WS}_S$ due to atmospheric effects
rad	αij_x	ALPHABL(i,j)	angle of attack of x-axis of $F_{B\ell i}$ for the j-th element of the i-th blade
rad	αij_{2D}	ALPHA2D(i,j)	two-dimensional angle of attack of the j-th element of the i-th blade
	-	ALPHA2DN	a single working value of ALPHA2D(i,j)
rad	β	B	deviation of β_I about the pre-set value of β_0
rad	β_W	BW	Euler angle which carries F_S into F_{IN}
rad	β_0	B0	pre-set value of coning angle in the main rotor blades
rad	β_1, β_2	B1, B2	flapping angles for the two rotor blades
m^2	δ	DELTA	area of the annulus swept out by each blade element, divided by π
rad	Δ	PHASE	cyclic control phase shift
s	Δt	DT	time step in numerical integration
s	ΔT	TSAMP	time interval in Flapping Equations Block
rad	θi_c	THETAC(i)	i-th blade pitch angle due to control inputs

Units	Variable	Computer Name	Description
rad	θij_b	PITCHBL(i,j)	pitch angle of the *j*-th element of the *i*-th blade
rad	θj_{twist}	TWIST(j)	blade twist
rad	θ_S	THETS	Euler angle which carries F_B into F_S
rad	θ_{zll}	ZLL	blade pitch offset at the root
	λ_{pp}	LAMPP	Pitt/Peters model variable (total relative inflow based on momentum theory)
m/s	μ_{pp}	MUPP	main rotor hub airspeed based on US_{xS} and US_{yS}
m^2/s^2	μ_{pp2}	MUPP2	$(\mu_{pp})^2$
m/s	ν	NU	Pitt/Peters model variable (uniform induced inflow based on momentum theory)
	-	NU_FUNC_DERIV	a derivative used in applying the Newton-Raphson method to find NU
kg/m^3	ρ	RHO	local atmospheric density
rad	χ	CHI	main rotor wake angle
rad	χ_{pp}	CHIPP	Pitt/Peters model variable (main rotor wake skew angle)
rad	ψ	PSI	main rotor azimuthal angle (referenced to Blade 1)
rad	ψ_{IN}	PSIIN	azimuthal location relative to the negative x_{IN}-axis
rad/s	$\underline{\omega B_B}$	OMEGAB_B(K)	inertial angular velocity of the helicopter body expressed in F_B
rad/s	-	OMEGABL_BL(K,i)	OMEGABLi_BLi(K)

Units	Variable	Computer Name	Description
rad/s	$\underline{\omega B\ell i}_{B\ell i}$	OMEGABLi_BLi(K)	inertial angular velocity of the i-th blade, expressed in $F_{B\ell i}$
rad/s	ω_{LP}	OMEGALP	break frequency of the main rotor force and moment low-pass filters
rad/s	$\underline{\omega\, i}_{B\ell i}$	-	angular velocity of $F_{B\ell i}$ relative to F_R expressed in $F_{B\ell i}$
rad/s	$\underline{\omega R}_S$	OMEGAR_S(K)	inertial angular velocity of F_R, expressed in F_S
rad/s	Ω	ROMEGA	angular velocity of the main rotor relative to the helicopter body
	-	$(\quad)_{ic}$	initial condition
	-	$(\quad)_{old}$	previous value

3.2 Landing Gear

3.2.1 Physical Model

The landing gear model employed in the present simulation is based on one described in References 3.10 and 3.11. The landing gear is assumed to consist of skids (see Figure 1.1). In order to apply the referenced model, its wheels have been replaced by 4 pads located at the ends of the skids (see Figure 3.2.3). These 4 pads then represent the skids in the model. The skids and their supporting struts are considered to be massless. Each of the 4 pads is attached to the fuselage by a strut as shown in Figure 3.2.1. The location of the attachment point for the *i*-th strut is specified by the vector **rSKi** from the origin of F_B. A reference frame F_{SKi} is located with its origin at the attachment point for the *i*-th strut, its *x*-axis parallel to the x-axis of F_B and its *z*-axis along the strut as shown in the figure. The strut can rotate about this *x*-axis in response to ground loads on the landing gear. This rotation is resisted by the structure which is modeled by a torsional spring and viscous damper. The resulting deflection is based on the assumption of a rigid strut. A quasi-static model is assumed in which the *i*-th strut angle relative to the helicopter body is a constant predetermined value when the *i*-th pad is in the air. Otherwise the *i*-th pad maintains ground contact when setting the strut angle to its airborne value would result in the *i*-th pad being located at or below the local ground plane.

3.2.1.1 Kinematics

Let the Euler angles which carry F_B into F_{SKi} be given by

$$\underline{E}_{SKi} = [\phi \quad 0 \quad 0]^T_{SKi} \tag{3.2.1}$$

In Figures 3.2.1 and 3.2.2 F_E is the Earth-fixed reference frame employed in the simulator's visual data base. Let F_V be a reference frame, vertically above F_E, with its *z*-axis parallel to the *z*-axis of F_E and its origin in the local ground plane. It will be assumed in the present model that the local ground plane is parallel to the xy-plane of F_E. The location of the local ground plane is given by **rG** relative to the origin of F_E. The location of the origin of F_V is specified by the vector **rV** from the origin of F_E. It follows from Figure 3.2.1 that

$$\underline{rV}_E = [0 \quad 0 \quad rG_{zE}]^T \tag{3.2.2}$$

The location of the i-th landing gear pad is given by the vector **POSNi** from the origin of F_V. From Figure 3.2.1 it is seen that in F_E components

$$\underline{POSNi}_E = \underline{rB}_E + \underline{rSKi}_E + \underline{rGSi}_E - \underline{rV}_E \tag{3.2.3}$$

and this applies both when the helicopter is on the ground and in the air. When the helicopter's i-th pad is on the ground it follows that

$$POSNi_{zE} = 0 \tag{3.2.4}$$

and (3.2.4) can be used to find the value of ϕ_{SKi} when the i-th pad is on the ground. Rewrite (3.2.3) as

$$\underline{POSNi}_E = \underline{rB}_E + \underline{L}_{EB}(\underline{rSKi}_B + \underline{L}_{BSKi}\,\underline{rGSi}_{SKi}) - \underline{rV}_E \tag{3.2.5}$$

In (3.2.5) $\underline{rB}_E$ and $\underline{L}_{EB}$ come from the flight equations of Section 3.8 of Reference 3.1, $\underline{rV}_E$ comes from the visual data base, $\underline{rSKi}_B$ and $\underline{rGSi}_{SKi}$ are fixed parameters for the helicopter and $\underline{L}_{BSKi}$ is found using (A.2.5) and (3.2.1) and thus involves ϕ_{SKi}. Combining (3.2.2), (3.2.4) and (3.2.5) leads to

$$0 = rB_{zE} + (\underline{L}_{EB}\,\underline{rSKi}_B)_z + (\underline{L}_{EB}\,\underline{L}_{BSKi}\,\underline{rGSi}_{SKi})_z - rG_{zE} \tag{3.2.6}$$

In (3.2.6)

$$\underline{rGSi}_{SKi} = [0 \quad 0 \quad rGSi]^T \tag{3.2.7}$$

where *rGSi* is the length of the *i*-th strut. Also, it follows from (3.2.7) that in (3.2.6)

$$\underline{L}_{BSKi}\,\underline{rGSi}_{SKi} = rGSi\begin{bmatrix} 0 \\ -\sin\phi_{SKi} \\ \cos\phi_{SKi} \end{bmatrix} \tag{3.2.8}$$

Equations (3.2.6) and (3.2.8) can be solved numerically for ϕ_{SKi}. In the present simulation this is accomplished by using the Newton-Raphson method described in Appendix A. This requires the use of the gradient of the right-hand side of (3.2.6) with respect to ϕ_{SKi}. The right-hand side of (3.2.6) is $POSNi_{zE}$ and from (3.2.6) the required gradient is

$$\frac{\partial POSNi_{zE}}{\partial\phi_{SKi}} = \left(\underline{L}_{EB}\,\frac{\partial(\underline{L}_{BSKi}\,\underline{rGSi}_{SKi})}{\partial\phi_{SKi}}\right)_z \tag{3.2.9}$$

and from (3.2.8) and (3.2.9) and $\underline{L}_{EB}$ from Section 3.8 of Reference 3.1,

$$\frac{\partial POSNi_{zE}}{\partial\phi_{SKi}} = -rGSi\cos\theta_B(\cos\phi_{SKi}\sin\phi_B + \sin\phi_{SKi}\cos\phi_B) \tag{3.2.10}$$

When in the air the strut Euler angles ϕ_{SKi} are constant. Figure 3.2.3 gives the numbering of the strut/pad combinations. It follows that (assuming that the magnitude of ϕ_{SKi} while in the air is the same for all struts) while in the air:

$$\phi_{SK1} = \phi_{SK2} = -\phi_{SK0} \tag{3.2.11}$$

$$\phi_{SK3} = \phi_{SK4} = \phi_{SK0} \tag{3.2.12}$$

where ϕ_{SK0} is a specified helicopter parameter.

While on the ground we need to know the values of $\dot{\phi}_{SKi}$ in order to determine the viscous damping about the strut attachment points. This can be found by noting that when the i-th pad is on the ground

$$\dot{POSNi}_{zE} = 0 \tag{3.2.13}$$

From (3.2.5)

$$\underline{\dot{POSNi}}_E = \underline{\dot{rB}}_E + \underline{\dot{L}}_{EB}(\underline{rSKi}_B + \underline{L}_{BSKi}\,\underline{rGSi}_{SKi}) + \underline{L}_{EB}\,\underline{\dot{L}}_{BSKi}\,\underline{rGSi}_{SKi} \tag{3.2.14}$$

and $\dot{POSNi}_{zE}$ follows from (3.2.14). By differentiating (3.2.8) obtain

$$\underline{\dot{L}}_{BSKi}\,\underline{rGSi}_{SKi} = -rGS_i\begin{bmatrix} 0 \\ \cos\phi_{SKi} \\ \sin\phi_{SKi}\end{bmatrix}\dot{\phi}_{Ski} \tag{3.2.15}$$

From (A.4,18) to (A.4,20) of Reference 3.6

$$\underline{\dot{L}}_{EB} = \underline{L}_{EB}\,\underline{\widetilde{\omega B}}_B \tag{3.2.16}$$

where $\underline{L}_{EB}$ and $\underline{\widetilde{\omega B}}_B$ come from the Flight Equations of Section 3.8 of Reference 3.1.

From the definition of inertial velocity

$$\underline{\dot{rB}}_E = \underline{VB}_E \tag{3.2.17}$$

and $\underline{VB}_E$ can also be found in the Flight Equations. From (3.2.10), (3.2.15) and $\underline{L}_{EB}$ it follows that

$$\left(\underline{L}_{EB}\,\underline{\dot{L}}_{BSKi}\,\underline{rGSi}_{SKi}\right)_z = -rGSi\cos\theta_B(\cos\theta_{SKi}\sin\phi_B + \sin\phi_{SKi}\cos\phi_B)\dot{\phi}_{SKi}$$

$$= \dot{\phi}_{SKi}\left(\frac{\partial POSNi_{zE}}{\partial \phi_{SKi}}\right) \qquad (3.2.18)$$

By combining (3.2.13) to (3.2.18) and solving for $\dot{\phi}_{SKi}$ obtain

$$\dot{\phi}_{SKi} = -\left(VB_{zE} + \left(\underline{L}_{EB}\,\underline{\tilde{\omega B}}_B\left(\underline{rSKi}_B + \underline{L}_{BSKi}\,\underline{rGSi}_{SKi}\right)\right)_z\right) \times \left(\frac{\partial POSNi_{zE}}{\partial \phi_{SKi}}\right)^{-1} \qquad (3.2.19)$$

3.2.1.2 Forces on Landing Gear When Sliding

In the present model each landing gear pad when in ground contact is either sliding or not sliding (sticking). The ground forces acting on the *i*-th pad are shown in Figure 3.2.1 for the sliding case. **Ni** is the normal reaction force acting on the landing gear and **Li** is the in-plane reaction force acting on the landing gear. Recall that the local ground plane is assumed to be parallel to the *xy*-plane of F_E. In Figure 3.2.1 we will further specify the orientation of F_V (which has thus far only been required to have its *z*-axis parallel to the *z*-axis of F_E). Let the Euler angles which carry F_E into F_V be given by

$$\underline{E}_V = [0 \quad 0 \quad \psi_B]^T \qquad (3.2.20)$$

where ψ_B is given by (3.8.1).

The components of **Li** in F_V are modeled in the present development by

$$\underline{Li}_V = [Li_x \quad Li_y \quad 0]^T \qquad (3.2.21)$$

The **Ni** are found by equating the torque about the x-axis of F_{SKi} due to the ground reaction forces to the torque produced by the torsional spring and damper. This is a direct result of assuming that the skids and their supporting structure are massless.

The torque applied to the helicopter fuselage about the fuselage attachment point by the i-th torsional spring and damper is

$$\underline{GJi}_{SKi} = \begin{bmatrix} \kappa 1 \Delta\phi_{Ski} + \kappa 2 \Delta\dot{\phi}_{Ski} \\ 0 \\ 0 \end{bmatrix} \tag{3.2.22}$$

where

$$\Delta\phi_{Ski} = \phi_{SKi} + \phi_{SK0} \qquad \text{for} \qquad i = 1, 2 \tag{3.2.23}$$

$$\Delta\phi_{Ski} = \phi_{SKi} - \phi_{SK0} \qquad \text{for} \qquad i = 3, 4 \tag{3.2.24}$$

and $\kappa 1$ and $\kappa 2$ are fixed helicopter parameters. The torque about the x-axis of F_{Ski} applied to the helicopter fuselage by the ground reaction forces on the i-th pad is

$$\underline{GSKi}_{SKi} = \underline{r\widetilde{GSi}}_{SKi}(\underline{Li}_{SKi} + \underline{Ni}_{Ski}) \tag{3.2.25}$$

where

$$\underline{Ni}_V = [0 \quad 0 \quad Ni]^T \tag{3.2.26}$$

From (3.2.21) and (3.2.26)

$$\underline{Li}_{SKi} + \underline{Ni}_{SKi} = \underline{L}_{SKiV}[Li_x \quad Li_y \quad Ni]^T \tag{3.2.27}$$

where $\underline{L}_{SKiV}$ is given by (3.2.69). Using (A.1.14) and (3.2.7) it follows that

$$\underline{\widetilde{rGSi}}_{SKi} = \begin{bmatrix} 0 & -rGSi & 0 \\ rGSi & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.2.28}$$

Equating the x-components of (3.2.22) and (3.2.25) results in (using (3.2.27) and (3.2.28)):

$$\begin{aligned} GJi_{xSKi} &= (\underline{\widetilde{rGSi}}_{SKi}(\underline{L}i_{SKi} + \underline{Ni}_{SKi}))_x \\ &= [0 \quad -rGSi \quad 0]\underline{L}_{SKiV}\begin{bmatrix} Li_x \\ Li_y \\ Ni \end{bmatrix} \\ &= -rGSi[Ri_{21} \quad Ri_{22} \quad Ri_{23}]\begin{bmatrix} Li_x \\ Li_y \\ Ni \end{bmatrix} \end{aligned} \tag{3.2.29}$$

where $[Ri_{21} \quad Ri_{22} \quad Ri_{23}]$ is the second row of $\underline{L}_{SKiV}$. Ni is then found from (3.2.29) as

$Ni = -(GJi_{xSKi} / rGSi + Ri_{21}Li_x + Ri_{22}Li_y) / Ri_{23}$ when the right-hand side is negative

$$Ni = 0 \text{ otherwise} \tag{3.2.30}$$

The limiting operation has been added to (3.2.30) to ensure that Ni is always negative, as required by (3.2.26) and Figure 3.2.1, despite any effects introduced by the quasi-static dynamics assumption.

Li when sliding is based on a Coulomb friction model. The position of the i-th pad relative to the origin of F_V is given by the vector **rUi** when the pad is on the ground (see Figure 3.2.2). From Figures 3.2.1 and 3.2.2 it can be seen that when the i-th pad is on the ground

$$\mathbf{rUi} = \mathbf{POSNi} \tag{3.2.31}$$

The velocity of the i-th pad over the ground when sliding on the ground is $\overset{\bullet}{\underline{rUi}}_V$. **Li** is then given by

$$\underline{Li}_V = -\mu_{SL} \frac{|Ni|}{\left|\underline{r\dot{U}i}_V\right|} \underline{r\dot{U}i}_V \tag{3.2.32}$$

where μ_{SL} is the coefficient of sliding friction. Since the direction of the normal reaction force in (3.2.26) is always in the direction of the negative z-axis of F_V and from (3.2.30) it follows that

$$Ni \leq 0 \tag{3.2.33}$$

Thus (3.2.32) can be written as

$$\underline{Li}_V = \mu_{SL} \frac{Ni}{\left|\underline{r\dot{U}i}_V\right|} \underline{r\dot{U}i}_V \tag{3.2.34}$$

In order to minimize jerkiness when the gear switches from sliding to sticking, the Coulomb friction model is replaced by a viscous friction model when $\left|\underline{r\dot{U}i}_V\right|$ falls below 0.05 m/s. Thus for

$$\left|\underline{r\dot{U}i}_V\right| > 0.05 \tag{3.2.35}$$

$$\underline{Li}_V \text{ is (3.2.34)}$$

and for

$$\left|\underline{r\dot{U}i}_V\right| \leq 0.05 \tag{3.2.36}$$

$$\underline{Li}_V = \mu_{SL}\, Ni\, \underline{r\dot{U}i}_V / 0.05 \tag{3.2.37}$$

From (3.2.31) it follows that

$$\underline{rUi}_V = \underline{L}_{VE}\,\underline{POSNi}_E \tag{3.2.38}$$

and

$$\underline{\dot{rUi}}_V = \underline{L}_{VE}\,\underline{\dot{POSNi}}_E + \underline{\dot{L}}_{VE}\,\underline{POSNi}_E \tag{3.2.39}$$

where $\underline{POSNi}_E$ and $\underline{\dot{POSNi}}_E$ come from Section 3.2.1.1. From (A.4,19) of Reference 3.6

$$\underline{\dot{L}}_{VE} = -\underline{\widetilde{\omega V}}_V\;\underline{L}_{VE} \tag{3.2.40}$$

Since (3.2.20) holds it follows that

$$\underline{\omega V}_V = [0 \quad 0 \quad \dot{\psi}_B]^T \tag{3.2.41}$$

where $\dot{\psi}_B$ comes from the Flight Equations and $\underline{L}_{VE}$ is given by (3.2.62).

3.2.1.3 Strut/Pad Deformation

The strut/pad combination is assumed to be rigid when calculating values for everything except the in-plane reaction force **Li** when the *i*-th pad is sticking to the ground. When sticking, a strut/pad deformation is allowed (which is assumed to be small) and is used to determine **Li**. The deformation geometry is shown in Figure 3.2.2. The deformation vector is given by

$$\underline{\Delta i}_V = \underline{rDi}_V - \underline{rUi}_V \tag{3.2.42}$$

where $\underline{rUi}_V$ is given by (3.2.38). **rDi** gives the location of the ground point to which the *i*-th pad is assumed to be stuck, relative to the origin of F_V. Note that

$$\underline{\Delta i}_V = [\Delta i_x \quad \Delta i_y \quad 0]^T \tag{3.2.43}$$

The force generation model is based on deformation and deformation rate. From (3.2.38) and (3.2.42)

$$\underline{\Delta i}_V = \underline{L}_{VE}\,\underline{rDi}_E - \underline{rUi}_V$$

$$= \underline{L}_{VE}(\underline{rDi}_E - \underline{POSNi}_E) \tag{3.2.44}$$

and

$$\dot{\underline{\Delta i}}_V = \dot{\underline{L}}_{VE}\,\underline{rDi}_E + \underline{L}_{VE}\,\dot{\underline{rDi}}_E - \dot{\underline{rUi}}_V \tag{3.2.45}$$

From (3.2.45) and noting that

$$\dot{\underline{rDi}}_E = 0 \tag{3.2.46}$$

it follows that

$$\dot{\underline{\Delta i}}_V = \dot{\underline{L}}_{VE}\,\underline{rDi}_E - \dot{\underline{rUi}}_V \tag{3.2.47}$$

where $\dot{\underline{L}}_{VE}$ is given by (3.2.40) and $\dot{\underline{rUi}}_V$ by (3.2.39).

When sliding it is assumed that deformation is present and governed by (3.2.49). This deformation can be found by solving the differential equation

$$\dot{\underline{\Delta i}}_V = (\underline{Li}_V - C1\,\underline{\Delta i}_V)/C2 \tag{3.2.48}$$

where $\underline{Li}_V$ is given by (3.2.34).

3.2.1.4 Forces on Landing Gear When Sticking

A description of the ground reaction forces can be found at the start of Section 3.2.1.2.

The normal ground reaction force when sticking or sliding is given by (3.2.30). The in-plane reaction force applied to the landing gear when sticking is modeled by

$$\underline{Li}_V = C1 \times \underline{\Delta i}_V + C2 \times \dot{\underline{\Delta i}}_V \tag{3.2.49}$$

where $\underline{\Delta i}_V$ is given by (3.2.44) and $\dot{\underline{\Delta i}}_V$ by (3.2.47). C1 and C2 are fixed helicopter parameters.

3.2.1.5 *Sticking/Sliding Transition*

In order to determine whether a pad is sticking or sliding the following physical models and logic are applied. When sticking, the *i*-th pad will transition to sliding if

$$|\,\underline{Li}_V\,| > \mu_{ST}|Ni| \tag{3.2.50}$$

where

$$\mu_{ST} \geq \mu_{SL} \tag{3.2.51}$$

When sliding, the *i*-th pad will transition to sticking if the sliding velocity over the ground becomes small enough. That is if

$$|\,\dot{\underline{rUi}}_V\,| < V_{REF} \tag{3.2.52}$$

provided that the sliding velocity has exceeded V_{REF} or T_{LG} seconds have passed since the previous transition from sticking to sliding.

Following the transition from sticking to sliding the deformation differential equation (3.2.48) must be solved. The initial condition on $\underline{\Delta i}_V$ used to start this process is taken to be its value at the end of the previous sticking phase.

Following the transition from sliding to sticking the value of $\underline{rDi}_E$ that holds for the duration of this sticking phase is given by the following relationship evaluated at the end of the previous sliding phase (based on (3.2.42))

$$\underline{rDi}_V = \underline{rUi}_V + \underline{\Delta i}_V \tag{3.2.53}$$

or, using (3.2.31) and (3.2.53)

$$\underline{rDi}_E = \underline{POSNi}_E + \underline{L}_{EV}\,\underline{\Delta i}_V \tag{3.2.54}$$

3.2.1.6 *Forces and Moments Applied to Helicopter*

As a result of the assumption that the landing gear is massless, the ground reaction forces and their resulting moments are applied directly to the helicopter body. Thus the applied force is

$$\begin{aligned}\underline{fLGi}_B &= \underline{L}_{BV}(\underline{Ni}_V + \underline{Li}_V)\\ &= \underline{L}_{BV}[Li_x \quad Li_y \quad Ni]^T\end{aligned} \tag{3.2.55}$$

where $\underline{L}_{BV}$ is given by (3.2.7), and from Figure 3.2.1 the moment applied about the helicopter body CG is

$$\underline{GLGi}_B = (\underline{\widetilde{rSKi}}_B + \underline{\widetilde{rGSi}}_B)\,\underline{fLGi}_B \tag{3.2.56}$$

where

$$\underline{rGSi}_B = \underline{L}_{BSKi}\,\underline{rGSi}_{SKi} \tag{3.2.57}$$

and $\underline{L}_{BSKi}$ is given by (3.2.61). The above applies when the i-th pad is on the ground. If the i-th pad is in the air then its applied forces and moments are zero. The total contribution from all 4 pads is then

$$\underline{fLG}_B = \sum_{i=1}^{4} \underline{fLGi}_B \tag{3.2.58}$$

$$\underline{GLG}_B = \sum_{i=1}^{4} \underline{GLGi}_B \tag{3.2.59}$$

3.2.1.7 Filtering Landing Gear Forces and Moments

As indicated in Reference 3.1, it is necessary to pass the outputs $\underline{fLG}_B$, and $\underline{GLG}_B$ from the Landing Gear through a low-pass filter before inputing them to the Flight Equations. The filter has the transfer function

$$\frac{\omega_{LG}{}^2}{s^2 + 2\zeta_{LG}\omega_{LG}s + \omega_{LG}{}^2} \tag{3.2.60}$$

where ω_{LG} is the break frequency and ζ_{LG} is the damping ratio. The filtered force and moment outputs are $\underline{fLGF}_B$, and $\underline{GLGF}_B$.

3.2.1.8 Rotation Matrices

Several rotation matrices are employed in Sections 3.2.1.1 to 3.2.1.6. They can be found as follows.

$\underline{L}_{EB}$ – from Section 3.8, Reference 3.1

By making use of (A.2.5) and (3.2.1) it follows that

$$\underline{L}_{BSKi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_{SKi} & -\sin\phi_{SKi} \\ 0 & \sin\phi_{SKi} & \cos\phi_{SKi} \end{bmatrix} \tag{3.2.61}$$

By making use of (A.2.4) and (3.2.20) it follows that

$$\underline{L}_{VE} = \begin{bmatrix} \cos\psi_B & \sin\psi_B & 0 \\ -\sin\psi_B & \cos\psi_B & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2.62}$$

Since the Euler angles that carry F_V into F_B are $[\phi_B \quad \theta_B \quad 0]^T$ it follows from (A.2.4) that

$$\underline{L}_{BV} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ \sin\phi\sin\theta & \cos\phi & \sin\phi\cos\theta \\ \cos\phi\sin\theta & -\sin\phi & \cos\phi\cos\theta \end{bmatrix}_B \tag{3.2.63}$$

Also

$$\begin{aligned} \underline{L}_{SKiV} &= \underline{L}_{SKiB}\,\underline{L}_{BV} \\ &= \underline{L}^T_{BSKi}\,\underline{L}_{BV} \end{aligned} \tag{3.2.64}$$

3.2.1.9 Landing Gear Logic

In order to calculate the ground forces acting on the landing gear and hence the corresponding forces and moments acting on the helicopter body it is necessary to determine whether the *i*-th pad is in ground contact and, if in ground contact, whether it is sliding or sticking to the ground. The logic variable *IGNDi* is used to indicate the ground contact status

$$\begin{aligned} IGNDi &= 1 \quad i\text{-th pad on ground} \\ &= 0 \quad i\text{-th pad in air} \end{aligned} \tag{3.2.65}$$

The logic variable *ISLIDEi* is used to indicate the sliding/sticking status when the *i*-th pad is on the ground

$ISLIDEi = 1 \quad i$-th pad sliding

$$= 0 \qquad i\text{-th pad sticking} \tag{3.2.66}$$

The flow of the landing gear logic for the *i*-th pad is shown in Figure 3.2.4. The block numbers referred to below are given in this figure. This process is repeated for each of the 4 pads.

When the process is entered, the values for *IGNDi* and *ISLIDEi* will initially be those from the last time the process was applied. Block $\boxed{1}$ is entered first.

Block $\boxed{1}$ – On-Ground Check

The on-ground check is carried out by determining whether the *i*-th pad would be in ground contact if the deflection angle ϕ_{SKi} were set to the value it has when the *i*-th pad is in the air as given by (3.2.11) and (3.2.12). Define *GNDCHKi* to be the value of $POSNi_{-E}$ when ϕ_{SKi} is set to its value when in the air. It then follows that

$$\begin{aligned} &\text{if } GNDCHKi \geq 0 \\ &\qquad\qquad \text{then } IGNDi = 1 \\ &\text{if } GNDCHKi < 0 \\ &\qquad\qquad \text{then } IGNDi = 0 \end{aligned} \tag{3.2.67}$$

In the present model it is assumed that immediately following touchdown, the *i*-th pad is sliding. Thus $IGNDi_{old} = 0$, for the previous value of *IGNDi*, indicates that the *i*-th pad was previously in the air, and if the current value of *IGNDi* indicates that it is now on the ground, then set

$$ISLIDEi_{old} = 1 \tag{3.2.68}$$

The logic flow is directed to Block $\boxed{6}$ if the *i*-th pad is in the air (i.e., $IGNDi = 0$). The logic flow is directed to Block $\boxed{2}$ if the *i*-th pad is on the ground (i.e., $IGNDi = 1$).

Block $\boxed{2}$ – Initial Sliding Check

This block directs the flow of logic based on $ISLIDEi_{old}$, the previous value of $ISLIDEi$. If the i-th pad is currently on the ground and was previously sliding (i.e., $ISLIDEi_{old} = 1$) then the logic flow is directed to Block $\boxed{3}$, otherwise it is directed to Block $\boxed{4}$.

Block $\boxed{3}$ – Next Sliding Check 1

This block is used to check the sliding status of the i-th pad given that it is currently on the ground and was previously sliding. The sliding check is based on (3.2.52).

$$\text{if } |\underline{r\dot{U}i}_V| > V_{REF}$$

$$\text{then } ISLIDEi = 1 \tag{3.2.69}$$

$$\text{if } |\underline{r\dot{U}i}_V| \leq V_{REF} \text{ and } |\underline{r\dot{U}i}_V|_{old} \leq V_{REF} \text{ and } t \leq T_{LG}$$

$$\text{then } ISLIDEi = 1 \tag{3.2.70}$$

$$\text{but if } t > T_{LG}$$

$$\text{then } ISLIDEi = 0$$

(where t is the time from the start of sliding)

$$\text{if } |\underline{r\dot{U}i}_V| \leq V_{REF} \text{ and } \qquad |\underline{r\dot{U}i}_V|_{old} > V_{REF}$$

$$\text{then } ISLIDEi = 0 \tag{3.2.71}$$

where $|\underline{r\dot{U}i}_V|_{old}$ is the previous value of $|\underline{r\dot{U}i}_V|$. $\underline{r\dot{U}i}_V$ is found using (3.2.39). If the i-th pad is currently sliding then the logic flow is directed to Block $\boxed{5}$, otherwise it is directed to Block $\boxed{4}$. If this block finds that $ISLIDEi = 0$ then the value of $\underline{rDi}_E$ is updated by (3.2.54).

Block $\boxed{4}$ – Next Sliding Check 0

This block is used to check the sliding status of the *i*-th pad given that it is currently on the ground and was previously sticking. The sliding check is based on (3.2.50)

$$\text{if } |\underline{Li}_V| > \mu_{ST}|N_i| \quad \text{then } ISLIDEi = 1 \qquad (3.2.72)$$

$$\text{if } |\underline{Li}_V| \le \mu_{ST}|N_i| \quad \text{then } ISLIDEi = 0 \qquad (3.2.73)$$

$\underline{Li}_V$ and *Ni* are found by employing procedure blocks Calculate L_V When Sticking and Calculate NF. These are described in Section 3.2.3 (see block diagram 1.1.2.3). If the *i*-th pad is currently sliding (*ISLIDEi* = 1) then the logic flow is directed to Block $\boxed{5}$, otherwise it is directed to Block $\boxed{6}$.

Block $\boxed{5}$ – Sliding Calculations

$\underline{Li}_V$ and *Ni* are found by employing procedure blocks L_V When Sliding and Calculate *NF*. These are described in Section 3.2.3 (see block diagram 1.1.2.4). The logic flow is then directed to Block $\boxed{6}$.

Block $\boxed{6}$ – Gear Forces and Moments

If this block is entered with *IGNDi* = 0 then the landing gear forces and moments applied to the helicopter body are set to zero

$$\begin{aligned} \underline{fLGi}_B &= \underline{0} \\ \underline{GLGi}_B &= \underline{0} \end{aligned} \qquad (3.2.74)$$

If *IGNDi* = 1 then the forces and moments are calculated using (3.2.55) and (3.2.56).

The logic flow is complete at this point.

3.2.2 MATRIXx

The block diagrams and block script code for the Landing Gear module are found in Appendix D and E, respectively. Tilda and Transpose Blocks are used at several locations as required.

The present module has several levels of blocks within blocks. This can be seen from the block diagrams at the end of this section. To help in tracing paths through these blocks, a Table of Contents for the pages of block diagrams is provided. The title of each page is given along with a numerical page identifier (appearing at the top of the block diagram page). The identifier may be construed as a section or subsection number in a written report. For example, the block of subsection 1.1.1.2 is the second block for which a separate block diagram is provided, on the page representing subsection 1.1.1. Since the same block form may be used at several locations in the Landing Gear module, in order to eliminate duplicate block diagrams, the form of a repeated block can be found by going to the page subsection number in brackets following the block title in the Table of Contents. For example the block diagram for 1.1.1.2.1.1, Calculate POSN_E, can be found by going to subsection 1.1.1.1. It should be noted that all the subsection numbers associated with a block diagram page are given at the top of that page. The blocks which are encompassed within the logic blocks of Figure 3.2.4 are also indicated in the Table of Contents. A Table of Contents is also provided for the pages of block script following the block diagrams. These are listed in alphabetical order.

The system of blocks is configured to find the forces and moments associated with the i-th strut/pad combination. The system is structured as a big loop with the index "gearN" serving as a counter. As gearN steps up from 1 to 4, the results for the 4 strut/pad combinations are determined in sequence. This can be seen in the use of the "while" loop in Block 1. (Landing Gear). When gearN reaches 4 the value of Break is changed from 0 to 1 and the Break Block terminates the loop. The same process is employed in Block 1.1.1.2.1 (Calculate PHISK) where a "while" loop is used to repeatedly improve the estimate of PHISK as part of the Newton-Raphson method (see Appendix A.2). When the tolerance set by the parameter TOLERANCEPHISK is met, the loop is broken.

In some blocks (e.g., Sticking ICs) the subscript $(X)_{old}$ is used in the name of the block input when the name (X) is used for a block output. This is necessary because MATRIXx does not allow the same name to be used for both an input and an output variable in the same block. In some instances, when a block is part of a "while" loop, it is necessary to refill the elements of the vector variable which is being worked on because the MATRIXx code resets them to zero when the block is first entered. The block DEF_V also serves the same purpose.

Several blocks in this module are Condition Blocks (e.g., Touchdown Variable Init). When the first (top) listed input to the block is positive it will execute, otherwise it is skipped. For the Calculate PHISK block it was found that the inputs were subjected to an unexplained shift in order (by one) when it was attempted to run it not as a condition block.

The OR, NOT, AND Logical Operator Functions are standard logic blocks. Several Logical Expression Blocks were used in this module to control the integrators. An example of this is block 61 within the block LF_V_DD Integrator. The equation $Y=U=i$ stands for the logic that when the input gearN equals i then the output equals 1. Otherwise the output equals 0. This output then goes into the Conditional Block containing the integrator, and is used to turn it on and off as required.

In block diagrams, inputs to a block can come from input lines or from Read from Blocks within the block. In the latter case, the variables must have been previously passed through a Write to Block. For example, in Block 1. (Landing Gear), the matrix L_B_V(L,M) is created and passed through a Write to Block. Then in Block 1.1.2.3.3 (Calculate NF), the input to block Calculate GJ and R appears as a single input line (gearN). However, in Block 1.1.2.3.3.1 showing the details of block Calculate GJ and R, it is seen that L_B_V(L,M) enters as an input through a Read from Block (along with many other variables).

The initial conditions applied to the integrators in this module are designed to handle the starting and stopping of the filters to which they apply. Consider the deformation filters. These filters are only active when the i-th strut/pad is sliding. When sliding starts, the integrators must be reset to the deformation values at the end of the previous sticking phase. This is handled by a combination of blocks Touchdown

Variable Init and Sliding Variable Init. Consider the integrator shown in Figure 3.2.5. The initial conditions are set by adding a fixed value of C to the output of the integrator. As implemented by MATRIXx, the output of the integrator, A, is incremented after each time step Δt by the integral of the input over Δt. When the filter is stopped, this value of A remains on the output and is there when the filter is started again. For this reason, if we wish to start up with a new initial condition x_{ic} following a start/stop cycle, then the new value of C used must be

$$C = x_{ic} - A_{old} \tag{3.2.75}$$

where A_{old} was the value of A when the filter was previously stopped. Now since

$$x = B = A + C \tag{3.2.76}$$

thus A_{old} is given by

$$A_{old} = B_{old} - C_{old} \tag{3.2.77}$$

and the new C is (from (3.2.75) and (3.2.77))

$$C = x_{ic} + C_{old} - B_{old} \tag{3.2.77}$$

and (3.2.78) is used to create DEF_Vic(K,gearN) in this module.

The variable INITVARS is used to initiate the initial condition determination process when it is switched from 0 to 1.

Following touchdown, the block Location Vars Init sets all the integrator initial conditions to zero in accordance with the physical model.

In order to prevent the unnecessary repetition of calculations of the same variable while progressing through the Landing Gear blocks shown in Figure 3.2.4, condition blocks have been used to skip around blocks when appropriate. For example, immediately following touchdown of the i-th strut/pad, the operations in Block 1.1.1.2 (Touchdown Variable Init), a block contained in Block 1.1.1 (On Ground Check), include the calculation of PHISK with the helicopter's i-th strut/pad on the ground. These operations are part of Block $\boxed{1}$ of Figure 3.2.4. On subsequent passes through Block $\boxed{1}$

with the *i*-th strut/pad on the ground, PHISK is not computed until either Block $\boxed{3}$ or $\boxed{4}$ of Figure 3.2.4 is reached. However, these latter PHISK calculations can be skipped on the pass immediately following touchdown of the *i*-th strut/pad as indicated above. This is accomplished by creating the logic variable CALCPHISK in Block 1.1.1 (On Ground Check). CALPHISK is set to zero if the *i*-th strut/pad has touched down during the present pass through the Landing Gear module. This zero value is used in Blocks 1.1.2.2.1 and 1.1.2.3.1 (Calculate Gear Orientation) to skip around the Condition Block Calculate PHISK.

In Figure 3.2.4 the path from Block $\boxed{3}$ to Block $\boxed{4}$ is taken if the *i*-th strut/pad has gone from sliding to sticking on the current pass through the Landing Gear module. The purpose of this is to allow the forces L and N (when sticking) to be calculated using the computations in Block $\boxed{4}$. The operations of Block $\boxed{4}$ in Figure 3.2.4 are implemented in MATRIXx Block 1.1.2.3 (Next Sliding Check 0). If Block $\boxed{4}$ is entered from Block $\boxed{3}$ then we want to skip the blocks Calculate Gear Orientation and Sliding Check contained in Block 1.1.2.3 because their operations would be redundant. This is accomplished by making them Condition Blocks controlled by the logic variable GOTO4FR2. When GOTO4FR2 is zero (which happens when Block $\boxed{4}$ is entered from Block $\boxed{3}$) then these blocks are skipped. The appropriate value for GOTO4FR2 is established in Block 1.1.2 (On Ground Calculations). If the flow is directed from Block $\boxed{2}$ to Block $\boxed{4}$ in Figure 3.2.4, then in MATRIXx Block 1.1.2 the block Initial Sliding Check sets GOTO4FR2 to unity and this value cannot be changed until the next pass through the Landing Gear module. If the flow is not directed from Block $\boxed{2}$ to Block $\boxed{4}$ then GOTO4FR2 is set to zero and the only other way Block $\boxed{4}$ can be reached is from Block $\boxed{3}$. OR Blocks and Condition Blocks are used to direct the flow to Blocks $\boxed{4}$ and $\boxed{5}$.

The flow from Block $\boxed{4}$ to either Block $\boxed{5}$ or Block $\boxed{6}$ of Figure 3.2.4 is accomplished by the combination of logic blocks at the output of block Sliding Check contained in Block 1.1.2.3 (Next Sliding Check 0). If Block $\boxed{4}$ was entered from Block $\boxed{3}$ (indicated by GOTO4FR2 equal to zero) then we know that the *i*-th strut/pad is sticking and that after finding L and N we want to enter Block $\boxed{6}$. This is achieved in Block 1.1.2.3 by the AND Blocks. GOTO5 is set to zero by the upper AND Block when GOTO4FR2 is zero. The zero output from the lower AND Block when combined with the output of unity from the NOT block (when GOTO4FR2 is zero) in the OR Block, results in GOTO6 equal to unity. The combined effect directs the flow to Block $\boxed{6}$. If Block $\boxed{4}$ was entered from Block $\boxed{2}$ then GOTO4FR2 is unity and the output from the NOT Block is zero. This results in the flow being directly controlled by the GOTO5 and GOTO6 outputs from the block Sliding Check.

In a similar fashion, if you enter Block $\boxed{4}$ from Block $\boxed{3}$ in Figure 3.2.4 RU_V_D will have been calculated in Block $\boxed{3}$. For this case the calculation of RU_V_D should be skipped in Block $\boxed{4}$. This calculation is performed in MATRIXx Block 1.1.2.3.2 (Calculate L_V When Sticking) by the Condition Block Calculate RU_V_D which is controlled by the logic variable GOTO4FR2. If GOTO4FR2 is zero, indicating that Block $\boxed{4}$ was entered from Block $\boxed{3}$, then the calculation of RU_V_D is skipped as desired.

Several of the block diagrams have a pair of vertical lines running from top to bottom, dividing them into a left side and a right side. This represents a sequencer and it ensures that all the operations to the left of the division will be carried out before those to the right.

Table 3.2.1 lists the parameter data required to run this module.

3.2.3 Landing Gear Notation

Units	Variable	Computer Name	Description
	-	Break	logic parameter to determine when a "while" loop is finished
	-	CALCPHISK	logic variable, normally zero but set to one on pass in which touchdown occurs for a strut/pad
	$C1, C2$	COEF1, COEF2	deformation coefficients
rad	$\underline{E}_B$	-	Euler angles $[\phi \quad \theta \quad \psi]_B^T$ which carry F_E into F_B
rad	$\underline{E}_{SKi}$	-	Euler angles $[\phi \quad 0 \quad 0]_{SKi}^T$ which carry F_B into F_{SKi}
N	$\underline{fLG}_B$	FLG_B(K)	total force applied to the helicopter by the landing gear, expressed in F_B
N	$\underline{fLGi}_B$	FLG_B_(K,i)	force applied to the helicopter by the i-th strut/pad, expressed in F_B
N.m	$\underline{GJi}_{SKi}$	-	torque applied to the helicopter body about the fuselage attachment point by the i-th torsional spring and damper
N.m	$\underline{GJi}_{xSKi}$	GJ(i)	GJi_{xSKi}
N.m	$\underline{GLG}_B$	GLG_B(K)	total moment applied about the helicopter body CG by the landing gear, expressed in F_B
N.m	$\underline{GLGi}_B$	GLG_B(K,i)	moment applied about the helicopter body CG by the i-th pad expressed in F_B
	$GNDCHKi$	GNDCHK(i)	value of $POSNi_{zE}$ when ϕ_{SKi} has its airborne value
	-	GOTOm	logic variable set to 1 to direct flow to Block $\boxed{m}$

Units	Variable	Computer Name	Description
	-	GOTO4FR2	logic variable set to 1 to direct flow from Block $\boxed{2}$ to Block $\boxed{4}$. It is zero otherwise
	-	GOTO4FR3	logic variable set to 1 to direct flow from Block $\boxed{3}$ to Block $\boxed{4}$. It is zero otherwise
m/rad	-	GRADPOSN_E3(i)	$\dfrac{\partial POSNi_{\underline{E}}}{\partial \phi_{SKi}}$
	i	gearN	index representing i-th strut/pad when i=*gearN*
	IGNDi	IGND(i)	logic parameter indicating i-th pad on ground
	-	INITVARS	logic parameter used to trigger a change in initial conditions for the integrators
	ISLIDEi	ISLIDE(i)	logic parameter indicating i-th pad sliding
	$\underline{L}_{BE}$	L_B_E(L,M)	rotation matrix from F_E to F_B
	$\underline{L}_{BSKi}$	L_B_SKN(L,M)	rotation matrix from F_{SKi} to F_B when i=gearN
	$\underline{L}_{BV}$	L_B_V(L,M)	rotation matrix from F_V to F_B
	$\underline{L}_{SKiV}$	L_SKN_V(L,M)	rotation matrix from F_V to F_{SKi} when i=gearN
	$\underline{L}_{VE}$	L_V_E(L,M)	rotation matrix from F_E to F_V
N	-	LF_TOT	magnitude of $\underline{LFi}_V$
N	$\underline{Li}_V$	L_V(K,i)	in-plane reaction force of the ground on the i-th pad. expressed in F_V

Units	Variable	Computer Name	Description
	-	NCOUNT(i)	counter for *i*-th strut/pad used to determine time since current sliding started
N	Ni	NF(i)	$Ni_{:V}$
N	$\underline{Ni}_V$	-	normal reaction force of the ground on the *i*-th pad, expressed in F_V
	-	NREF	value of NCOUNT(i) when time T_{LG} has been reached during sliding
rad^2/s^2	-	OMEGALG2	ω_{LG}^2
m	$\underline{POSNi}_E$	POSN_E(K,i)	location of the *i*-th pad relative to the origin of F_E
m	$\underline{rDi}_V$	RD_V(K,i)	location of the deformed *i*-th pad relative to the origin of F_V
m	$\underline{rG}_E$	RG_E(K)	location of local ground plane reference point relative to the origin of F_E
m	$rGSi$	RGS(i)	$rGSi_{:SKi}$
m	$\underline{rGSi}_{SKi}$	-	location of the *i*-th pad relative to the origin of F_{SKi}
m	$rG_{:E}$	RG_E3	
m	$\underline{rSKi}_B$	RSK_B(K,i)	location of *i*-th strut attachment point relative to origin of F_B
m	$\underline{rUi}_V$	-	$\underline{POSNi}_V = \underline{L}_{VE}\, \underline{POSNi}_E$
m	$\underline{rV}_E$	RV_E(K)	location of the origin of F_V relative to origin of F_E
	$Ri_{21}, Ri_{22}, Ri_{23}$	R21(i), R22(i), R23(i)	second row of $\underline{L}_{SKiV}$

Units	Variable	Computer Name	Description
m	-	RGSN_B(K)	$\underline{L}_{BSKi}\underline{rGS}_i$ when i=gearN
m	-	RSKN_B(K)	RSK_B(K,gearN)
m	-	RSK_RGS_B(K)	RSK_B(K)+L_B_SKN(L,3)RGS(gearN)
m/s	$RUVDi$	RUVD	magnitude of $\underline{r\dot{U}i}_V$
s	T_{LG}	TLG	timing parameter when sliding
	-	TOLERANCEPHISK	tolerance parameter in the Newton-Raphson method
m/s	V_{REF}	VREF	velocity parameter when sliding
m/s	$\underline{VB}_E$	VB_E(K)	inertial velocity of the CG of the helicopter body expressed in F_E
m	$\underline{\Delta i}_V$	DEF_V(K,i)	deformation of the i-th pad expressed in F_V
s	Δt	DT	time step in numerical integration
rad	$\Delta\phi_{SKi}$	-	change in ϕ_{SKi} from its airborne value
	$\kappa 1$, $\kappa 2$	KAPPA1, KAPPA2	spring and damping coefficients for the struts
	μ_{SL}	MUSL	sliding friction coefficient
	μ_{ST}	MUST	sticking friction coefficient
rad	ϕ_B, θ_B, ψ_B	PHIB, THETB, PSIB	Euler angles which carry F_E into F_B
rad	ϕ_{SK0}	PHISK0	value of $\lvert\phi_{SKi}\rvert$ when the i-th pad is airborne
rad	ϕ_{SKi}	PHISK(i)	Euler angle which carries F_B into F_{SKi}

Units	Variable	Computer Name	Description
rad/s	$\underline{\omega B_B}$	OMEGAB_B(K)	inertial angular velocity of the helicopter body, expressed in F_B
rad/s	$\underline{\omega V_V}$	OMEGAV_V(K)	angular velocity of F_V relative to F_E, expressed in F_V
	-	$(\ )_{ic}$	initial condition
	-	$(\ )_{\text{slic}}$	part of initial condition to be used when the next slide starts
	-	$(\ )_{stic}$	part of initial condition to be used when the next sticking condition starts

3.3 References

3.1 Reid, L.D., Haycock, B.C., de Leeuw, J.H., and Graf, W.O., 2000, "Flight Dynamics for Helicopter Emergency Maneuvers Trainer", DCIEM Contract W7711-005923/001/TOR.

3.2 Howlett, J. J., 1981, "UH-60A Black Hawk Engineering Simulation Program: Volume I-Mathematical Model," NASA CR 166309.

3.3 Chen, R. T. N., 1989, "A Survey of Nonuniform Inflow Models for Rotorcraft Flight Dynamics and Control Applications," presented at the 15th European Rotorcraft Forum.

3.4 Bramwell, A. R. S., 1976, "Helicopter Dynamics," Edward Arnold Publishers Ltd.

3.5 Ballin, M. G., and Dalang-Secretan, M. A., 1990, "Validation of the Dynamic Response of a Blade-Element UH-60 Simulation Model in Hovering Flight," presented at the 46th Annual Forum of the American Helicopter Society, Washington, D.C.

3.6 Etkin, B., and Reid, L. D., 1996, "Dynamics of Flight, Stability and Control," John Wiley and Sons, New York.

3.7 McCormick, B. W., 1995, "Aerodynamics, Aeronautics and Flight Mechanics," John Wiley and Sons, New York.

3.8 Padfield, G. D., 1996, "Helicopter Flight Dynamics: The Theory and Application of Flying Qualities and Simulation Modeling," AIAA Education Series.

3.9 Talbot, P. D., Tinling, B. E., Decker, W. A., and Chen, R. T. N., 1982, "A Mathematical Model of a Single Main Rotor Helicopter for Piloted Simulation," NASA TM 84281.

3.10 Blackwell, J., 1991, "Structure of Undercarriage Model as Used in Seahawk/FFG-7 Simulation Program," DSTO, ARL Melbourne, Group Report RW 91/2.

3.11 Reid, L. D., Pierce, A. A., Graf, W. O., Zielinski, A. W., and Dufort, P. A., 1996, "Study of Human Factors Affecting the Use of Head-Mounted Displays in a Helicopter Deck Landing Simulator," UTIAS Report No. 353.

3.12 Reid, L. D., Rajagopal, P., and Graf, W. O., 1993, "A Simulator Investigation of Helicopter Flight Control System Mode Transitions," University of Toronto Institute for Aerospace Studies, Report No. 348.

3.13 Newman, S., 1994, "The Foundations of Helicopter Flight," Edward Arnold, London.

3.14 Etkin, B., 1972, "Dynamics of Atmospheric Flight," John Wiley and Sons, New York.

3.15 ARMCOP-UH1 Simulation

Table 3.1.1

<u>Main Rotor Module</u>

<u>Required Helicopter Data</u>

B0

CD(I,J)

CL(I,J)

IBL(L,M)

K0

K1

MBLADE

MINV(L,M)

NL

OMEGALP

PHASE

PI

R

RCG

RS_B(K)

R0

R2D

WBLADE

ZLL

Table 3.2.1

Landing Gear Module

Required Helicopter Data

COEF1

COEF2

KAPPA1

KAPPA2

MUSL

MUST

PHISKO

RGS(N)

RSK_B(K,I)

TOLERANCEPHISK

VREF

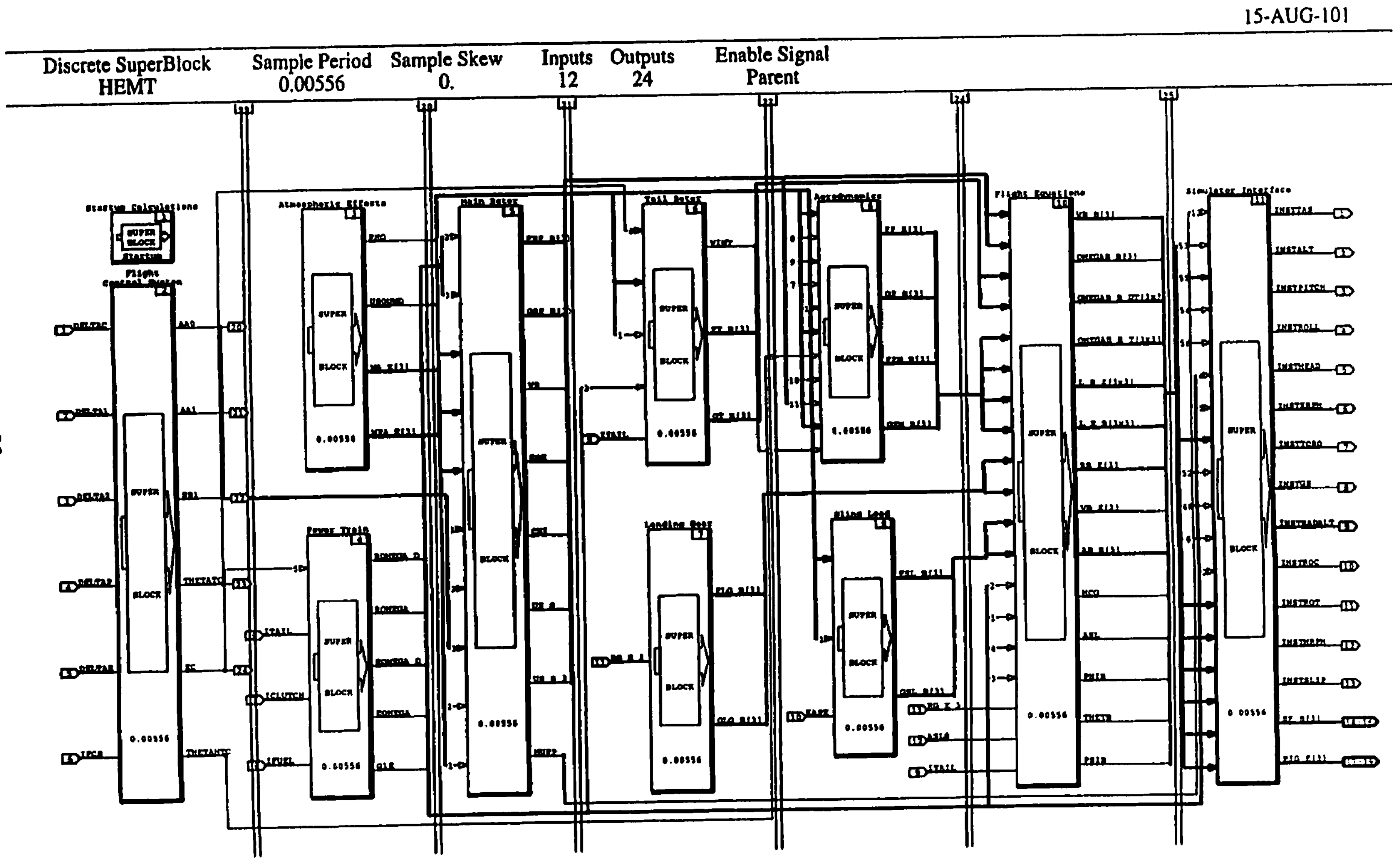


Figure 3.1 Module Interconnections

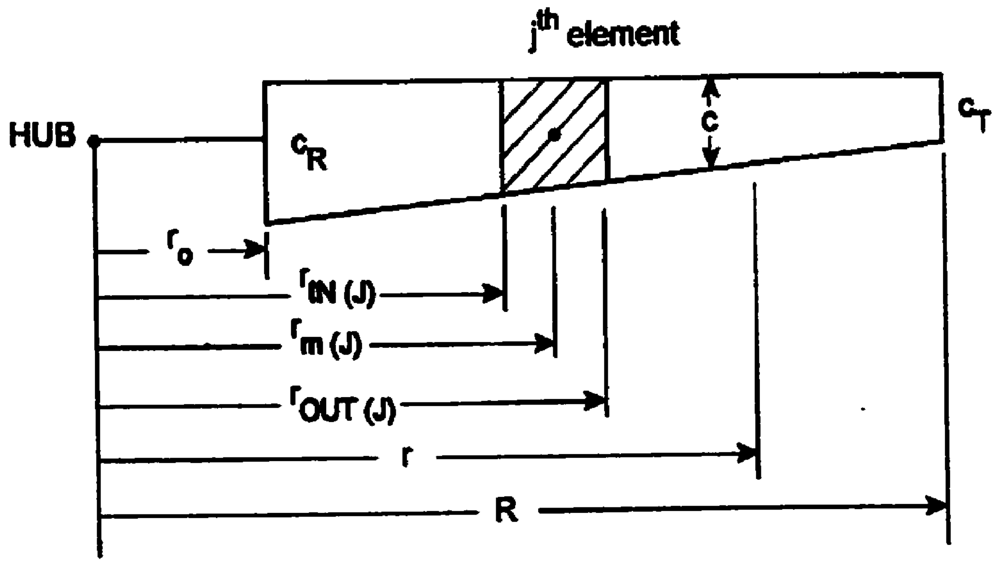


Figure 3.1.1 Main Rotor Blade Geometry

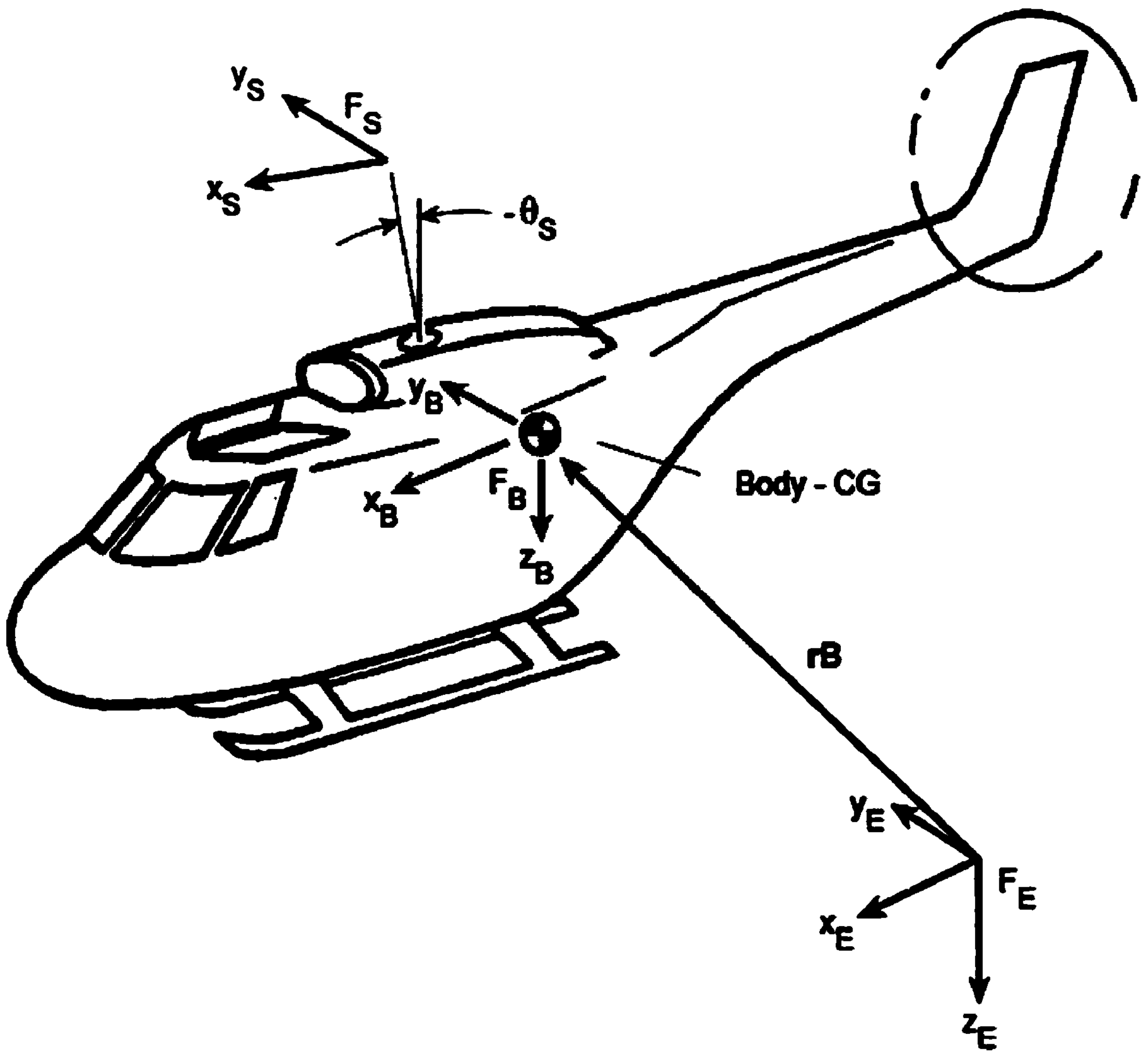


Figure 3.1.2 Helicopter Reference Frames

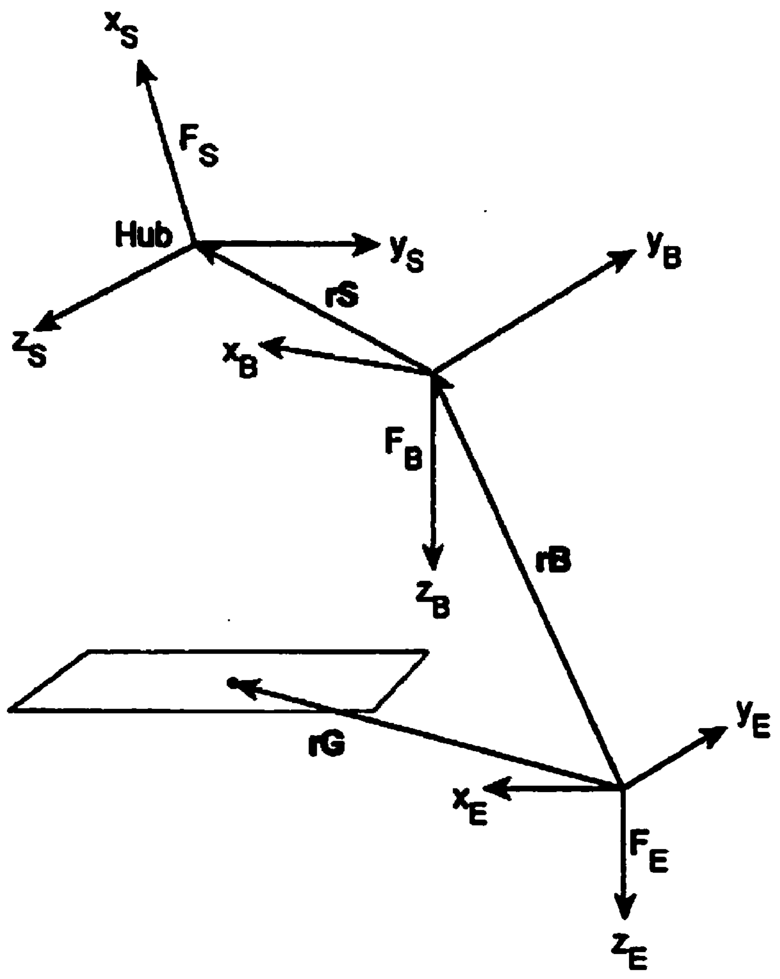


Figure 3.1.3 Simulation Reference Frames

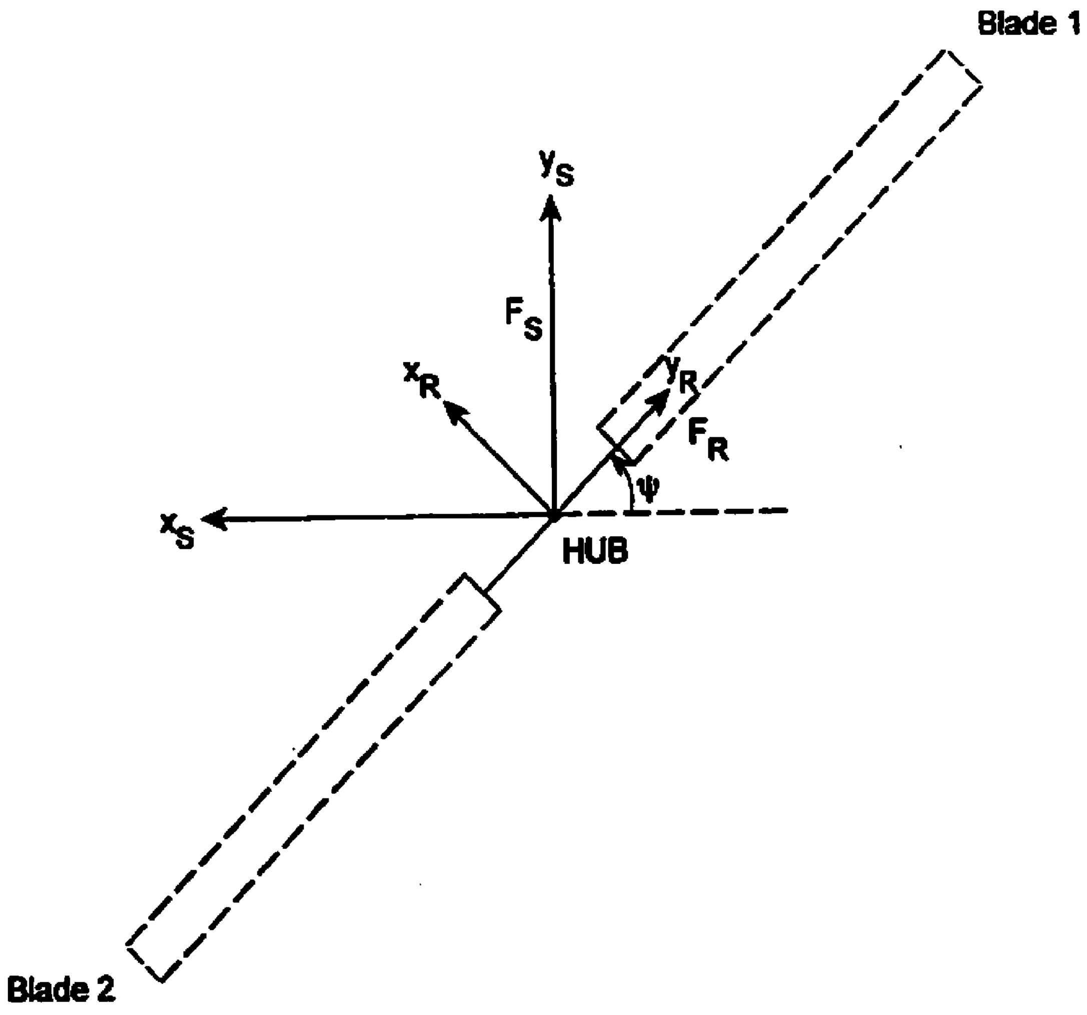


Figure 3.1.4 Main Rotor Reference Frames

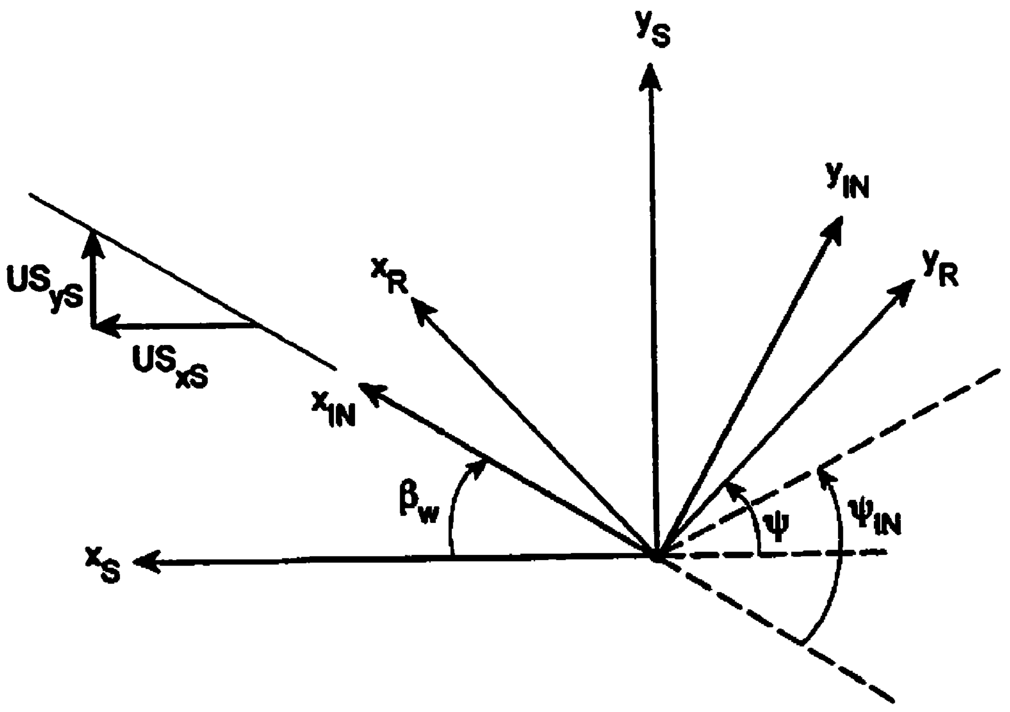


Figure 3.1.5 Reference Frame F_{IN}

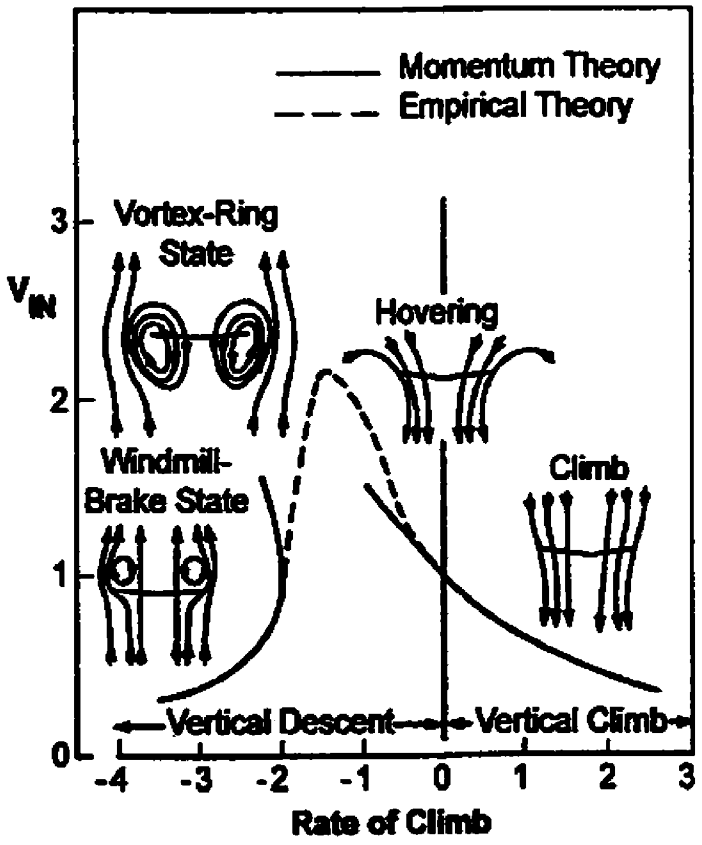


Figure 3.1.6 Main Rotor Inflow (Ref. 3.3)

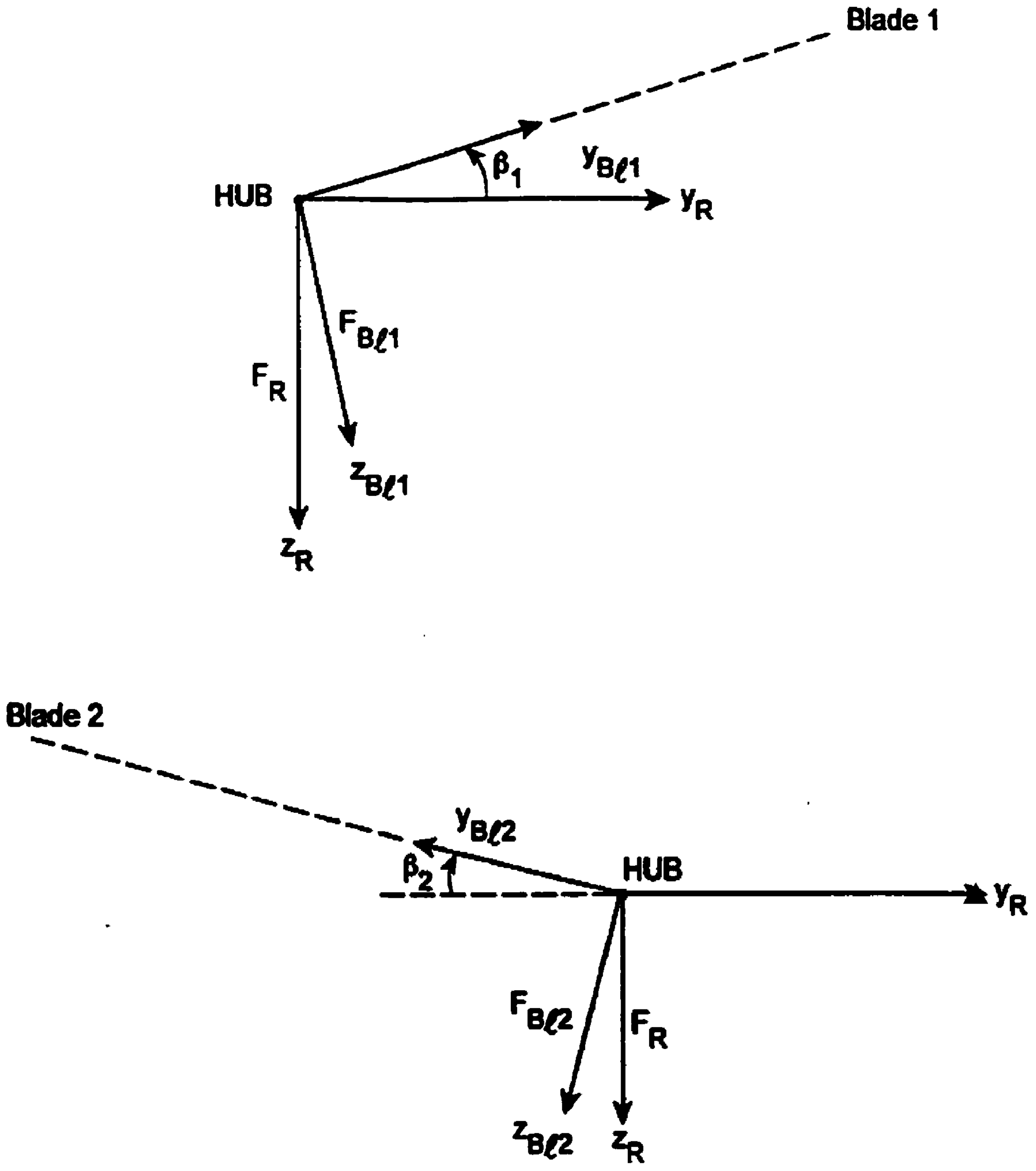


Figure 3.1.7 Blade Flapping Angles

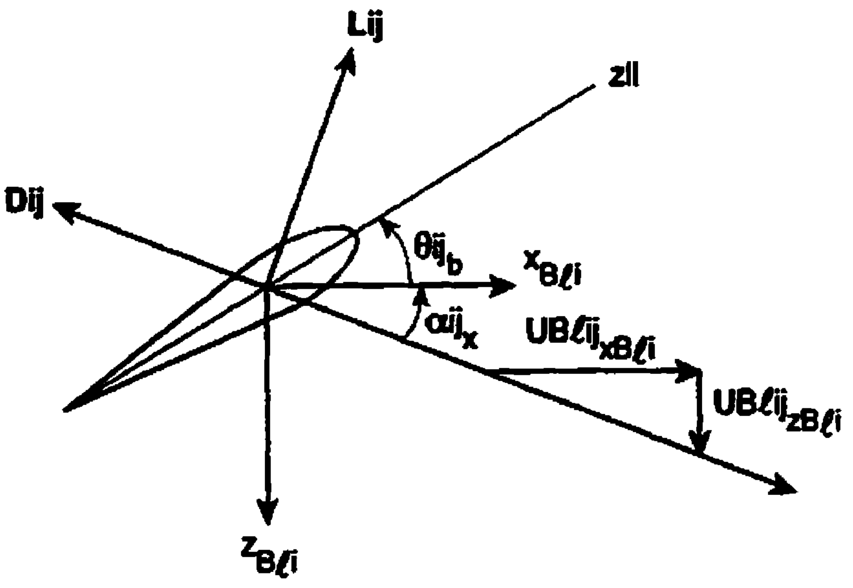


Figure 3.1.8 Blade Element Angle of Attack

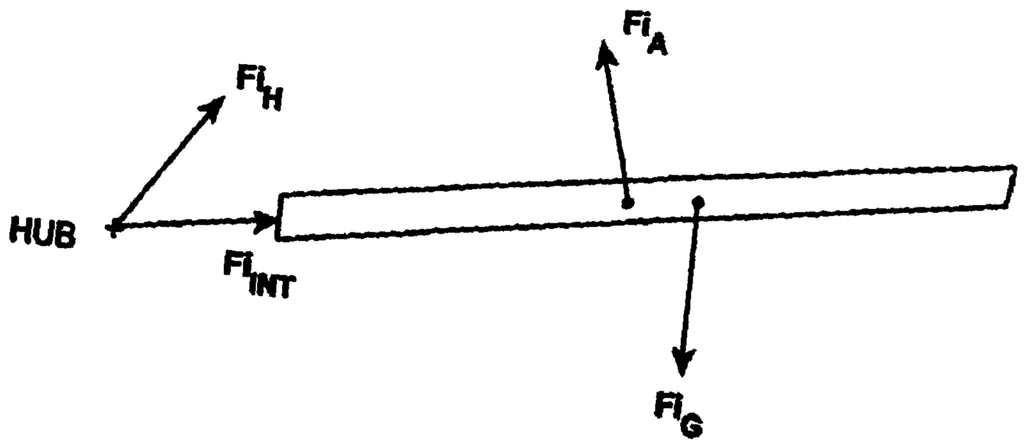


Figure 3.1.9 Forces Acting on a Single Main Rotor Blade

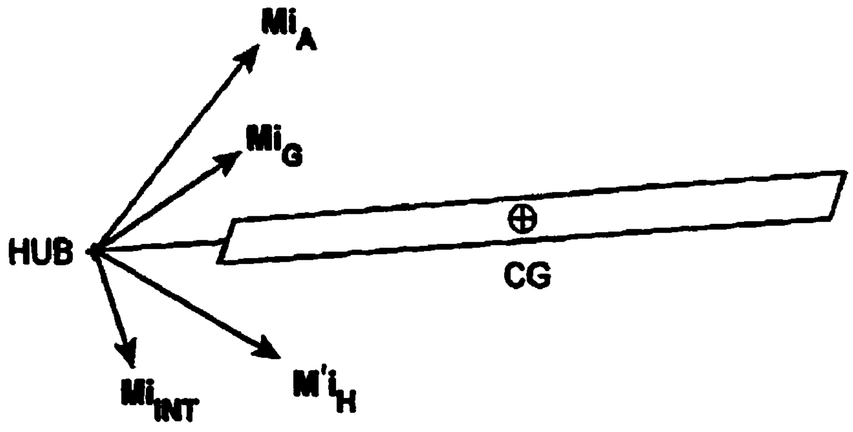


Figure 3.1.10 Moments Acting on a Single Main Rotor Blade

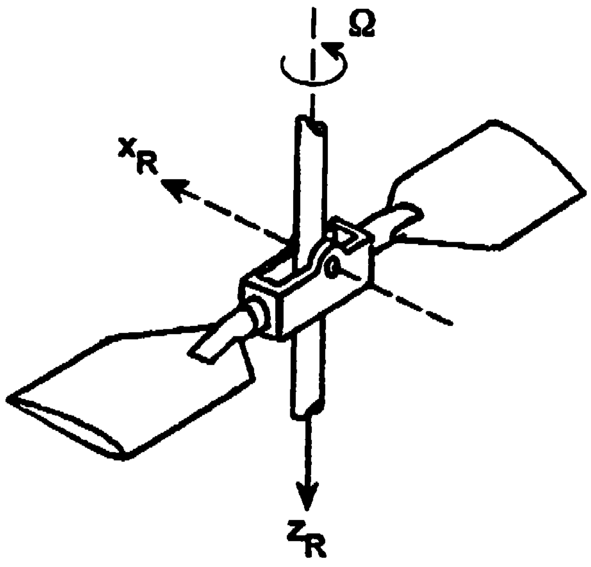


Figure 3.1.11 Teetering or See-Saw Rotor

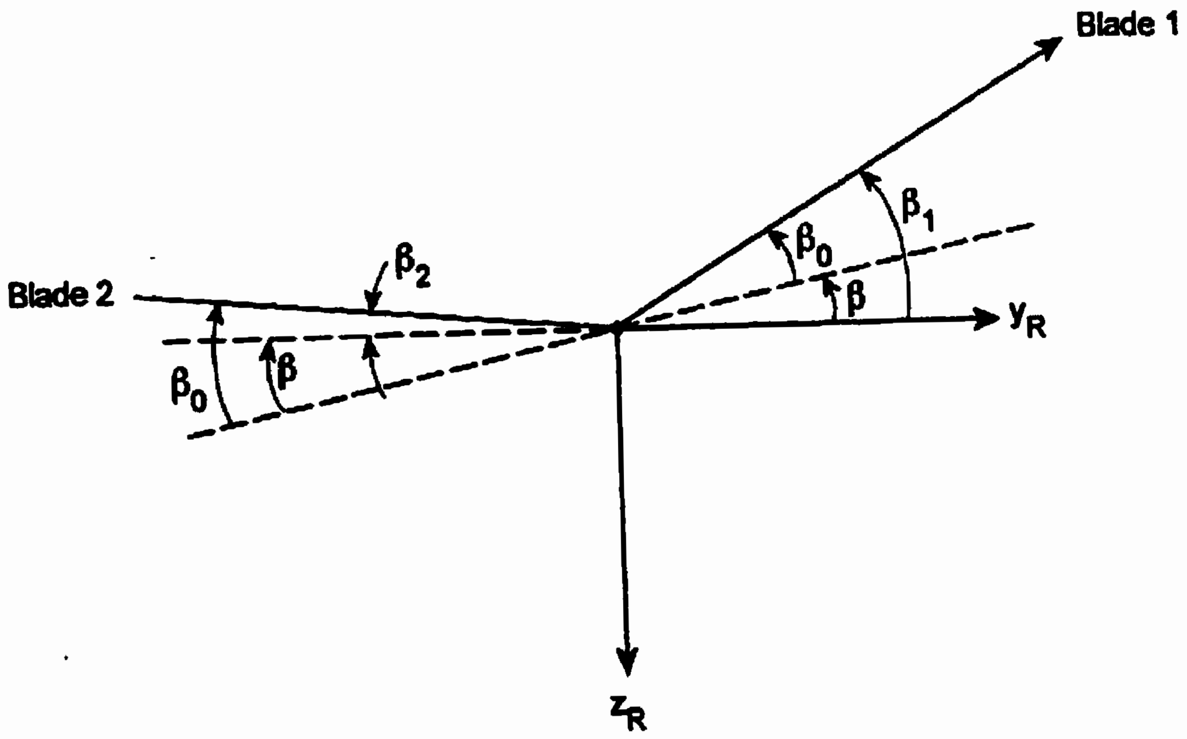


Figure 3.1.12 Main Rotor Flapping Geometry

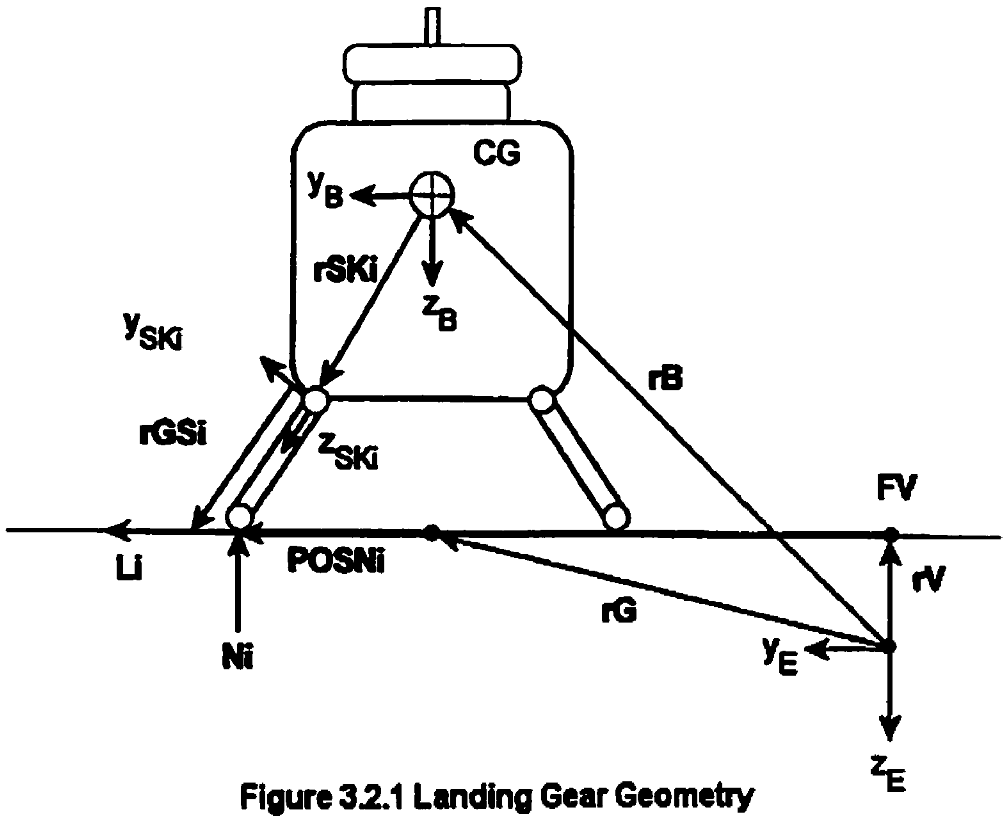


Figure 3.2.1 Landing Gear Geometry

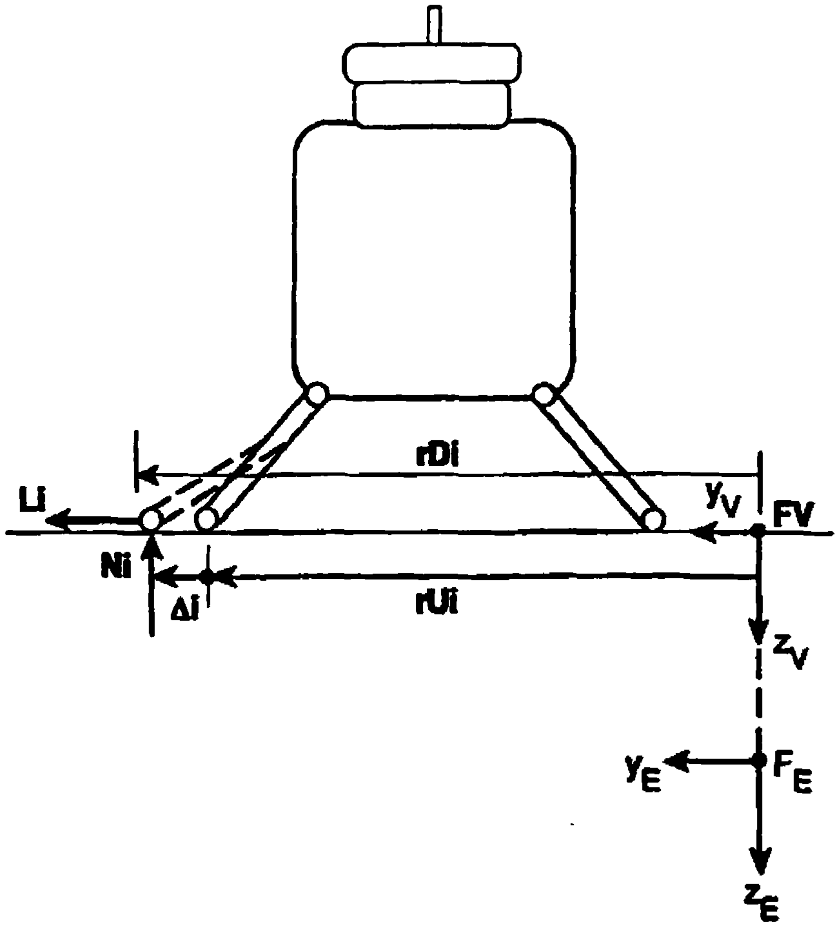


Figure 3.2.2 Landing Gear Deformation Geometry

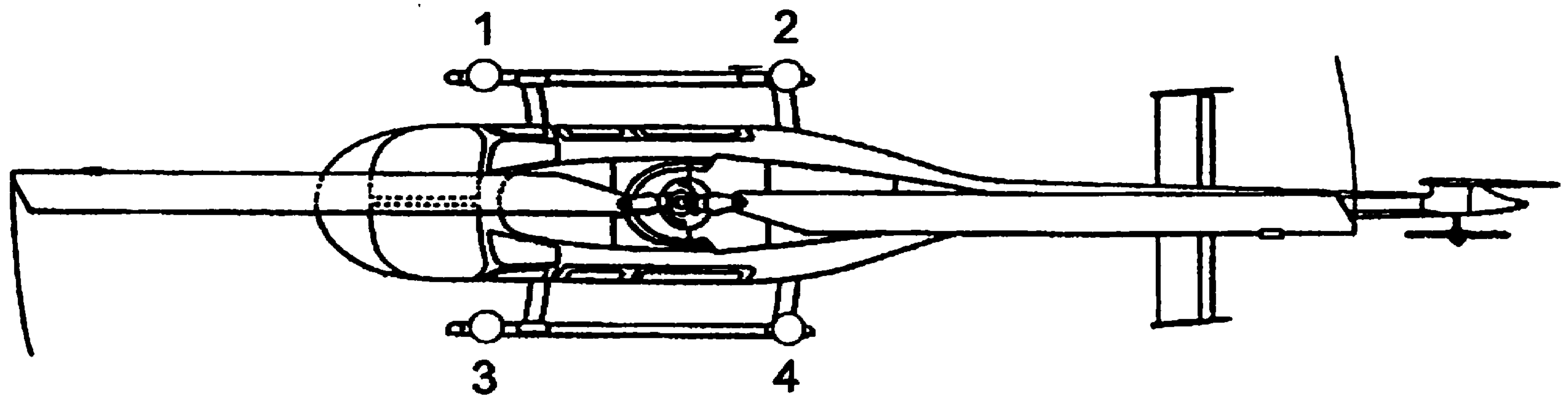


Figure 3.2.3 Landing Gear Pads

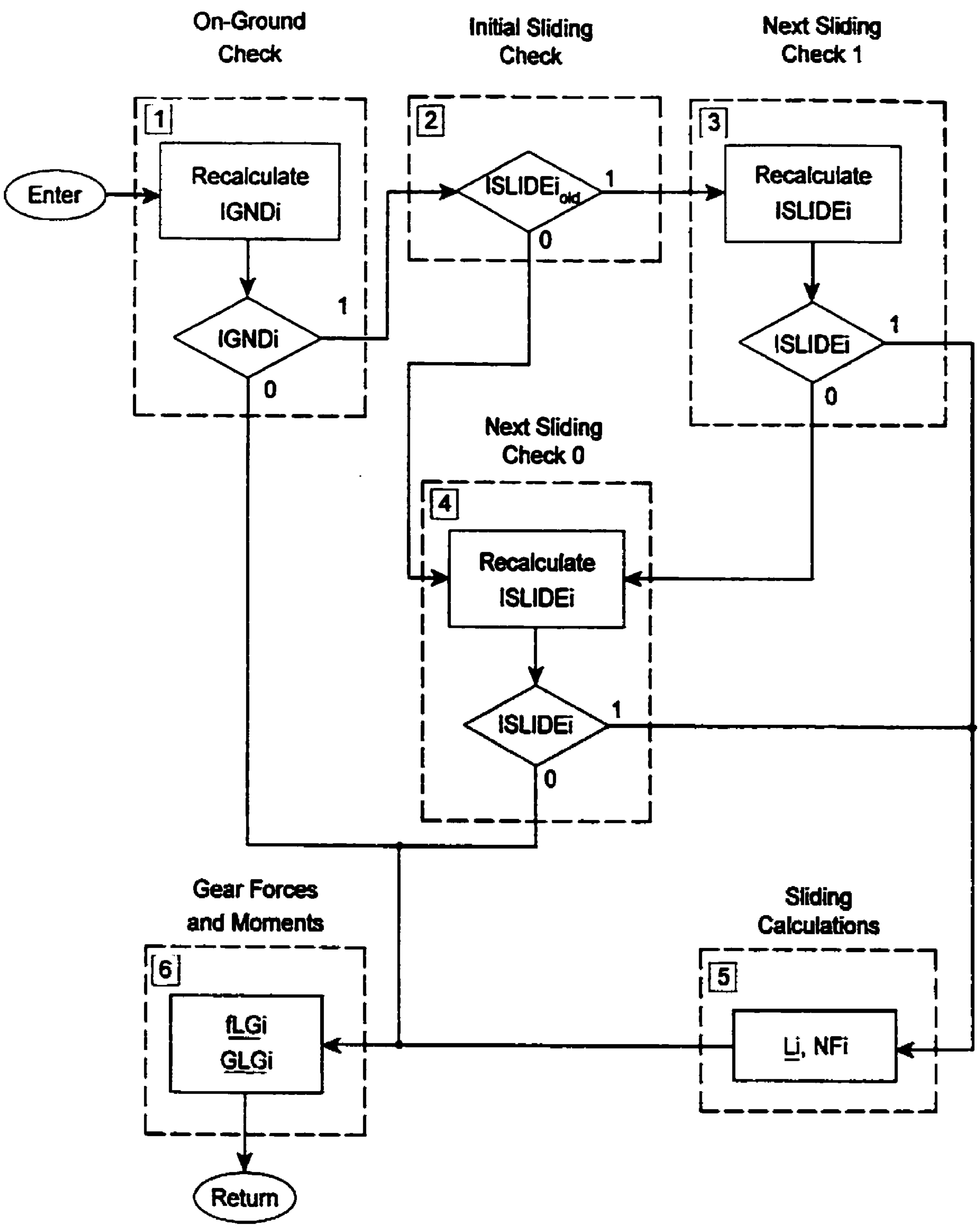


Figure 3.2.4 Logic Blocks For i-th Strut/Pad

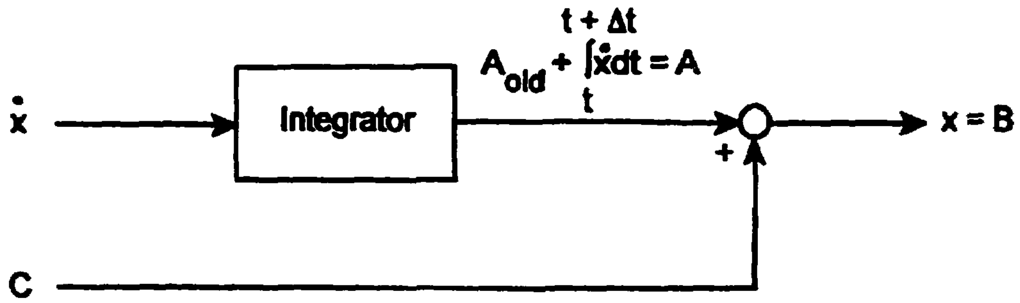


Figure 3.2.5 Integrator Initial Condition

4. Notable Developments

4.1 SystemBuild

As one might expect, quite a few obstacles were encountered while developing this model that had to be overcome to achieve the desired results from the SystemBuild code. Many of these problems were due to either SystemBuild not being able to do what was expected, or a lack of guidance in the documentation as to how these things could be accomplished. A few notable cases are documented here.

4.1.1 Matrices

Early on in the development it was discovered that blocks within SystemBuild could only handle single dimensional arrays as inputs and outputs, preventing the use of matrices, quite a surprise for a program named MATRIXx. To get around this issue, matrices had to be converted into vectors by stringing the rows together before being output from blocks, and then reformed into the original matrix after being taken as input to the next block. This problem was partially addressed with the release of MATRIXx Version 6.1. The available blocks within SystemBuild now include a subset for matrix operations, including such things as matrix multiplication and transposing matrices. A closer look reveals that actually MATRIXx has simply used a similar scheme to produce these blocks, with the inputs and outputs vectorized and labelled as a matrix. Block Script blocks remain unchanged in that they still require single dimensioned inputs and outputs forcing the retention of the vectorizing scheme, although now inputs and outputs can be labelled as a matrix while in vector form.

4.1.2 Algebraic Loops

Another issue uncovered was the program's inability to resolve algebraic loops for complex systems (see Figure 4.1 for an example of a block diagram including an algebraic loop). For very simple feedback models, such as a basic feedback control model, MATRIXx could resolve the loop automatically by adding a one step delay to the output of the final block before being fed back making it available at the input for the start of the next iteration. For more complex models, such as the highly in depth simulation of a helicopter, these delays had to be added manually. The first attempt to manually add this delay was to use z-transform blocks with a transfer function of 1/z, a single step delay (see Figure 4.2). This worked on a few test cases, but led to highly unpredictable behaviour. MATRIXx would at times add additional delays and produce varying results. The solution ultimately employed was to make use of the common memory, using a pair of blocks to write to a variable and read from it on the next iteration (see Figure 4.3).

It was later discovered that a common practice for dealing with algebraic loops within MATRIXx is to make use of Block Script State and Next State variables. They are intended to be used with the State variable representing data within a Block Script Block from the previous time step, while the corresponding Next State variable represents data to be used on the next time step. Although this works well, it is only useful for a loop within a single Block Script Block and cannot deal with a loop containing multiple blocks. Since the use of Read/Write blocks accomplishes the desired result just as well, and can be used for loops containing multiple blocks, their use was retained in all cases, keeping the methodology consistent for loops of all sizes.

4.2 AutoCode

Once the development and testing of the model within the SystemBuild environment was completed, the AutoCode feature was used to generate C code. This code was then compiled and run first as a stand-alone program for testing, and later integrated into the complete simulator architecture for a real-time man-in-the-loop helicopter simulation.

This transition from the SystemBuild environment to a real-time program resulted in a new batch of unforeseen difficulties in the simulator development. In many ways, the generated code differed in execution from the SystemBuild Simulate feature. Although the use of AutoCode and the methods used to ensure proper execution of the program are not within the scope of this thesis, some of the major issues are described below for completeness and to give an idea of the type of difficulties encountered.

4.2.1 Variable Initialization

One of the earliest discovered differences that caused a wide range of errors when attempting to compile and run the C code was a difference in the way variables are initialized. In the SystemBuild environment, all required variables both within Block Script blocks and for Read/Write blocks are initialized to zero if not specifically set to some other value. When AutoCode is run, all variables required for Read/Write blocks are initialized, but local and output variables for Block Script blocks are not automatically zeroed. As a result, these variables can potentially have any value, causing abnormal results for calculations that use the non-zeroed value. Once discovered, this was fairly easy to rectify. To ensure the code executes as expected, all variables within Block Script blocks or for Read/Write blocks must be initialized, either to a suitable value or to zero.

One further problem with variable initialization concerns the choice of an initial value. Originally, all variables that would be used before being recalculated, such as feedback terms, were set to a suitable initial value, while those calculated before being used, such as in a start-up calculation, were simply set to zero. Although this worked well for the most part, there are instances where this strategy will not work. If the variable in question is being used as an integrator initial condition, the generated code will initialize the integrator before doing the start up calculations. If at this point the variable has a zero value, the zero becomes hard coded in as the initial condition rather than the variable. To correct this, variables that are used as integrator initial conditions are now initialized to some non-zero value, typically a small value ε.

4.2.2 Execution

Various other minor differences that led to unexpected behaviour in the real-time simulation were found along the way. This would include such things as various blocks executing in a different order in some situations. Typically this was dealt with through small modifications in the code to ensure proper execution, such as the addition of sequencer bars to ensure correct execution order. In all cases, SystemBuild simulation results were used to verify the results of the real-time simulation.

4.3 Main Rotor

The major difficulty with the main rotor model is stability, particularly in flapping angles, causing unexpected results in a number of situations. This problem was identified early in the development process, while the rotor was still being tested in isolation in the SystemBuild simulate environment, without the added complexity of interactions with the rest of the helicopter model. Significant time has been spent working to correct this and the rotor has been improved as much as possible.

4.3.1 Initial Testing

Once the rotor model was coded into MATRIXx, it was run in isolation, first in a steady state scenario and then with varying speeds, control inputs, and atmospheric conditions. In the steady state runs, the initial rotor model performed well, with all force and moment outputs as expected. When simulated with varying collective, once again results were as expected. However, once a cyclic input was added, the instability of the rotor began to show. Forces and moments still appeared to vary in a reasonable fashion, but the flapping angles were excessively large, an order of magnitude larger than expected. With a moderate cyclic input, flapping angles could get as large as 30°, and with a larger cyclic input flapping angles could get even larger to the point of obviously unrealistic results.

To determine if this was a numerical problem due to the discrete nature of the simulation, or a physical shortcoming in the model, the rotor model was run non-real-time at a very high update rate (1kHz) to compare results. With this update rate, which obviously requires too much computation time to be implemented real-time, the rotor flapping displayed significantly stronger damping and resulted in smaller flapping angles. As such, the 120 Hz update rate case was modified in an attempt to mimic the higher update case.

4.3.2 Improving Stability

A variety of integration methods were tested for the flapping equations, and the best results were obtained from a Fourier method successfully used in the past on rotor disk models, such as in Reference 4.3. The equations are simplified on the assumption that flapping will take place with an angular frequency that is an even multiple of the rotor frequency. By comparing with the 1kHz case, the best agreement was found using a flapping frequency that is double the main rotor angular frequency.

Although this change in integration method greatly improved the rotor response in offline testing, stability was still a problem in piloted tests. The next step to improve this was to add roll and pitch stabilizing to the rotor through adding a model representation of a Bell stabilizing bar. This is a rigid rod with weights at its extremities, rotating with the rotor. All pitch changes to the rotor from the flight control system are passed via this bar. Due to connections at the pitch horns, the rotor plane of rotation is driven to be parallel to that of the bar, which imparts added stability to the rotor through the bar's gyroscopic inertia. See Section 3.9 of Reference 4.1 for a complete description of the stabilizer and its MATRIXx implementation. The results of its addition on the rotor inputs can be seen in Figures 4.5 and 4.6 as described below.

The final step to improving rotor response was to add a low pass filter to the output forces and moments. As can be seen in the previously mentioned figures, the forces and moments display quite large spikes, which occur at the two-per-revolution frequency. Not only does this contribute to rotor instability, but it can also exceed the allowable limits on the motion system. The filter used is a low-pass filter with the transfer function

$$\frac{\omega_{LP}}{s+\omega_{LP}} \tag{4.1}$$

where ω_{LP} is the break frequency. The MATRIXx implementation of this filter can be found in Appendix B, Block 1.5. A number of values for the break frequency were tested in piloted runs to establish a suitable value that would attenuate the spikes as desired, while retaining a realistic feel to the rotor, resulting in a final value of $\omega_{LP} = 10.0$ rad/s. The effects of the filter can be seen below in Figures 4.5 and 4.6 as described below.

4.3.3 Final Results

Following these modifications, the rotor response is now significantly improved over the original version developed. Figure 4.4 shows a sample time history of rotor output with steps in control inputs while in a steady hover condition in its original form, while Figure 4.5 shows the same response of the updated rotor under the same conditions. Figure 4.6 is included to show these plots side by side for easier comparison. In all three figures, the rotor starts up in a simplified steady hover scenario, with helicopter velocity and airspeed set to zero, zero cyclic inputs, collective set to provide a lift force approximately equal to the helicopter weight, and the helicopter held at a constant attitude and position. After 2.5 seconds, the collective is then increased to roughly double the total rotor lift while still holding attitude and position constant. After 5 seconds, a longitudinal cyclic step input is introduced. Finally after 7.5 seconds the collective is then lowered back to its original value until the end of the simulation at 10 seconds. The plotted variables are described in Table 4.1.

In Figure 4.4 showing the original rotor response, the first 2.5 seconds are fairly uneventful. Following a small initial transient in the startup, all forces and moments remain constant. The z-force is approximately equal to 42,000 N, the weight of the helicopter, with a small x-force due to the slight forward pitch of the rotor shaft. Flapping angles are constant at the coning angle of the rotor of 0.024 radians, and the constant input of collective at 0.25 radians and zero cyclic are shown. After 2.5 seconds, the collective is increased to 0.4 radians. Once again, following a short transient as the inflow establishes, forces, moments, and flapping angles remain constant. The rotor z-force has now increased to approximately twice its original value, resulting in a corresponding increase in x-force. Also note the large increase in the rotor moment about the z-axis showing the large increase in torque necessary to create this additional lift.

After 5 seconds, a step cyclic input is added with a magnitude of 0.1 radians, or approximately a 6° blade angle of attack. As one would expect, this produces an oscillation with the same frequency as the rotor rotation and flapping angles that are 180° out of phase. The magnitude of oscillation however is not ideal. Looking at the flapping angles, the stability problem becomes apparent. The magnitude of oscillation grows fairly quickly at first to adjust to the new inputs, and then settles into a slow growth with flapping magnitudes continually growing until the inputs are once again changed. This flapping motion produces a similar result in rotor torque, with the moment about the z-axis oscillating with increasing amplitude as the flapping oscillation increases. Rotor lift is barely changed, showing only a small oscillation compared to its total value. The remaining forces and moments however show a different trend. The application of longitudinal cyclic produces a forward force along with a nose-down pitching moment. Due to cyclic phasing, this also produces a side force and rolling moment, although with a smaller net magnitude than the forward force and pitching moment. These forces have a large oscillation amplitude initially due to the step input and then decay with time, as they should. However, after approximately half a second, the amplitude of oscillation begins to grow once again, steadily growing until inputs are once again changed.

Note that although amplitude of oscillation increases, the overall net forces and moments remain steady. The result of this is that in piloted tests, the main rotor would often appear to be properly functioning until the amplitude of oscillation became quite large. At this point, the simulation would experience a seemingly unexplainable sudden crash, as it could occur at any time, caused by an event some time earlier producing a steadily increasing oscillation that had reached a critical value.

When the simulation had run for 7.5 seconds, the collective was lowered back to its original value while maintaining the cyclic input. For this section, the oscillations reach a steady state, albeit with a notable amplitude. For example, the cyclic input has produced a forward force of approximately 7500 N, and a nose down pitching moment of approximately 9000 N.m, however they oscillate with peak-to-peak amplitudes in the range of 7000 N and 18000 N.m respectively. With a helicopter mass of just over 4300 kg, this leads to a much larger than desired oscillation of the helicopter body during piloted simulations. The flapping angles also reach a steady oscillation with a peak-to-peak amplitude of approximately 0.2 radians or 12°.

As noted earlier, Figure 4.5 shows the same series of events as Figure 4.4 but with the modified rotor model, and Figure 4.6 contains these two plots side by side for easy reference. The first notable difference occurs from the beginning of the simulation. The effects of the stabilizer bar can be seen in the cyclic input, adding a small oscillating input to the pilots controls, causing a small amplitude flapping motion while in the steady hover. After 2.5 seconds when the collective step is applied, the effects of the filter can be seen in the smoothing effect on the step changes in forces and moments. Apart from these two points, the output remains largely the same as in Figure 4.4 for the first 5 seconds.

From when the cyclic step is introduced 5 seconds into the simulation until the end of the simulation at 10 seconds, the stabilizing effects can be clearly seen. Oscillations in all forces and moments are now notably reduced. For example, in the latter portions of the simulation where forward force and pitching moment were oscillating with amplitudes in the range of 7000 N and 18000 N.m respectively, these values have been reduced to 1000 N and 2000 N.m. In addition, after 5 seconds in the previously example, several forces and moments oscillated with large amplitude initially in response to the step, decayed with time, and then increased again. Following modifications, this oscillation is much better controlled, with moderate amplitude in response to the step, which then decreases to a steady oscillation without later increasing in amplitude. This effect can also be seen in the flapping angles. The flapping motion increases in amplitude across the first half second in response to the step input and then settles into a steady oscillation without increasing in amplitude further. The overall magnitude of flapping is also reduced to a more moderate level in this section with a peak-to-peak amplitude approximately half that shown in the previous plot. After 7.5 seconds when the collective is once again lowered, the flapping angles increase to roughly the same levels observed in the previous figure. However, due to the lower inflow velocity with lowered collective, there is a reduction in aerodynamic damping. This scenario should generate a fair amount of flapping, so this is an acceptable result.

Although a few limitations remain, the simulation has been improved as much as possible and is now able to realistically simulate most flight conditions (see Section 5 for flight test results).

4.4 Landing Gear

A number of modifications were made between the initial landing gear model and the version currently in use. Like the main rotor, wherever possible modifications have been made to improve the simulation. Both landings and take-offs have been successfully flown in the current simulation. As described below, Figure 4.7 plots the helicopter velocity, attitude, and specific forces during a typical landing on the current landing gear model, with the plotted variables described in Table 4.2.

The simulation starts with the helicopter in a level TRC controlled hover just above the ground. The collective is then lowered slightly to allow the helicopter to descend to the ground, at which time the collective is fully lowered and fuel to the engine cut to allow the helicopter to settle fully onto the ground. As shown in the plots, the helicopter descends approximately 1 ft before touching down approximately 1.5 seconds after the simulation begins. At this point, the vertical velocity drops to zero and the helicopter rolls level from its slightly banked hover attitude. At the point of touchdown, fairly sharp spikes in specific force are observed, due to the stiff nature of strut-type landing gear. From this point until approximately 7 seconds into the simulation, the helicopter settles as the main rotor is unloaded. This causes the helicopter to pitch up, as the centre of gravity is closer to the aft struts than the front pair. A little more than 5.5 seconds into the simulation, the landing gear pads transition from sliding to sticking, causing specific force spikes in all three axes and following some initial small amplitude oscillation brings the helicopter to rest. One final feature to note is the specific force oscillations between 4 and 5 seconds, and again after 7 seconds. This is a ground resonance to be addressed below, caused by interactions between the main rotor and landing gear, with slightly different characteristics due to being in the sliding mode for the first set and sticking for the second set of oscillations.

4.4.1 Testing Procedure

As with all other subsystems, the landing gear module was tested as much as possible in isolation within the SystemBuild simulate environment before being combined with the rest of the helicopter simulation. Steady state forces and moments were first checked using constant values for a range of helicopter positions, velocities, and attitudes. Once satisfied with the static results, the gear calculations were checked using just the helicopter body equations in combination with the landing gear, without any rotors, rotating subsystems, or aerodynamic forces. Finally, the landing gear was used in conjunction with the entire helicopter simulation.

4.4.2 Test Results and Improvements Implemented

In its initial form, transitions between the sticking and sliding states would cause a large spike in resulting forces and moments. Even with identical values for sticking and sliding coefficients of friction, which should produce a fairly smooth transition, the discrete nature of the simulation would produce a significant spike. The sharp nature of the sliding/sticking transitions has been encountered previously in simulations at UTIAS, with varying degrees of severity. In this case, the sharp peak was large enough to cause the helicopter to either flip or jump back into the air.

The solution employed to smooth these transitions was to filter the in-plane forces. The filter used was a second-order low-pass filter with the transfer function

$$\frac{\omega_{LG}^2}{s^2 + 2\zeta_{LG}\omega_{LG}s + \omega_{LG}^2} \tag{4.2}$$

where ω_{LG} is the break frequency and ζ_{LG} is the damping ratio. Optimum performance was obtained using a break frequency of 95 rad/s and a damping ratio of 0.65. This attenuated the sharp spike resulting in a much smoother transition between sliding and sticking, although the filtering itself was rather complex due to the way in which the in-plane forces were calculated. A large number of variables had to be passed between mirrored filters in the sliding and sticking calculations for the filtering to be performed correctly, as well as requiring numerous logic switches to ensure variables were being calculated for the correct strut-pad, all greatly adding to the complexity of the model.

With filters in place, the landing gear appeared to produce excellent results in both static and dynamic testing. The forces and moments were as expected, and the simple landing gear/helicopter body tests proved to be quite robust, and able to bring the helicopter to rest from a very wide range of horizontal or vertical velocities and orientations. In all cases, transitions from sliding to sticking or vice versa were accomplished smoothly, with no unexpected transients, and would come to a complete stop.

Tests performed using the full helicopter simulation unfortunately were not nearly as smooth. The forces and moments once again contained large spikes, causing the helicopter to jitter. Rather than a smooth transition from sliding to sticking, the landing gear would rapidly transition back and forth between the two states ten or more times creating large force spikes before finally either sticking or sliding. In addition while sitting on the ground, oscillations in position and orientation were observed initially with small amplitude, but with increasing amplitude the longer the simulation was run.

The cause of both problems was traced to interactions with the rotor and other rotating subsystems. Removal of the rotor, tail rotor, and all rotating subsystems allowed the landing gear to once again perform as desired. See Figure 4.8 for a plot of velocity, attitude, and specific force as in Figure 4.7, but with no rotating subsystems acting on the helicopter. As in the previous case, the simulation begins with the helicopter in a level attitude, approximately 1 ft above the ground. Without the presence of a main rotor to support the helicopter however, the helicopter body descends to the ground much more quickly (note the different time scales, with a 10 second simulation in Figure 4.7 and 3 seconds in Figure 4.8), touching down approximately 0.25 seconds into the simulation. For the same reason, the touchdown is much harder than in the previous example, generating a larger specific force along the z-axis and putting a larger demand on the landing gear. From this point on, the helicopter is quickly and smoothly brought to rest. Once again, the helicopter pitches up while settling onto the ground. At the same time, some motion in the x-direction results, which is halted slightly more than 1 second into the simulation. The helicopter body also displays a small rolling oscillation, once again stopped fairly quickly. In this simulation, all motion has stopped even before 2 seconds have passed since the start of the run (about the time the previous example was first touching down), and contains no unexpected spikes or oscillations in the specific force. Without rotating systems, the landing gear model is clearly able to quickly and smoothly bring the helicopter to rest. Through testing, it appears that this interaction with the rotating subsystems is greatly exaggerated by the previously noted rotor instability. With a more stable rotor model, the landing gear should operate trouble free as shown in this example.

Since these rotating systems are obviously required, the landing gear model had to be altered to compensate for this interaction. Initially, it was thought that by altering the spring rates and damping coefficients used in the model, the interaction could be subdued. When this proved unsuccessful, a low-pass filter was inserted, filtering the landing gear forces and moments applied to the helicopter body. The filter used was again a second-order low-pass filter with the transfer function given in Equation 4.2. The MATRIXx implementation of this filter can be found in Appendix D, Block 1.2. The final values used were a break frequency of 95 rad/s and a damping ratio of 0.65.

With this filtering in place, filters on the in-plane force calculations become redundant and unnecessary, so they were removed, giving the results shown in Figure 4.7. For comparison with Figure 4.7, Figure 4.9 shows helicopter velocity, attitude, and specific force with no filters present. This figure is quite similar to Figure 4.7, as it displays the same scenario with identical control inputs. Once again, the helicopter touches down a little less than 2 seconds into the simulation. Through the initial touchdown phase, the two plots look very similar, with nearly identical velocity and attitude plots. The only noticeable difference is the sharper nature of the specific force without filters. As the helicopter settles, without the filters, the ground resonance in the x- and z-directions are reduced, although the y-force remains fairly similar. Once the landing gear transitions from sliding to sticking approximately 5.5 seconds into the simulation, the difference becomes quite obvious. Altitude and pitch plots remain relatively unchanged, while roll response is actually somewhat improved without the filters, as the oscillation in roll immediately following sticking is notably smaller. Velocity and specific force plots however show why the filtering is necessary. Immediately upon sticking, these plots show a very high frequency, high amplitude oscillation. Specific forces oscillate with a peak-to-peak amplitude in the range of 10 m/s^2 (or approximately 1g) in all three axes. Not only is this unrealistic, but can also cause problems with the motion base.

4.4.3 Final Results

Obviously, this solution does not completely eliminate the coupling that occurs between the landing gear and rotating systems, so the final simulation has some imperfections remaining as shown in Figure 4.7. The oscillations are still present, although much smaller in amplitude. If left on the ground for an extended period of time, the oscillation will eventually grow large enough to be felt by the pilot, and can eventually cause a crash in the simulation. This also makes the landing gear model somewhat less robust. Speeds and attitudes must be limited more so than in an actual helicopter to prevent a rollover. However, under normal take-off and landing conditions, the landing gear model now provides a realistic feel that accurately mimics skid-type landing gear.

4.5 References

4.1 Reid, L.D., Haycock, B.C., de Leeuw, J.H., and Graf, W.O., 2000, "Flight Dynamics for Helicopter Emergency Maneuvers Trainer", DCIEM Contract W7711-005923/001/TOR.

4.2 —— 1996, "MATRIXx Product Family; Getting Started (UNIX).

4.3 Howlett, J. J., 1981, "UH-60A Black Hawk Engineering Simulation Program: Volume I-Mathematical Model," NASA CR 166309.

Table 4.1 Variables shown in Figures 4.4 through 4.6

Variable	Units	Description
FR_B(1)	N	Rotor x-force in body frame
FR_B(2)	N	Rotor y-force in body frame
FR_B(3)	N	Rotor z-force in body frame
GR_B(1)	N.m	Rotor moment about body x-axis
GR_B(2)	N.m	Rotor moment about body y-axis
GR_B(3)	N.m	Rotor moment about body z-axis
B1	rad	Flapping angle of blade 1
B2	rad	Flapping angle of blade 2
AA0	rad	Rotor collective angle
BB1	rad	Rotor longitudinal cyclic angle
Time (x-axis)	s	Time since start of simulation

Table 4.2 Variables shown in Figures 4.7 through 4.9

Variable	Units	Description
INSTALT	feet	C of G altitude
VB_E(1)	m/s	Helicopter x-velocity in earth frame
VB_E(2)	m/s	Helicopter y-velocity in earth frame
VB_E(3)	m/s	Helicopter z-velocity in earth frame
THETB	rad	Helicopter pitch angle
PHIB	rad	Helicopter roll angle
SF_B(1)	m/s^2	Specific x-force in body frame
SF_B(2)	m/s^2	Specific y-force in body frame
SF_B(3)	m/s^2	Specific z-force in body frame
Time (x-axis)	s	Time since start of simulation

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Engine Governor Core	0.00833	0.	4	1	Parent

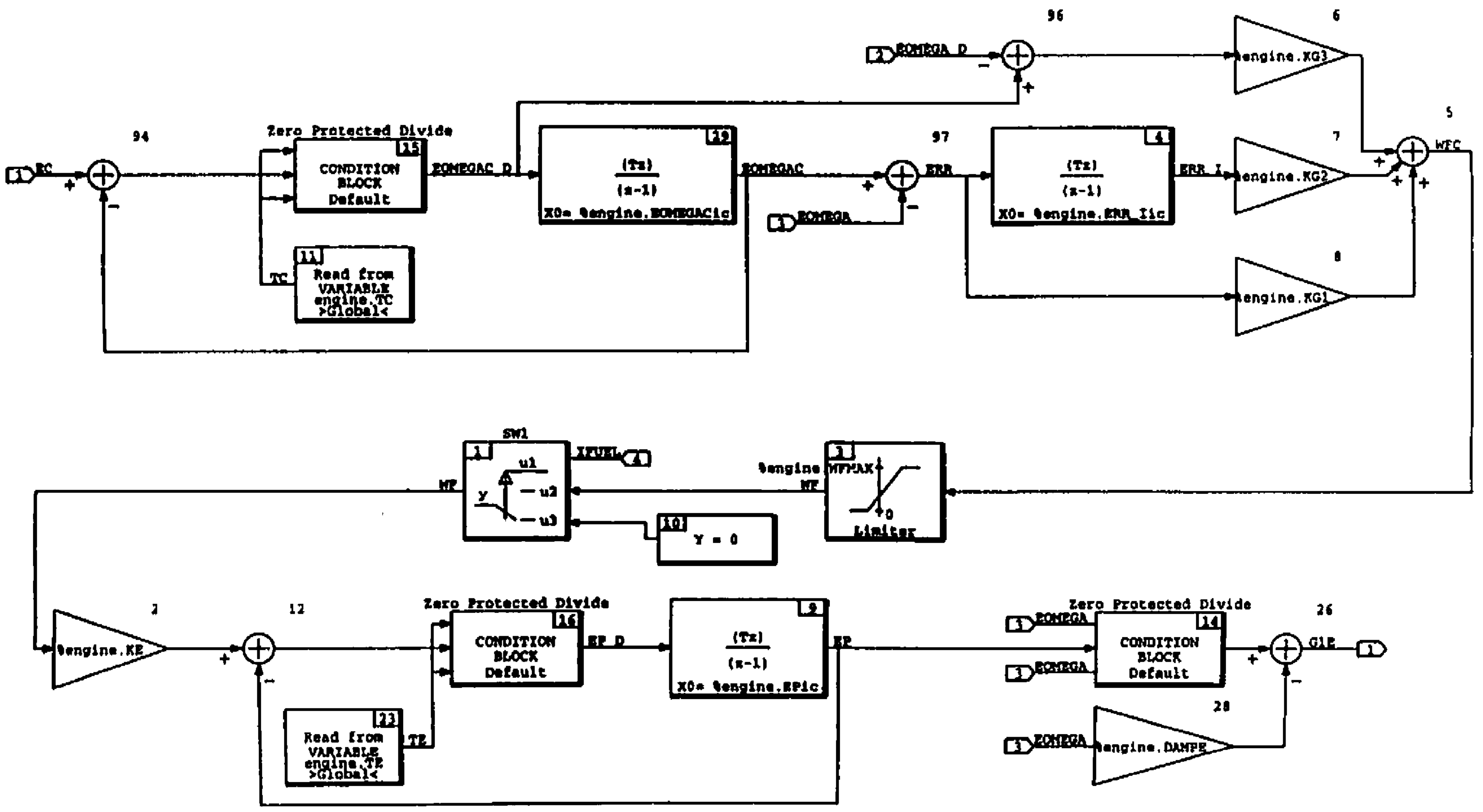


Figure 4.1 Block Diagram Containing Algebraic Loops

11-OCT-100

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Engine Governor Core	0.00833	0.	4	1	Parent

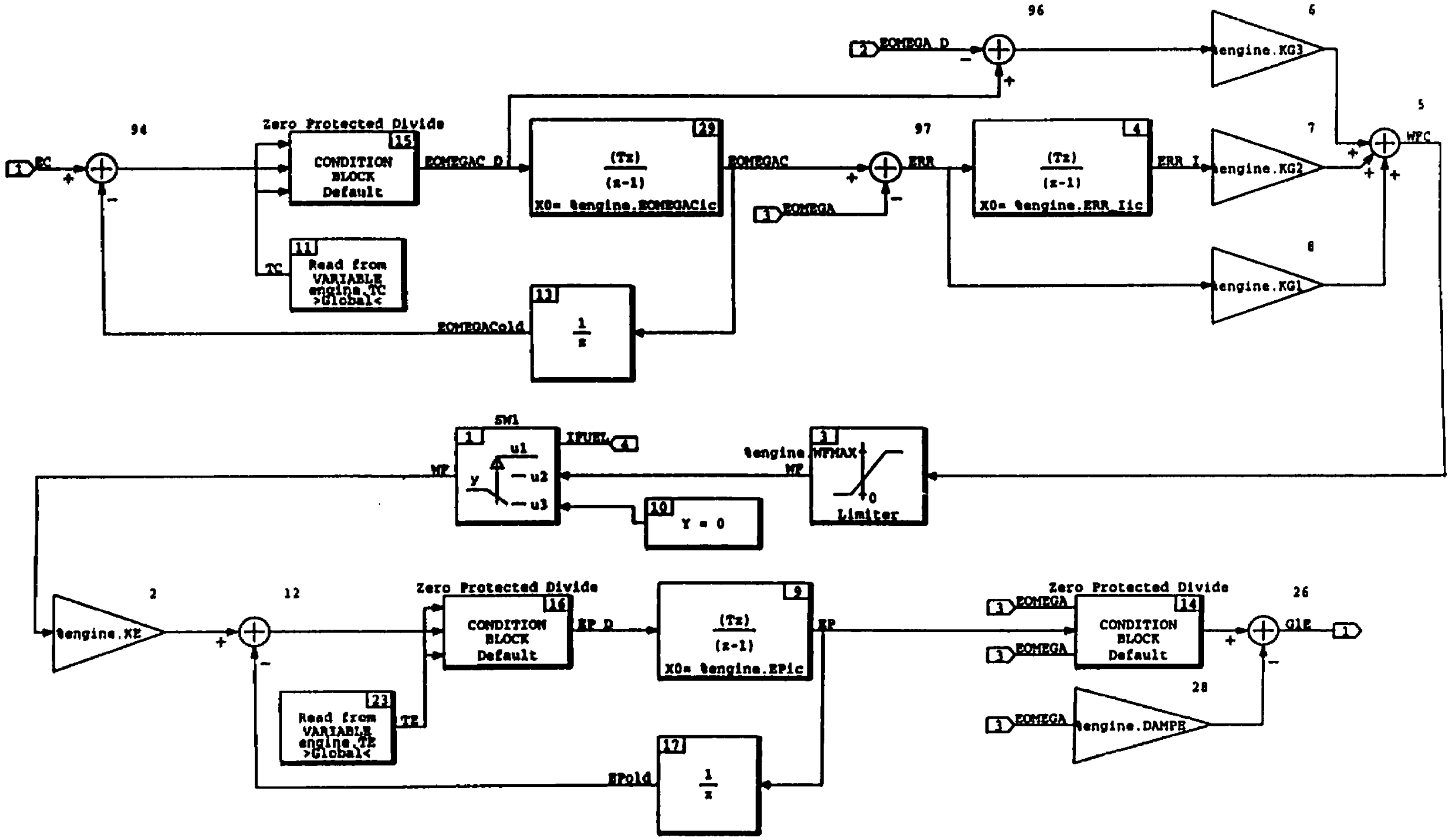


Figure 4.2 Algebraic Loops Broken Using 1/z Blocks

11-OCT-100

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Engine Governor Core	0.00833	0.	4	1	Parent

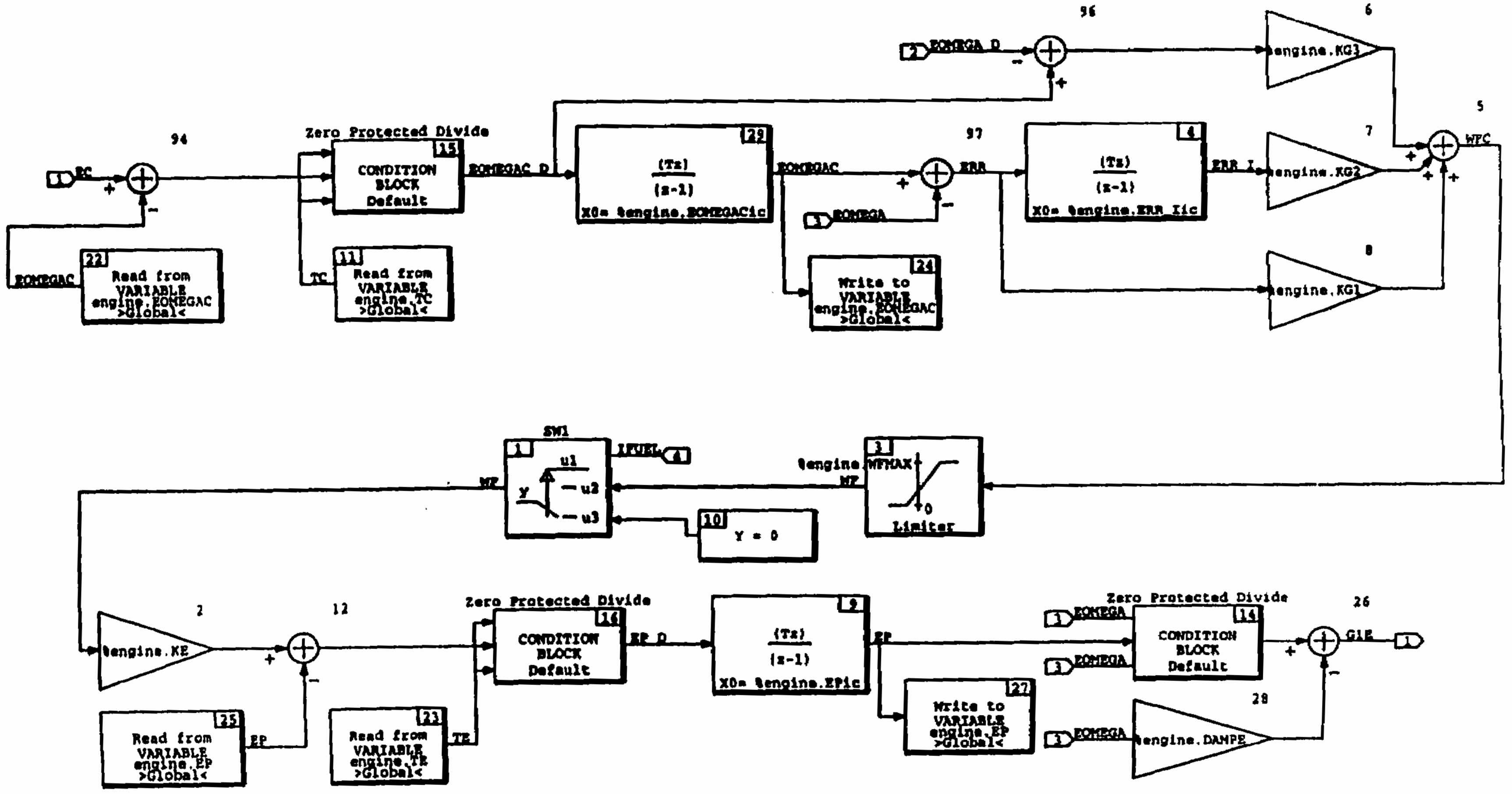


Figure 4.3 Algebraic Loops Broken Using Read/Write Blocks

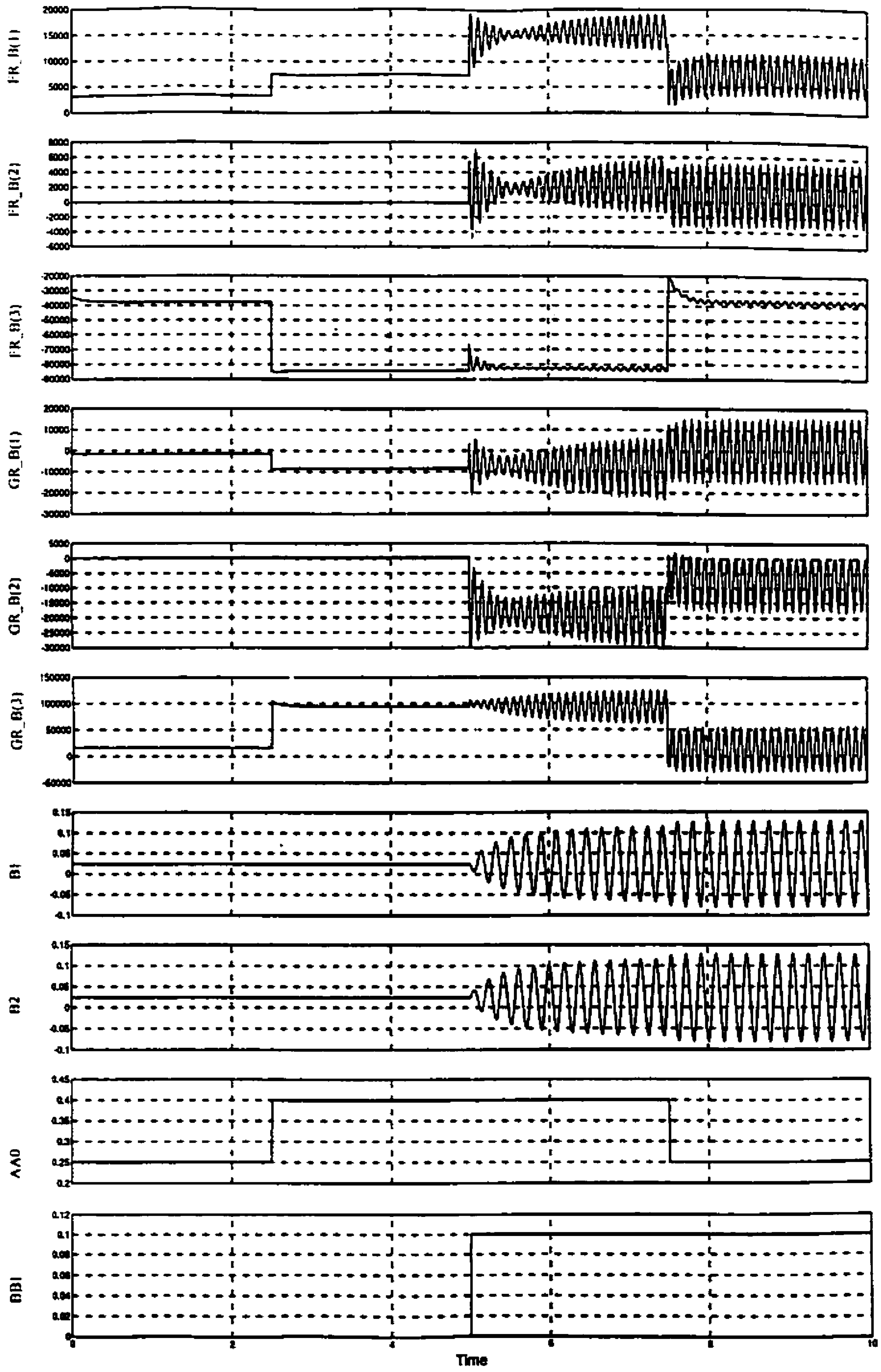


Figure 4.4 Original Rotor Response

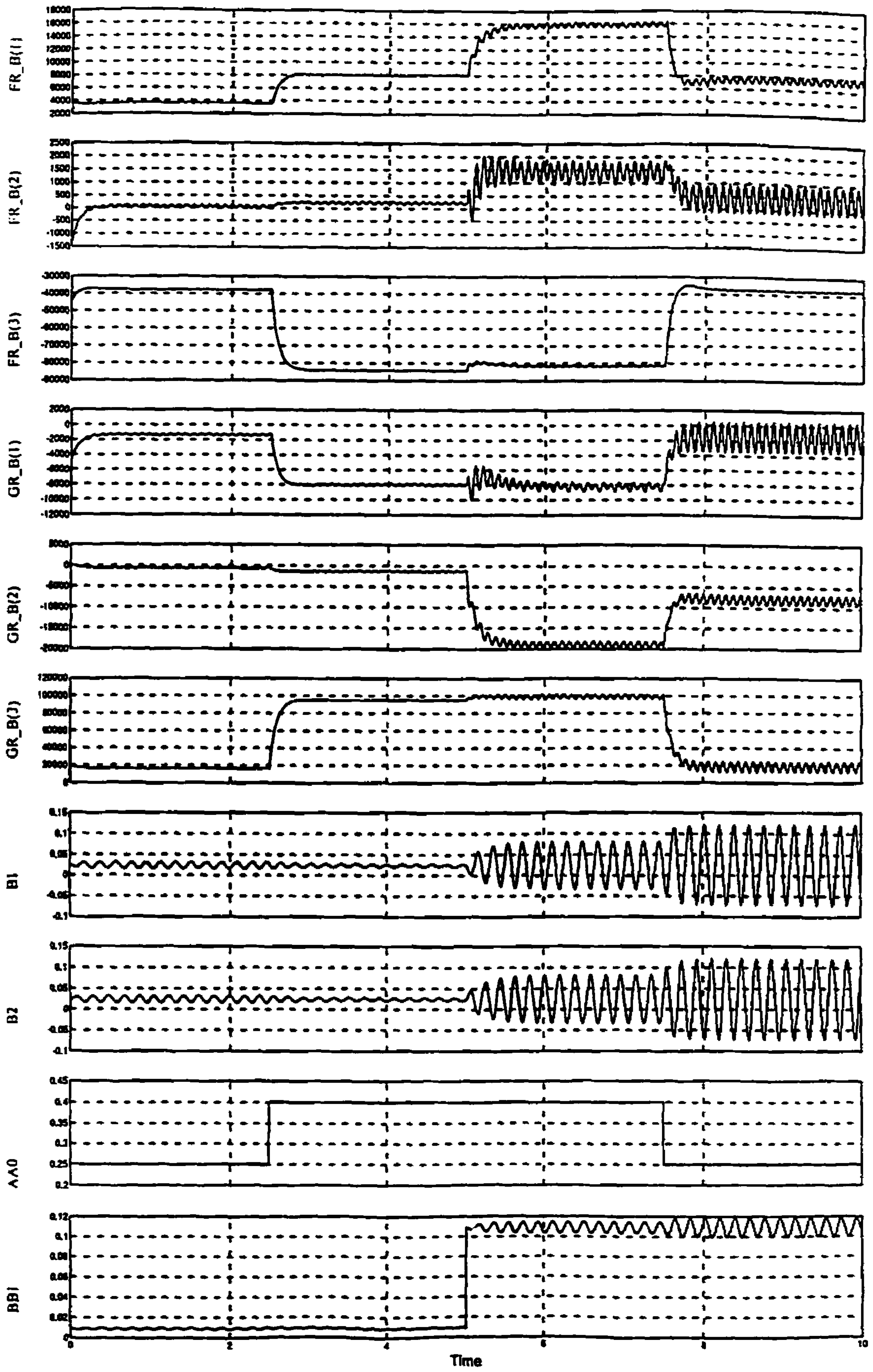


Figure 4.5 Rotor Response with Modifications

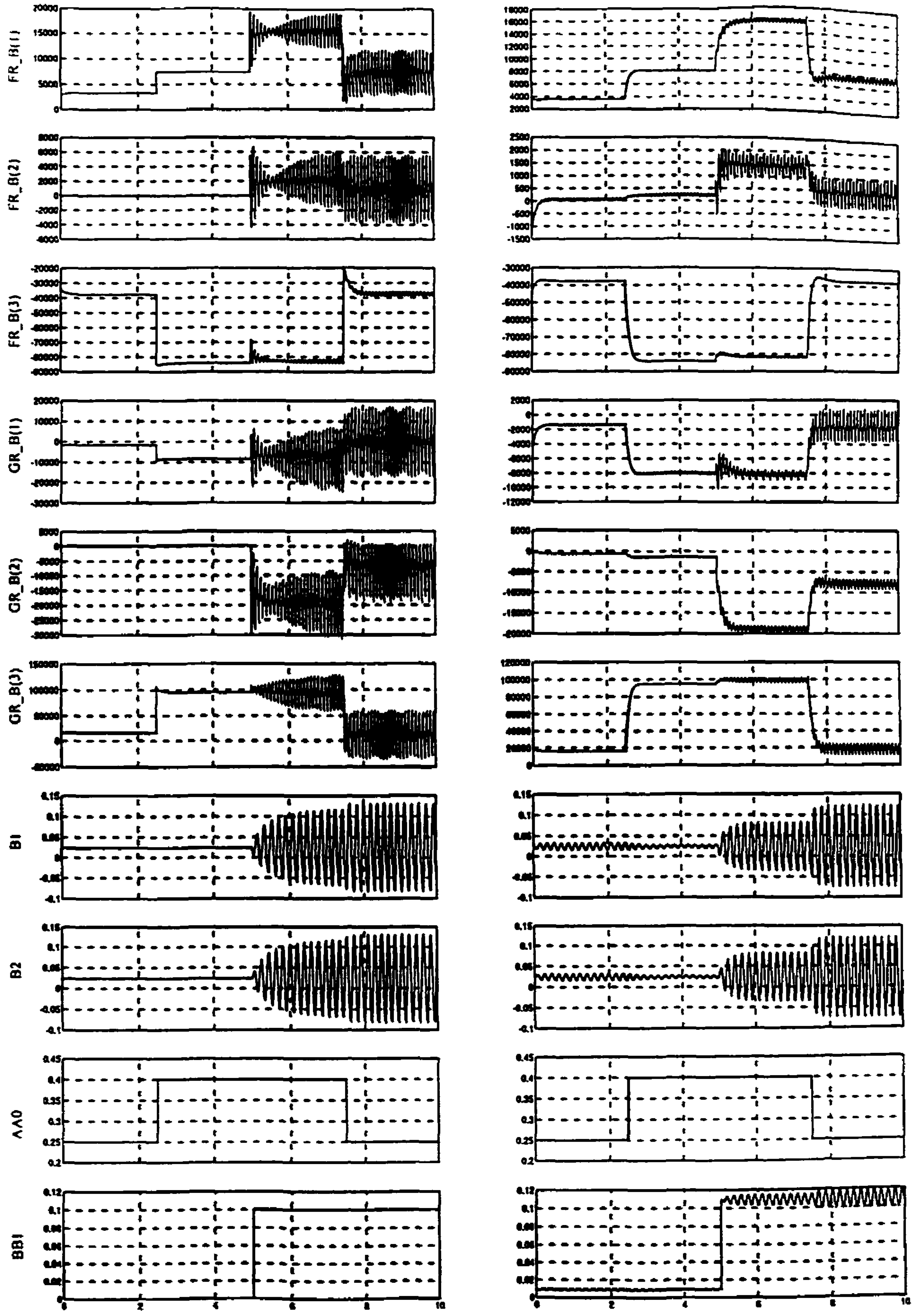


Figure 4.6 Comparison of Rotor Response Before (Left) and After (Right) Modifications

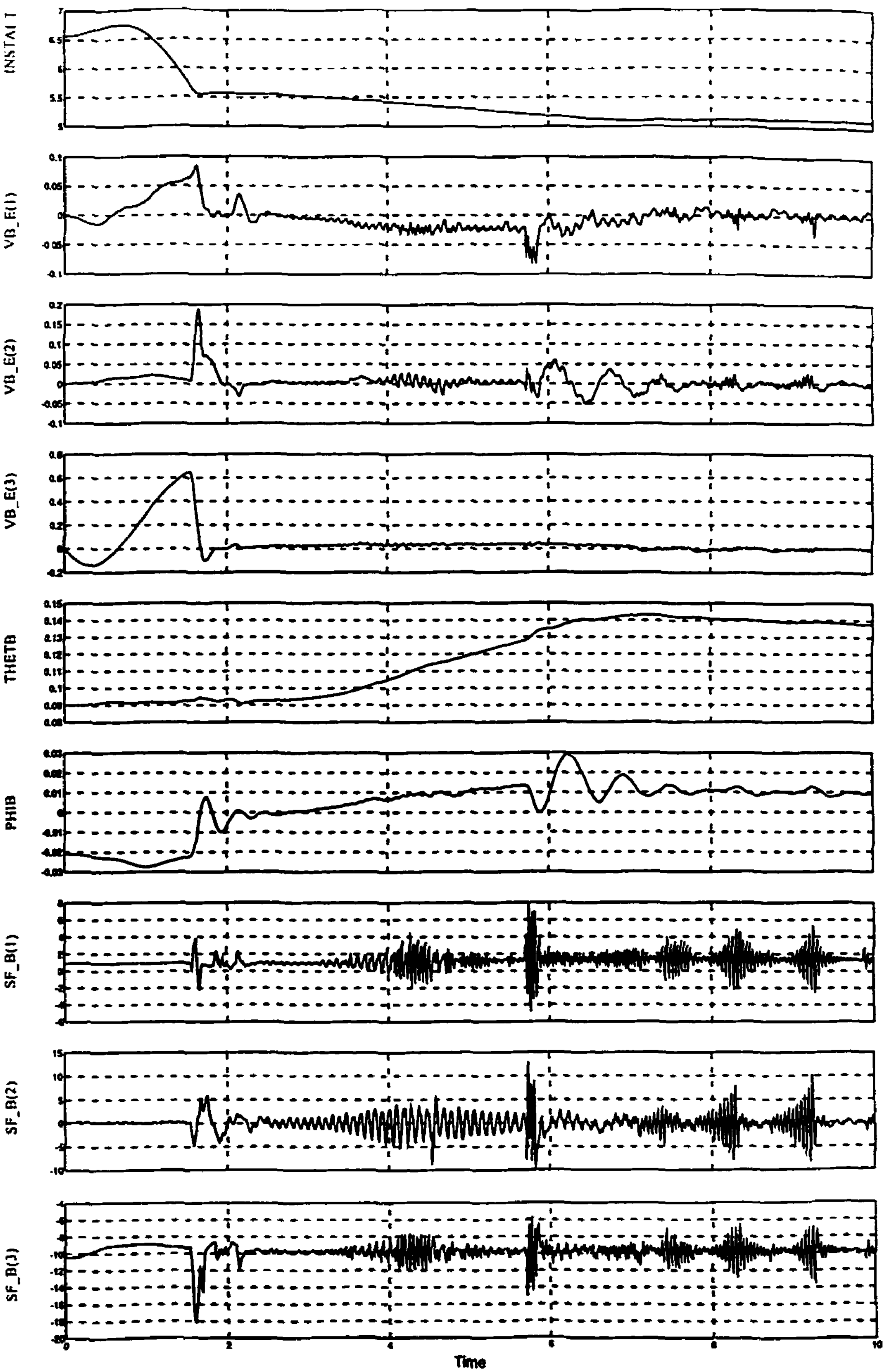


Figure 4.7 Landing from Level Hover

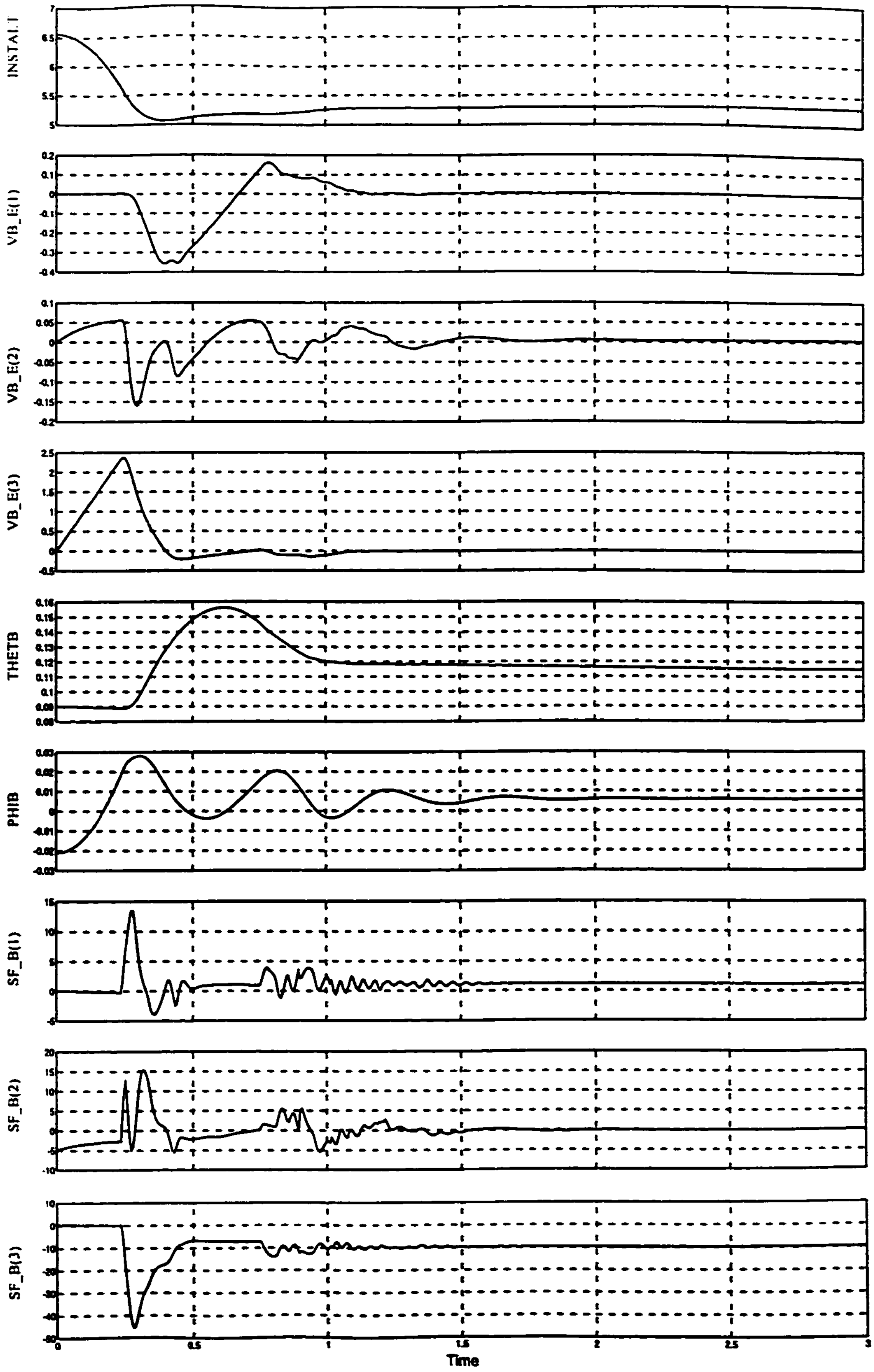


Figure 4.8 Landing from Hover, no Rotating Subsystems

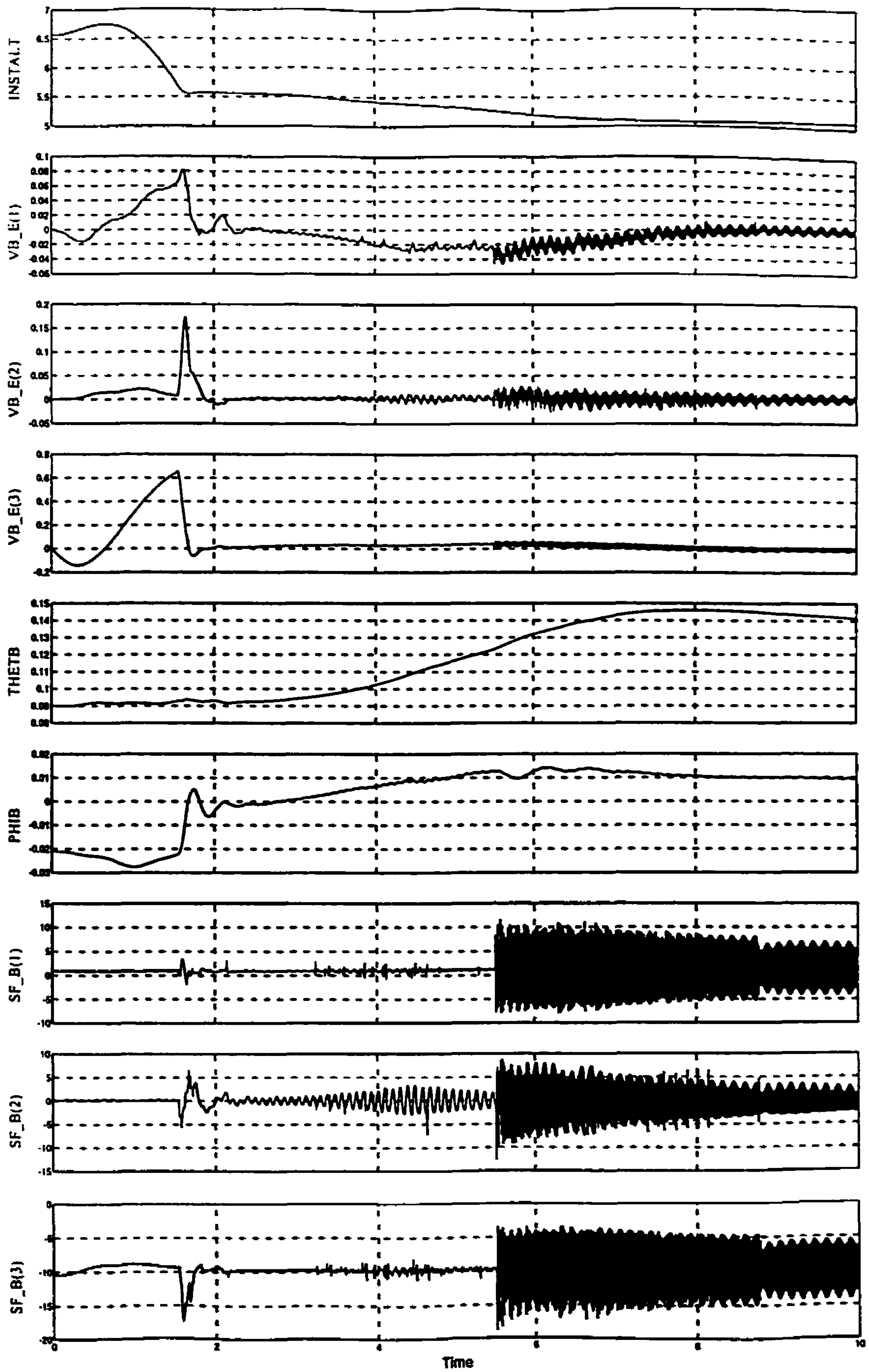


Figure 4.9 Landing from Hover with Filters Removed

5. Flight-Testing

Piloted flight-testing of the man-in-the-loop simulation was carried out in a number of steps. The simulation was first checked for steady trimmed flight with null control inputs, followed by basic checks of various control inputs, and finally fully piloted flights. For all testing, the simulation was configured as a Bell 205 as best as possible with the helicopter data presented in Reference 5.1. Accurate data for the Bell 205 was used wherever possible, and for those parameters that were unavailable, reasonable estimates or approximations were used.

5.1 Preliminary Testing

All initial test work was done in-house without the assistance of qualified helicopter pilots, to ensure a fair level of operation before enlisting the aid of test pilots.

5.1.1 Tests Performed

This initial testing began with flying the helicopter in an open-loop fashion. This included a trimmed cruise condition, and a steady hover under the control of the TRC controller. The next step was to check control responses. From the cruise, controls were checked to produce a response in the desired direction, in the basic sense of an increase in collective resulted in a positive rate of climb for example. From the TRC controlled hovering flight, all controls were checked to control the desired movement, where in this case the example of an increase in collective should produce a steady rate of climb.

In addition, all of the logic switches were tested. This includes the failure modes by switching the fuel, clutch, and tail rotor logic flags, as well as testing the pick-up, carrying, and dropping of a slung load. These tests were performed both under human control and with the assistance of the flight controller.

5.1.2 Test Phase Results

At the end of this phase of testing, it was determined the simulation was ready to be flown by an experienced pilot. All basics had been checked, and test flying had been performed up to the abilities of a non-helicopter pilot, with the simulation performing in an acceptable fashion. Limitations discovered in this phase were an occasional uncontrollable roll due to the rotor problems previously noted, and very limited forward speed under the control of the TRC controller.

5.2 Flight Tests

The helicopter simulator was then tested with the assistance of a qualified commercial helicopter pilot. The aim of this phase of testing was to check flying characteristics in the sense of a generic helicopter. The aid of a commercial pilot was used to check that flying the HEMT simulator was in fact similar to flying a helicopter, although not necessarily perfectly matching the flying characteristics of the Bell 205. Between the initial in-house testing and this phase of testing, the HEMT simulator has logged many hours of operation, allowing a fairly comprehensive examination of the simulator characteristics.

5.2.1 Tests Performed

In this phase, the HEMT simulator was flown in a variety of configurations. Starting from a trimmed cruise, the pilot was given a range of tasks including straight cruise, accelerating and decelerating to new cruise speeds, climbs and descents, and turns to specified headings. Starting from a TRC controlled hover, the pilot performed vertical climbs and descents, forward and aft hover taxiing, sideways flight, pedal turns, as well as landings and take-offs, initially under TRC control, and then without the aid of the controller.

During the testing process, a number of unrealistic qualities were observed, which are described below. Wherever possible, a remedy was applied to improve the simulation, resulting in the model currently in place. With the current rotor model, the simulation has been improved as much as possible, and is able to fly a large portion of the desired flying maneuvers.

5.2.2 Discrepancies Identified

Engine Torque – The torque indicator on the instrument panel appeared non-functional. It was discovered that although the instrument was working, the scale of the instrument did not suit the magnitude of numbers output from the simulation due to a difference in units. Multiplying the simulator output by a suitable factor for the instrument gauge corrected this.

Lateral Cyclic – The lateral cyclic response was found to be much too sensitive. This produced a seemingly twitchy response, made the helicopter difficult to control in the roll axis, and was deemed unrealistically sensitive. To correct this, the lateral cyclic gain was halved and retested. This produced a cyclic response that allowed much smoother control of the aircraft as well as yielding a realistic control feel.

Tail Rotor – When one of the pedals is depressed, the immediate response in an actual helicopter is a yaw in the direction of the pedal. In this simulation, that was not the case. Both in hover and cruise, pushing a pedal would cause a roll in the opposite direction much stronger than the intended yaw, so much so that the aircraft would actually turn in the direction of the roll, opposite the intended direction of yaw. The tail rotor produces a side force and its location relative to the CG determines the resulting moment. Since this location in the simulation is the same as the location in the actual helicopter, this is a rather unexpected result. The only possible physical explanation for the discrepancy is that the rotor in the actual helicopter gyro-stabilizes the aircraft, resisting the rolling motion, while in the simulation this stabilization effect is not as strong due to rotor instabilities. The rotor itself rolls rather than remaining stationary, thus allowing the body to roll. It is believed this lack of damping in roll is due to the rotor instabilities previously mentioned, as well as delays in the feedback from rotor forces and moments due to the necessary filtering. To correct this, the tail rotor location in the simulation was moved down, directly behind the CG. This eliminated the rolling moment, resulting in a far more realistic yawing motion upon use of the pedals, as demonstrated below.

See Figures 5.1 through 5.4 for helicopter response to a step input to the pedal before and after relocation of the tail rotor. The plotted variables are described in Table 5.1.

Figure 5.1 shows the step input while in a TRC controlled hover with the original tail rotor location. The first plot is the step input, consisting of a right pedal input (positive DELTAP). While in TRC mode, this commands a steady rate of yaw to the right. The effects of the TRC controller can be seen in the tail rotor pitch angle, which is quickly reduced by the controller following the initial application of pedal. The side force and moments produced mimic the pitch angle due to the fairly linear nature of the model. From the plots of GT_B(1) and GT_B(3), it becomes clear that in addition to the desired right (positive) yawing moment, a notable negative (left) rolling moment is also produced. The effect of this rolling moment can be seen in the plot of roll angle, PHIB, which shows an initial roll to the left (negative), before being corrected by the TRC controller. Due to a combination of effects, the application of pedal also results in a nose-down pitch (negative THETB). The bottom two plots of heading (PSIB) and rate of turn (OMEGAB_B(3)) show the desired result of the input – a steady yaw to the right with increasing PSIB.

Figure 5.2 shows the same step input after tail rotor relocation. The pedal input and tail rotor pitch angle remains the same as before relocation. The first notably changed plot is that of the rolling moment, GT_B(1). In this case, since the tail rotor has been relocated to eliminate rolling moment, no change in moment is present. The net moment is non-zero however. When the tail rotor was moved, a steady offset to the tail rotor moments was added to maintain the same trim attitude and control inputs as required for the original location. The result of the relocation can be observed in the plot of roll angle. In this case, the tendency to roll left has been almost completely eliminated, with only a small initial deviation to the left due to other effects. Once again, the plots of heading and rate of turn show a steady pedal turn to the right being controlled by the TRC flight controller.

Figure 5.3 shows the step input in an open-loop trimmed cruise with the original tail rotor location. Starting from a trimmed cruise at 50 m/s, the pedal step input is once again applied while keeping all other controls constant, this time with no feedback or flight controller present. Due to this lack of feedback control, a smaller step input was used to achieve a reasonable length of time before the helicopter deviates excessively far from its trim state. In this case, the plot of tail rotor pitch angle mimics the step input of the pedal deflection due to the open loop control law as shown in the first two plots. Once again, both a rolling and pitching moment can be observed upon initial application of the pedal input, although the yawing moment is notably larger, on the order of 2000 N.m compared to the 500 N.m rolling moment. In the remaining plots, the problem becomes quite apparent. The helicopter rolls further and further to the left (negative), while pitching up (positive) following the initial nose-down pitch observed in the TRC case. This results in the heading angle showing only a very small and very brief turn to the right (positive) before swinging quickly left, opposite the intended direction of yaw. This can also be seen in the rate of turn plot, where the desired positive rate of turn quickly swings negative due to the induced roll.

Figure 5.4 shows the open-loop step response with the new tail rotor location. Once again, pedal deflection and tail rotor pitch show the step input. As in the hover case, the rolling moment has been eliminated, and a steady offset included to maintain the original trim state, while the yawing moment remains relatively unchanged. As a result of the relocation, the plot of roll has notably changed. The helicopter body now rolls initially to the right (the desired direction of yaw) before slowly rolling over to the left. The initial nose-down pitch response is once again apparent, although the pitch up of the previous case has now been considerably calmed. Finally, both the heading and rate of turn show an initial turn to the right (positive). Following this, the rate of turn does decay, however this is not surprising due to the open-loop control being used.

Perhaps more importantly, pilot comments following the change were quite favourable. Originally the pedal response was described as quite unrealistic producing unexpected reactions. With the tail rotor relocated, the pedal response now much more accurately reflects that of the actual aircraft.

Main Rotor – As noted earlier, the rotor was a primary source of concern with the HEMT simulator. At this phase, considerable work had already been performed to improve rotor behaviour and as such worked acceptably in most situations, although some unrealistic conditions continued to occur. In the hover, the helicopter can be controlled and maneuvered as desired, however all control movements must be kept small and speeds limited. Large deviations can push the rotor into an unstable oscillation, making the helicopter nearly uncontrollable. The same can occur with speeds in the range of 5 to 25 kts, particularly when attempting to decelerate. Once above this speed, the rotor once again becomes sufficiently damped and can be flown normally. Finally, as speed increases, the rotor can once again become poorly damped making control difficult. This occurs above approximately 140-150 kts, although this is not a large concern as it is above typical operating speeds.

5.2.3 Test Phase Results

In its current form, the following flight sequences can be performed:

- Straight and Level Flight from approximately 25 kts up to approximately 140 kts with no flight controller
- Climbs, Descents, and Turns under the same conditions
- Hover, including slow speed forward, rearward, and sideways flight as well as pedal turns both with no flight controller or under TRC control
- Take-offs and Landings with no flight controller or under TRC control
- Sling Load operations in cruise or hover
- Autorotations

In its current form, the following limitations exist for a successful flight:

- All landings must take place with the helicopter fairly level, with no excessive translational speed or rapid rate of descent. Touching down with high speeds or not level typically causes a rollover.
- The helicopter should not be left sitting on the ground for long periods of time, particularly with positive collective. In this situation, the interaction between the rotor and landing gear causes a resonance, which grows steadily. This causes a jitter that can eventually become large enough to cause a crash.
- Flight in the range of 5 kts to 25 kts should be avoided. In this region, the rotor model becomes quite unstable, frequently causing an uncontrollable roll to one side. The transitional speed range can and has been traversed

successfully both accelerating and decelerating, although is typically only successful if performed as quickly as possible. In particular, the take-off or hover to forward flight can be accomplished if the helicopter is steadily accelerated until a safe speed is reached, while the deceleration to hover is rather challenging, as it takes much longer to pass through this region and into a stable hover.

- Flight above 140 kts should be avoided. As above, the rotor becomes unstable in this range making it much more difficult to keep level.
- Failing the engine to perform autorotations works best if started from moderate cruise speeds, in the range of 60 to 90 kts, and with relatively low power settings before toggling the switch. As above, the rotor has instabilities whereby too sudden a change in engine torque can cause the helicopter to roll over, causing a crash. If the engine is failed in this speed range with the collective already partially lowered, the transition from powered flight to autorotation can be accomplished smoothly.

5.3 References

5.1 Reid, L.D., Haycock, B.C., de Leeuw, J.H., and Graf, W.O., 2000, "Flight Dynamics for Helicopter Emergency Maneuvers Trainer", DCIEM Contract W7711-005923/001/TOR.

Table 5.1 Variables shown in Figures 5.1 through 5.4

Variable	Units	Description
DELTAP	cm	Distance of pedal travel
THETATC	rad	Tail rotor pitch angle
FT_B(2)	N	Force along y-axis due to tail rotor
GT_B(1)	N.m	Rolling moment due to tail rotor
GT_B(3)	N.m	Yawing moment due to tail rotor
PHIB	rad	Helicopter roll angle
THETB	rad	Helicopter pitch angle
PSIB	rad	Helicopter heading
OMEGAB_B(3)	rad/s	Rate of turn
Time (x-axis)	s	Time since start of simulation

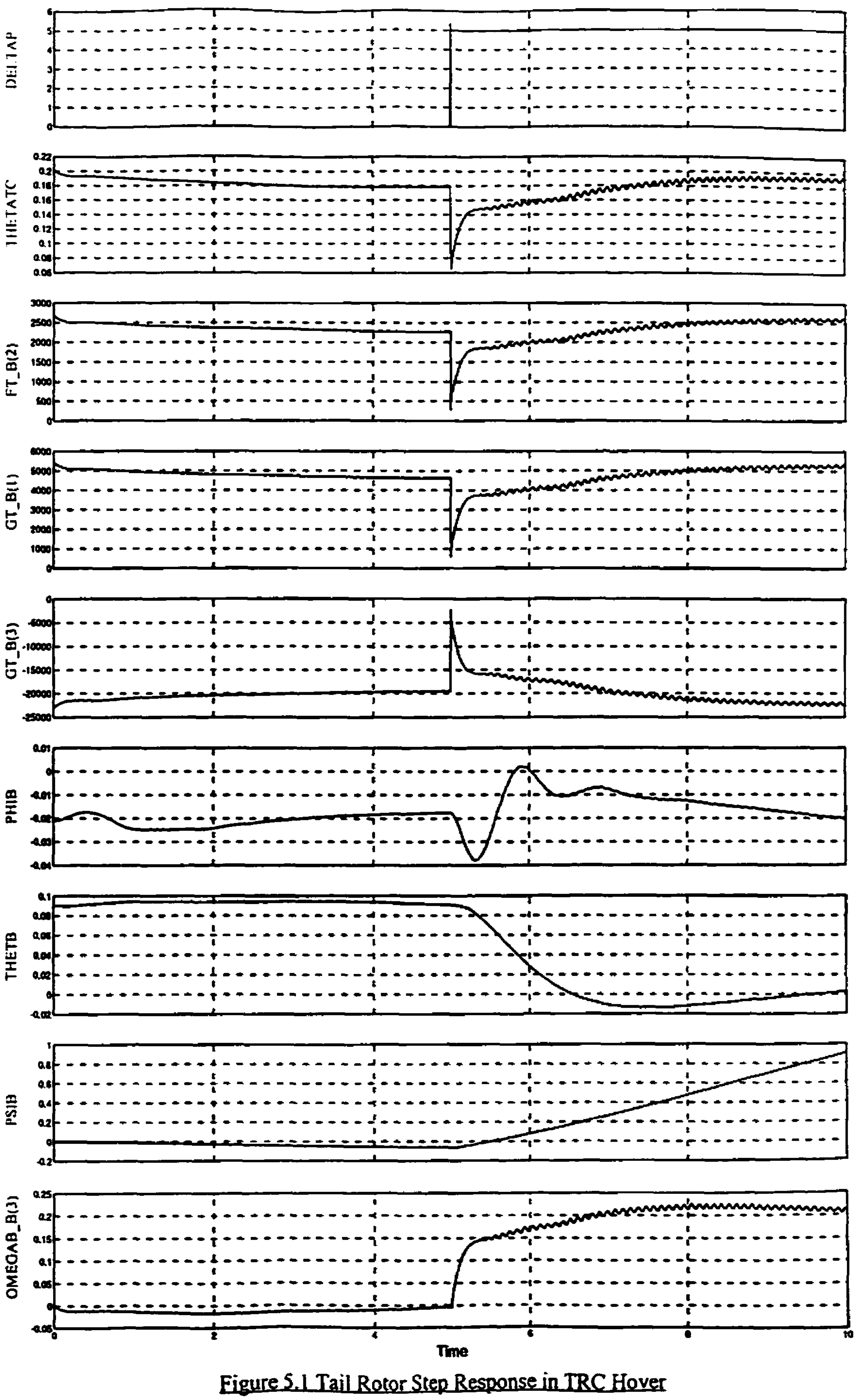


Figure 5.1 Tail Rotor Step Response in TRC Hover

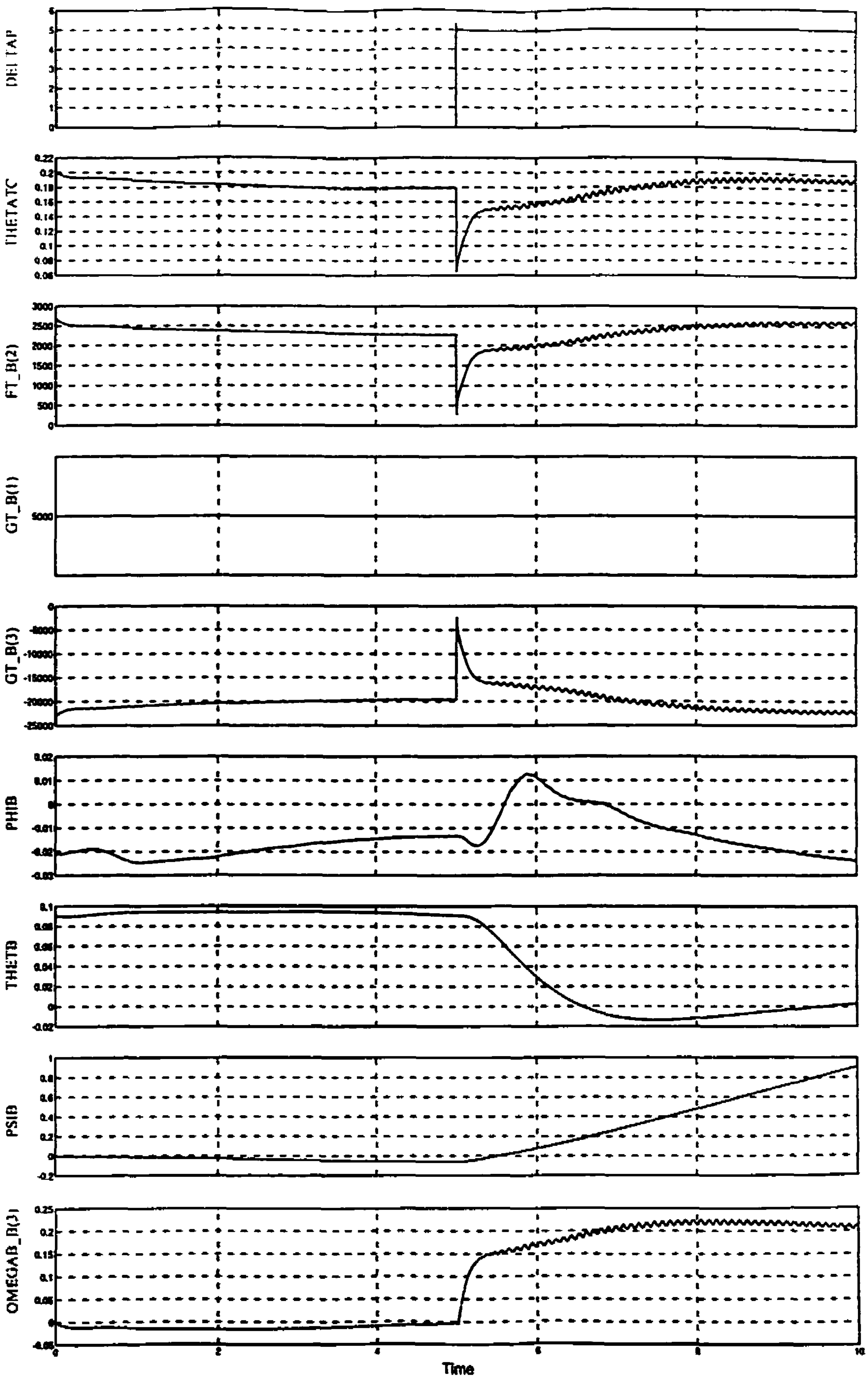


Figure 5.2 Tail Rotor Step Response in Hover after Relocation

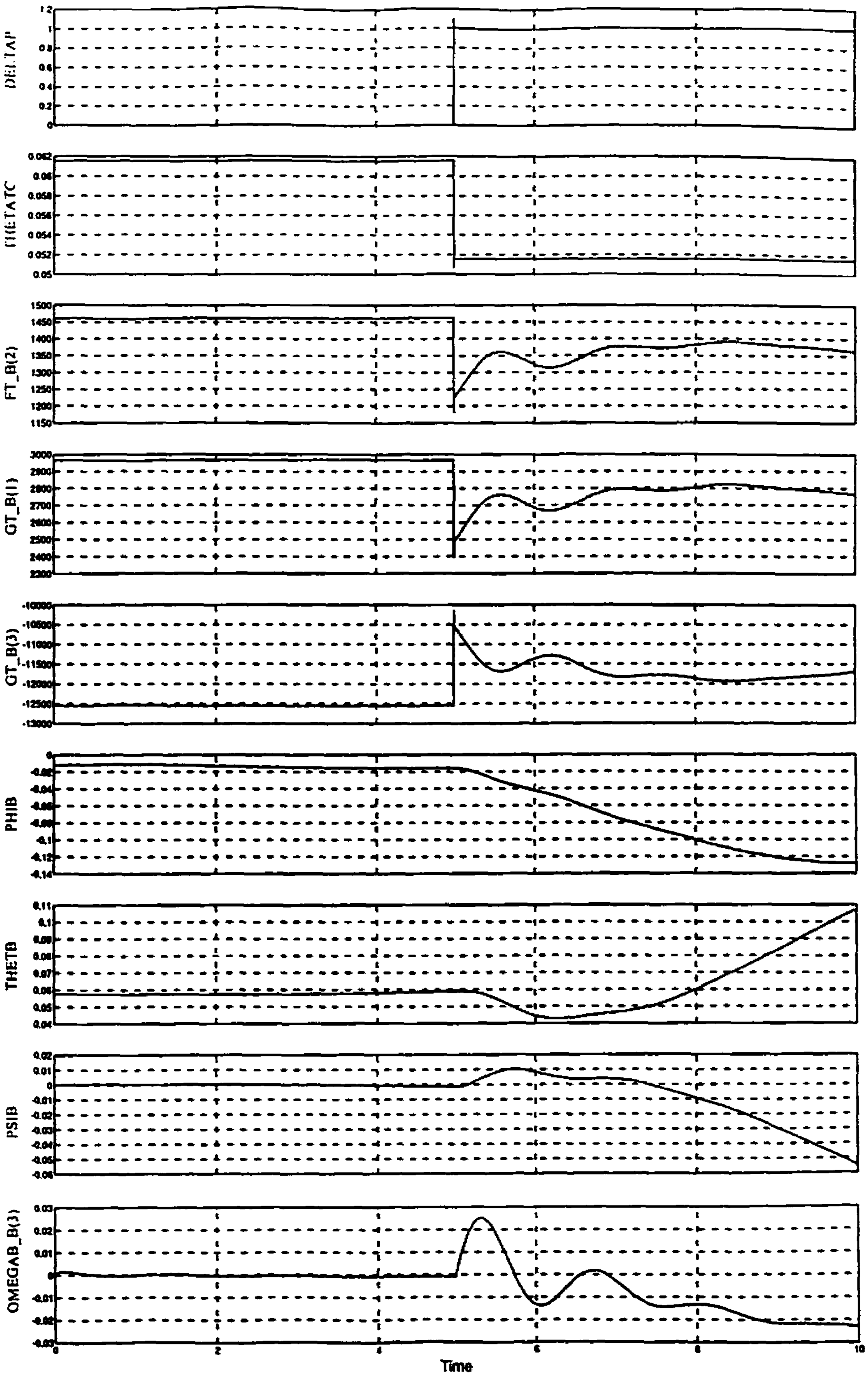


Figure 5.3 Tail Rotor Step Response in Open Loop Cruise

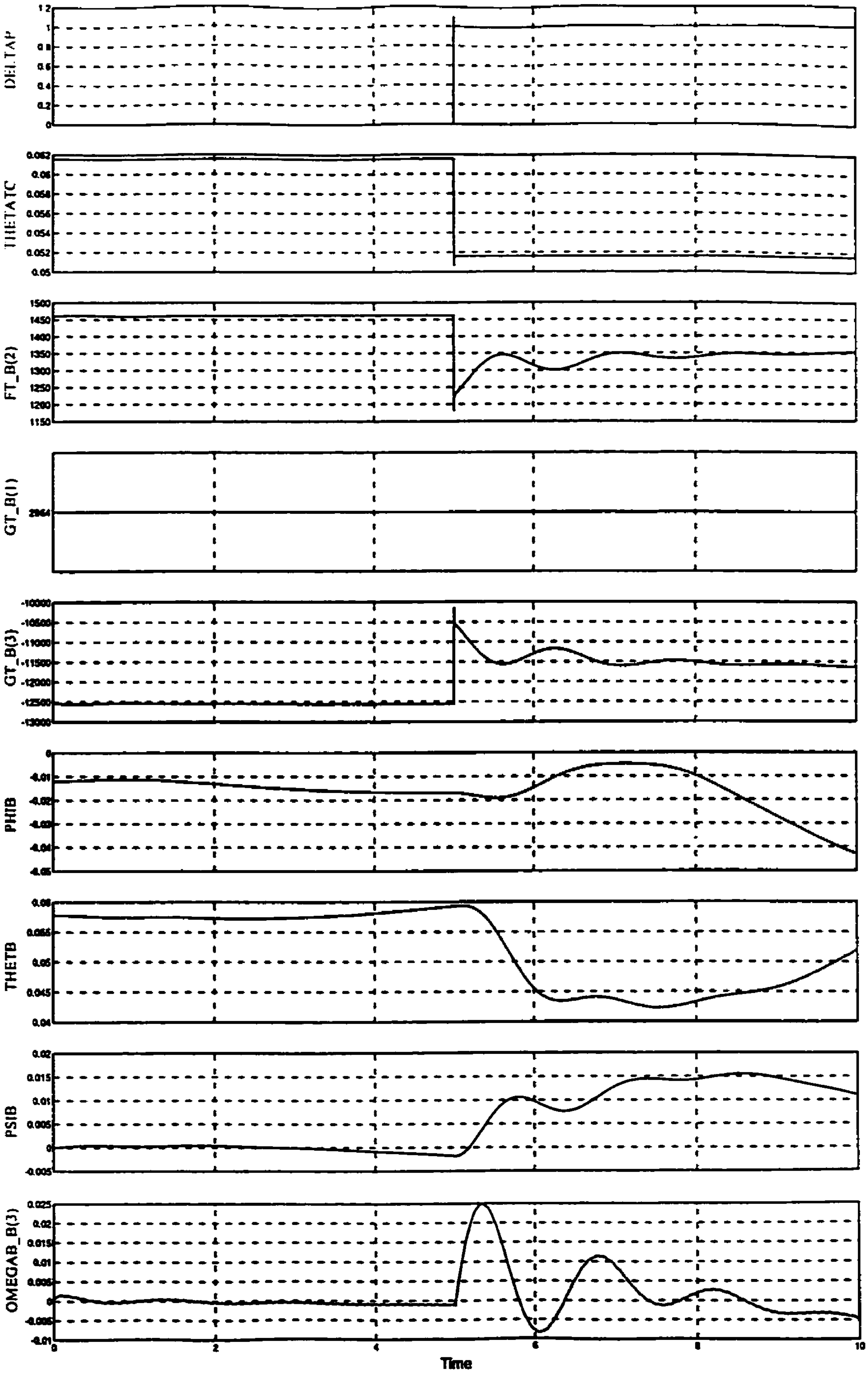


Figure 5.4 Tail Rotor Step Response in Cruise after Relocation

6. Conclusions and Recommendations

In its current form, the HEMT simulator model is fully functional and able to carry out a wide range of flight maneuvers as demonstrated in the flight-testing program. However, it also still has a number of issues to be addressed before use in a training application.

As it stands, the HEMT simulator has been proven to fly as a helicopter, however it has not been proven to fly exactly like any one particular model of helicopter. In order to accomplish this, a more complete and formal testing process will have to take place using highly experienced test pilots who are familiar with the intended helicopter. These pilots could then provide the necessary feedback to fine tune the generic helicopter simulation currently running into a realistic representation of the Bell 205.

Before carrying out this in-depth study of the simulator's flying characteristics, it is highly desirable to make a few modifications to ensure a successful program. As discussed earlier in this thesis, the current rotor model has caused a wide range of problems. This has resulted in instability in various flight conditions, less than ideal landing characteristics, and marginal maneuverability under the control of the TRC flight controller, all requiring fixes placed in the code. As such, it would seem the best option is to explore the possibility of removing the blade element rotor model in favour of the much simpler rotor disk model. Rotor disk models have been used in the past to effectively simulate a teetering rotor without the instabilities and difficulties encountered with the blade element model. The downside to this change would be a drastic reduction in the resolution of rotor behaviour, resulting in a less realistic simulation. However, the unrealistic instability of the blade element model does not present a realistic simulation either, so the rotor disk model should be an improvement.

In order for the simulation to be used effectively for emergency training, a visual database suitable for performing autorotation training is required, although this is beyond the scope of this thesis. In order to perform autorotations in the simulator and have positive transfer to the real aircraft, strong visual clues to give height above ground information must be present. Unfortunately, what elements are important to perception during the autorotation approach is not well documented. In addition, those features typically the most notable during training in actual aircraft are difficult to simulate in an artificial environment. These cues are fairly subtle, such as the effect of ground rush, where suddenly ground speed becomes much more apparent due an increase in the amount of detail visible on the ground, and the use of peripheral vision to judge altitude at low heights. The database currently being used for testing is a previously developed database in the Fredericton area. In this database it is possible to get height cues from alternate sources, such as nearby buildings, allowing for testing in the approach and landing phase although not necessarily giving cues similar to those in a real helicopter.

Appendices

A. Background Material

This section contains background material which applies to the rest of the thesis. This includes notational information, computer variable names, and numerical techniques. The material includes *pseudocode* notation. In the present development, *pseudocode* refers to equations written in a form that employs the computer names of variables. It serves as a bridge between the physical equations of the helicopter and the computer code written in MATRIXx.

A.1 Vector/Matrix Notation

In general a vector is represented by a bold letter such as **V** (for inertial velocity). The components of a vector **V** expressed in a particular reference frame F_A are represented by a column matrix using the notation

$$\underline{V}_A = \begin{bmatrix} V_{xA} \\ V_{yA} \\ V_{zA} \end{bmatrix} = [V_{xA} \quad V_{yA} \quad V_{zA}]^T \tag{A.1.1}$$

The pseudocode representation of the above is V_A(K). In general a matrix is represented by a letter with an underscore such as $\underline{C}$ and in pseudocode by $C(I, J)$ where I represents the row number and J the column number. A particular row of the matrix is represented by replacing I by the number of the row. For example, the row matrix representing the second row of $C(I, J)$ would be indicated by $C(2, J)$. In a similar fashion the third column of $C(I, J)$ is indicated by $C(I, 3)$.

Matrix multiplication is represented by

$$\underline{A} = \underline{B}\,\underline{C} \tag{A.1.2}$$

and in pseudocode notation by

$$A(I, J) = B(I, L)\; C(L, J)$$

$$= \sum_{L} B(I,L)\,C(L,J) \tag{A.1.3}$$

or, if $\underline{C}$ is a column matrix

$$A(I) = B(I,\, L)\; C(L)$$
$$= \sum_{L} B(I,L)\,C(L) \tag{A.1.4}$$

The repeated upper case index, on the right-hand side in the above, implies summation over that index.

A single numerical value for an index can be indicated by replacing the upper case letter by a lower case letter. For example, whereas in pseudocode

$$A(I,\, J) = B(I,\, L)\; C(L,\, J)$$
$$= \sum_{L} B(I,L)\,C(L,J) \tag{A.1.5}$$

produces a full matrix $A(I,\, J)$ the following

$$A(I, j) = B(I,\, L)\; C(L, j)$$
$$= \sum_{L} B(I,L)\,C(L,j) \tag{A.1.6}$$

produces a column matrix $A(I, j)$. Similarly $V(k)$ is a scalar and

$$A(I,\, J) = B(I,\, J)\;\; V(k) \tag{A.1.7}$$

produces a full matrix $A(I,\, J)$. A repeated lower case index does not imply summation over the index, thus

$$A(I, j) = B(I, j)\;\; V(j) \tag{A.1.8}$$

represents a column matrix $B(I, j)$ multiplied by a single scalar $V(j)$.

The components of a vector in a particular reference frame may be indicated alternatively by

$$[\mathbf{V}]_B \equiv \underline{V}_B \tag{A.1.9}$$

and a particular component of a vector $\underline{V}_B$ can be indicated by, for example,

$$V_{yB} \equiv 2^{\text{nd}} \text{ element in } \underline{V}_B \tag{A.1.10}$$

or

$$[\underline{V}_B]_y \equiv V_{yB} \tag{A.1.11}$$

A.1.1 Tilda Operator

The cross-product of two vectors is often required. This is expressed by

$$\mathbf{a} = \mathbf{b} \times \mathbf{c} \tag{A.1.12}$$

In matrix notation (see, for example, Reference A.1) this can be expressed using components in reference frame F_A by

$$\underline{a}_A = \tilde{\underline{b}}_A \, \underline{c}_A \tag{A.1.13}$$

where the *tilda* operator produces the matrix

$$\tilde{\underline{b}}_A = \begin{bmatrix} 0 & -b_{zA} & b_{yA} \\ b_{zA} & 0 & -b_{xA} \\ -b_{yA} & b_{xA} & 0 \end{bmatrix} \tag{A.1.14}$$

An example of the use of a vector cross-product is the moment about a point produced by a force. In Figure A.1 consider two points on a body represented by O and P. The location of P relative to O is given by the vector $\mathbf{r}$. A force $\mathbf{f}$ is applied to the body at point P. The moment $\mathbf{m}$ about O due to this force is given by the cross-product

$$\mathbf{m} = \mathbf{r} \times \mathbf{f} \tag{A.1.15}$$

or expressed in reference frame F_A

$$\underline{m}_A = \underline{\tilde{r}}_A \underline{f}_A \tag{A.1.16}$$

A.2 Reference Frames and Euler Angles

A number of different reference frames are required when modeling a helicopter. Reference frames are designated by the letter F with a subscript, e.g., F_A. The orientation of one frame relative to another is represented by 3 Euler angles (ϕ, θ, ψ) with suitable subscripts to identify them. The Euler angle convention employed follows that described in Reference A.1. When the Euler angles are first introduced in a development, the starting reference frame and final reference frame must be specified. For example: let the Euler angles which carry F_α into F_β be

$$\underline{E} = [\phi \quad \theta \quad \psi]^T \tag{A.2.1}$$

Starting with F_α the following steps are carried out in order to rotate it into coincidence with F_β. Place the origins of F_α and F_β at a common point. First rotate F_α about its z-axis by ψ to create an intermediate frame F_a. Next rotate F_a about its y-axis by θ to create another intermediate frame F_b. Finally rotate F_b about its x-axis by ϕ to create F_β.

Euler angles can be used to determine the rotation matrix $\underline{L}$ that allows vector components to be specified in different reference frames. For example, for frames F_α and F_β

$$\underline{V}_\beta = \underline{L}_{\beta\alpha} \underline{V}_\alpha \tag{A.2.2}$$

$$\underline{V}_\alpha = \underline{L}_{\alpha\beta} \underline{V}_\beta \tag{A.2.3}$$

With the Euler angles specified above, which carry F_α into F_β, it follows from Reference A.1 that

$$\underline{L}_{\beta\alpha} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \quad \text{(A.2.4)}$$

$$\underline{L}_{\alpha\beta} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \quad \text{(A.2.5)}$$

The properties of rotation matrices result in

$$\underline{L}_{AB} = \underline{L}_{BA}^{T} \quad \text{(A.2.6)}$$

$$\underline{L}_{AC} = \underline{L}_{AB}\,\underline{L}_{BC} \quad \text{(A.2.7)}$$

In pseudocode the rotation matrix $\underline{L}_{AB}$ is represented by L_A_B(M,N).

A.3 Variable Names

The names of scalars, vectors and matrices reflect their underlying form and the operations carried out on them. The following pseudocode conventions have been followed in this report.

NAME	- a scalar
NAME_D	- the time derivative of NAME
NAME_DD	- the second time derivative of NAME
NAME_A(K)	- a vector expressed in F_A
NAME_A_D(K)	- the time derivative of NAME_A(K)
NAME_A_DD(K)	- the second time derivative of NAME_A(K)

NAME_A_T(L,M) - the resulting matrix after applying the tilda operator to NAME_A(K) (see Section (A.1.1)

NAME_A_DT(L,M)	- the first time derivative of NAME_A_T(L,M)
NAME(L,M)	- a matrix
NAME_D(L,M)	- the first time derivative of NAME(L,M)
NAME_V(P)	- the vectorized version of NAME(L,M), see Section 2

A.3.1 Euler Angles

The set of Euler angles for a specific rotation B is given by

$$\underline{E}_B = [\phi \quad \theta \quad \psi]_B^T$$

$$= [\phi_B \quad \theta_B \quad \psi_B]^T \tag{A.3.1}$$

In pseudocode notation this is represented by

$$\text{E\_B(K)} = [\text{PHIB} \quad \text{THETB} \quad \text{PSIB}]^T \tag{A.3.2}$$

For other rotations the B is replaced by suitable letters. The sines and cosines of angles are represented by the notation

SIPHIB	= SIN(PHIB)	(A.3.3)
COPHIB	= COS(PHIB)	(A.3.4)
SITHETB	= SIN(THETB)	(A.3.5)
COTHETB	= COS(THETB)	(A.3.6)

$$\text{SIPSIB} = \text{SIN(PSIB)} \tag{A.3.7}$$

$$\text{COPSIB} = \text{COS(PSIB)} \tag{A.3.8}$$

The same notation applies to the sine and cosine of all angles.

A.3.2 Velocity and Acceleration

In this simulation there are three linear velocities: inertial velocity, airspeed, and wind speed. Since the Earth-fixed frame F_E is assumed to be an inertial reference frame, the inertial velocity of a point is also its ground speed. These velocities refer to conditions at a specific poinit on the helicopter and vary from location to location. In general at a specific point, with all velocities referring to that point, it follows that

$$\mathbf{V} = \mathbf{U} + \mathbf{W} \tag{A.3.9}$$

where **V** is the inertial velocity of the point

U is the airspeed of the point

W is the wind speed at the point

W is composed of atmospheric effects and interference effects from other parts of the helicopter such that

$$\mathbf{W} = \mathbf{WA} + \mathbf{WI} \tag{A.3.10}$$

where **WA** is that part of **W** due to atmospheric effects

WI is that part of **W** due to interference effects from the helicopter

The letter *V* is used to represent the inertial velocity of a point. The *V* is followed by letters to indicate the point to which it applies. For example **VB** is the inertial velocity of the helicopter body CG. The letter *U* is used to represent the airspeed of a point. The *U* is followed by letters to represent the point to which it applies. For example, **UB** is the airspeed of the helicopter body CG. The letter *W* is used to represent the wind speed at a point, and as above, **WB** is the wind speed at the helicopter body CG. In a similar fashion to (A.3.10)

$$\mathbf{WB} = \mathbf{WBA} + \mathbf{WBI} \tag{A.3.11}$$

In the case of points on the main and tail rotors, the inflow created by the rotors is included separately. For points on the main rotor

$$\mathbf{W} = \mathbf{WA} + \mathbf{V}_{\mathrm{IN}} \tag{A.3.12}$$

where $\mathbf{V}_{\mathrm{IN}}$ is the main rotor inflow. For points on the tail rotor

$$\mathbf{W} = \mathbf{WA} + \mathbf{WI} + \mathbf{V}_{\mathrm{INT}} \tag{A.3.13}$$

where $\mathbf{V}_{\mathrm{INT}}$ is the tail rotor inflow.

The inertial angular velocity of a reference frame F_A is represented by $\omega\mathbf{A}$. The inertial linear acceleration of a point is expressed as **a** and **aB** would represent the value for the helicopter body CG.

A.3.3 Indices

In order to assist in clarifying the structure of the physical equations and pseudocode, an attempt has been made to relate the letters used for indices to specific helicopter and vector/matrix structures. The index K has been used in general for the elements of a vector; for example, in pseudocode, V_A(K) represents the vector **V** in F_A components. The indices L, M, N, O, P have been used in general for the elements of a matrix; for example, in pseudocode A(L,M). In the case of the main rotor, I and i have been used to indicate the blade number. J and j have been used to represent the particular blade element. Thus in pseudocode a matrix such as $A(I, J)$ would contain data in which each row would pertain to a single blade and each entry in the row would pertain to a particular element of that blade. In the case of the landing gear, I and i have been used to indicate a particular strut/pad combination.

In cases where the name of a reference frame contains the index i, for example the blade-fixed frame $F_{B\ell i}$, if the value of i is indicated elsewhere in the name of a variable, the i may be dropped from the letters representing the frame; for example, OMEGABL_BL(K,i) pertains to components in $F_{B\ell i}$. Also, if a variable applies unchanged to all frames $F_{B\ell i}$, then it may have a name which incorporates the letters $B\ell$ without including the i, for example $\underline{I}_{B\ell}$.

A.4 Numerical Techniques

A.4.1 Algebraic Equation Solvers

When the solution to a nonlinear algebraic equation is required, either the Relaxation Method or the Newton-Raphson method is employed.

The Relaxation Method is employed in the Main Rotor module and is described below in Appendix A.4.1.1. It is an iterative method that can either be run for a fixed number of steps (the current approach in the simulation) or run until a tolerance condition on the solution is met.

The Newton-Raphson method is employed in the Landing Gear module and is described below in Appendix A.4.1.2. It is an iterative method that continually tries to reach the solution in a single step. It has good convergence properties and tends to require very few iterations in the present application. A tolerance parameter is used which stops the process when changes in the trial solutions drop below its specified threshold.

A.4.1.1 Relaxation Method

The relaxation method is used to numerically solve nonlinear algebraic equations. It is an iterative process employed in Reference A.2 based on the following.

For a general algebraic equation of the form:

$$x = f(g(x)) \tag{A.4.1}$$

where $f(y)$ and $g(y)$ are general algebraic functions of one variable, create the following sequence for the n-th step of the iteration process:

$$DUM_n = g(X2est_{n-1}) \tag{A.4.2}$$

$$X1est_n = f(DUM_n) \tag{A.4.3}$$

$$X2est_n = 0.1X1est_n + 0.9X2est_{n-1} \tag{A.4.4}$$

The process is begun with a supplied starting value for $X2est_{n-1}$ (note that x_{n-1} is the value from the n-1 iteration). Equations (A.4.2) to (A.4.4) are executed iteratively either a fixed number of times or until

$$|X2est_{n-1} - X2est_n| < \varepsilon \tag{A.4.5}$$

where ε is a specified tolerance parameter. The solution is

$$X = X2est_n \tag{A.4.6}$$

at the end of the iterative process.

A.4.1.2 Newton-Raphson Method

The Newton-Raphson Method (Reference A.3) is a numerical method for solving nonlinear algebraic equations. It is a single step gradient method. Consider a nonlinear equation in the variable x. Arrange it into the form

$$f(x) = 0 \tag{A.4.7}$$

Now consider the relationship

$$y = f(x) \tag{A.4.8}$$

as plotted in Figure A.2. The task is to find x^*, the value of x that results in $y = 0$ and hence satisfies (A.4.7). Assume that after n steps in an iterative solution method we have an estimate x_n of the desired x-value, x^*. Construct a tangent to (A.4.8) at $x = x_n$ and extend it to reach the x-axis at $x = x_{n+1}$. x_{n+1} is taken to be the next estimate of x^*. This process is repeated until

$$|x_{n+1} - x_n| \leq \varepsilon \tag{A.4.9}$$

where ε is a small tolerance parameter.

In order to carry out the above process we need an expression for the slope of the curve in (A.4.8). This is found from (see Figure A.2)

$$\begin{aligned} \tan\theta &= dy/dx \\ &= f'(x) \end{aligned} \tag{A.4.10}$$

For the case with $x = x_n$ it is seen that

$$\tan\theta = f(x_n)/(x_n - x_{n-1}) \tag{A.4.11}$$

Setting $x = x_n$ in (A.4.10) and combining it with (A.4.11) obtain

$$x_{n+1} = x_n - f'(x_n)/f(x_n) \tag{A.4.12}$$

(A.4.12) is employed iteratively until (A.4.9) is satisfied, to obtain the desired estimate of x^*.

A.4.2 Euler Integration Scheme

A numerical integration scheme is employed to solve the nonlinear differential equations representing the helicopter dynamics. An Euler integration method is used as described below. It was found to be reasonably stable and sufficiently accurate for the present application. The time step is Δt (or ΔT) seconds.

The Euler method is a numerical integration scheme employed to evaluate the integral

$$x(t) = \int_o^t y(t)dt \tag{A.4.13}$$

where

$$y(t) = \dot{x}(t) \tag{A.4.14}$$

At the end of the n-th step in its implementation the following holds

$$x_n = x_{n-1} + \Delta t\, y_{n-1} \tag{A.4.15}$$

where

$$x_n = x(n\Delta t) \tag{A.4.16}$$

and Δt is the time step employed. In the present simulation, when feedback loops are employed, y_{n-1} may contain contributions involving x_{n-1} and other system variables. The most recently computed values of the other system variables are used.

The above is achieved when using MATRIXx by employing the integration method labeled *Backward Euler* which is represented in the block diagrams by the z-transform:

$$\frac{Tz}{z-1} \tag{A.4.17}$$

The feedback of x_{n-1} when required in the input to the integrator is achieved by using Write to and Read from Blocks as described in Section 2.1.

A.5 References

A.1 Etkin, B., and Reid, L. D., 1996, "Dynamics of Flight, Stability and Control," John Wiley and Sons, New York.

A.2 Howlett, J. J., 1981, "UH-60A Black Hawk Engineering Simulation Program: Volume I – Mathematical Model," NASA CR 166309.

A.3 Dieudonne, J. E., Parrish, R. B., and Bardusch, R. E., 1972, "An Actuator Extension Transformation for a Motion Simulator and an Inverse Transformation Applying Newton-Raphson's Method," NASA TN D-7067.

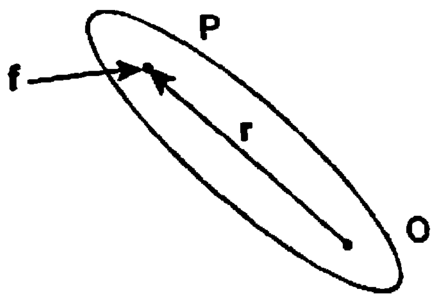


Figure A.1 Force and Moment

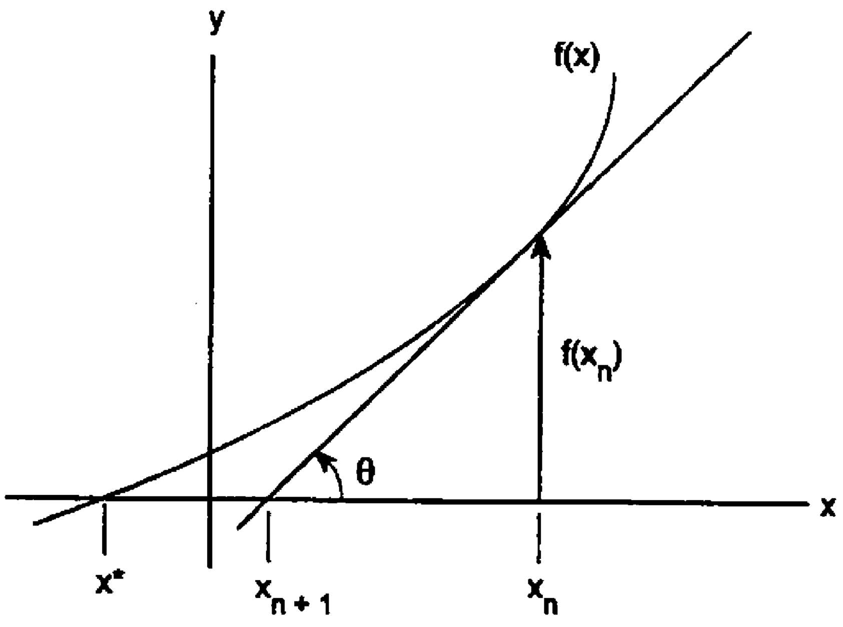


Figure A.2 Newton-Raphson Method

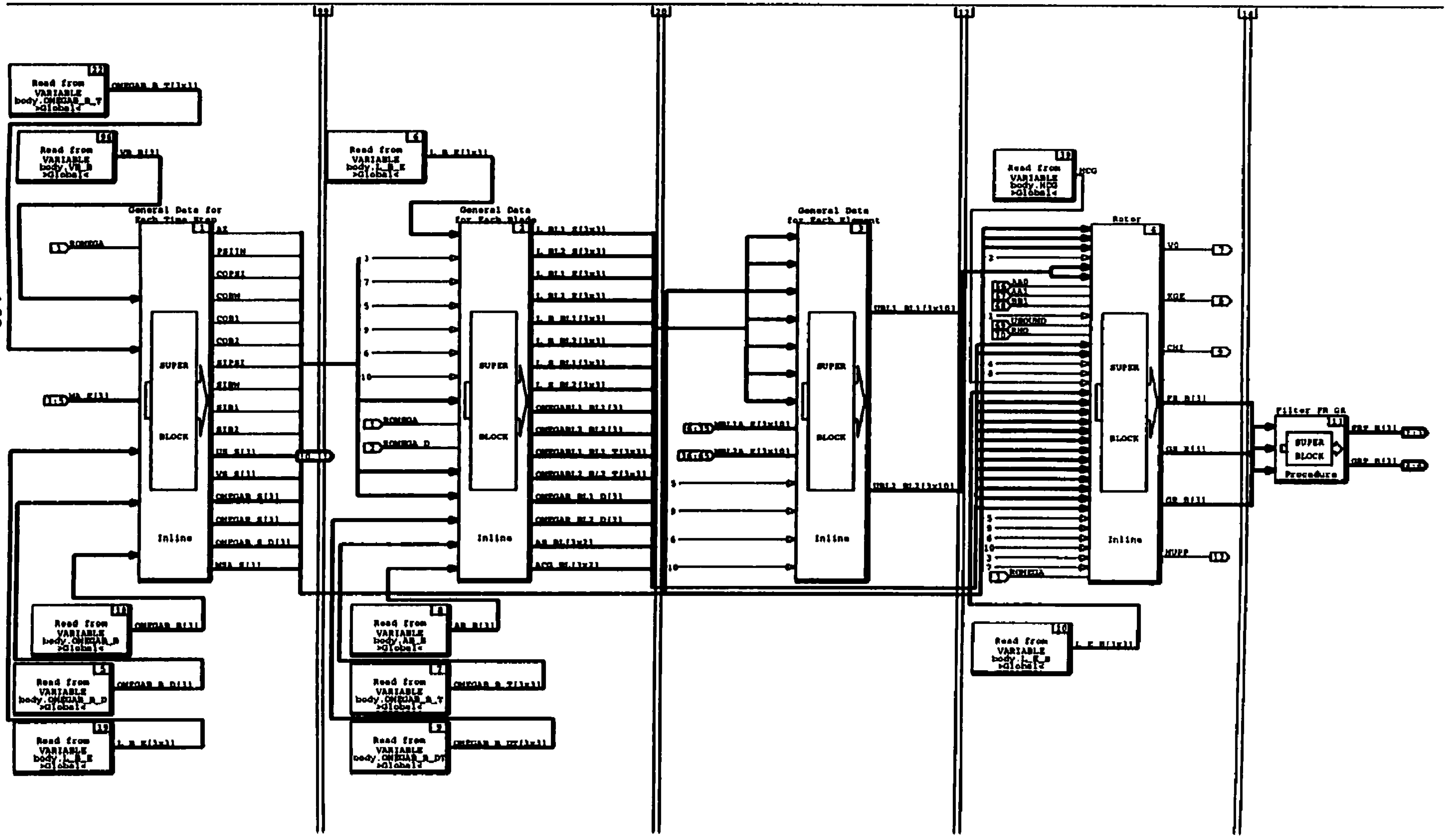
B. Main Rotor Block Diagrams

Table of Contents

1.

24-JUL-101

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Main Rotor	0.00556	0.	70	12	Parent

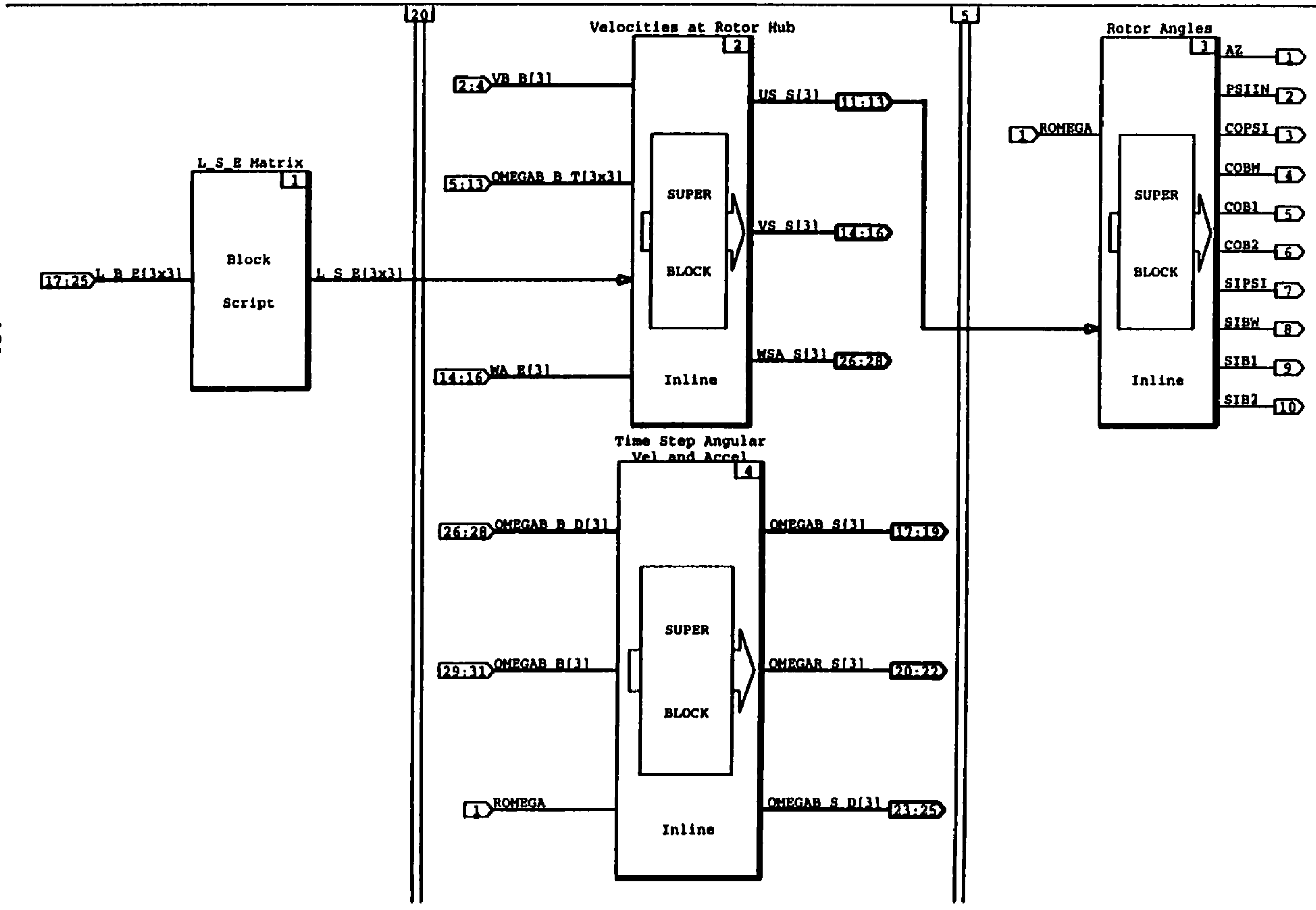


General Data for Each Time Step
General Data for Each Blade
General Data for Each Element
Rotor
Filter FR GR
SUPER BLOCK
Inline
Procedure
Read from VARIABLE

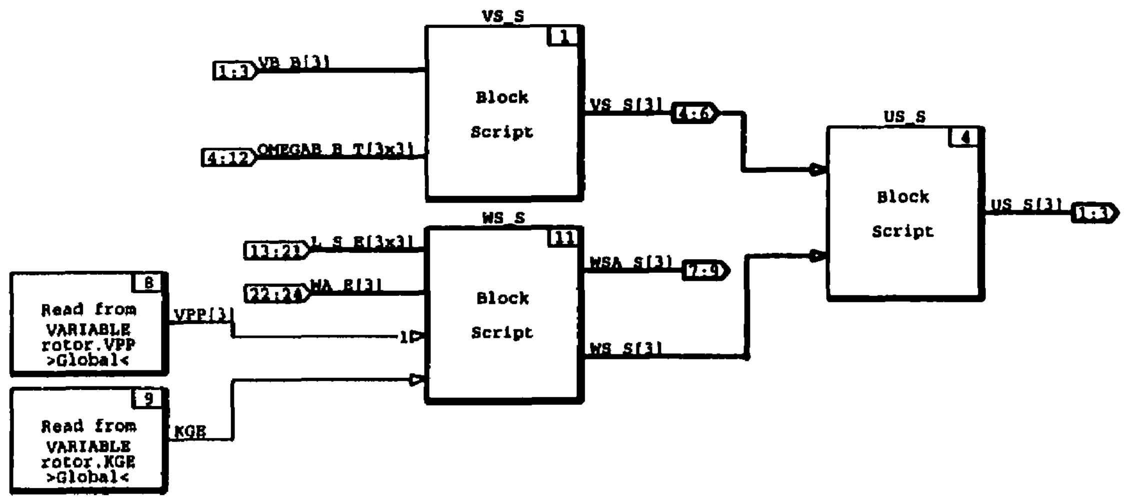
1.1

24-JUL-101

Discrete Procedure SuperBlock General Data for Each Time Step	Procedure Class	Inputs	Outputs
	Inline	31	28



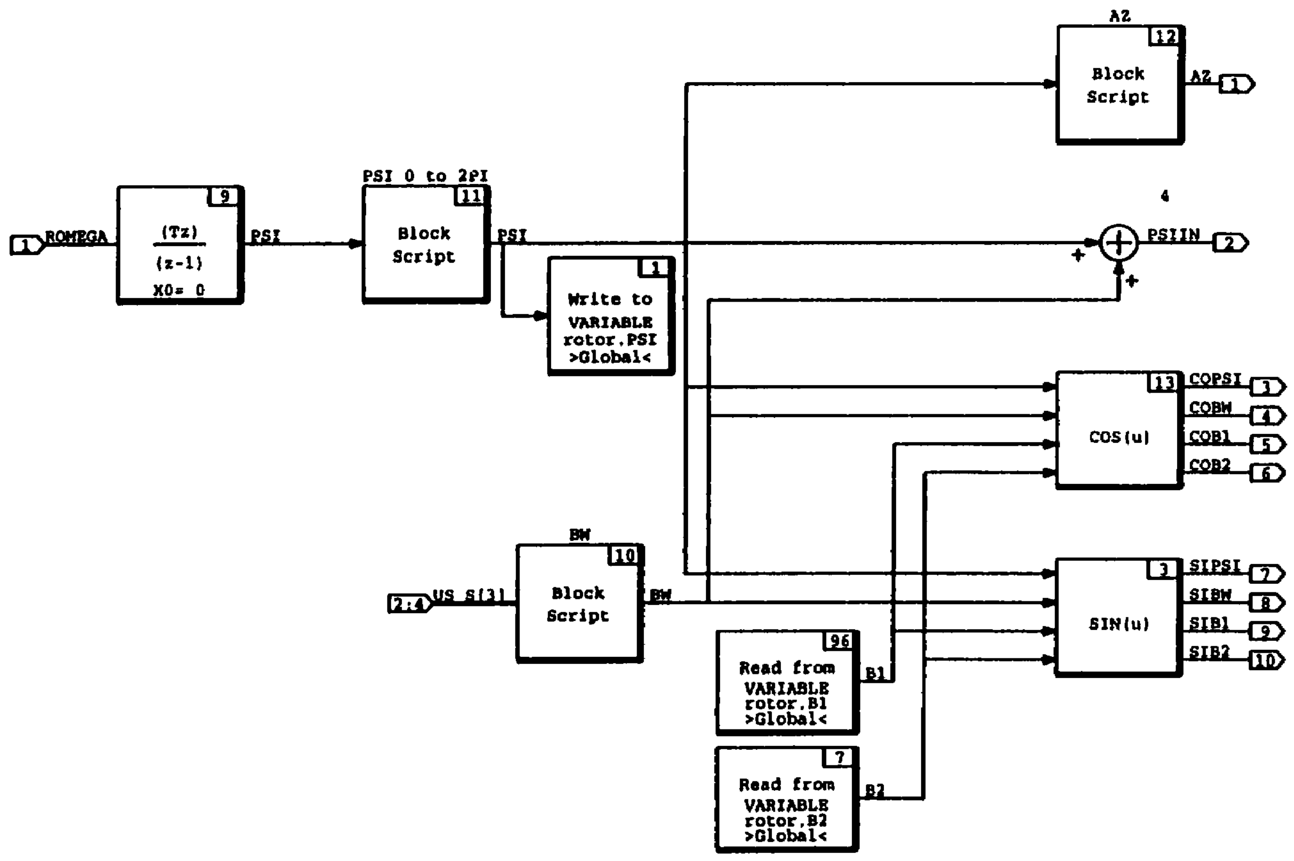
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Velocities at Rotor Hub	Inline	24	9



1.1.2

24-JUL-101

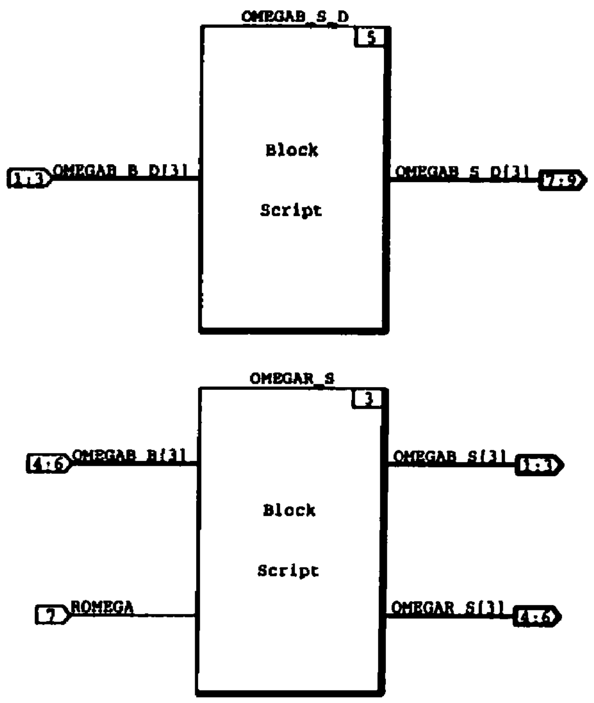
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Rotor Angles	Inline	4	10



1.1.3

24-JUL-101

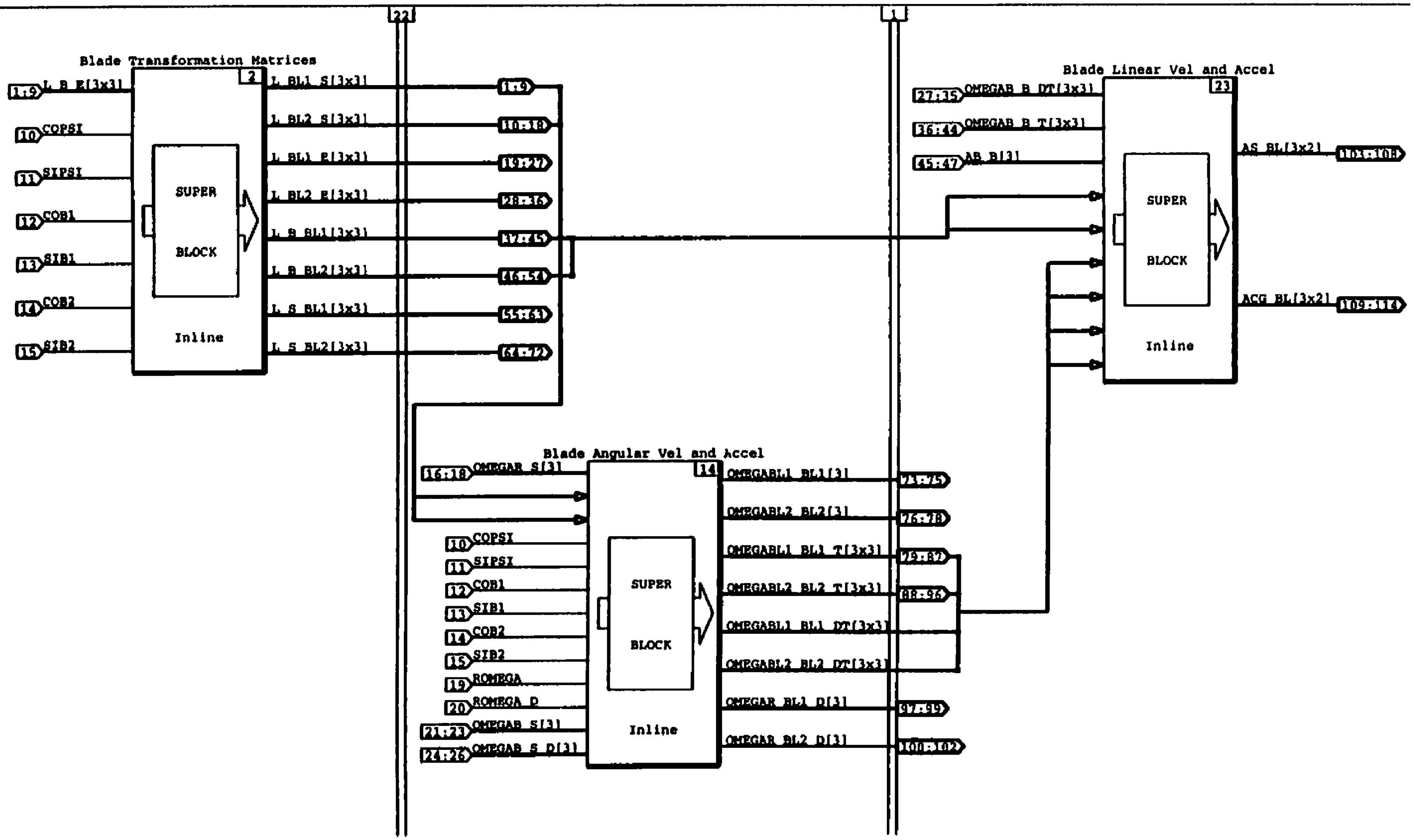
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Time Step Angular Vel and Accel	Inline	7	9



1.2

24-JUL-101

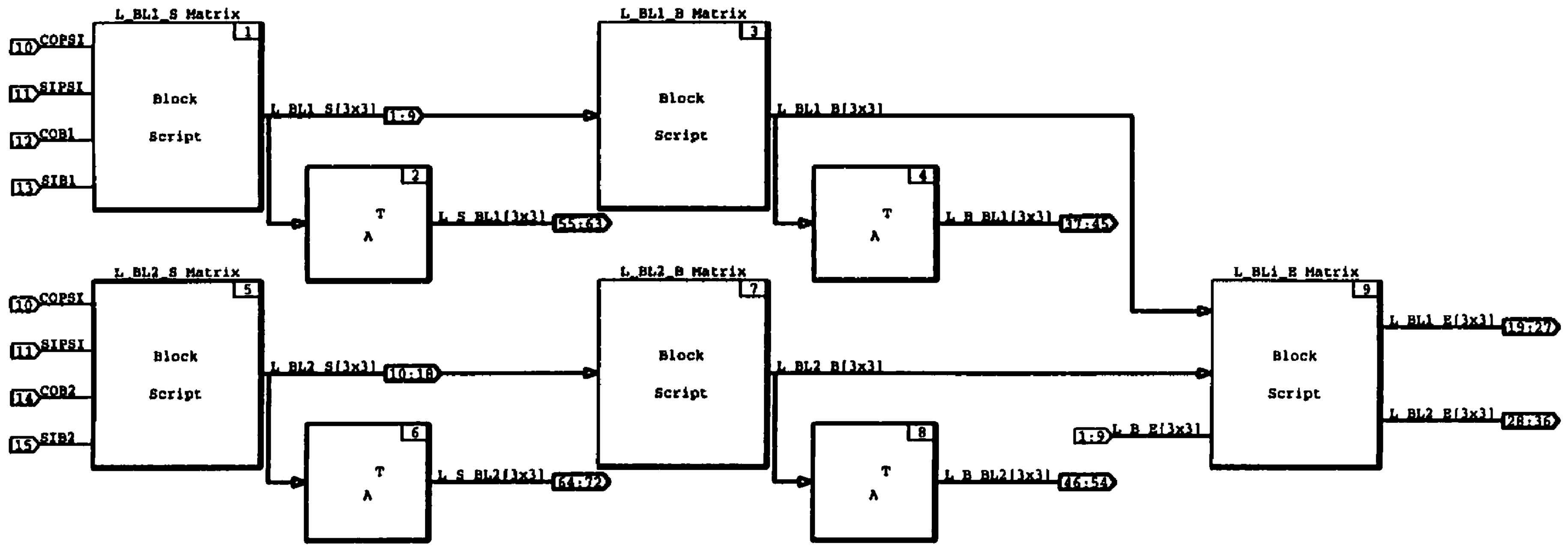
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
General Data for Each Blade	Inline	47	114



1.2.1

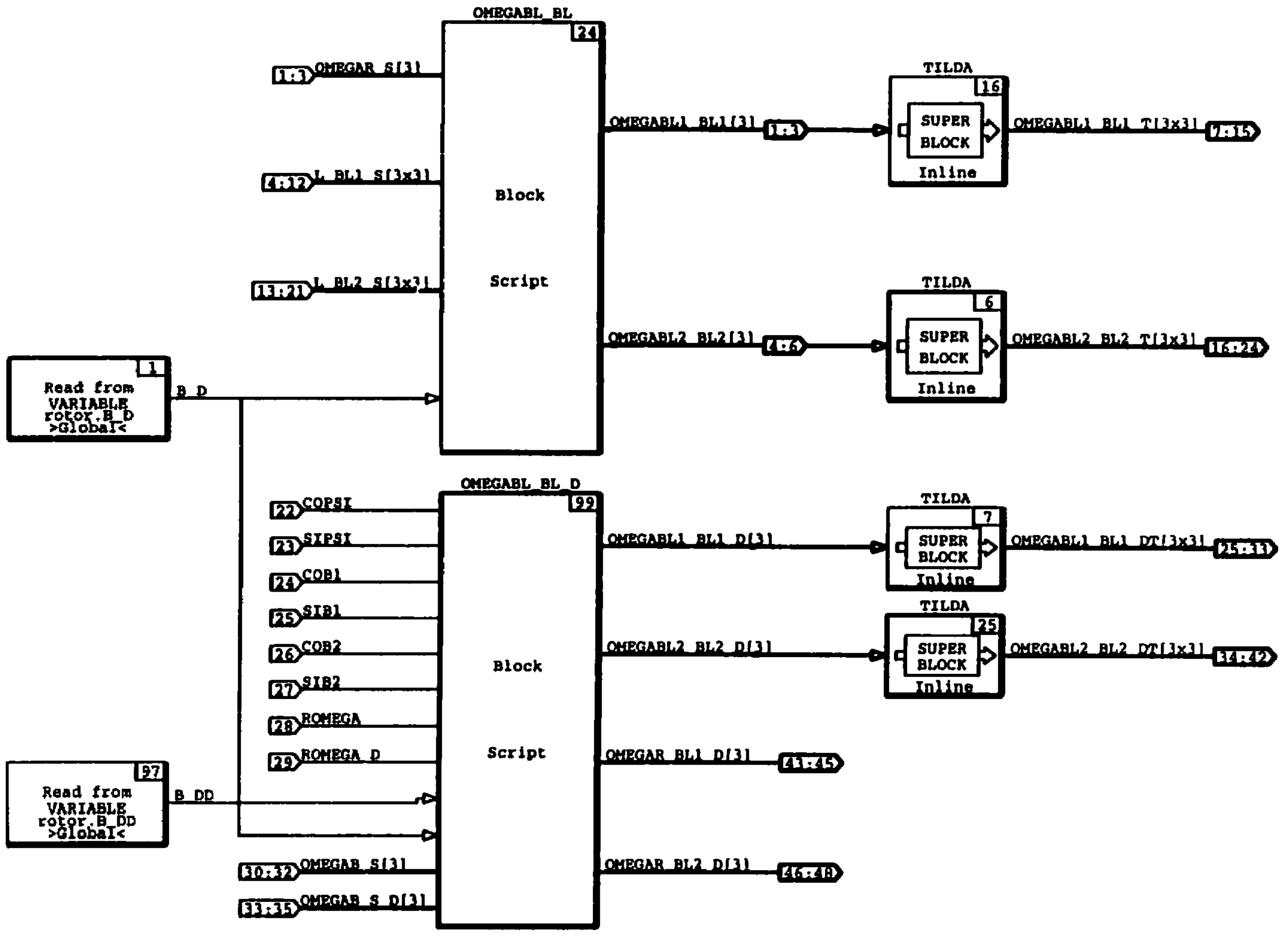
24-JUL-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Blade Transformation Matrices	Inline	15	72



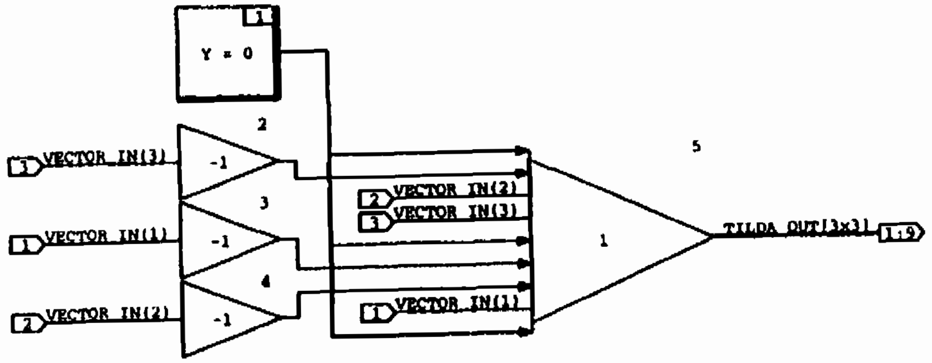
1.2.2

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Blade Angular Vel and Accel	Inline	35	48

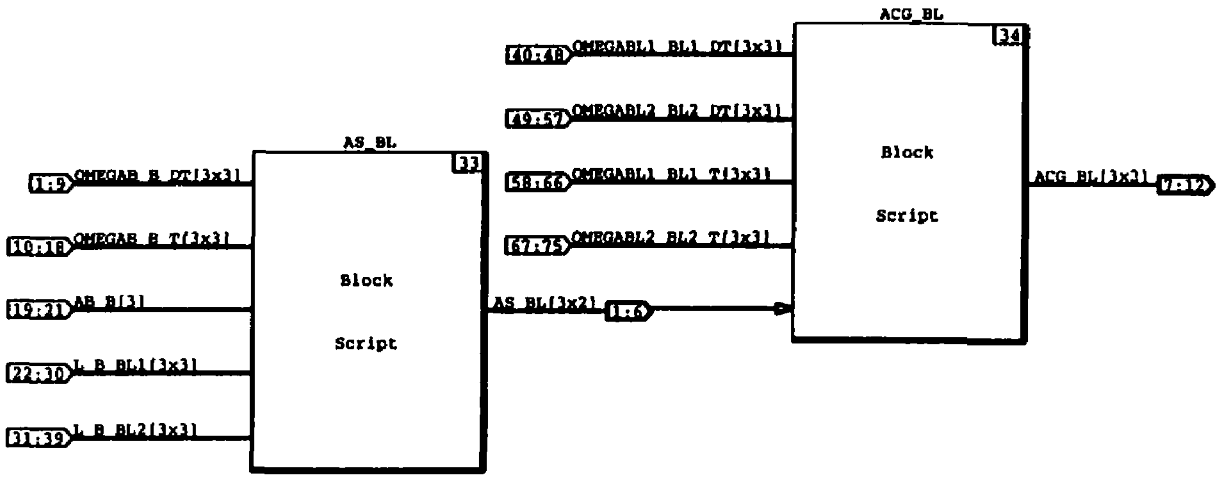


1.2.2.1

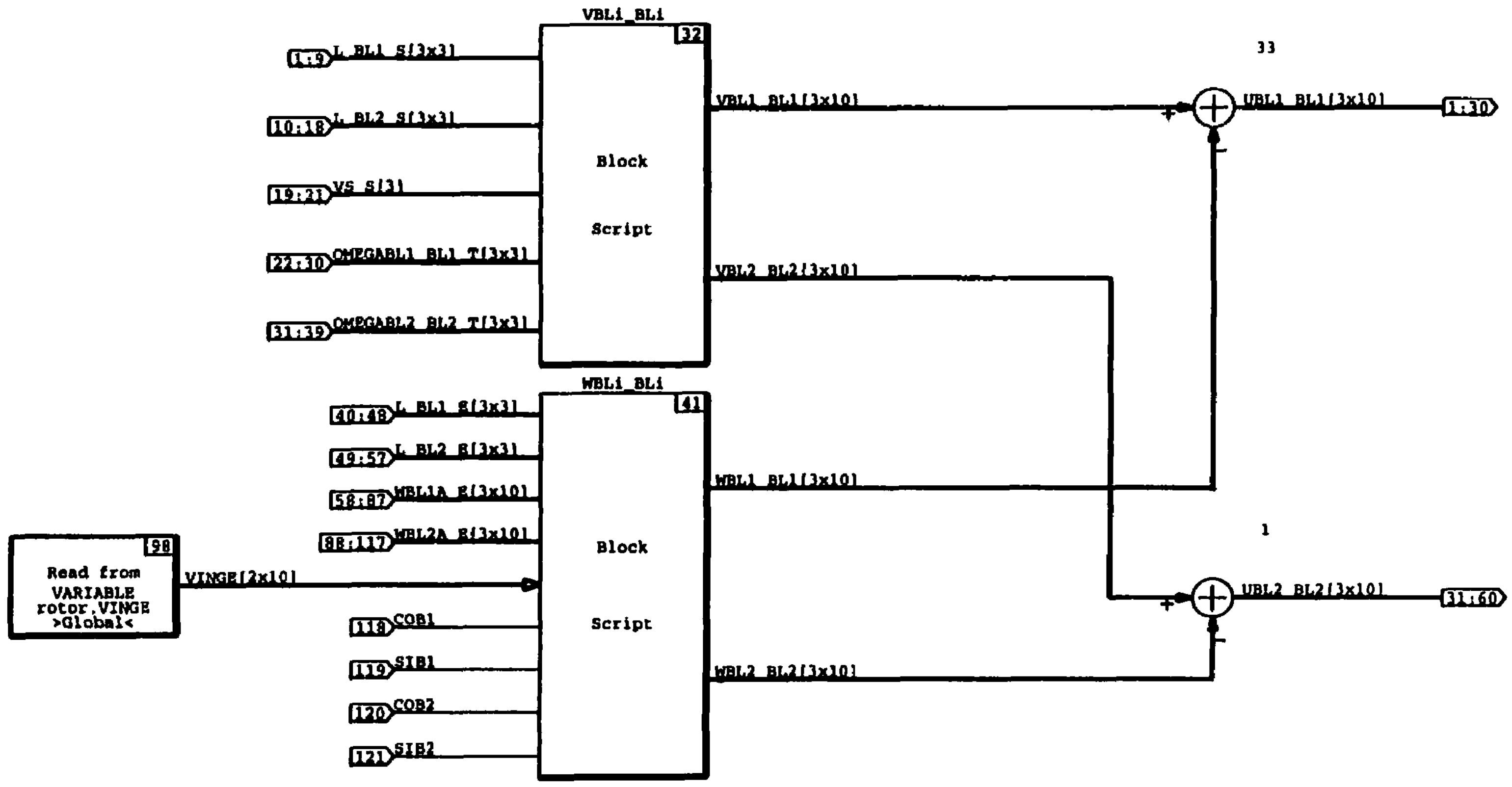
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
TILDA	Inline	3	9



Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Blade Linear Vel and Accel	Inline	75	12

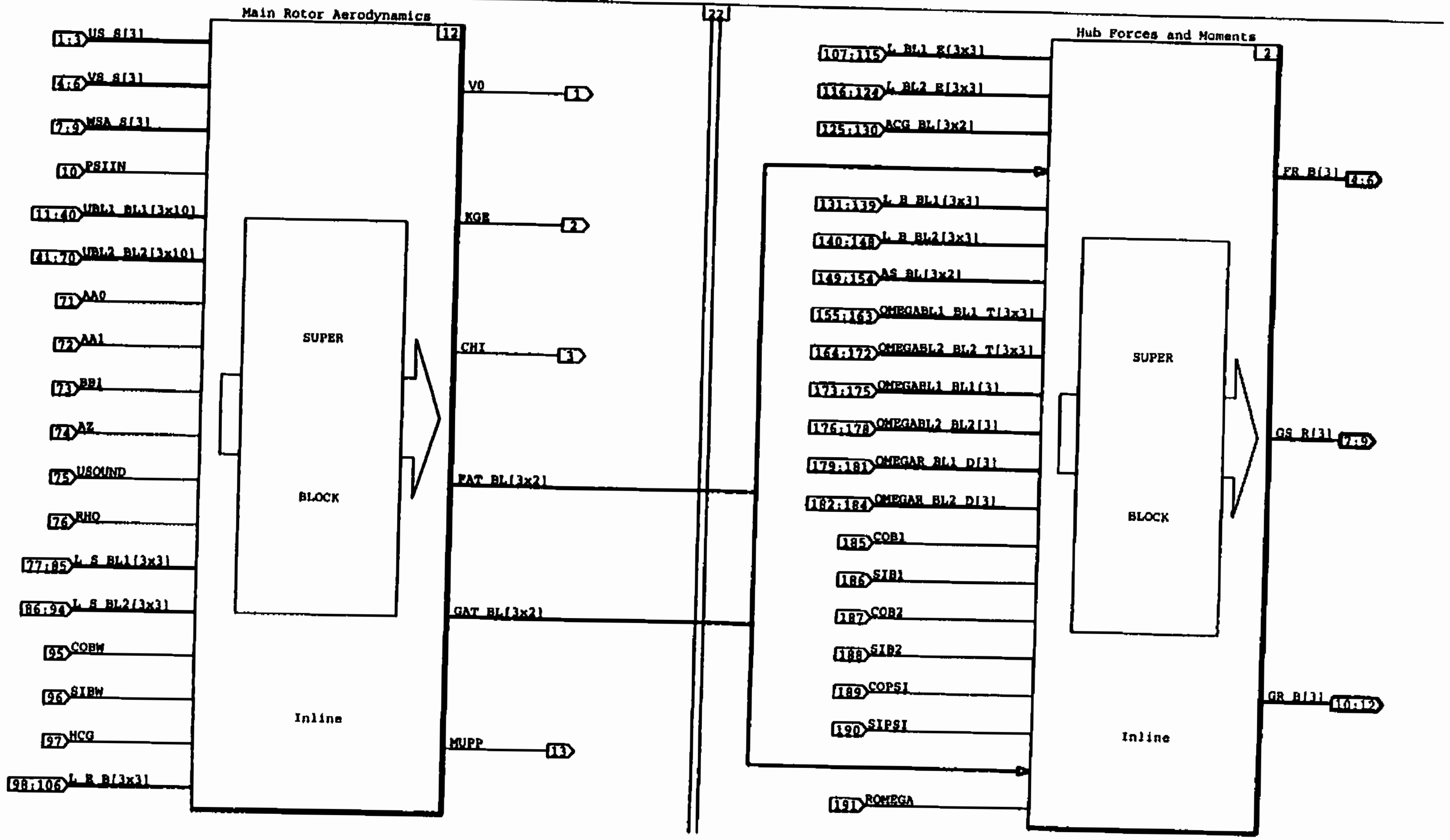


Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
General Data for Each Element	Inline	121	60



1.4

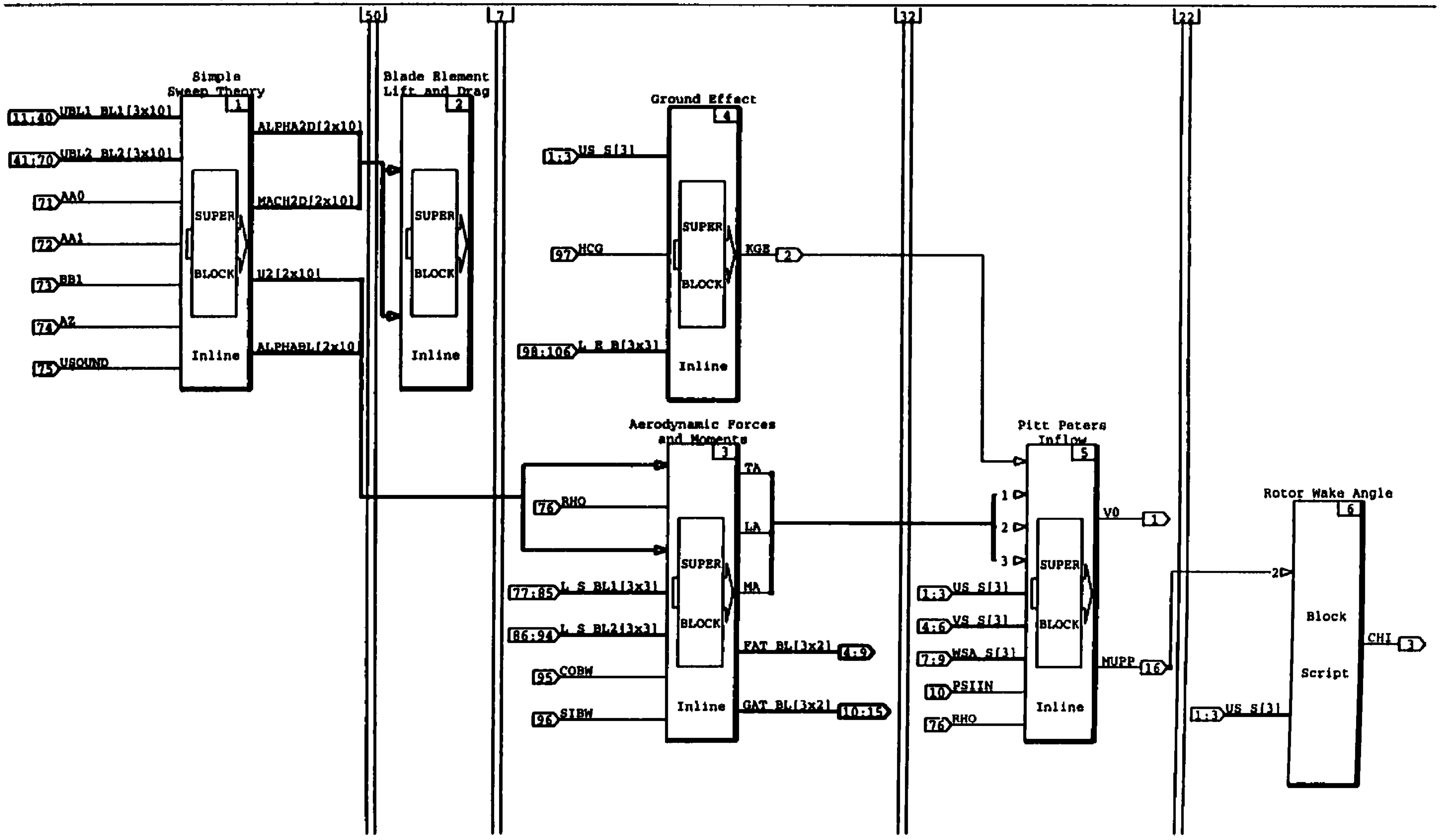
Discrete Procedure SuperBlock Rotor | Procedure Class Inline | Inputs 191 | Outputs 13 | 24-JUL-101



1.4.1

24-JUL-101

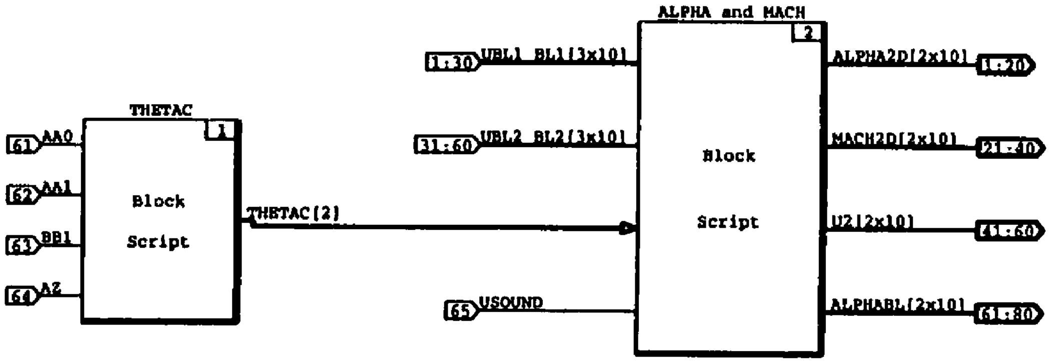
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Main Rotor Aerodynamics	Inline	106	16



1.4.1.1

24-JUL-101

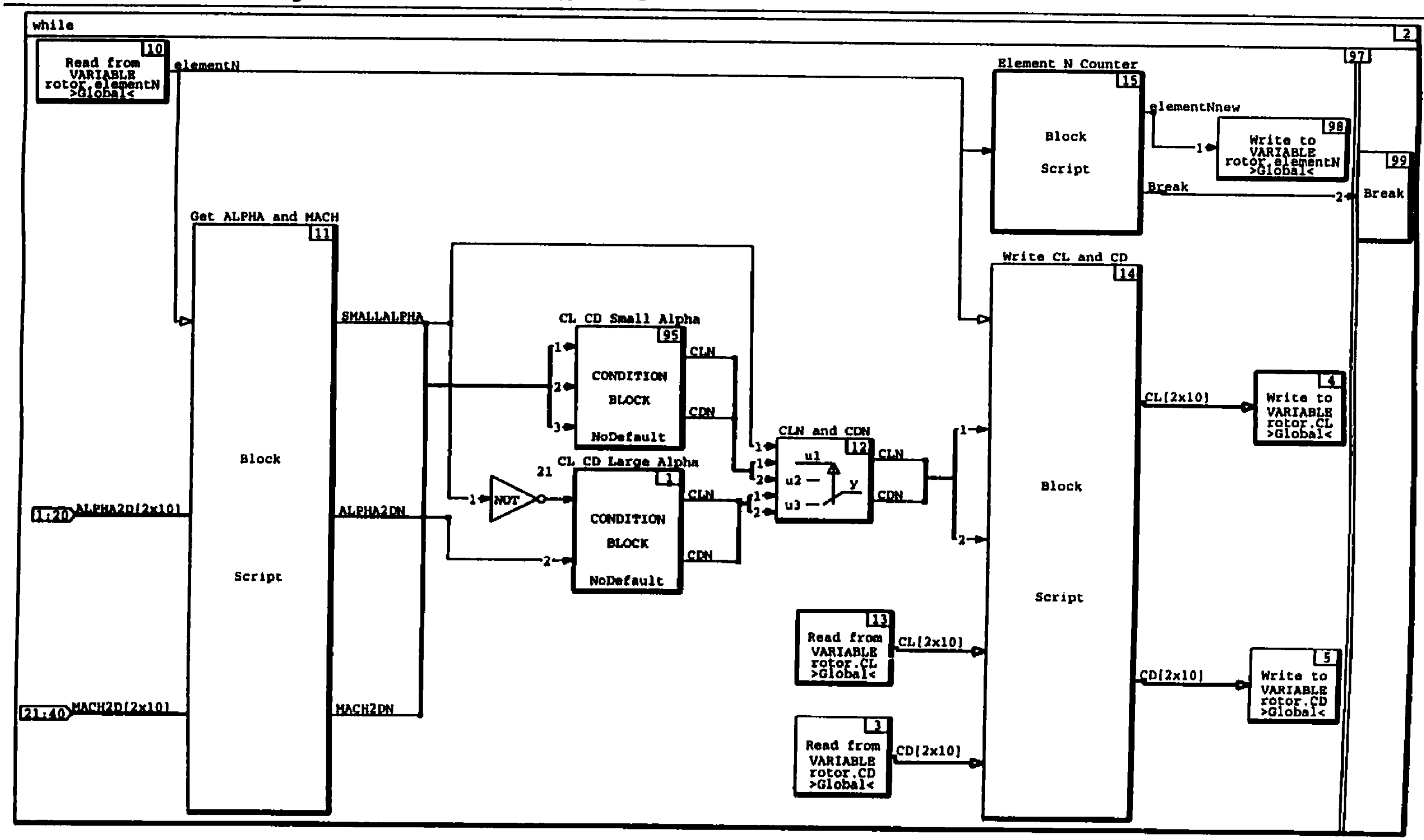
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Simple Sweep Theory	Inline	65	80



1.4.1.2

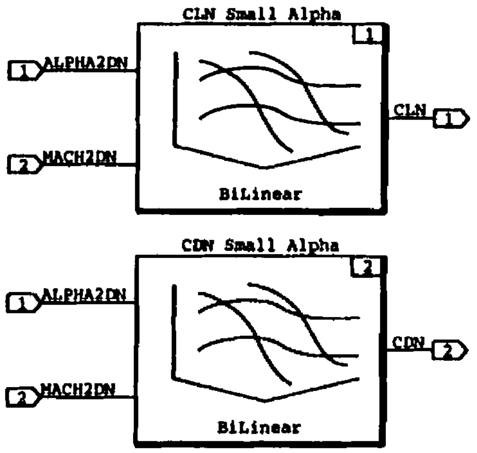
24-JUL-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Blade Element Lift and Drag	Inline	40	0



1.4.1.2.1

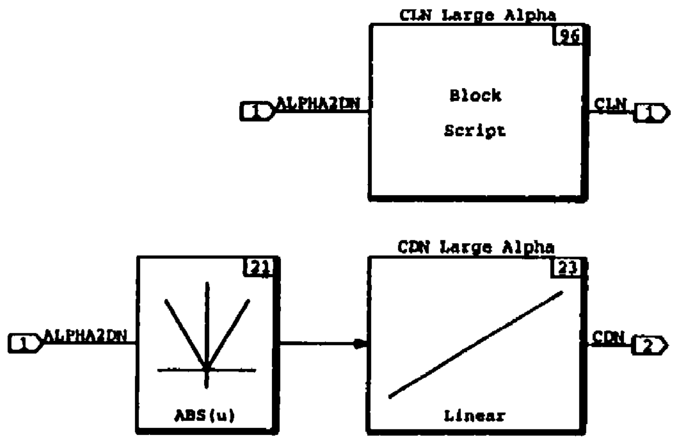
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
CL CD Small Alpha	Standard	2	2



1.4.1.2.2

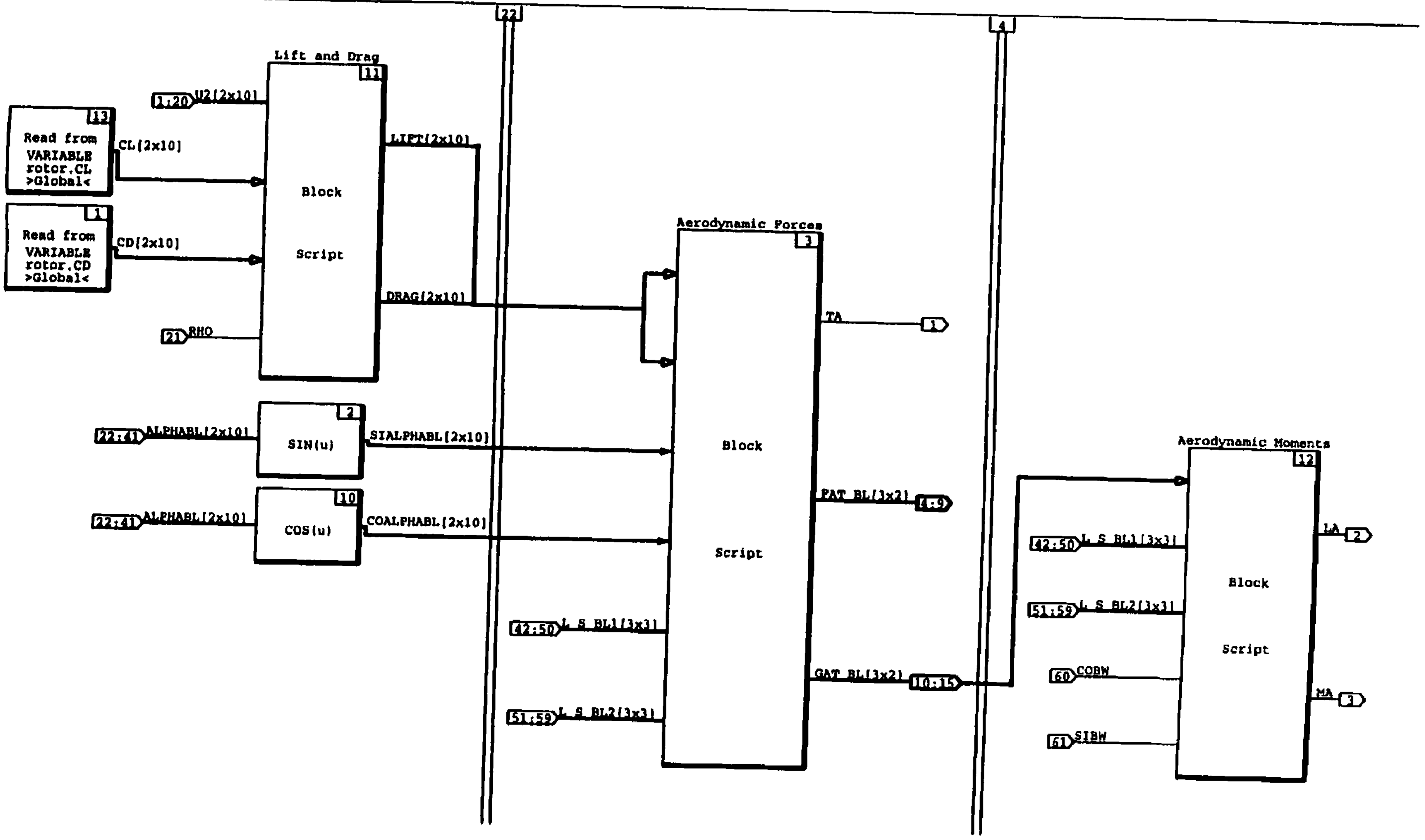
24-JUL-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
CL CD Large Alpha	Standard	1	2



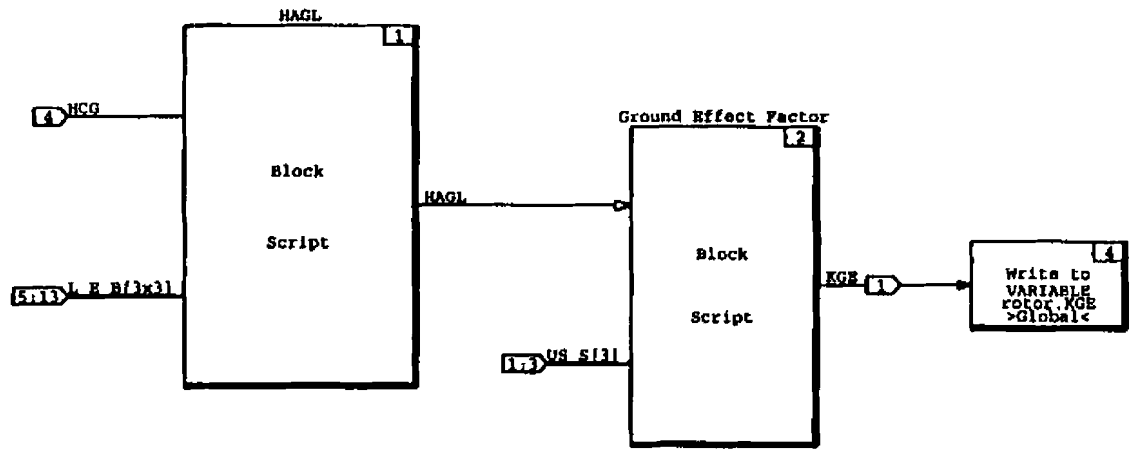
1.4.1.3

Discrete Procedure SuperBlock Aerodynamic Forces and Moments	Procedure Class Inline	Inputs 61	Outputs 15	24-JUL-101



1.4.1.4

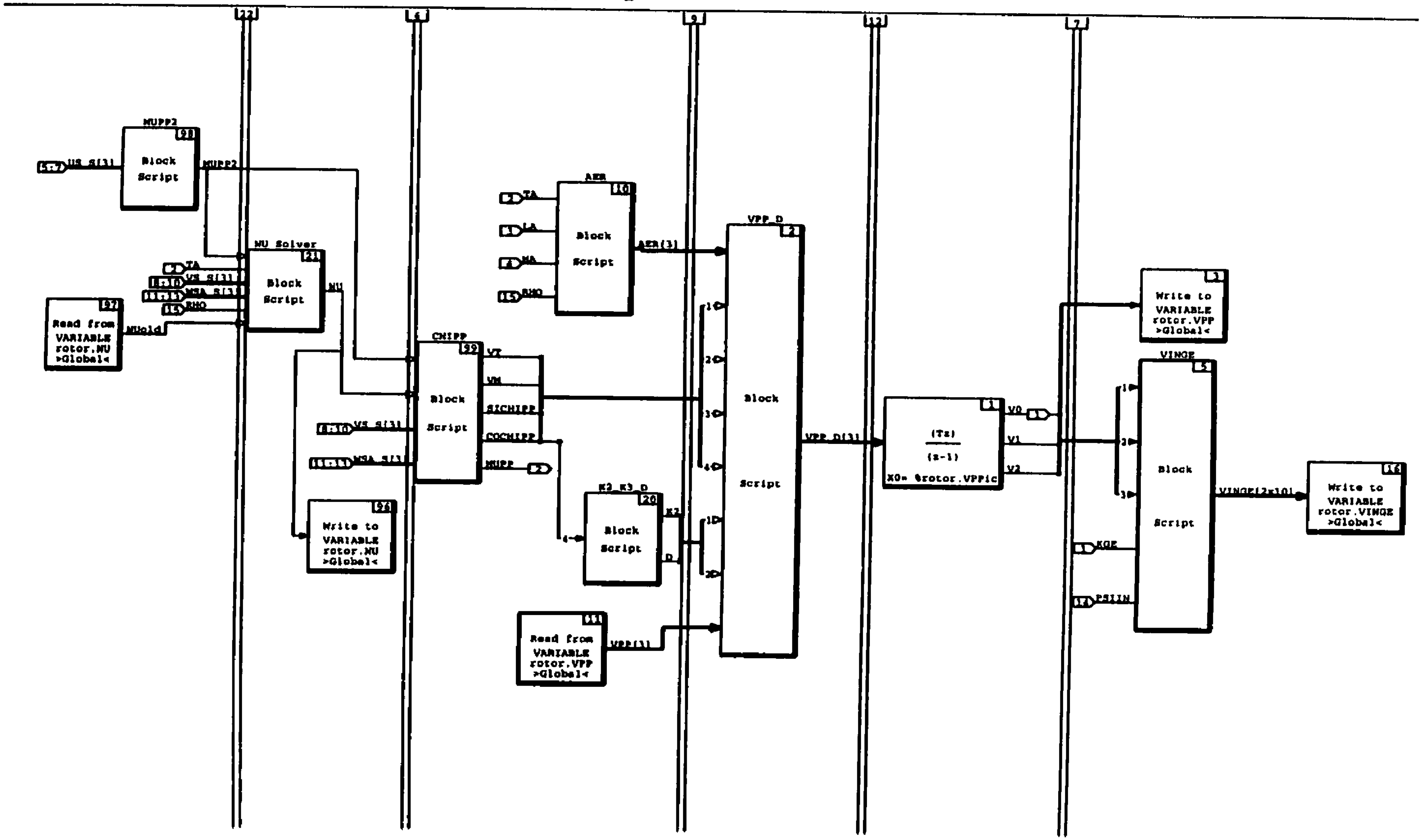
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Ground Effect	Inline	13	1



1.4.1.5

24-JUL-101

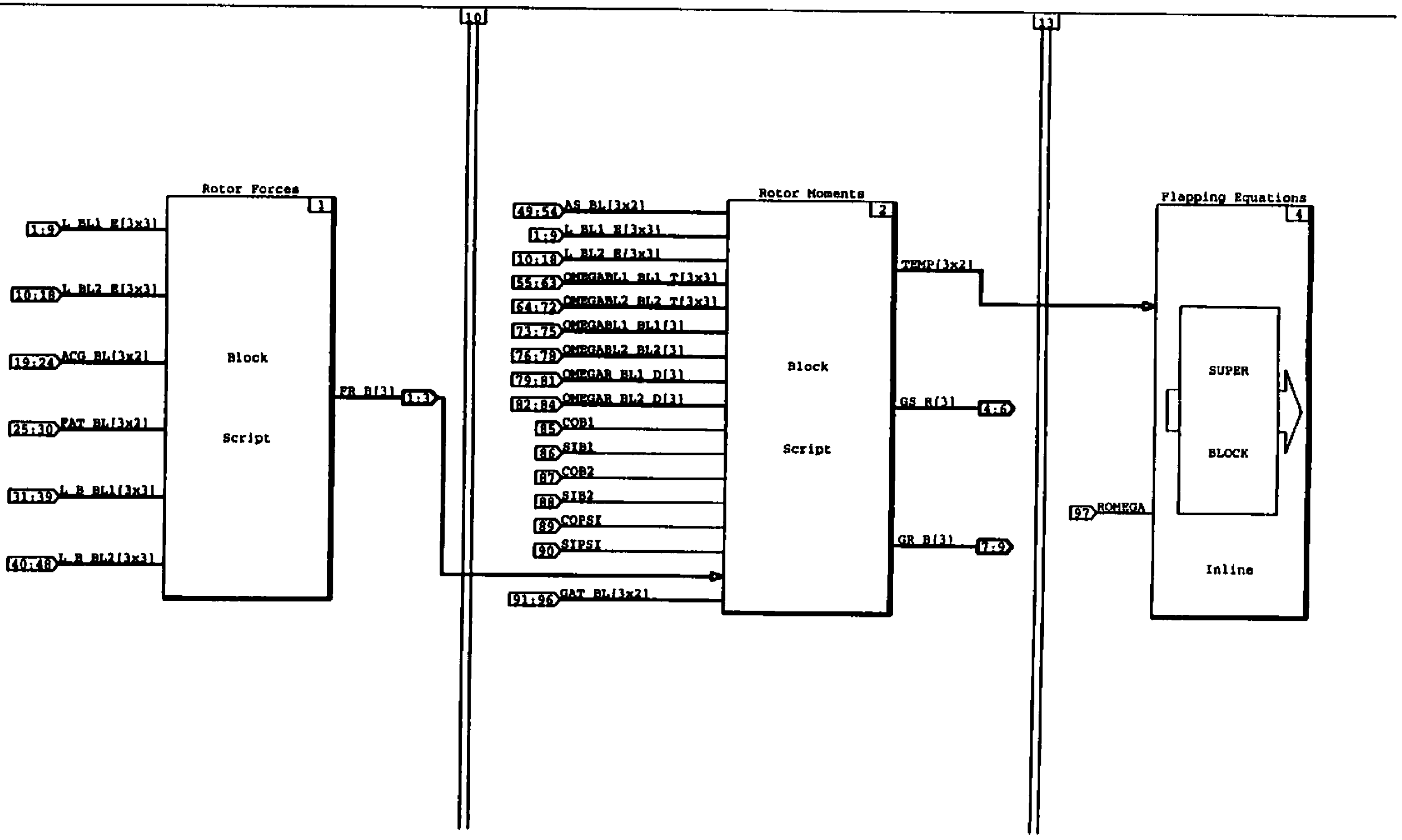
Discrete Procedure SuperBlock Pitt Peters Inflow | Procedure Class Inline | Inputs 15 | Outputs 2



1.4.2

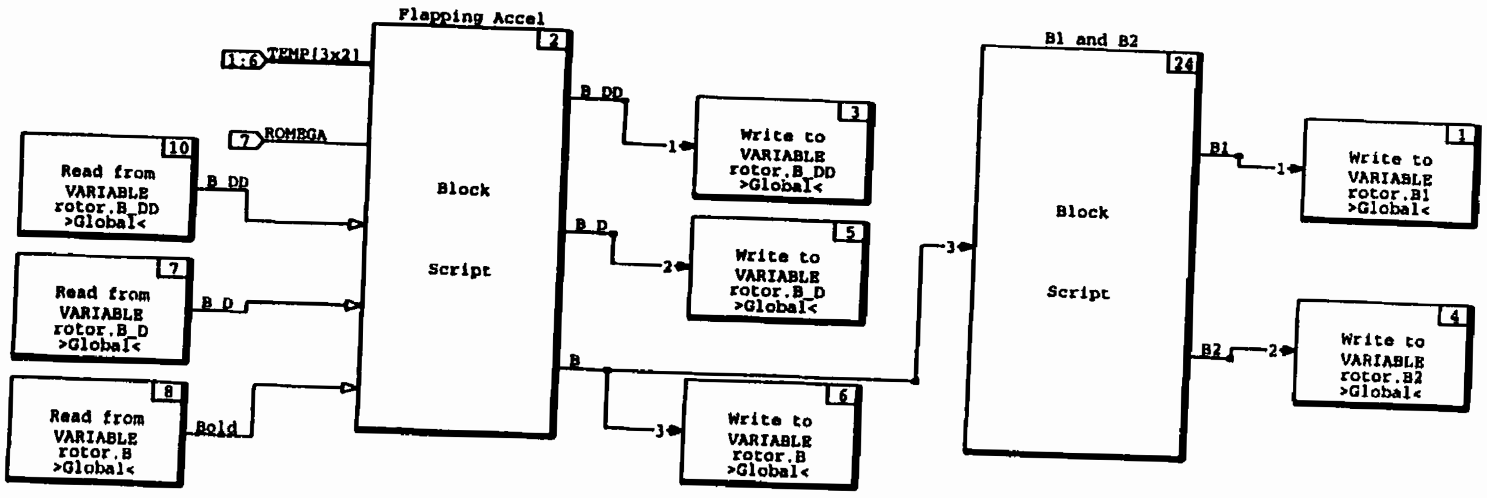
24-JUL-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Hub Forces and Moments	Inline	97	9

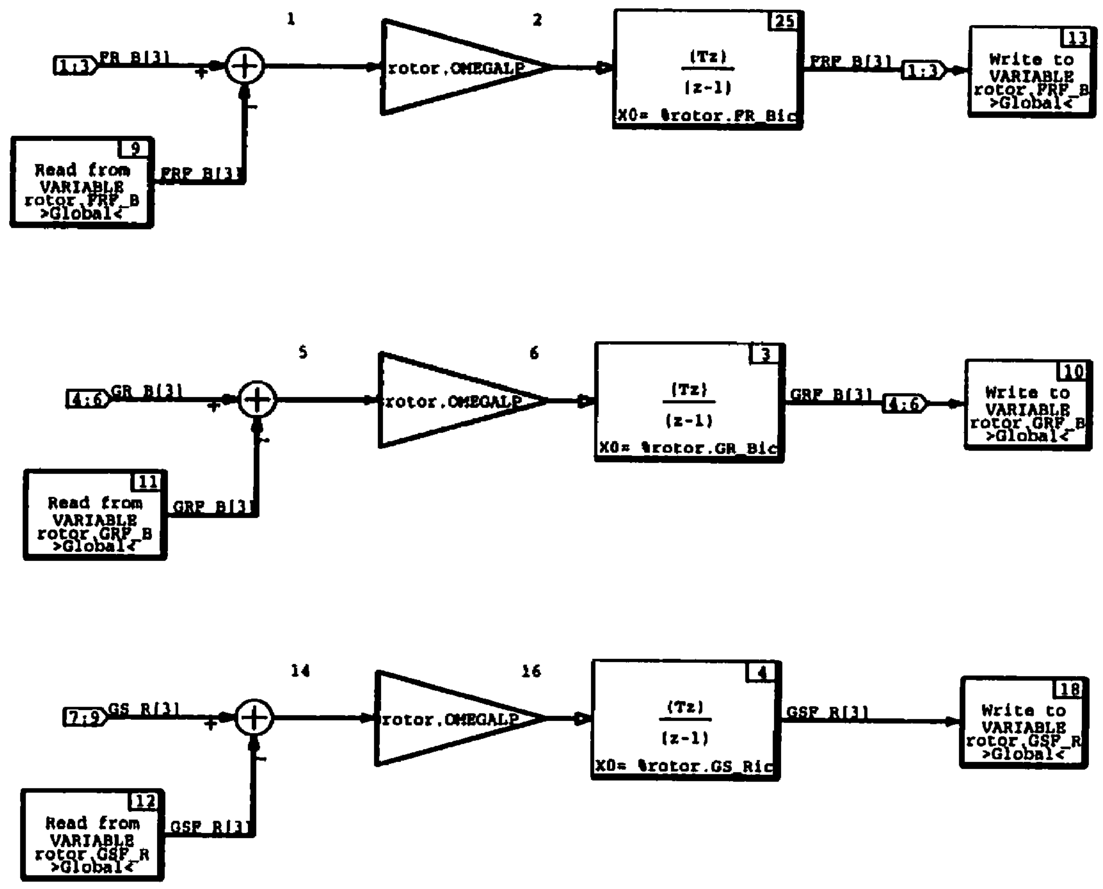


1.4.2.1

Discrete Procedure SuperBlock Flapping Equations	Procedure Class Inline	Inputs 7	Outputs 0	24-JUL-101



Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Filter FR GR	Standard	9	6


1:3
FR B(3)
+
-
1
rotor.OMEGALP
2
25
(Tz)
(z-1)
X0= %rotor.FR_Bic
FRP B(3)
1:3
13
Write to VARIABLE rotor.FRP_B >Global<
9
Read from VARIABLE rotor.FRP_B >Global<
FRP B(3)
4:6
GR B(3)
5
6
3
X0= %rotor.GR_Bic
GRP B(3)
10
Write to VARIABLE rotor.GRP_B >Global<
11
Read from VARIABLE rotor.GRP_B >Global<
7:9
GS R(3)
14
16
4
X0= %rotor.GS_Ric
GSP R(3)
18
Write to VARIABLE rotor.GSP_R >Global<
12
Read from VARIABLE rotor.GSP_R >Global<

C. Main Rotor Block Script

Table of Contents

L_BLi_E Matrix

Lift and Drag

L_S_E Matrix

MUPP2

NU Solver

OMEGABL_BL

OMEGABL_BL_D

OMEGAB_S_D

OMEGAR_S

PSI 0 to 2PI

Rotor Forces

Rotor Moments

Rotor Wake Angle

THETAC

US_S

VBLi_BLi

VINGE

VPP_D

VS_S

WBLi_BLi

Write CL and CD

WS_S

```
inputs: (OMEGABL1_BL1_DT_V,OMEGABL2_BL2_DT_V,
         OMEGABL1_BL1_T_V,OMEGABL2_BL2_T_V,AS_BL_V);
outputs: ACG_BL_V;
parameters: RCG;

float OMEGABL1_BL1_DT_V(9),OMEGABL2_BL2_DT_V(9),
      OMEGABL1_BL1_T_V(9),OMEGABL2_BL2_T_V(9),AS_BL_V(6);
float ACG_BL_V(6);
float RCG;

float OMEGABL1_BL1_DT(3,3),OMEGABL2_BL2_DT(3,3),
      OMEGABL1_BL1_T(3,3),OMEGABL2_BL2_T(3,3),AS_BL(3,2),
      DUMMY_1(3,3),DUMMY_2(3,3),ACG_BL(3,2);

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    OMEGABL1_BL1_DT(K,L) = OMEGABL1_BL1_DT_V(M);
    OMEGABL2_BL2_DT(K,L) = OMEGABL2_BL2_DT_V(M);
    OMEGABL1_BL1_T(K,L) = OMEGABL1_BL1_T_V(M);
    OMEGABL2_BL2_T(K,L) = OMEGABL2_BL2_T_V(M);
  endfor;
endfor;

M = 0;
for K=1:3 do
  for I=1:2 do
    M = M + 1;
    AS_BL(K,I) = AS_BL_V(M);
  endfor;
endfor;


for K=1:3 do
  DUMMY_1(K,2) = OMEGABL1_BL1_DT(K,2);
  DUMMY_2(K,2) = OMEGABL2_BL2_DT(K,2);
  for M=1:3 do
    DUMMY_1(K,2) = DUMMY_1(K,2) + OMEGABL1_BL1_T(K,M)*OMEGABL1_BL1_T(M,2);
    DUMMY_2(K,2) = DUMMY_2(K,2) + OMEGABL2_BL2_T(K,M)*OMEGABL2_BL2_T(M,2);
  endfor;
endfor;

for K=1:3 do
  ACG_BL(K,1) = AS_BL(K,1) + DUMMY_1(K,2)*RCG;
  ACG_BL(K,2) = AS_BL(K,2) + DUMMY_2(K,2)*RCG;
endfor;

M = 0;
for K=1:3 do
  for I=1:2 do
    M = M + 1;
    ACG_BL_V(M) = ACG_BL(K,I);
  endfor;
endfor;
```

```
inputs: (TA,LA,MA,RHO);
outputs: AER;
parameters: (R);
environment: PI;

float TA,LA,MA,RHO;
float AER(3);
float R;
float RHOPIR3;

RHOPIR3 = RHO*PI*R^3;
AER(1) =  TA    / (RHOPIR3);
AER(2) = (LA/R) / (RHOPIR3);
AER(3) = (MA/R) / (RHOPIR3);
```

1

Aug 1 2001 18:53:50 **Aerodynamic_Forces** Page 1

```
inputs: (LIFT_V,DRAG_V,SIALPHABL_V,COALPHABL_V,L_S_BL1_V,L_S_BL2_V);
outputs: (TA,FAT_BL_V,GAT_BL_V);
parameters: (NL,RM);

float LIFT_V(20),DRAG_V(20),SIALPHABL_V(20),COALPHABL_V(20),L_S_BL1_V(9),L_S_BL2_V(9
);
float TA,FAT_BL_V(6),GAT_BL_V(6);
integer NL;
float RM(10);
float LIFT(2,10),DRAG(2,10),SIALPHABL(2,10),COALPHABL(2,10),
      FATX_BL(2),FATZ_BL(2),FAX_BL(2,10),FAZ_BL(2,10),FAT_BL(3,2),
      GAT_BL(3,2),GATX_BL(2),GATZ_BL(2),L_S_BL1(3,3),L_S_BL2(3,3),FA_S(3);


Z = 0;
for I=1;2 do
  for J=1;10 do
    Z = Z + 1;
    LIFT(I,J) = LIFT_V(Z);
    DRAG(I,J) = DRAG_V(Z);
    SIALPHABL(I,J) = SIALPHABL_V(Z);
    COALPHABL(I,J) = COALPHABL_V(Z);
  endfor;
endfor;

Z = 0;
for K=1;3 do
  for L=1;3 do
    Z = Z + 1;
    L_S_BL1(K,L) = L_S_BL1_V(Z);
    L_S_BL2(K,L) = L_S_BL2_V(Z);
  endfor;
endfor;

for I=1;2 do
  FATX_BL(I) = 0;
  FATZ_BL(I) = 0;
  GATX_BL(I) = 0;
  GATZ_BL(I) = 0;
  for J=1;NL do
    FAX_BL(I,J) =  LIFT(I,J)*SIALPHABL(I,J) - DRAG(I,J)*COALPHABL(I,J);
    FAZ_BL(I,J) = -LIFT(I,J)*COALPHABL(I,J) - DRAG(I,J)*SIALPHABL(I,J);

    FATX_BL(I) = FATX_BL(I) + FAX_BL(I,J);
    FATZ_BL(I) = FATZ_BL(I) + FAZ_BL(I,J);

    GATX_BL(I) = GATX_BL(I) + RM(J)*FAZ_BL(I,J);
    GATZ_BL(I) = GATZ_BL(I) - RM(J)*FAX_BL(I,J);
 endfor;
endfor;

FAT_BL(1,1) = FATX_BL(1);
FAT_BL(3,1) = FATZ_BL(1);
FAT_BL(1,2) = FATX_BL(2);
FAT_BL(3,2) = FATZ_BL(2);

GAT_BL(1,1) = GATX_BL(1);
GAT_BL(3,1) = GATZ_BL(1);
GAT_BL(1,2) = GATX_BL(2);
GAT_BL(3,2) = GATZ_BL(2);

for K=1;3 do
  FA_S(K) = L_S_BL1(K,1)*FAT_BL(1,1) + L_S_BL1(K,3)*FAT_BL(3,1) +
            L_S_BL2(K,1)*FAT_BL(1,2) + L_S_BL2(K,3)*FAT_BL(3,2);
endfor;

TA = -FA_S(3).

Z = 0;
for K=1;3 do
  for I=1;2 do
    Z = Z + 1;
    FAT_BL_V(Z) = FAT_BL(K,I);
```

Aug 1 2001 18:53:50 **Aerodynamic_Forces** Page 2

```
    GAT_BL_V(Z) = GAT_BL(K,I);
  endfor;
endfor;
```

```
inputs: (GAT_BL_V,L_S_BL1_V,L_S_BL2_V,COBW,SIBW);

outputs: (LA,MA);

float GAT_BL_V(6),L_S_BL1_V(9),L_S_BL2_V(9);
float LA,MA;
float GAT_BL(3,2),L_S_BL1(3,3),L_S_BL2(3,3),GA_S(3);

Z = 0;
for K=1;3 do
  for L=1;3 do
    Z = Z + 1;
    L_S_BL1(K,L) = L_S_BL1_V(Z);
    L_S_BL2(K,L) = L_S_BL2_V(Z);
  endfor;
endfor;

Z = 0;
for K=1;3 do
  for I=1;2 do
    Z = Z + 1;
    GAT_BL(K,I) = GAT_BL_V(Z);
  endfor;
endfor;

for K=1;3 do
  GA_S(K) = L_S_BL1(K,1)*GAT_BL(1,1) + L_S_BL1(K,3)*GAT_BL(3,1) +
            L_S_BL2(K,1)*GAT_BL(1,2) + L_S_BL2(K,3)*GAT_BL(3,2);
endfor;

LA = GA_S(1)*COBW + GA_S(2)*SIBW;
MA = GA_S(2)*COBW - GA_S(1)*SIBW;
```

```
inputs: (UBL1_BL1_V,UBL2_BL2_V,THETAC,USOUND);

outputs: (ALPHA2D_V,MACH2D_V,U2_V,ALPHABL_V);

parameters: (NL,ZLL,TWIST);

float UBL1_BL1_V(30),UBL2_BL2_V(30),THETAC(2),USOUND;
float ALPHA2D_V(20),MACH2D_V(20),U2_V(20),ALPHABL_V(20);
integer NL;
float ZLL,TWIST(10);
float UBL1_BL1(3,10),UBL2_BL2(3,10),UBLX_BL(2,10),UBLZ_BL(2,10),
      ALPHABL(2,10),PITCHBL(2,10),ALPHA2D(2,10),MACH2D(2,10),U2(2,10);


M = 0;
for K=1:3 do
  for J=1:10 do
    M = M + 1;
    UBL1_BL1(K,J) = UBL1_BL1_V(M);
    UBL2_BL2(K,J) = UBL2_BL2_V(M);
  endfor;
endfor;

for J=1:NL do
  UBLX_BL(1,J) = UBL1_BL1(1,J);
  UBLX_BL(2,J) = UBL2_BL2(1,J);
  UBLZ_BL(1,J) = UBL1_BL1(3,J);
  UBLZ_BL(2,J) = UBL2_BL2(3,J);
endfor;

for I=1:2 do
  for J=1:NL do

    if (UBLZ_BL(I,J) == 0) & (UBLX_BL(I,J) == 0) then
        ALPHABL(I,J) = 0;
    else
      ALPHABL(I,J) = ATAN2( UBLZ_BL(I,J) , UBLX_BL(I,J) );
    endif;

    PITCHBL(I,J) = THETAC(I) + ZLL + TWIST(J);
    ALPHA2D(I,J) = ALPHABL(I,J) + PITCHBL(I,J);
    U2(I,J) = UBLX_BL(I,J)^2 + UBLZ_BL(I,J)^2;
    MACH2D(I,J) = (U2(I,J)^0.5) / USOUND;

  endfor;
endfor;

M = 0;
for I=1:2 do
  for J=1:10 do
    M = M + 1;
    ALPHA2D_V(M) = ALPHA2D(I,J);
    MACH2D_V(M) = MACH2D(I,J);
    ALPHABL_V(M) = ALPHABL(I,J);
    U2_V(M) = U2(I,J);
  endfor;
endfor;
```

```
inputs: (OMEGAB_B_DT_V,OMEGAB_B_T_V,AB_B,L_BL1_B_V,L_BL2_B_V);

outputs: AS_BL_V;

parameters: RS_B;

float OMEGAB_B_DT_V(9),OMEGAB_B_T_V(9),AB_B(3),L_BL1_B_V(9),L_BL2_B_V(9);
float AS_BL_V(6);
float RS_B(3);
float OMEGAB_B_DT(3,3),OMEGAB_B_T(3,3),L_BL1_B(3,3),L_BL2_B(3,3),TEMP(3,3),AS_B(3),A
S_BL(3,2);

Z = 0;
for K=1;3 do
  for L=1;3 do
    Z = Z + 1;
    OMEGAB_B_DT(K,L) = OMEGAB_B_DT_V(Z);
    OMEGAB_B_T(K,L) = OMEGAB_B_T_V(Z);
    L_BL1_B(K,L) = L_BL1_B_V(Z);
    L_BL2_B(K,L) = L_BL2_B_V(Z);
  endfor;
endfor;

for K=1;3 do
  AS_B(K) = AB_B(K);
  for L=1;3 do
    TEMP(K,L) = 0;
    for M=1;3 do
      TEMP(K,L) = TEMP(K,L) + OMEGAB_B_T(K,M)*OMEGAB_B_T(M,L);
    endfor;
    AS_B(K) = AS_B(K) + (TEMP(K,L) + OMEGAB_B_DT(K,L))*RS_B(L);
  endfor;
endfor;

for K=1;3 do
  AS_BL(K,1) = 0;
  AS_BL(K,2) = 0;
  for L=1;3 do
    AS_BL(K,1) = AS_BL(K,1) + L_BL1_B(K,L)*AS_B(L);
    AS_BL(K,2) = AS_BL(K,2) + L_BL2_B(K,L)*AS_B(L);
  endfor;
endfor;

M = 0;
for K=1;3 do
  for I=1;2 do
    M = M + 1;
    AS_BL_V(M) = AS_BL(K,I);
  endfor;
endfor;
```

Aug 1 2001 18:53:50 **AZ** Page 1

```
inputs: PSI;
outputs: AZ;
parameters: PHASE;

float PSI;
float AZ;
float PHASE;

AZ = PSI + PHASE;
```

1

Aug 1 2001 18:53:50 **B1_and_B2** Page 1

```
inputs: B;

outputs: (B1,B2);

parameters: B0;

environment: PI;

float B;
float B1,B2;
float B0;

B1 = B0 + B;
B2 = B0 - B;
```

Aug 1 2001 18:19:22 **BW** Page 1

```
inputs: US_S;
outputs: BW;

float US_S(3);
float BW;

if (US_S(1) == 0) then
  if (US_S(2) == 0) then
    BW = 0;
  else
    BW = ATAN2(US_S(2),US_S(1));
  endif;
else
  BW = ATAN2(US_S(2),US_S(1));
endif;
```

Aug 1 2001 18:53:50 **CHIPP** Page 1

```
inputs: (MUPP2,NU,VS_S,WSA_S);

outputs: (VT,VM,SICHIPP,COCHIPP,MUPP);

float MUPP2,NU,VS_S(3),WSA_S(3);
float VT,VM,SICHIPP,COCHIPP,MUPP,LAMPP;
float CHIPP;

MUPP = (MUPP2)^0.5;

LAMPP =  NU - (VS_S(3) - WSA_S(3));

if ( MUPP == 0 ) & ( LAMPP == 0 ) then
  CHIPP = 0;
else
  CHIPP = ATAN2( MUPP,LAMPP );
endif;

SICHIPP = SIN(CHIPP);
COCHIPP = COS(CHIPP);

VT = (MUPP2 + LAMPP^2)^0.5;
if (VT <> 0.0) then
  VM = (MUPP2 + LAMPP*(LAMPP + NU))/VT;
endif;
```

Aug 1 2001 18:53:50 **CLN_Large_Alpha** Page 1

```
inputs: ALPHA2DN;

outputs: CLN;

float ALPHA2DN;
float CLN;

if (abs(ALPHA2DN) <= 180) & (abs(ALPHA2DN) >= 174) then
  CLN = 0.12833*abs(ALPHA2DN) - 23.1;
elseif (abs(ALPHA2DN) < 174) & (abs(ALPHA2DN) >= 158) then
  CLN = -0.0075*abs(ALPHA2DN) + 0.535;
elseif (abs(ALPHA2DN) < 158) & (abs(ALPHA2DN) >= 150) then
  CLN = 0.0375*abs(ALPHA2DN) - 6.575;
elseif (abs(ALPHA2DN) < 150) & (abs(ALPHA2DN) >= 32) then
  CLN = -0.01625*abs(ALPHA2DN) + 1.4875;
endif;

if (ALPHA2DN < -32) then
  CLN = -CLN;
endif;
```

Aug 1 2001 18:53:50 **Element_N_Counter** Page 1

```
inputs: elementN;

outputs: (elementNnew,Break);

parameters: NL;

integer elementN;
integer elementNnew,Break;
integer NL;

if (elementN < 2*NL) then
   elementNnew = elementN + 1;
   Break = 0;
else
   elementNnew = 1;
   Break = 1;
endif;
```

```
inputs: (TEMP_V,ROMEGA,B_DDold,B_Dold,Bold);
outputs: (B_DD,B_D,B);
parameters: IBL;
environment: TSAMP;

float TEMP_V(6),ROMEGA,B_DDold,B_Dold,Bold;
float B_DD,B_D,B;
float IBL(3,3);
float TEMP(3,2),DPHI,R2;

Z = 0;
for K = 1:3 do
  for L = 1:2 do
    Z = Z + 1;
    TEMP(K,L) = TEMP_V(Z);
  endfor;
endfor;

B_DD = (TEMP(1,1) - TEMP(1,2))/(2*IBL(1,1));

R2 = 2.0*ROMEGA;
DPHI = R2*TSAMP;
SIDPHI = sin(DPHI);
CODPHI = cos(DPHI);

B_D = B_Dold*CODPHI + B_DDold*SIDPHI/R2;
B = Bold + B_Dold*SIDPHI/R2 + (1. - CODPHI)*B_DDold/(R2*R2);
```

```
inputs: (elementN,ALPHA2D_V,MACH2D_V);

outputs: (SMALLALPHA,ALPHA2DN,MACH2DN);

parameters: (NL,R2D);

environment: PI;

integer elementN;
float ALPHA2D_V(20),MACH2D_V(20);
integer SMALLALPHA;
float ALPHA2DN,MACH2DN;
integer NL;
float R2D;

if (elementN <= NL) then
  ALPHA2DN = ALPHA2D_V(elementN);
  MACH2DN = MACH2D_V(elementN);
else
  ALPHA2DN = ALPHA2D_V(10 + elementN - NL);
  MACH2DN = MACH2D_V(10 + elementN - NL);
endif;

if (ALPHA2DN < -PI) then
  ALPHA2DN = ALPHA2DN + 2*PI;
elseif (ALPHA2DN > PI) then
  ALPHA2DN = ALPHA2DN - 2*PI;
endif;

ALPHA2DN = R2D*ALPHA2DN;

if (abs(ALPHA2DN) <= 32) then
  SMALLALPHA = 1;
else
  SMALLALPHA = 0;
endif;
```

Aug 1 2001 18:53:50 **Ground_Effect_Factor** Page 1

```
inputs: (HAGL,US_S);

outputs: (KGE);

parameters: R;

float HAGL,US_S(3);
float KGE,UTOT;
float R;

UTOT = (US_S(1)^2. + US_S(2)^2. + US_S(3)^2.)^0.5;

if (HAGL > (5.*R)) then
   KGE = 1.0;
else
   if (HAGL > 0) & (UTOT > 0) then
      KGE = (1.0 - 0.115*((R/HAGL)^2.)*US_S(3)/UTOT)^(-.6667);
   else
      KGE = 0.1;
   endif;
endif;
```

Aug 1 2001 18:53:50 **HAGL** Page 1

```
inputs: (HCG,L_E_B_V);
outputs: HAGL;
parameters: (RS_B);

float HCG,L_E_B_V(9);
float HAGL;
float RS_B(3);

float L_E_B(3,3);

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_E_B(K,L) = L_E_B_V(M);
  endfor;
endfor;

HAGL = HCG - ( L_E_B(3,1)*RS_B(1) + L_E_B(3,2)*RS_B(2) + L_E_B(3,3)*RS_B(3) );
```

1

Aug 1 2001 18:53:50 **K2_K3_D** Page 1

```
inputs: (COCHIPP);

outputs: (K2,D);

parameters: K1;

float COCHIPP;
float K2,D;
float K3;

K2 = 1 + COCHIPP;
K3 = 1 - COCHIPP;
D = 2*COCHIPP +  K1*K3;
```

Aug 1 2001 18:53:50 **L_BL1_B_Matrix** Page 1

```
inputs: L_BL1_S_V;

outputs: L_BL1_B_V;

parameters: (COTHETS,SITHETS);

float L_BL1_S_V(9);
float L_BL1_B_V(9);
float L_S_B(3,3);
float L_BL1_S(3,3),L_BL1_B(3,3);

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_BL1_S(K,L) = L_BL1_S_V(M);
  endfor;
endfor;

L_BL1_B(1,1) =  L_BL1_S(1,1)*COTHETS;
L_BL1_B(1,2) =  L_BL1_S(1,2);
L_BL1_B(1,3) = -L_BL1_S(1,1)*SITHETS;

L_BL1_B(2,1) =  L_BL1_S(2,1)*COTHETS + L_BL1_S(2,3)*SITHETS;
L_BL1_B(2,2) =  L_BL1_S(2,2);
L_BL1_B(2,3) = -L_BL1_S(2,1)*SITHETS + L_BL1_S(2,3)*COTHETS;

L_BL1_B(3,1) =  L_BL1_S(3,1)*COTHETS + L_BL1_S(3,3)*SITHETS;
L_BL1_B(3,2) =  L_BL1_S(3,2);
L_BL1_B(3,3) = -L_BL1_S(3,1)*SITHETS + L_BL1_S(3,3)*COTHETS;

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_BL1_B_V(M) = L_BL1_B(K,L);
  endfor;
endfor;
```

Aug 1 2001 18:53:50 **L_BL1_S_Matrix** Page 1

```
inputs: (COPSI,SIPSI,COB1,SIB1);
outputs: L_BL1_S_V;

float COPSI,SIPSI,COB1,SIB1;
float L_BL1_S_V(9);

float L_BL1_S(3,3);

L_BL1_S(1,1) = SIPSI;
L_BL1_S(1,2) = COPSI;
L_BL1_S(1,3) = 0;

L_BL1_S(2,1) = -COB1*COPSI;
L_BL1_S(2,2) =  COB1*SIPSI;
L_BL1_S(2,3) = -SIB1;

L_BL1_S(3,1) = -SIB1*COPSI;
L_BL1_S(3,2) =  SIB1*SIPSI;
L_BL1_S(3,3) =  COB1;

M = 0;
for K=1:3 do
   for L=1:3 do
      M = M + 1;
      L_BL1_S_V(M) = L_BL1_S(K,L);
   endfor;
endfor;
```

1

Aug 1 2001 18:53:50 **L_BL2_B_Matrix** Page 1

```
inputs: L_BL2_S_V;

outputs: L_BL2_B_V;

parameters: (COTHETS,SITHETS);

float L_BL2_S_V(9);
float L_BL2_B_V(9);
float L_S_B(3,3);
float L_BL2_S(3,3),L_BL2_B(3,3);

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_BL2_S(K,L) = L_BL2_S_V(M);
  endfor;
endfor;

L_BL2_B(1,1) =  L_BL2_S(1,1)*COTHETS;
L_BL2_B(1,2) =  L_BL2_S(1,2);
L_BL2_B(1,3) = -L_BL2_S(1,1)*SITHETS;

L_BL2_B(2,1) =  L_BL2_S(2,1)*COTHETS + L_BL2_S(2,3)*SITHETS;
L_BL2_B(2,2) =  L_BL2_S(2,2);
L_BL2_B(2,3) = -L_BL2_S(2,1)*SITHETS + L_BL2_S(2,3)*COTHETS;

L_BL2_B(3,1) =  L_BL2_S(3,1)*COTHETS + L_BL2_S(3,3)*SITHETS;
L_BL2_B(3,2) =  L_BL2_S(3,2);
L_BL2_B(3,3) = -L_BL2_S(3,1)*SITHETS + L_BL2_S(3,3)*COTHETS;

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_BL2_B_V(M) = L_BL2_B(K,L);
  endfor;
endfor;
```

Aug 1 2001 18:53:50 **L_BL2_S_Matrix** Page 1

```
inputs: (COPSI,SIPSI,COB2,SIB2);
outputs: L_BL2_S_V;

float COPSI,SIPSI,COB2,SIB2;
float L_BL2_S_V(9);

float L_BL2_S(3,3);

L_BL2_S(1,1) = -SIPSI;
L_BL2_S(1,2) = -COPSI;
L_BL2_S(1,3) =  0;

L_BL2_S(2,1) =  COB2*COPSI;
L_BL2_S(2,2) = -COB2*SIPSI;
L_BL2_S(2,3) = -SIB2;

L_BL2_S(3,1) =  SIB2*COPSI;
L_BL2_S(3,2) = -SIB2*SIPSI;
L_BL2_S(3,3) =  COB2;

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_BL2_S_V(M) = L_BL2_S(K,L);
  endfor;
endfor;
```

```
inputs: (L_BL1_B_V,L_BL2_B_V,L_B_E_V);
outputs: (L_BL1_E_V,L_BL2_E_V);

float L_BL1_B_V(9),L_BL2_B_V(9),L_B_E_V(9);
float L_BL1_E_V(9),L_BL2_E_V(9);

float L_BL1_B(3,3),L_BL2_B(3,3),L_B_E(3,3),L_BL1_E(3,3),L_BL2_E(3,3);

Z = 0;
for K=1:3 do
  for L=1:3 do
    Z = Z + 1;
    L_BL1_B(K,L) = L_BL1_B_V(Z);
    L_BL2_B(K,L) = L_BL2_B_V(Z);
    L_B_E(K,L) = L_B_E_V(Z);
  endfor;
endfor;

for L=1:3 do
  for M=1:3 do
    L_BL1_E(L,M) = 0;
    L_BL2_E(L,M) = 0;
    for K=1:3 do
      L_BL1_E(L,M) = L_BL1_E(L,M) + L_BL1_B(L,K)*L_B_E(K,M);
      L_BL2_E(L,M) = L_BL2_E(L,M) + L_BL2_B(L,K)*L_B_E(K,M);
    endfor;
  endfor;
endfor;

Z = 0;
for K=1:3 do
  for L=1:3 do
    Z = Z + 1;
    L_BL1_E_V(Z) = L_BL1_E(K,L);
    L_BL2_E_V(Z) = L_BL2_E(K,L);
  endfor;
endfor;
```

```
inputs: (U2_V,CL_V,CD_V,RHO);
outputs: (LIFT_V,DRAG_V);
parameters: (NL,S,TLF);

float U2_V(20),CL_V(20),CD_V(20),RHO;
float LIFT_V(20),DRAG_V(20);
integer NL;
float S(10),TLF(10);
float FREF(2,10),U2(2,10),CL(2,10),CD(2,10),LIFT(2,10),DRAG(2,10);

M = 0;
for I=1:2 do
   for J=1:10 do
      M = M + 1;
      U2(I,J) = U2_V(M);
      CL(I,J) = CL_V(M);
      CD(I,J) = CD_V(M);
   endfor;
endfor;

for I=1:2 do
   for J=1:NL do
      FREF(I,J) = S(J)*U2(I,J)*RHO/2;
      LIFT(I,J) = TLF(J) * FREF(I,J) * CL(I,J);
      DRAG(I,J) = FREF(I,J) * CD(I,J);
   endfor;
endfor;

M = 0;
for I=1:2 do
   for J=1:10 do
      M = M + 1;
      LIFT_V(M) = LIFT(I,J);
      DRAG_V(M) = DRAG(I,J);
   endfor;
endfor;
```

Aug 1 2001 18:19:22 **L_S_E_Matrix** Page 1

```
inputs: L_B_E_V;
outputs: L_S_E_V;
parameters: L_S_B;

float L_B_E_V(9);
float L_S_E_V(9);
float L_S_B(3,3);

float L_B_E(3,3),L_S_E(3,3);

M = 0;
for K=1;3 do
  for L=1;3 do
    M = M + 1;
    L_B_E(K,L) = L_B_E_V(M);
  endfor;
endfor;

L_S_E(1,1) = L_B_E(1,1)*L_S_B(1,1) + L_B_E(3,1)*L_S_B(1,3);
L_S_E(1,2) = L_B_E(1,2)*L_S_B(1,1) + L_B_E(3,2)*L_S_B(1,3);
L_S_E(1,3) = L_B_E(1,3)*L_S_B(1,1) + L_B_E(3,3)*L_S_B(1,3);

L_S_E(2,1) = L_B_E(2,1);
L_S_E(2,2) = L_B_E(2,2);
L_S_E(2,3) = L_B_E(2,3);

L_S_E(3,1) = L_B_E(1,1)*L_S_B(3,1) + L_B_E(3,1)*L_S_B(3,3);
L_S_E(3,2) = L_B_E(1,2)*L_S_B(3,1) + L_B_E(3,2)*L_S_B(3,3);
L_S_E(3,3) = L_B_E(1,3)*L_S_B(3,1) + L_B_E(3,3)*L_S_B(3,3);

M = 0;
for K=1;3 do
  for L=1;3 do
    M = M + 1;
    L_S_E_V(M) = L_S_E(K,L);
  endfor;
endfor;
```

Aug 1 2001 18:53:50 **MUPP2** Page 1

```
inputs: US_S;
outputs: MUPP2;

float US_S[3];
float MUPP2;

MUPP2 = US_S(1)^2 + US_S(2)^2;
```

```
# NU solver
# Uses relaxation method

inputs: (MUPP2,TA,VS_S,WSA_S,RHO,NUold);
outputs: NU;
parameters: R;
environment: PI;

float MUPP2,TA,VS_S(3),WSA_S(3),RHO,NUold;
float NU;
float R;
float NU_TEMP,NU_FUNC,NU_FUNC_DERIV,RHOPIR2,DUM;
integer ICNT;

NU = NUold;
NU_TEMP = NUold;

for ICNT = 1:50 do
   DUM = (MUPP2 + (NU_TEMP - VS_S(3) + WSA_S(3))^2);
   if (DUM == 0) then
      DUM = 1.E-6;
   endif;

   NU_TEMP = TA/(2*RHO*PI*R*R * DUM^0.5);
   NU_TEMP = 0.1*NU_TEMP + 0.9*NU;
   NU = NU_TEMP;
endfor;
```

```
inputs: (OMEGAR_S,L_BL1_S_V,L_BL2_S_V,B_D);
outputs: (OMEGABL1_BL1,OMEGABL2_BL2);

float OMEGAR_S(3),L_BL1_S_V(9),L_BL2_S_V(9),B_D;
float OMEGABL1_BL1(3),OMEGABL2_BL2(3);
float OMEGAR_BL(3,2),L_BL1_S(3,3),L_BL2_S(3,3),OMEGABL_BL(3,2);

Z = 0;
for K=1:3 do
  for L=1:3 do
    Z = Z + 1;
    L_BL1_S(K,L) = L_BL1_S_V(Z);
    L_BL2_S(K,L) = L_BL2_S_V(Z);
  endfor;
endfor;

for K=1:3 do
  OMEGAR_BL(K,1) = 0;
  OMEGAR_BL(K,2) = 0;
  for L=1:3 do
    OMEGAR_BL(K,1) = OMEGAR_BL(K,1) + L_BL1_S(K,L)*OMEGAR_S(L);
    OMEGAR_BL(K,2) = OMEGAR_BL(K,2) + L_BL2_S(K,L)*OMEGAR_S(L);
  endfor;
endfor;

for K = 1:3 do
  for L = 1:2 do
    OMEGABL_BL(K,L) = OMEGAR_BL(K,L);
  endfor;
endfor;
OMEGABL_BL(1,1) = OMEGABL_BL(1,1) - B_D;
OMEGABL_BL(1,2) = OMEGABL_BL(1,2) + B_D;

for K=1:3 do
  OMEGABL1_BL1(K) = OMEGABL_BL(K,1);
  OMEGABL2_BL2(K) = OMEGABL_BL(K,2);
endfor;
```

```
inputs: (COPSI,SIPSI,COB1,SIB1,COB2,SIB2,ROMEGA,ROMEGA_D,B_DD,B_D,OMEGAB_S,OMEGAB_S_
D);

outputs: (OMEGABL1_BL1_D,OMEGABL2_BL2_D,OMEGAR_BL1_D,OMEGAR_BL2_D);

float COPSI,SIPSI,COB1,SIB1,COB2,SIB2,ROMEGA,ROMEGA_D,B_DD,B_D,OMEGAB_S(3),OMEGAB_S_
D(3);
float OMEGABL1_BL1_D(3),OMEGABL2_BL2_D(3),OMEGAR_BL1_D(3),OMEGAR_BL2_D(3);
float OMEGABL_BL_D(3,2),OMEGAR_BL_D(3,2);

OMEGAR_BL_D(1,1) = (OMEGAB_S(1)*ROMEGA + OMEGAB_S_D(2))*COPSI -
                   (OMEGAB_S(2)*ROMEGA - OMEGAB_S_D(1))*SIPSI;
OMEGAR_BL_D(1,2) = -OMEGAR_BL_D(1,1);

OMEGAR_BL_D(2,1) =  SIPSI*(COB1*(OMEGAB_S(1)*ROMEGA + OMEGAB_S_D(2)) - SIB1*OMEGAB_S
(2)*B_D) +
                    COPSI*(COB1*(OMEGAB_S(2)*ROMEGA - OMEGAB_S_D(1)) + SIB1*OMEGAB_S
(1)*B_D) +
                    SIB1*(ROMEGA_D - OMEGAB_S_D(3)) + COB1*(ROMEGA - OMEGAB_S(3))*B_
D;
OMEGAR_BL_D(2,2) =  SIPSI*(COB2*(-OMEGAB_S(1)*ROMEGA - OMEGAB_S_D(2)) - SIB2*OMEGAB_
S(2)*B_D) +
                    COPSI*(COB2*(-OMEGAB_S(2)*ROMEGA + OMEGAB_S_D(1)) + SIB2*OMEGAB_
S(1)*B_D) +
                    SIB2*(ROMEGA_D - OMEGAB_S_D(3)) - COB2*(ROMEGA - OMEGAB_S(3))*B_
D;

OMEGAR_BL_D(3,1) =  SIPSI*(SIB1*(OMEGAB_S(1)*ROMEGA + OMEGAB_S_D(2)) + COB1*OMEGAB_S
(2)*B_D) +
                    COPSI*(SIB1*(OMEGAB_S(2)*ROMEGA - OMEGAB_S_D(1)) - COB1*OMEGAB_S
(1)*B_D) +
                    SIB1*(ROMEGA - OMEGAB_S(3))*B_D - COB1*(ROMEGA_D - OMEGAB_S_D(3)
);
OMEGAR_BL_D(3,2) =  SIPSI*(SIB2*(-OMEGAB_S(1)*ROMEGA - OMEGAB_S_D(2)) + COB2*OMEGAB_
S(2)*B_D) +
                    COPSI*(SIB2*(-OMEGAB_S(2)*ROMEGA + OMEGAB_S_D(1)) - COB2*OMEGAB_
S(1)*B_D) -
                    SIB2*(ROMEGA - OMEGAB_S(3))*B_D -  COB2*(ROMEGA_D - OMEGAB_S_D(3
));

OMEGABL_BL_D(1,1) =  OMEGAR_BL_D(1,1) - B_DD;
OMEGABL_BL_D(1,2) = -OMEGABL_BL_D(1,1);

OMEGABL_BL_D(2,1) = OMEGAR_BL_D(2,1);
OMEGABL_BL_D(2,2) = OMEGAR_BL_D(2,2);

OMEGABL_BL_D(3,1) = OMEGAR_BL_D(3,1);
OMEGABL_BL_D(3,2) = OMEGAR_BL_D(3,2);

for K=1:3 do
  OMEGABL1_BL1_D(K) = OMEGABL_BL_D(K,1);
  OMEGABL2_BL2_D(K) = OMEGABL_BL_D(K,2);
  OMEGAR_BL1_D(K) = OMEGAR_BL_D(K,1);
  OMEGAR_BL2_D(K) = OMEGAR_BL_D(K,2);
endfor;
```

Aug 1 2001 18:19:22 **OMEGAB_S_D** Page 1

```
inputs: OMEGAB_B_D;
outputs: OMEGAB_S_D;
parameters: L_S_B;

float OMEGAB_B_D(3);
float OMEGAB_S_D(3);
float L_S_B(3,3);

for K=1:3 do
    OMEGAB_S_D(K) = 0;
  for L=1:3 do
    OMEGAB_S_D(K) = OMEGAB_S_D(K) + L_S_B(K,L)*OMEGAB_B_D(L);
  endfor;
endfor;
```

```
inputs: (OMEGAB_B,ROMEGA);
outputs: (OMEGAB_S,OMEGAR_S);
parameters: L_S_B;

float OMEGAB_B(3);
float OMEGAB_S(3),OMEGAR_S(3);
float L_S_B(3,3);

OMEGAB_S(1) = L_S_B(1,1)*OMEGAB_B(1) + L_S_B(1,3)*OMEGAB_B(3);
OMEGAB_S(2) = OMEGAB_B(2);
OMEGAB_S(3) = L_S_B(3,1)*OMEGAB_B(1) + L_S_B(3,3)*OMEGAB_B(3);

OMEGAR_S(1) = OMEGAB_S(1);
OMEGAR_S(2) = OMEGAB_S(2);
OMEGAR_S(3) = OMEGAB_S(3) - ROMEGA;
```

Aug 1 2001 18:19:22 **PSI_0_to_2PI** Page 1

```
inputs: PSI_IN;
outputs: PSI;
environment: PI;

float PSI_IN;
float PSI;

PSI = MOD(PSI_IN,2*PI);
```

1

```
inputs: (L_BL1_E_V,L_BL2_E_V,ACG_BL_V,FAT_BL_V,L_B_BL1_V,L_B_BL2_V);

outputs: (FR_B);

parameters: (MBLADE,WBLADE);

float L_BL1_E_V(9),L_BL2_E_V(9),ACG_BL_V(6),FAT_BL_V(6),L_B_BL1_V(9),L_B_BL2_V(9);
float FR_B(3);
float MBLADE,WBLADE;
float L_BL1_E(3,3),L_BL2_E(3,3),ACG_BL(3,2),FAT_BL(3,2),L_B_BL1(3,3),L_B_BL2(3,3),
      FG_BL(3,2),FHI_BL(3,2);

Z = 0;
for K=1:3 do
  for L=1:3 do
    Z = Z + 1;
    L_BL1_E(K,L) = L_BL1_E_V(Z);
    L_BL2_E(K,L) = L_BL2_E_V(Z);
    L_B_BL1(K,L) = L_B_BL1_V(Z);
    L_B_BL2(K,L) = L_B_BL2_V(Z);
  endfor;
endfor;

Z = 0;
for K=1:3 do
  for I=1:2 do
    Z = Z + 1;
    ACG_BL(K,I) = ACG_BL_V(Z);
    FAT_BL(K,I) = FAT_BL_V(Z);
  endfor;
endfor;

for K=1:3 do
  FG_BL(K,1) = WBLADE * L_BL1_E(K,3);
  FG_BL(K,2) = WBLADE * L_BL2_E(K,3);
endfor;

for K=1:3 do
  for I=1:2 do
    FHI_BL(K,I) = MBLADE * ACG_BL(K,I) - FAT_BL(K,I) - FG_BL(K,I);
  endfor;
endfor;

for K=1:3 do
  FR_B(K) = 0;
  for L=1:3 do
    FR_B(K) = FR_B(K) - L_B_BL1(K,L)*FHI_BL(L,1) - L_B_BL2(K,L)*FHI_BL(L,2) ;
  endfor;
endfor;
```

Aug 1 2001 18:53:50 **Rotor_Moments** Page 1

```
# Rotor Moments block

inputs: (AS_BL_V,L_BL1_E_V,L_BL2_E_V,OMEGABL1_BL1_T_V,OMEGABL2_BL2_T_V,
         OMEGABL1_BL1,OMEGABL2_BL2,OMEGAR_BL1_D,OMEGAR_BL2_D,COB1,SIB1,
         COB2,SIB2,COPSI,SIPSI,FR_B,GAT_BL_V);

outputs: (TEMP_V,GS_R,GR_B);

parameters: (MBLADE,WBLADE,IBL,RCG,COTHETS,SITHETS,RS_B_T);

environment: TIME;

float AS_BL_V(6),L_BL1_E_V(9),L_BL2_E_V(9),OMEGABL1_BL1_T_V(9),OMEGABL2_BL2_T_V(9),
      OMEGABL1_BL1(3),OMEGABL2_BL2(3),OMEGAR_BL1_D(3),OMEGAR_BL2_D(3),COB1,SIB1,
      COB2,SIB2,COPSI,SIPSI,FR_B(3),GAT_BL_V(6);
float TEMP_V(6),GS_R(3),GR_B(3);
float MBLADE,WBLADE,IBL(3,3),RCG,COTHETS,SITHETS,RS_B_T(3,3);
float AS_BL(3,2),GAT_BL(3,2),L_BL1_E(3,3),L_BL2_E(3,3),OMEGABL1_BL1_T(3,3),
      OMEGABL2_BL2_T(3,3),DUM(3,2),TEMP(3,2),GS_B(3);

Z = 0;
for K = 1:3 do
  for L = 1:2 do
    Z = Z + 1;
    AS_BL(K,L) = AS_BL_V(Z);
    GAT_BL(K,L) = GAT_BL_V(Z);
  endfor;
endfor;

Z = 0;
for K = 1:3 do
  for L = 1:3 do
    Z = Z + 1;
    L_BL1_E(K,L) = L_BL1_E_V(Z);
    L_BL2_E(K,L) = L_BL2_E_V(Z);
    OMEGABL1_BL1_T(K,L) = OMEGABL1_BL1_T_V(Z);
    OMEGABL2_BL2_T(K,L) = OMEGABL2_BL2_T_V(Z);
  endfor;
endfor;

DUM(1,1) = AS_BL(1,1)*MBLADE - L_BL1_E(1,3)*WBLADE;
DUM(1,2) = AS_BL(1,2)*MBLADE - L_BL2_E(1,3)*WBLADE;
DUM(3,1) = AS_BL(3,1)*MBLADE - L_BL1_E(3,3)*WBLADE;
DUM(3,2) = AS_BL(3,2)*MBLADE - L_BL2_E(3,3)*WBLADE;

for L = 1:3 do
  for N = 1:2 do
    TEMP(L,N) = 0;
  endfor;
endfor;

for M = 1:3 do
  TEMP(1,1) = TEMP(1,1) + OMEGABL1_BL1_T(1,M)*OMEGABL1_BL1(M)*IBL(M,M);
  TEMP(1,2) = TEMP(1,2) + OMEGABL2_BL2_T(1,M)*OMEGABL2_BL2(M)*IBL(M,M);
endfor;
TEMP(1,1) = TEMP(1,1) + OMEGAR_BL1_D(1)*IBL(1,1) + RCG*DUM(3,1) - GAT_BL(1,1);
TEMP(1,2) = TEMP(1,2) + OMEGAR_BL2_D(1)*IBL(1,1) + RCG*DUM(3,2) - GAT_BL(1,2);

for M = 1:3 do
  TEMP(2,1) = TEMP(2,1) + OMEGABL1_BL1_T(2,M)*OMEGABL1_BL1(M)*IBL(M,M);
  TEMP(2,2) = TEMP(2,2) + OMEGABL2_BL2_T(2,M)*OMEGABL2_BL2(M)*IBL(M,M);
endfor;
TEMP(2,1) = TEMP(2,1) + OMEGAR_BL1_D(2)*IBL(2,2) - GAT_BL(2,1);
TEMP(2,2) = TEMP(2,2) + OMEGAR_BL2_D(2)*IBL(2,2) - GAT_BL(2,2);

for M = 1:3 do
  TEMP(3,1) = TEMP(3,1) + OMEGABL1_BL1_T(3,M)*OMEGABL1_BL1(M)*IBL(M,M);
  TEMP(3,2) = TEMP(3,2) + OMEGABL2_BL2_T(3,M)*OMEGABL2_BL2(M)*IBL(M,M);
endfor;
TEMP(3,1) = TEMP(3,1) + OMEGAR_BL1_D(3)*IBL(3,3) - RCG*DUM(1,1) - GAT_BL(3,1);
TEMP(3,2) = TEMP(3,2) + OMEGAR_BL2_D(3)*IBL(3,3) - RCG*DUM(1,2) - GAT_BL(3,2);

GS_R(1) =  0;
```

Aug 1 2001 18:53:50 **Rotor_Moments** Page 2

```
GS_R(2) = -TEMP(2,1)*COB1 - TEMP(3,1)*SIB1 + TEMP(2,2)*COB2 + TEMP(3,2)*SIB2;
GS_R(3) =  TEMP(2,1)*SIB1 - TEMP(3,1)*COB1 + TEMP(2,2)*SIB2 - TEMP(3,2)*COB2;

GS_B(1) = -COPSI*COTHETS*GS_R(2) + SITHETS*GS_R(3);
GS_B(2) =  SIPSI*GS_R(2);
GS_B(3) =  COPSI*SITHETS*GS_R(2) + COTHETS*GS_R(3);

for K = 1:3 do
  GR_B(K) = GS_B(K);
  for L = 1:3 do
    GR_B(K) = GR_B(K) + RS_B_T(K,L)*FR_B(L);
  endfor;
endfor;

Z = 0;
for K = 1:3 do
  for L = 1:2 do
    Z = Z + 1;
    TEMP_V(Z) = TEMP(K,L);
  endfor;
endfor;
```

Aug 1 2001 18:53:50 **Rotor_Wake_Angle** Page 1

```
inputs: (MUPP,US_S);

outputs: CHI;

float MUPP,US_S(3);
float CHI;

if (MUPP == 0) & (US_S(3) == 0) then
  CHI = 0;
else
  CHI = ATAN2( MUPP, -US_S(3) );
endif;
```

Aug 1 2001 18:53:50 **THETATC** Page 1

```
inputs: (AA0,AA1,BB1,AZ);
outputs: (THETAC);

float AA0,AA1,BB1,AZ;
float THETAC(2);

float A1C,B1S;

A1C = AA1 * COS(AZ);
B1S = BB1 * SIN(AZ);

THETAC(1) = AA0 - A1C - B1S;
THETAC(2) = AA0 + A1C + B1S;
```

```
inputs: (VS_S,WS_S);
outputs: US_S;

float VS_S(3),WS_S(3);
float US_S(3);

for K=1;3 do
  US_S(K) = VS_S(K) - WS_S(K);
endfor;
```

Aug 1 2001 18:53:50 **VBLi_BLi** Page 1

```
inputs: (L_BL1_S_V,L_BL2_S_V,VS_S,OMEGABL1_BL1_T_V,OMEGABL2_BL2_T_V);

outputs: (VBL1_BL1_V,VBL2_BL2_V);

parameters: (NL,RM);

float L_BL1_S_V(9),L_BL2_S_V(9),VS_S(3),OMEGABL1_BL1_T_V(9),OMEGABL2_BL2_T_V(9);
float VBL1_BL1_V(30),VBL2_BL2_V(30);
integer NL;
float RM(10);
float L_BL1_S(3,3),L_BL2_S(3,3),OMEGABL1_BL1_T(3,3),OMEGABL2_BL2_T(3,3),
      VBL1_BL1(3,10),VBL2_BL2(3,10),VS_BL1(3),VS_BL2(3);

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_BL1_S(K,L) = L_BL1_S_V(M);
    L_BL2_S(K,L) = L_BL2_S_V(M);
    OMEGABL1_BL1_T(K,L) = OMEGABL1_BL1_T_V(M);
    OMEGABL2_BL2_T(K,L) = OMEGABL2_BL2_T_V(M);
  endfor;
endfor;

for K=1:3 do
  VS_BL1(K) = 0;
  VS_BL2(K) = 0;
  for L=1:3 do
    VS_BL1(K) = VS_BL1(K) + L_BL1_S(K,L)*VS_S(L);
    VS_BL2(K) = VS_BL2(K) + L_BL2_S(K,L)*VS_S(L);
  endfor;
endfor;

for K=1:3 do
  for J=1:NL do
    VBL1_BL1(K,J) = VS_BL1(K) + OMEGABL1_BL1_T(K,2)*RM(J);
    VBL2_BL2(K,J) = VS_BL2(K) + OMEGABL2_BL2_T(K,2)*RM(J);
  endfor;
endfor;

M = 0;
for K=1:3 do
  for J=1:10 do
    M = M + 1;
    VBL1_BL1_V(M) = VBL1_BL1(K,J);
    VBL2_BL2_V(M) = VBL2_BL2(K,J);
  endfor;
endfor;
```

Aug 1 2001 18:53:50 **VINGE** Page 1

```
inputs: (V0,V1,V2,KGE,PSIIN);
outputs: (VINGE_V);
parameters: (NL,RH,R);

float V0,V1,V2,KGE,PSIIN;
float VINGE_V(20);
integer NL;
float RH(10),R;
float V3,V4(10),VINGE(2,10);

V3 = V0 * KGE;

for J=1:NL do
  V4(J) = RH(J) * (V1*SIN(PSIIN) + V2*COS(PSIIN))/R;
  VINGE(1,J) = V3 + V4(J);
  VINGE(2,J) = V3 - V4(J);
endfor;

N = 0;
for I=1:2 do
  for J=1:10 do
    N = N + 1;
    VINGE_V(N) = VINGE(I,J);
  endfor;
endfor;
```

```
# VPP_D block

inputs: (AER,VT,VM,SICHIPP,COCHIPP,K2,D,VPP);
outputs: (VPP_D);
parameters: (MINV,R,K0);
environment: INIT;

float AER(3),VT,VM,SICHIPP,COCHIPP,K2,D,VPP(3);
float VP?_D(3);
float MINV(3,3),R,K0;
float LINV13,LINV31;

float LINV(3,3);

if (INIT) then
    for K = 1:3 do
        for L = 1:3 do
            LINV(K,L) = 0.;
         endfor;
    endfor;
endif;

LINV(2,2) = -(VM*K2)/4;
if (D <> 0.0) then
  LINV(1,1) = 4*VT*COCHIPP/D;
  LINV(1,3) = VT*K0*SICHIPP/D;
  LINV(3,3) = 2*LINV(2,2)/D;
endif;
if (VT <> 0.0) then
  LINV(3,1) = LINV(1,3)*VM/VT;
endif;

for K=1:3 do
  VPP_D(K) = MINV(K,K) * (AER(K) - (LINV(K,1)*VPP(1) + LINV(K,2)*VPP(2) + LINV(K,3)*
VPP(3))/R);
endfor;
```

```
inputs: (VB_B,OMEGAB_B_T_V);
outputs: VS_S;
parameters: (RS_B,L_S_B);

float VB_B(3),OMEGAB_B_T_V(9);
float VS_S(3);
float RS_B(3),L_S_B(3,3);
float OMEGAB_B_T(3,3),VS_B(3);

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    OMEGAB_B_T(K,L) = OMEGAB_B_T_V(M);
  endfor;
endfor;

for K=1:3 do
  VS_B(K) = VB_B(K);
  for L=1:3 do
    VS_B(K) = VS_B(K) + OMEGAB_B_T(K,L)*RS_B(L);
  endfor;
endfor;

VS_S(1) = L_S_B(1,1)*VS_B(1) + L_S_B(1,3)*VS_B(3);
VS_S(2) = VS_B(2);
VS_S(3) = L_S_B(3,1)*VS_B(1) + L_S_B(3,3)*VS_B(3);
```

```
inputs: (L_BL1_E_V,L_BL2_E_V,WBL1A_E_V,WBL2A_E_V,VINGE_V,COB1,SIB1,COB2,SIB2);

outputs: (WBL1_BL1_V,WBL2_BL2_V);

parameters: NL;

float L_BL1_E_V(9),L_BL2_E_V(9),WBL1A_E_V(30),WBL2A_E_V(30),VINGE_V(20),COB1,SIB1,CO
B2,SIB2;
float WBL1_BL1_V(30),WBL2_BL2_V(30);
integer NL;
float L_BL1_E(3,3),L_BL2_E(3,3),WBL1A_E(3,10),WBL2A_E(3,10),VINGE(2,10),
      WBL1_BL1(3,10),WBL2_BL2(3,10),
      WBL1A_BL1(3,10),WBL2A_BL2(3,10),VINBL1(3,10),VINBL2(3,10);

M = 0;
for K=1;3 do
  for L=1;3 do
    M = M + 1;
    L_BL1_E(K,L) = L_BL1_E_V(M);
    L_BL2_E(K,L) = L_BL2_E_V(M);
  endfor;
endfor;

M = 0;
for K=1;3 do
  for J=1;10 do
    M = M + 1;
    WBL1A_E(K,J) = WBL1A_E_V(M);
    WBL2A_E(K,J) = WBL2A_E_V(M);
  endfor;
endfor;

M = 0;
for I=1;2 do
  for J=1;10 do
    M = M + 1;
    VINGE(I,J) = VINGE_V(M);
  endfor;
endfor;

for K=1;3 do
  for J=1;NL do
    WBL1A_BL1(K,J) = 0;
    WBL2A_BL2(K,J) = 0;
    for L=1;3 do
      WBL1A_BL1(K,J) = WBL1A_BL1(K,J) + L_BL1_E(K,L)*WBL1A_E(L,J);
      WBL2A_BL2(K,J) = WBL2A_BL2(K,J) + L_BL2_E(K,L)*WBL2A_E(L,J);
    endfor;
  endfor;
endfor;

for J=1;NL do
  VINBL1(1,J) =  0;
  VINBL1(2,J) = -VINGE(1,J)*SIB1;
  VINBL1(3,J) =  VINGE(1,J)*COB1;

  VINBL2(1,J) =  0;
  VINBL2(2,J) = -VINGE(2,J)*SIB2;
  VINBL2(3,J) =  VINGE(2,J)*COB2;
endfor;

for K=1;3 do
  for J=1;NL do
    WBL1_BL1(K,J) = WBL1A_BL1(K,J) + VINBL1(K,J);
    WBL2_BL2(K,J) = WBL2A_BL2(K,J) + VINBL2(K,J);
  endfor;
endfor;

M = 0;
for K=1;3 do
  for J=1;10 do
    M = M + 1;
    WBL1_BL1_V(M) = WBL1_BL1(K,J);
```

```
    WBL2_BL2_V(M) = WBL2_BL2(K,J);
  endfor;
endfor;
```

```
inputs: (elementN,CLN,CDN,CL_Vold,CD_Vold);

outputs: (CL_V,CD_V);

parameters: NL;

integer elementN;
float CLN,CDN,CL_Vold(20),CD_Vold(20);
float CL_V(20),CD_V(20);
integer NL;

for N = 1:20 do
  CL_V(N) = CL_Vold(N);
  CD_V(N) = CD_Vold(N);
endfor;

if (elementN <= NL) then
  CL_V(elementN) = CLN;
  CD_V(elementN) = CDN;
else
  CL_V(10 + elementN - NL) = CLN;
  CD_V(10 + elementN - NL) = CDN;
endif;
```

Aug 1 2001 18:19:22 **WS_S** Page 1

```
inputs: (L_S_E_V,WSA_E,V0,KGE);
outputs: (WSA_S,WS_S);

float L_S_E_V(9),WSA_E(3),V0,KGE;
float WSA_S(3),WS_S(3);
float L_S_E(3,3);

M = 0;
for K=1:3 do
  for L=1:3 do
    M = M + 1;
    L_S_E(K,L) = L_S_E_V(M);
  endfor;
endfor;

for K=1:3 do
  WSA_S(K) = 0;
  for L=1:3 do
    WSA_S(K) = WSA_S(K) + L_S_E(K,L)*WSA_E(L);
  endfor;
endfor;

WS_S(1) = WSA_S(1);
WS_S(2) = WSA_S(2);
WS_S(3) = WSA_S(3) + V0*KGE;
```

D. Landing Gear Block Diagrams

Table of Contents

Block 4

- 1.1.2.3 Next Sliding Check 0
  - 1.1.2.3.1 Calculate Gear Orientation (1.1.2.2.1)
    - 1.1.2.3.1.1 Calculate PHISK (1.1.1.2.1)
      - 1.1.2.3.1.1.1 Calculate POSN_E (1.1.1.1)
        - 1.1.2.3.1.1.1.1 Calculate L_B_SKN (1.1.1.1.1)
    - 1.1.2.3.1.2 Calculate PHISK_D (1.1.2.2.1.2)
  - 1.1.2.3.2 Calculate L_V When Sticking
    - 1.1.2.3.2.1 Calculate RU_V_D (1.1.2.2.2)
  - 1.1.2.3.3 Calculate NF
    - 1.1.2.3.3.1 Calculate GJ and R
  - 1.1.2.3.4 Sliding Check
    - 1.1.2.3.4.1 Sliding Variable Init

Block 5

- 1.1.2.4 Sliding Calculations
  - 1.1.2.4.1 DEF_V_D Integrator
    - 1.1.2.4.1.1 Integratorx2
  - 1.1.2.4.2 Calculate NF (1.1.2.3.3)
    - 1.1.2.4.2.1 Calculate GJ and R (1.1.2.3.3.1)

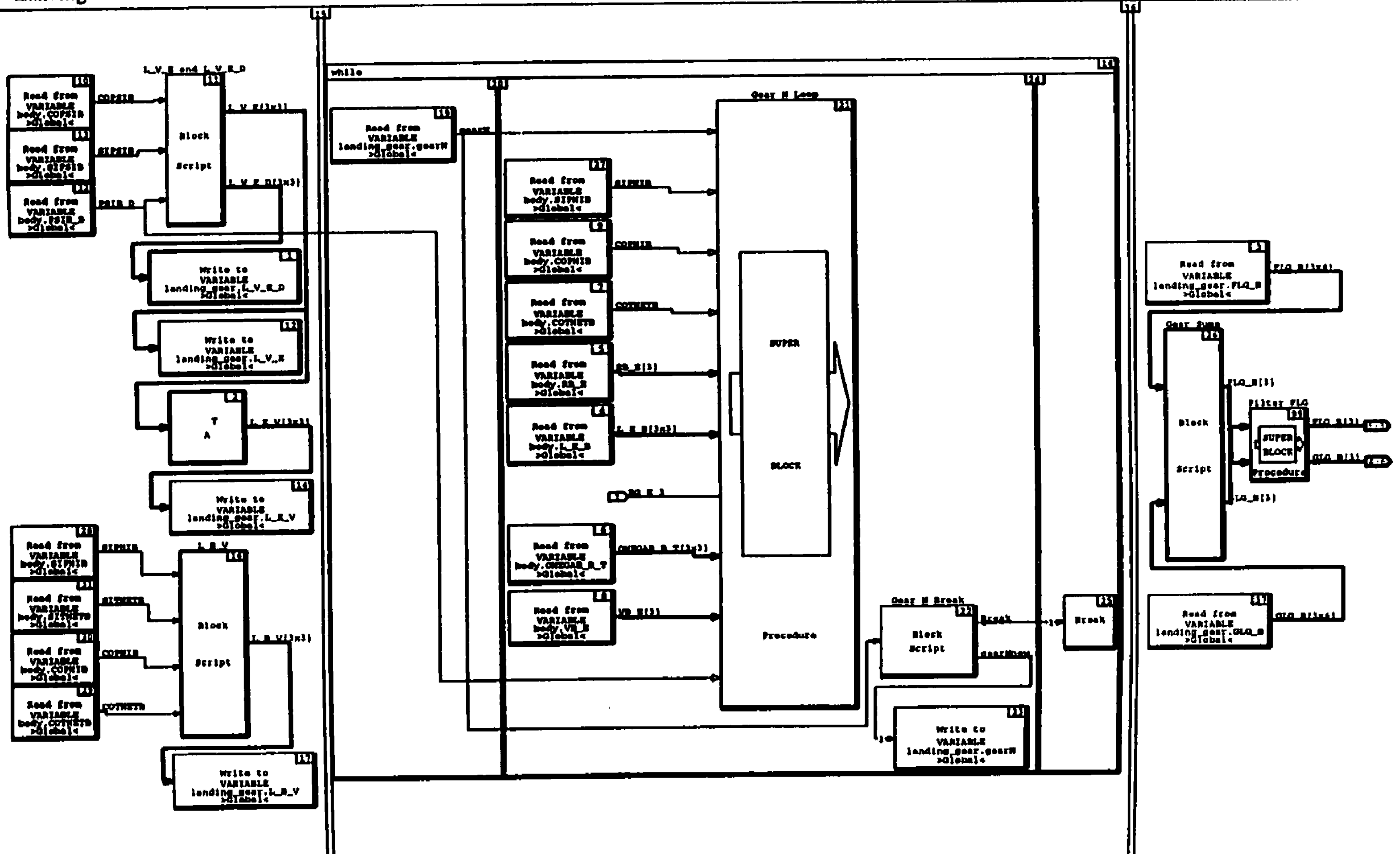
Block 6

- 1.1.3 Gear Forces and Moments
  - 1.1.3.1 TILDA
- 1.2 Filter FLG

1.

7-AUG-101

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Landing Gear	0.00556	0.	1	6	Parent

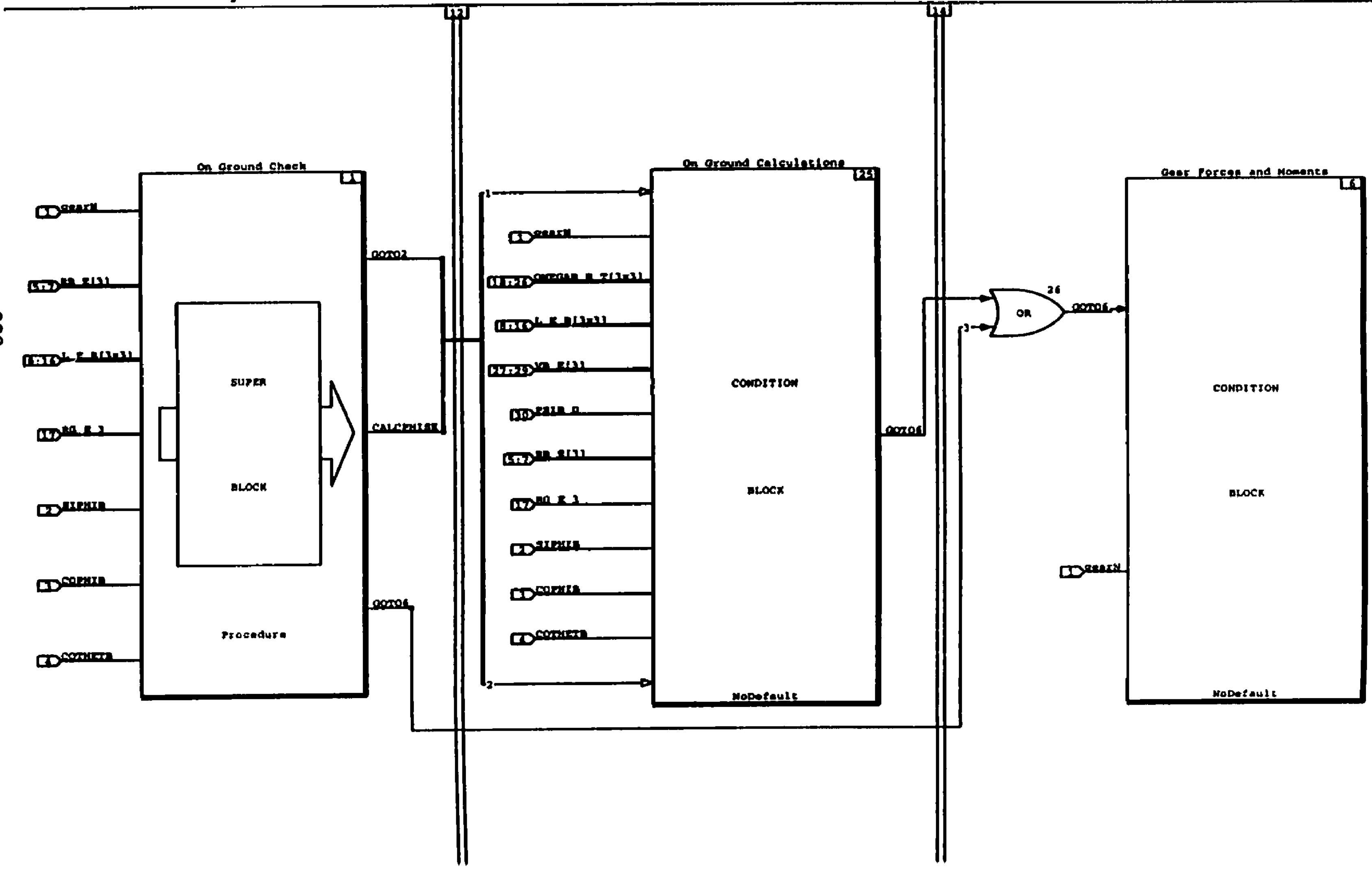


Read from VARIABLE body.COPSIB >Global<
Read from VARIABLE body.SIPSIB >Global<
Read from VARIABLE body.PSIB_D >Global<
L_V_E and L_V_E_D
Block Script
Write to VARIABLE landing_gear.L_V_E_D >Global<
Write to VARIABLE landing_gear.L_V_E >Global<
T A
Write to VARIABLE landing_gear.L_E_V >Global<
Read from VARIABLE body.SIPHIB >Global<
Read from VARIABLE body.SITHETB >Global<
Read from VARIABLE body.COPHIB >Global<
Read from VARIABLE body.COTHETB >Global<
L_B_V
Write to VARIABLE landing_gear.L_B_V >Global<
while
Read from VARIABLE landing_gear.gearN >Global<
Read from VARIABLE body.SIPHIB >Global<
Read from VARIABLE body.COPHIB >Global<
Read from VARIABLE body.COTHETB >Global<
Read from VARIABLE body.RB_E >Global<
Read from VARIABLE body.L_E_B >Global<
Read from VARIABLE body.OMEGAB_B_T >Global<
Read from VARIABLE body.VB_E >Global<
Gear N Loop
SUPER BLOCK
Procedure
Gear N Break
Block Script
Break
Write to VARIABLE landing_gear.gearN >Global<
Read from VARIABLE landing_gear.FLO_B >Global<
Gear Sums
Block Script
Filter FLG
SUPER BLOCK
Procedure
Read from VARIABLE landing_gear.GLO_B >Global<

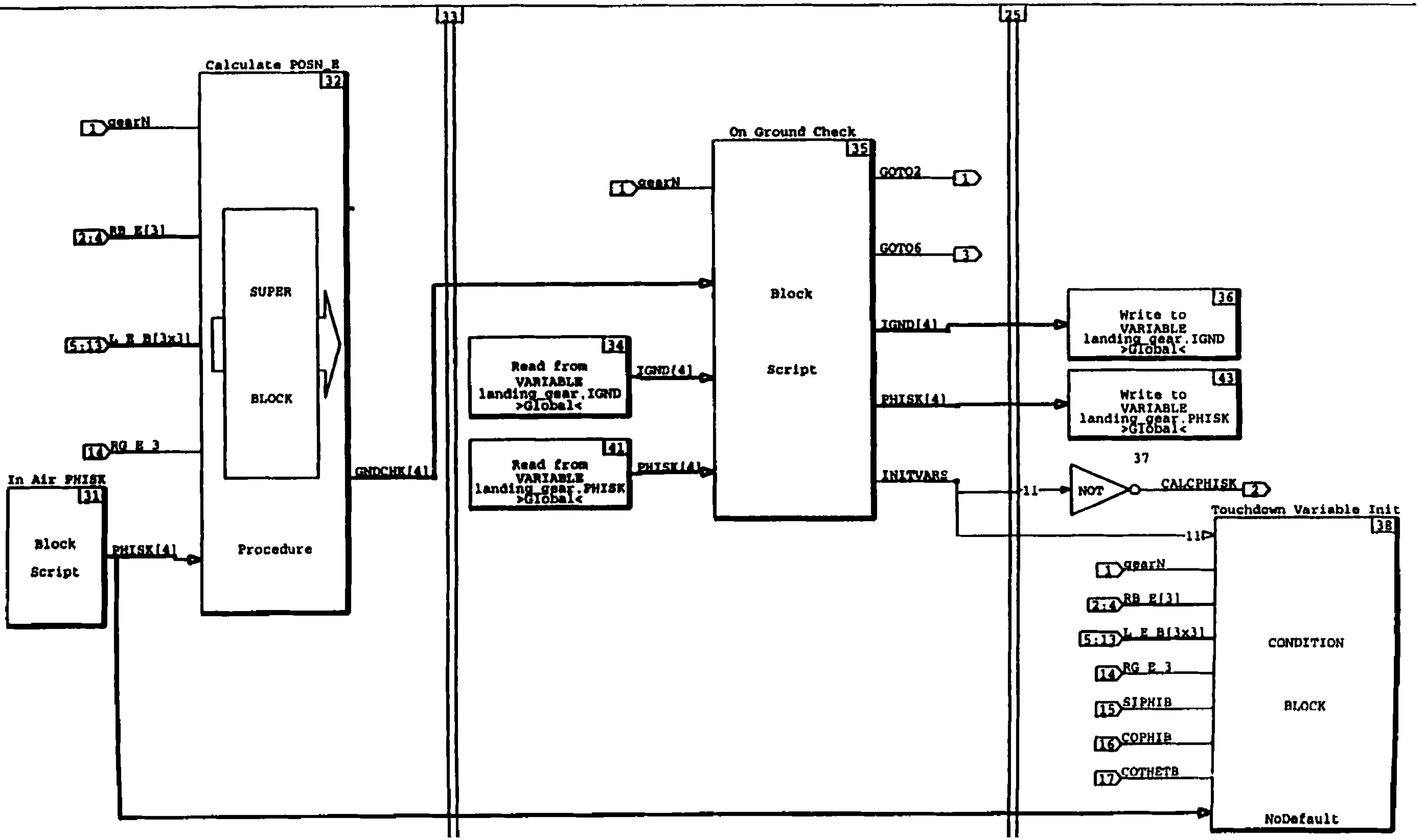
1.1

7-AUG-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Gear N Loop	Standard	30	0

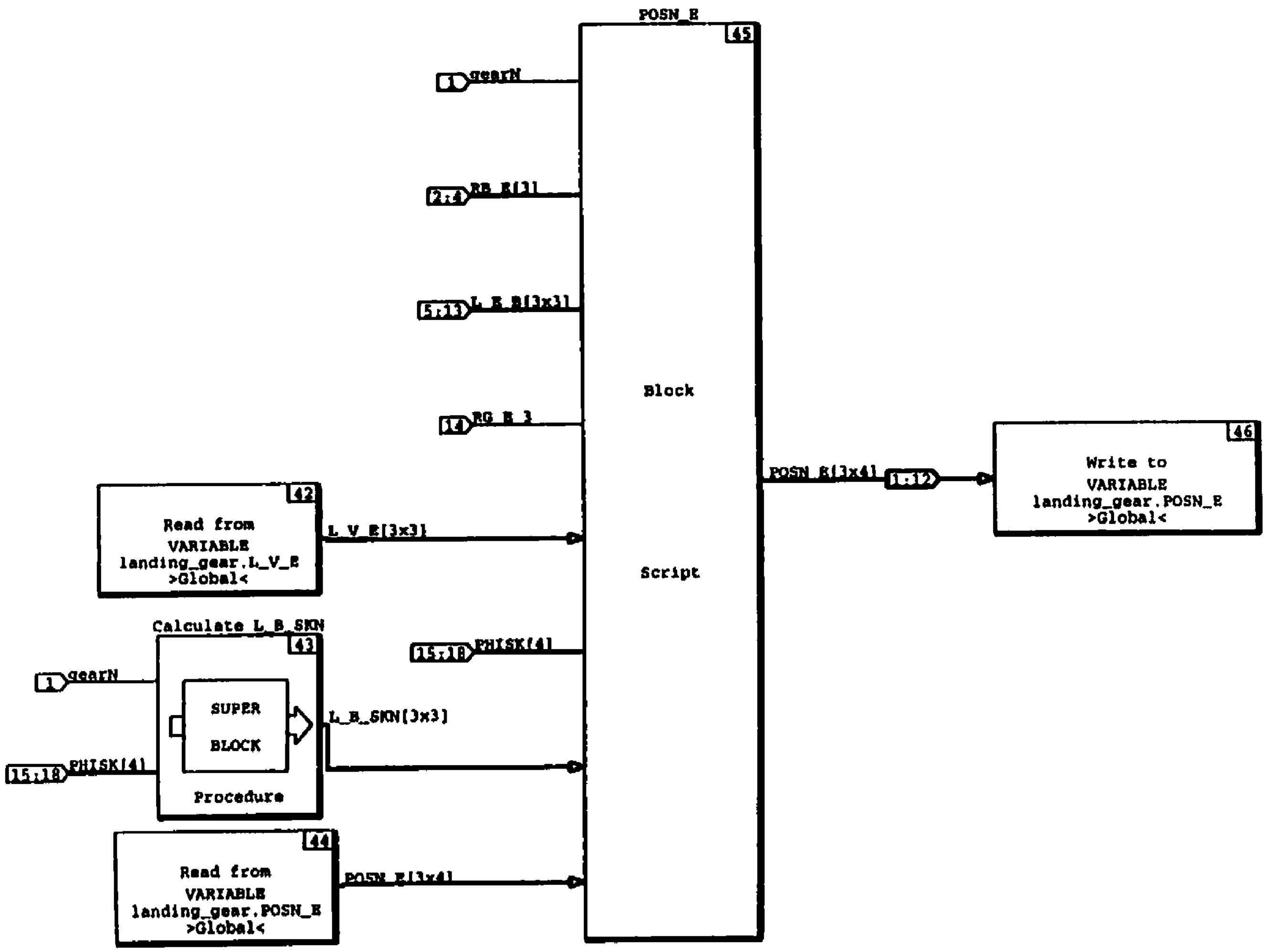


Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
On Ground Check	Standard	17	3



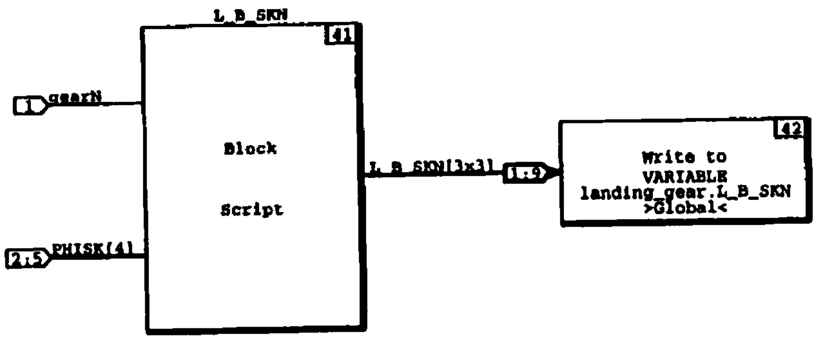
1.1.1.1

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate POSN_E	Standard	18	12

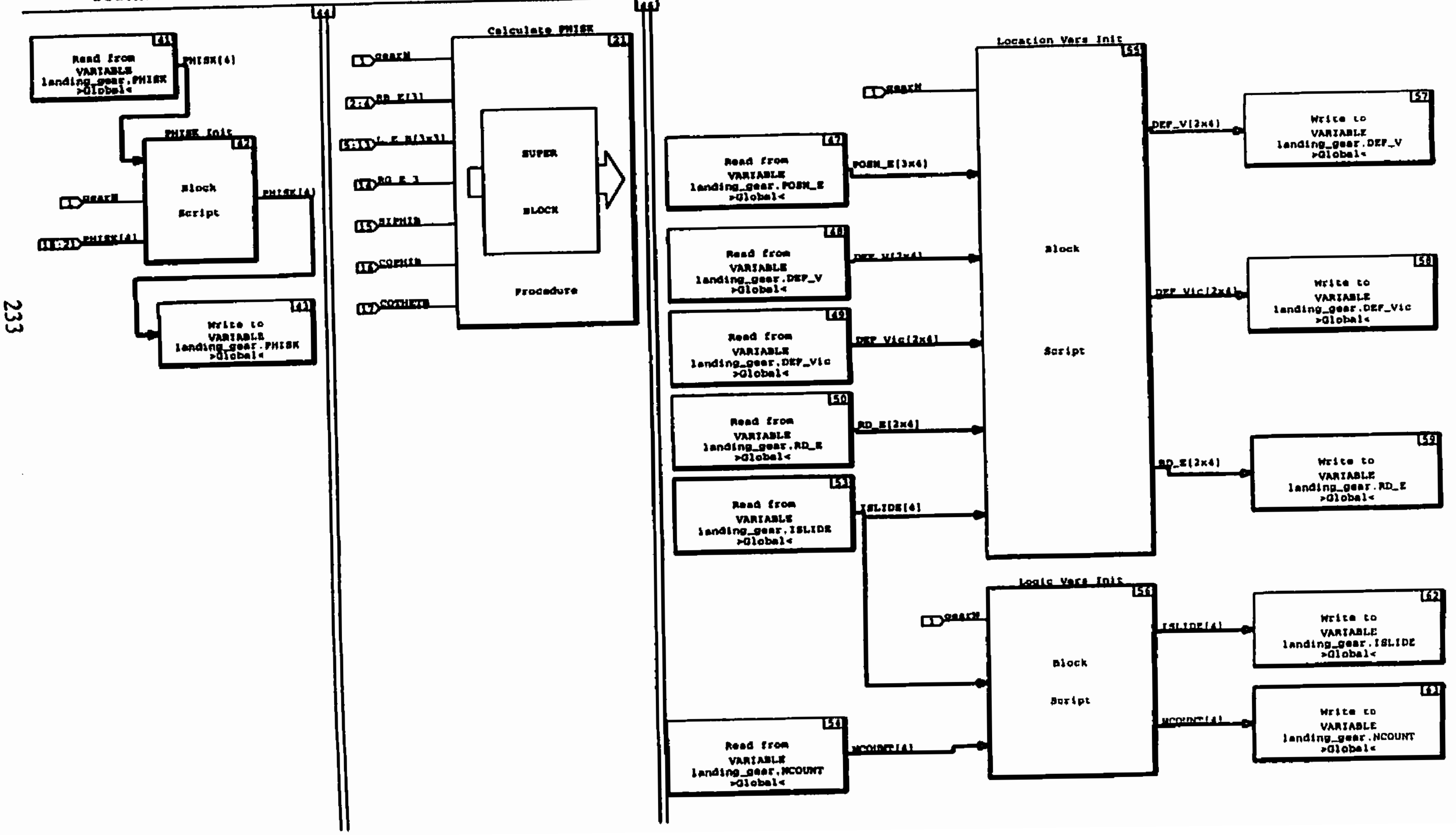


1.1.1.1.1 7-AUG-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate L_B_SKN	Standard	5	9



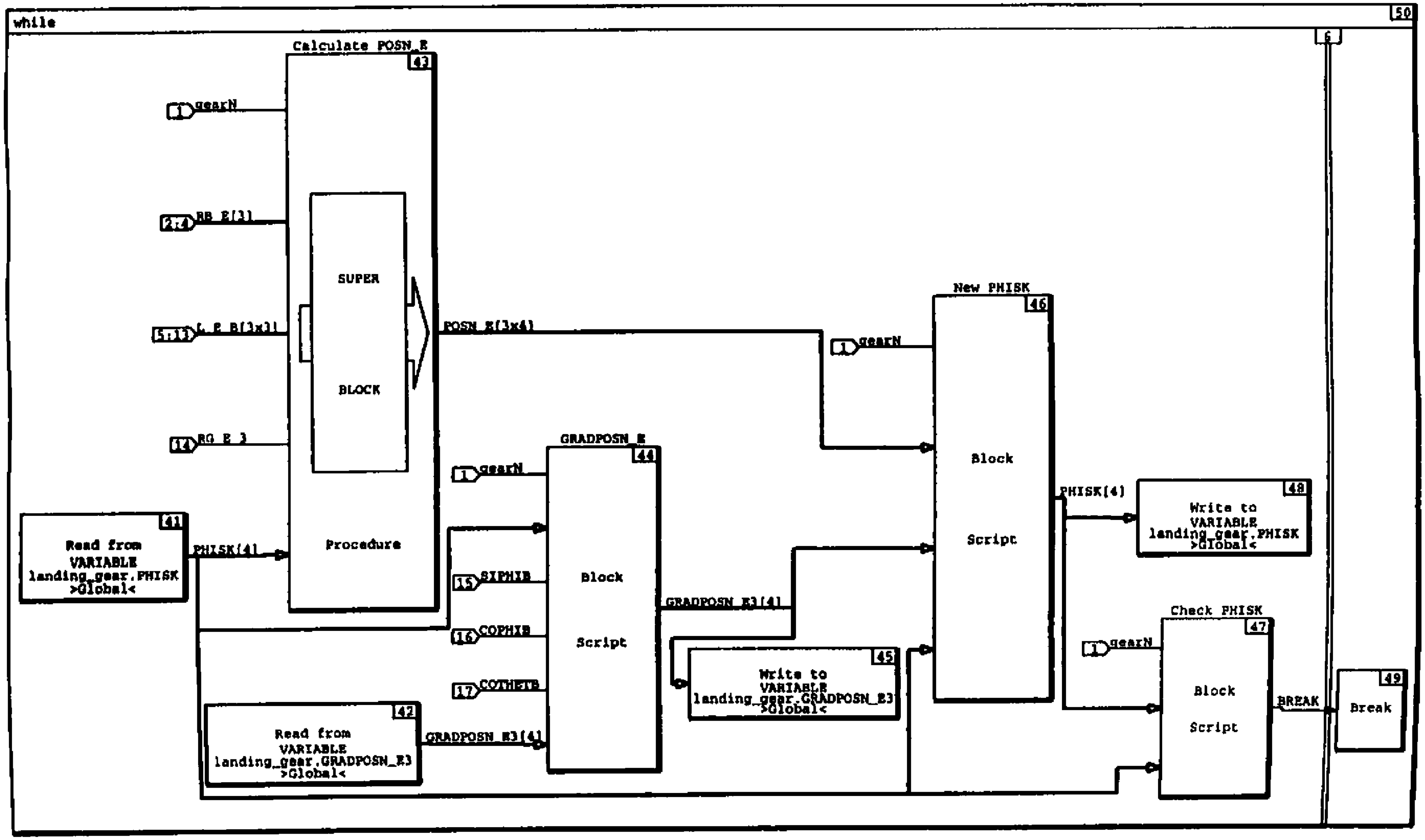
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Touchdown Variable Init	Standard	21	0


Read from VARIABLE landing_gear.PHISK >Global<
PHISK(4)
PHISK_Init
Block Script
gearN
PHISK(4)
Write to VARIABLE landing_gear.PHISK >Global<
Calculate PHISK
SUPER BLOCK
Procedure
gearN
RD_E(3)
SIPHIS
COPHIS
COTHETA
Location Vars Init
Block Script
Read from VARIABLE landing_gear.POSN_E >Global<
POSN_E(3x4)
Read from VARIABLE landing_gear.DEF_V >Global<
Read from VARIABLE landing_gear.DEF_Vic >Global<
DEF_Vic(2x4)
Read from VARIABLE landing_gear.RD_E >Global<
RD_E(2x4)
Read from VARIABLE landing_gear.ISLIDE >Global<
ISLIDE(4)
Read from VARIABLE landing_gear.NCOUNT >Global<
NCOUNT(4)
DEF_V(2x4)
Write to VARIABLE landing_gear.DEF_V >Global<
DEF_Vic(2x4)
Write to VARIABLE landing_gear.DEF_Vic >Global<
RD_E(2x4)
Write to VARIABLE landing_gear.RD_E >Global<
Logic Vars Init
Block Script
ISLIDE(4)
Write to VARIABLE landing_gear.ISLIDE >Global<
NCOUNT(4)
Write to VARIABLE landing_gear.NCOUNT >Global<

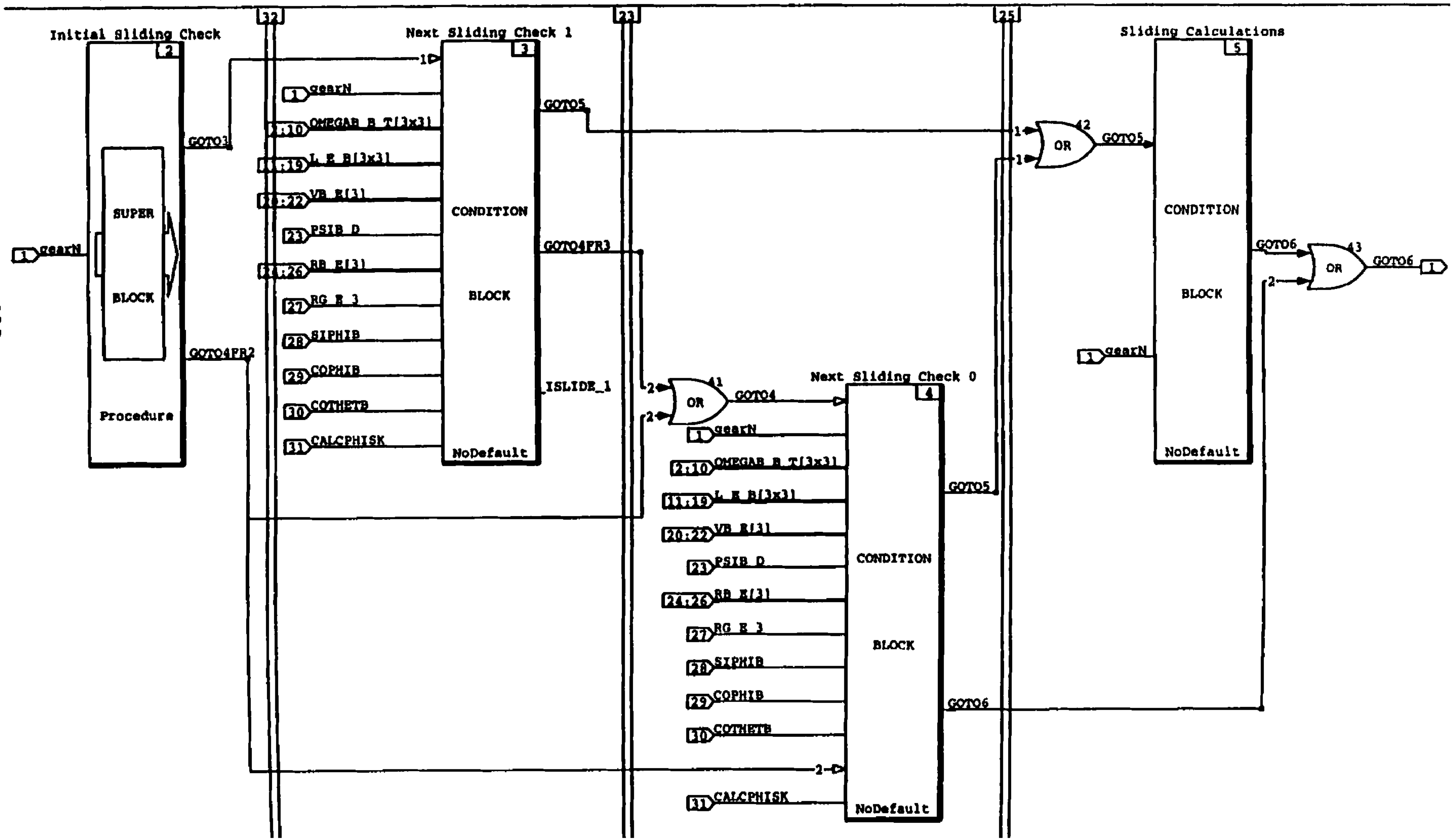
1.1.1.2.1

7-AUG-101

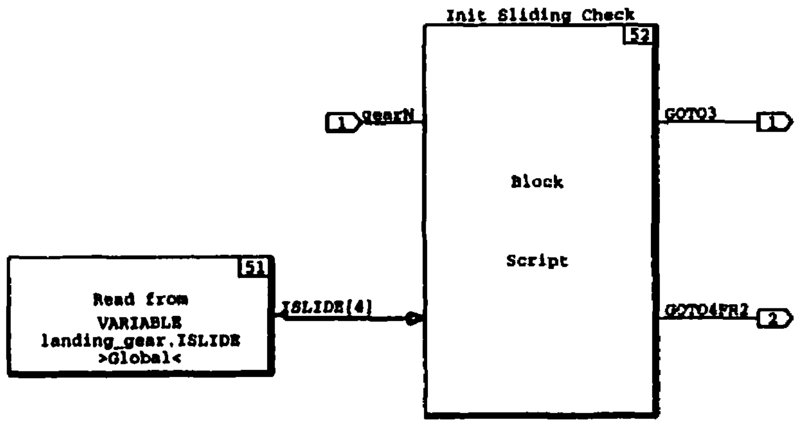
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate PHISK	Standard	17	0



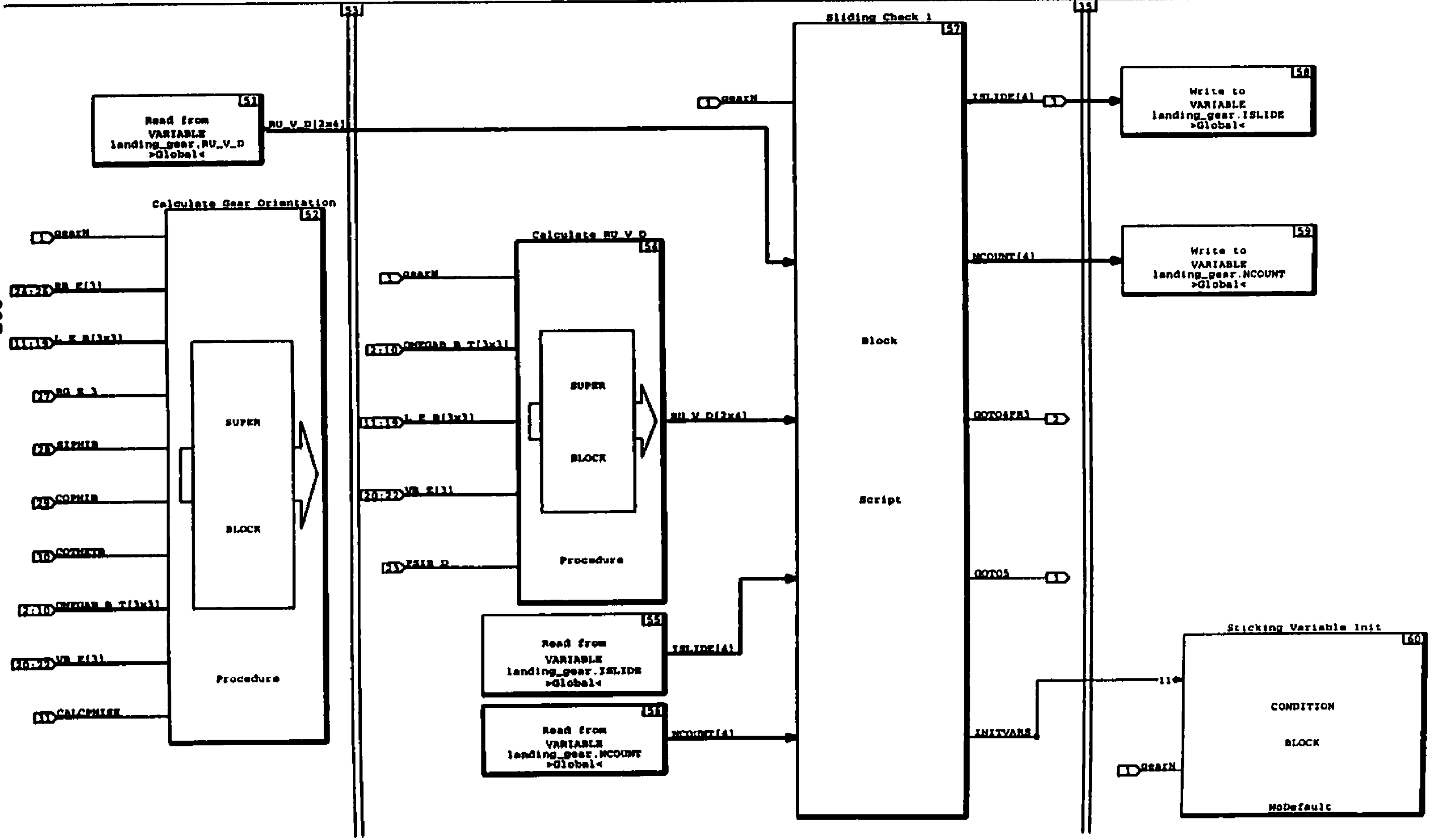
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
On Ground Calculations	Standard	31	1



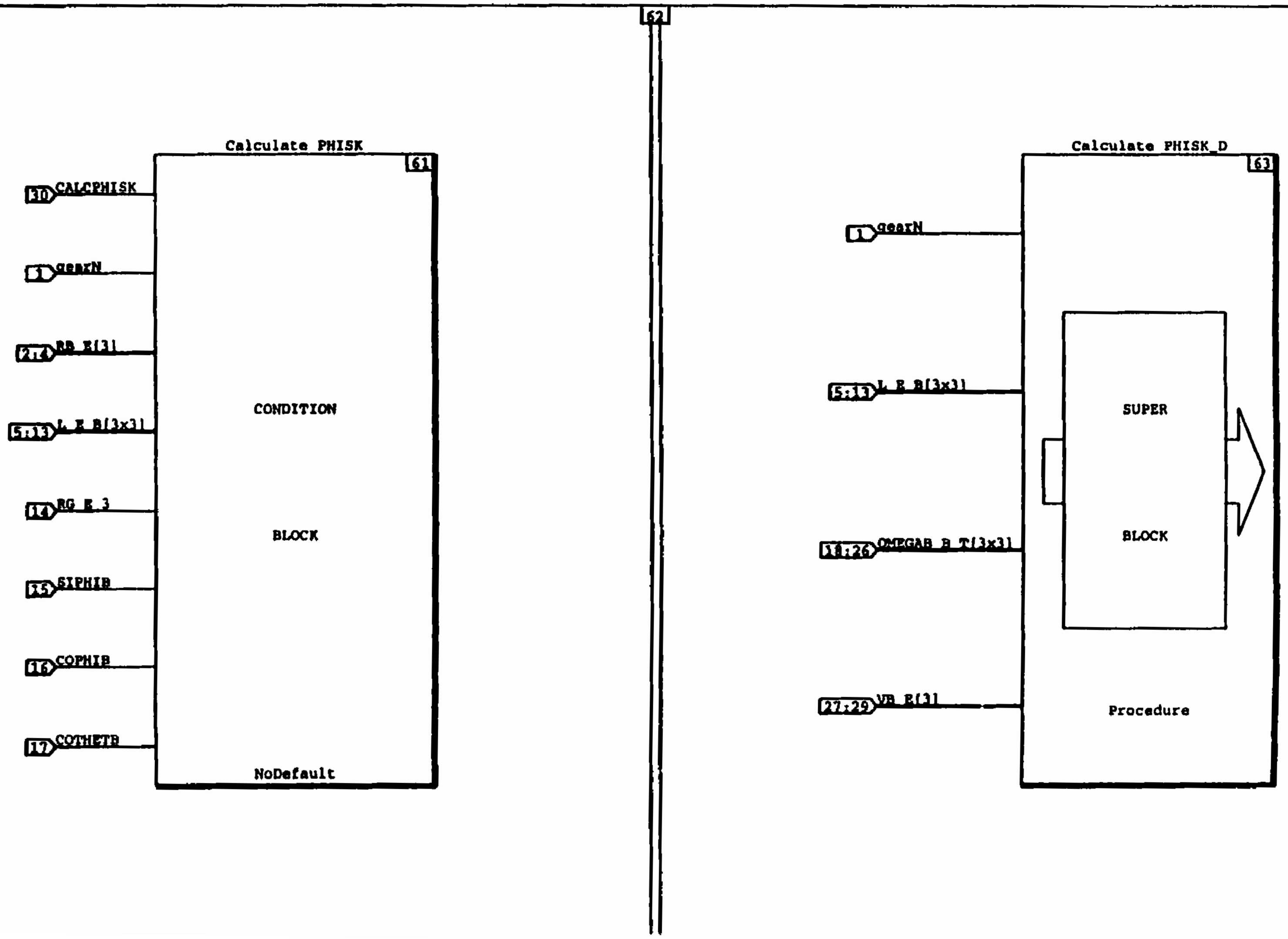
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Initial Sliding Check	Standard	1	2



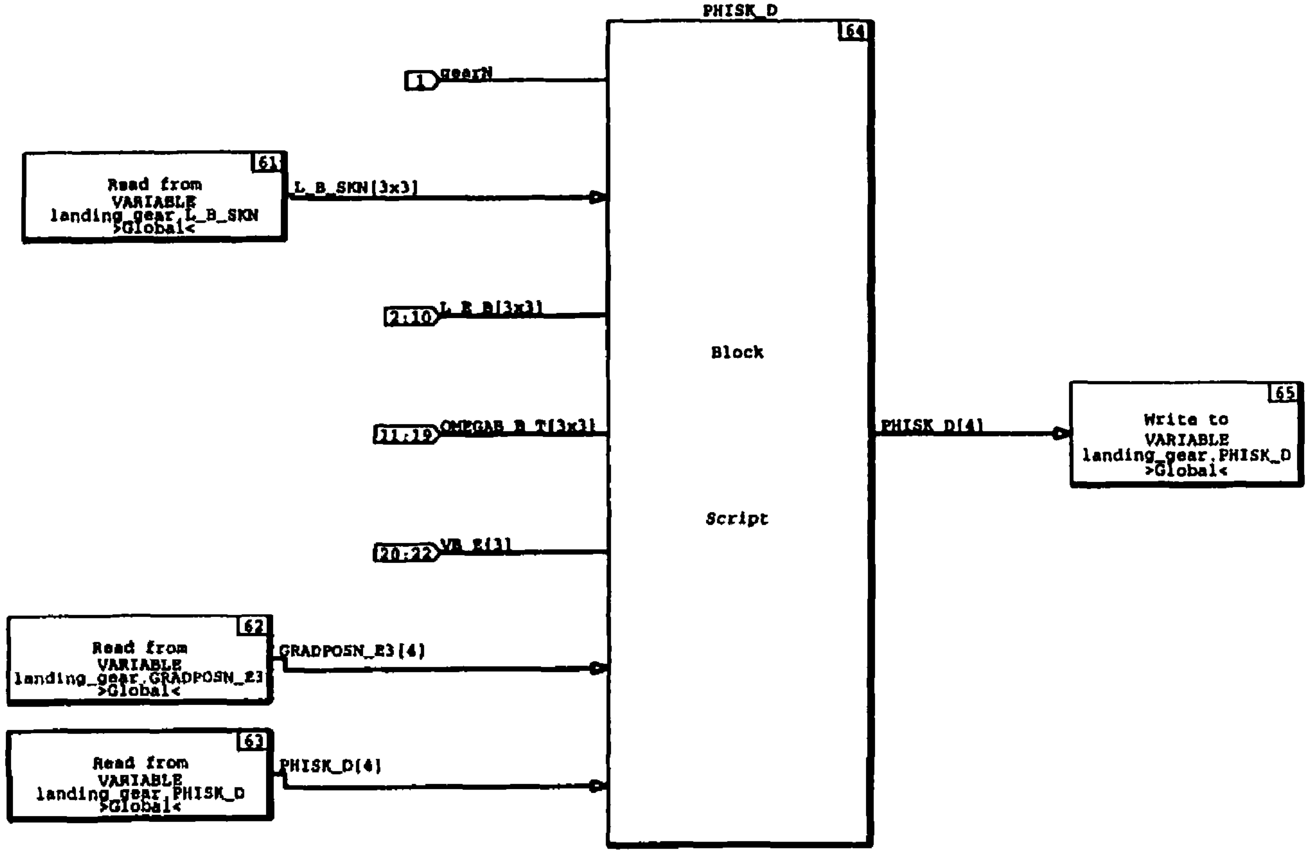
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Next Sliding Check 1	Standard	31	3



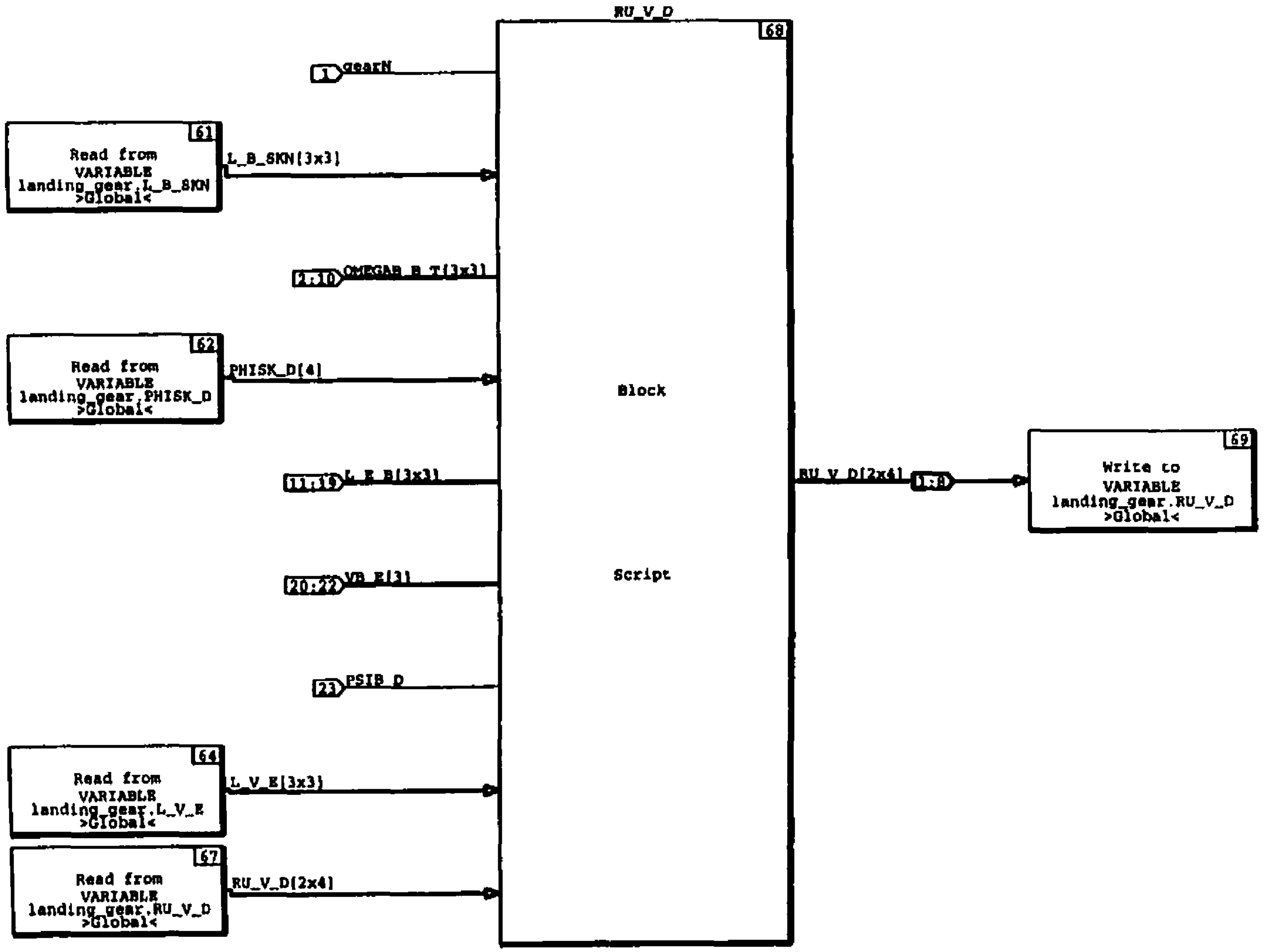
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate Gear Orientation	Standard	30	0



Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate PHISK_D	Standard	22	0


PHISK_D
64
1
gearN
61
Read from VARIABLE landing_gear.L_B_SKN >Global<
L_B_SKN[3x3]
2:10
L_B_B[3x3]
Block
Script
11:19
OMEGAB_B_T[3x3]
20:22
VB_E[3]
62
Read from VARIABLE landing_gear.GRADPOSN_E3 >Global<
GRADPOSN_E3[4]
63
Read from VARIABLE landing_gear.PHISK_D >Global<
PHISK_D[4]
PHISK_D[4]
65
Write to VARIABLE landing_gear.PHISK_D >Global<

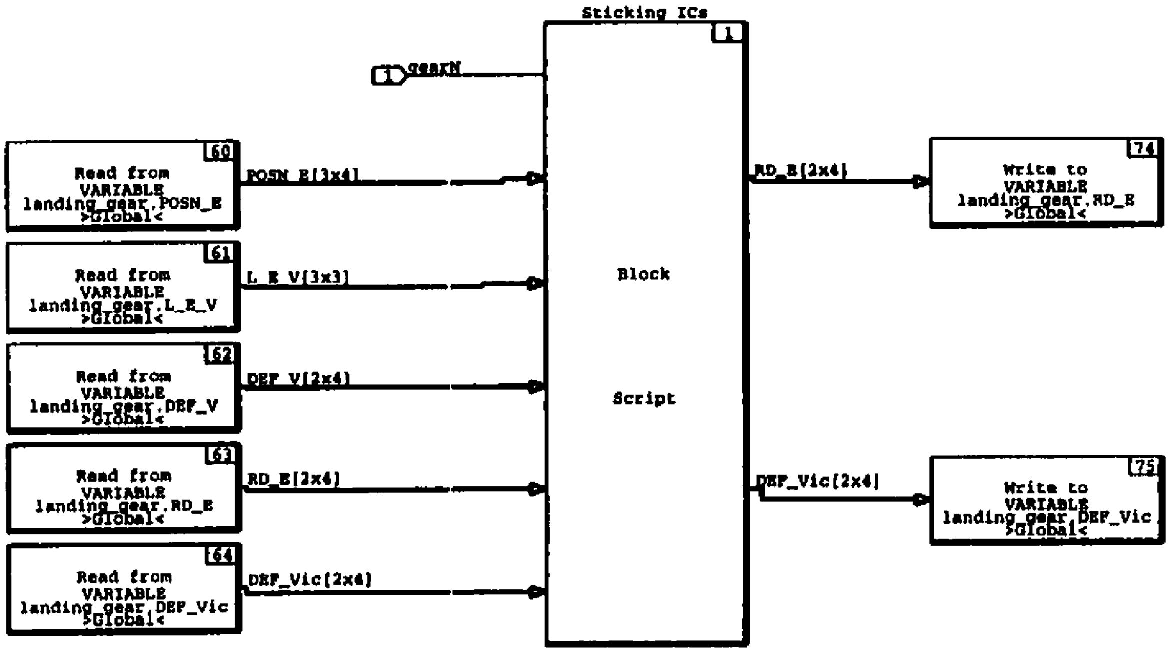
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate RU_V_D	Standard	23	8


RU_V_D
68
1
gearN
61
Read from VARIABLE landing_gear.L_B_SKN >Global<
L_B_SKN[3x3]
2:10
OMEGAB_B_T[3x3]
62
Read from VARIABLE landing_gear.PHISK_D >Global<
PHISK_D[4]
Block
11:19
L_E_B[3x3]
Script
20:22
VB_E[3]
23
PSIB_D
64
Read from VARIABLE landing_gear.L_V_E >Global<
L_V_E[3x3]
67
Read from VARIABLE landing_gear.RU_V_D >Global<
RU_V_D[2x4]
RU_V_D[2x4]
1:8
69
Write to VARIABLE landing_gear.RU_V_D >Global<

1.1.2.2.3

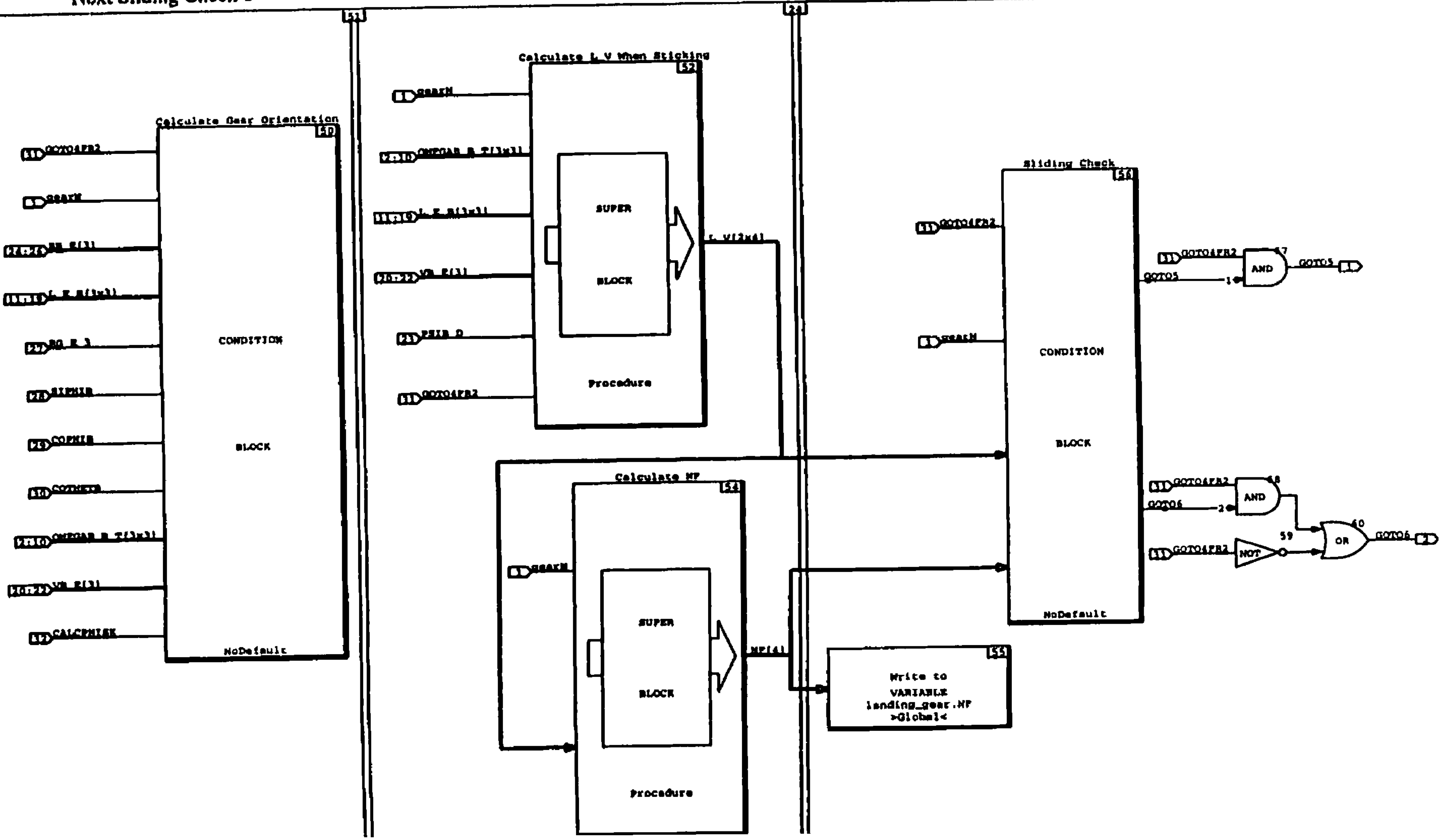
7-AUG-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Sticking Variable Init	Standard	1	0



Sticking ICs
1
1
gearN
60
Read from VARIABLE landing_gear.POSN_E >Global<
POSN_E[3x4]
61
Read from VARIABLE landing_gear.L_E_V >Global<
L_E_V[3x3]
62
Read from VARIABLE landing_gear.DEF_V >Global<
DEF_V[2x4]
63
Read from VARIABLE landing_gear.RD_E >Global<
RD_E[2x4]
64
Read from VARIABLE landing_gear.DEF_Vic >Global<
DEF_Vic[2x4]
Block
Script
RD_E[2x4]
74
Write to VARIABLE landing_gear.RD_E >Global<
DEF_Vic[2x4]
75
Write to VARIABLE landing_gear.DEF_Vic >Global<

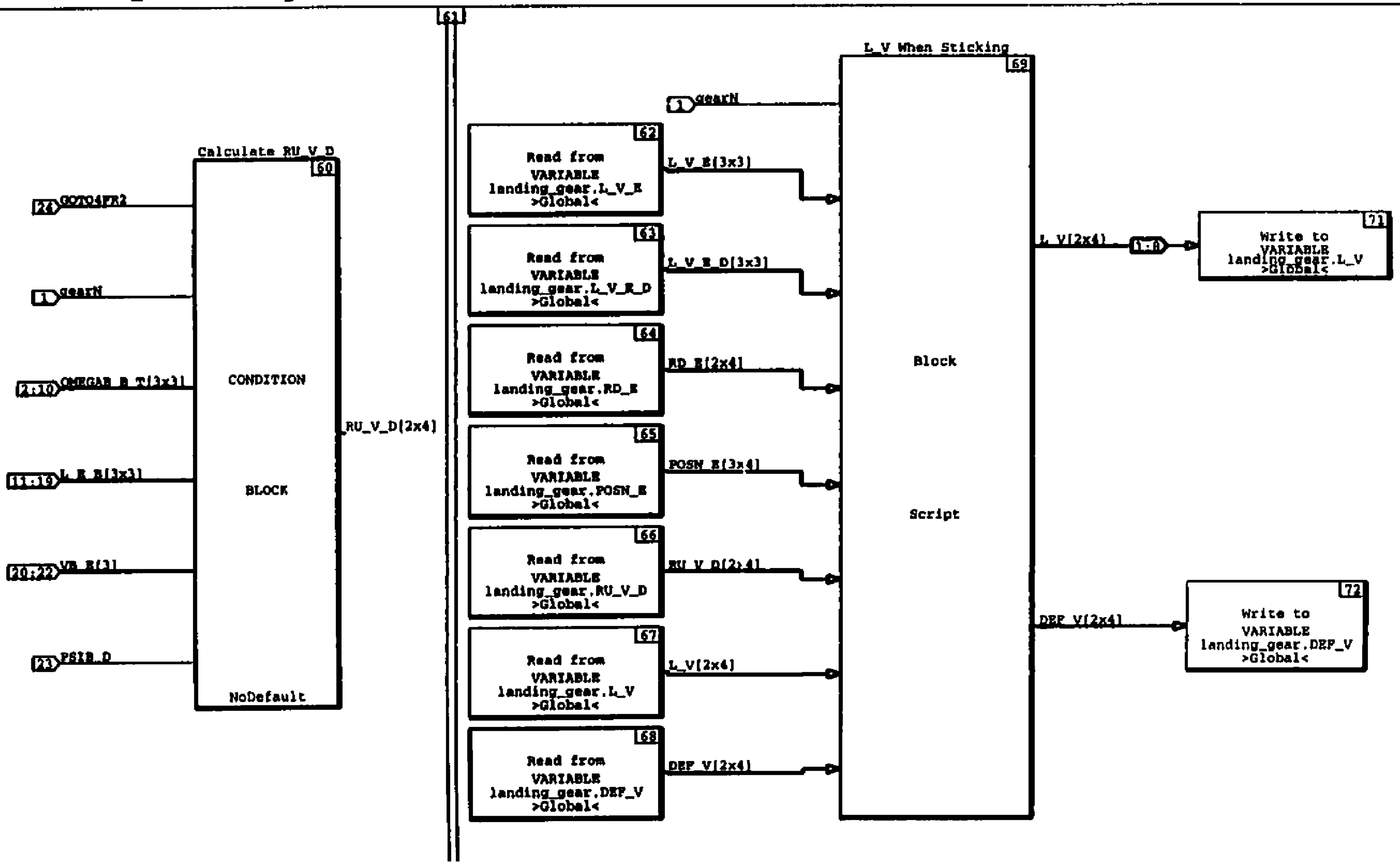
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Next Sliding Check 0	Standard	32	2



1.1.2.3.2

7-AUG-101

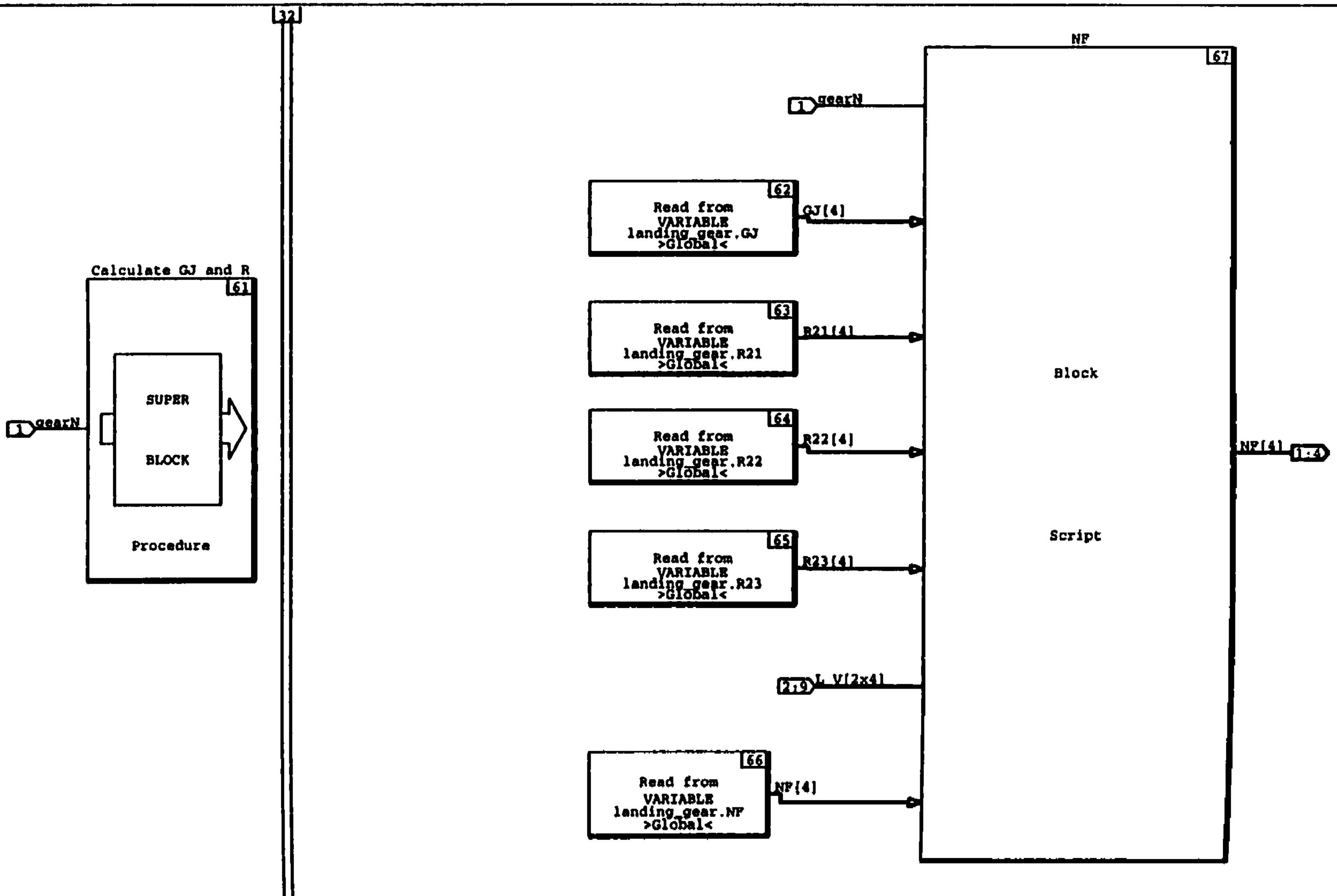
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate L_V When Sticking	Standard	24	8


61
Calculate RU_V_D
60
24 GOTO4FR2
1 gearN
2:10 OMEGAB_B_T[3x3]
11:19 L_E_B[3x3]
20:22 VB_E[3]
23 PSIB_D
CONDITION
BLOCK
NoDefault
RU_V_D[2x4]
62 Read from VARIABLE landing_gear.L_V_E >Global<
63 Read from VARIABLE landing_gear.L_V_E_D >Global<
64 Read from VARIABLE landing_gear.RD_E >Global<
65 Read from VARIABLE landing_gear.POSN_E >Global<
66 Read from VARIABLE landing_gear.RU_V_D >Global<
67 Read from VARIABLE landing_gear.L_V >Global<
68 Read from VARIABLE landing_gear.DEF_V >Global<
1 gearN
L_V_E[3x3]
L_V_E_D[3x3]
RD_E[2x4]
POSN_E[3x4]
RU_V_D[2x4]
L_V[2x4]
DEF_V[2x4]
L_V When Sticking
69
Block
Script
L_V[2x4]
1:8
71 Write to VARIABLE landing_gear.L_V >Global<
DEF_V[2x4]
72 Write to VARIABLE landing_gear.DEF_V >Global<

1.1.2.3.3

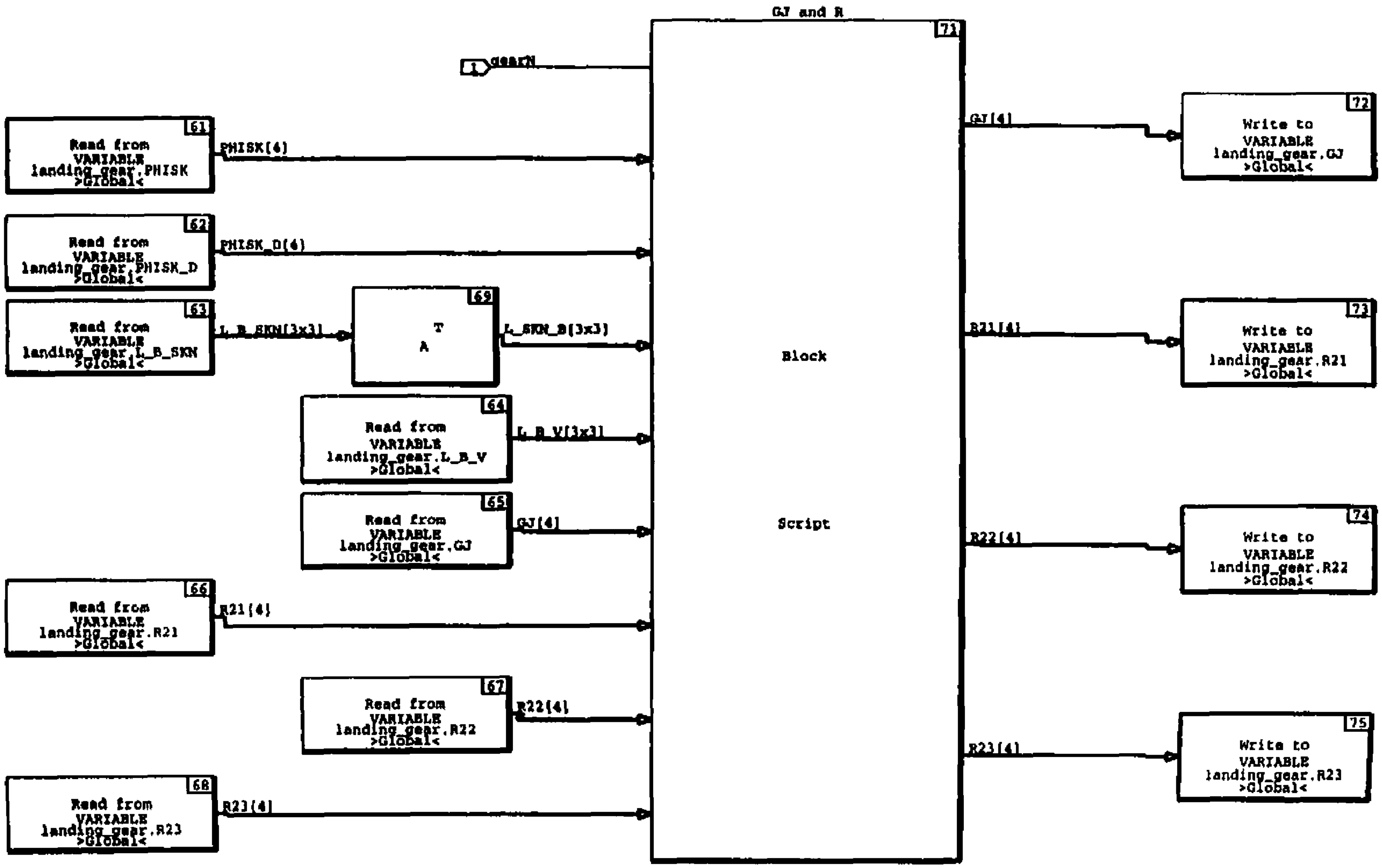
7-AUG-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate NF	Standard	9	4


32
Calculate GJ and R
61
SUPER
BLOCK
Procedure
1 gearN
62
Read from VARIABLE landing_gear.GJ >Global<
GJ[4]
63
Read from VARIABLE landing_gear.R21 >Global<
R21[4]
64
Read from VARIABLE landing_gear.R22 >Global<
R22[4]
65
Read from VARIABLE landing_gear.R23 >Global<
R23[4]
2:9 L V[2x4]
66
Read from VARIABLE landing_gear.NF >Global<
NF[4]
NF
67
Block
Script
NF[4]
1:4

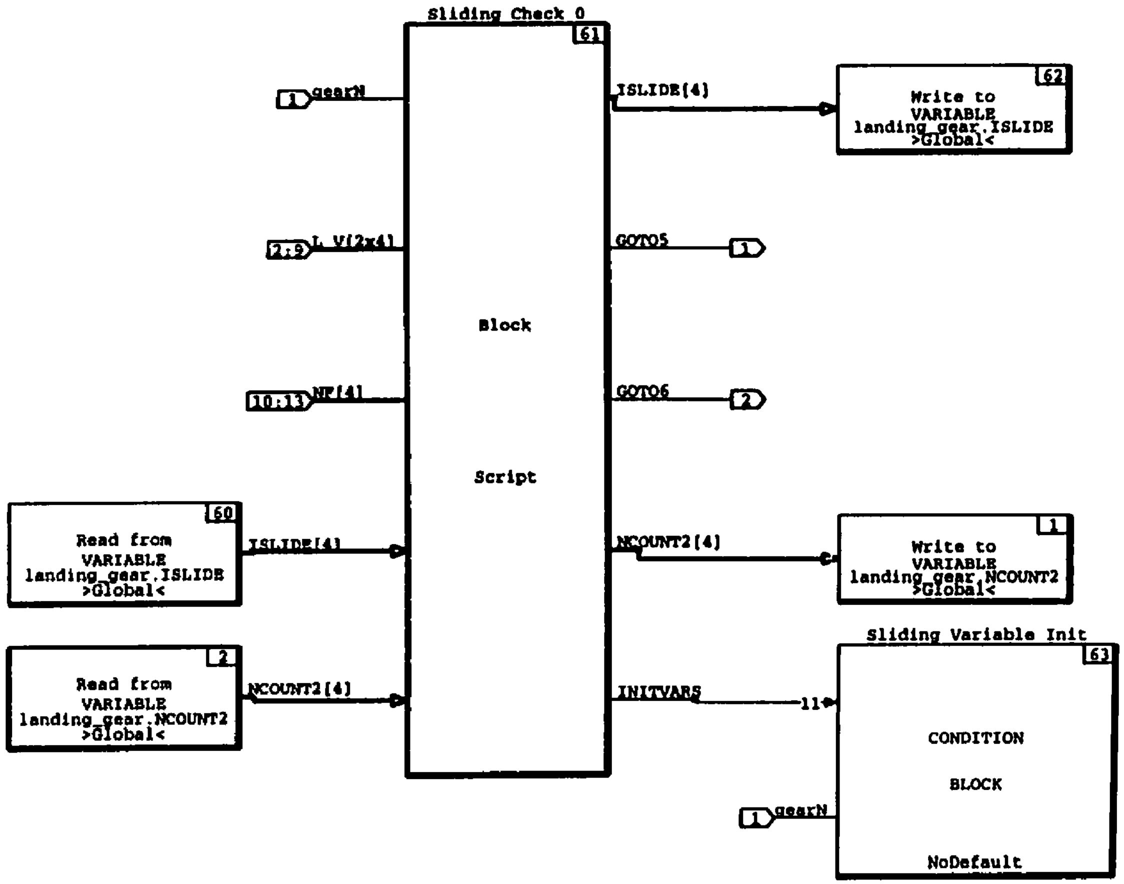
1.1.2.3.3.1

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Calculate GJ and R	Standard	1	0


GJ and R
71
1 gearN
Block
Script
61 Read from VARIABLE landing_gear.PHISK >Global<
PHISK(4)
62 Read from VARIABLE landing_gear.PHISK_D >Global<
PHISK_D(4)
63 Read from VARIABLE landing_gear.L_B_SKN >Global<
L_B_SKN(3x3)
69 A T
L_SKN_B(3x3)
64 Read from VARIABLE landing_gear.L_B_V >Global<
L_B_V(3x3)
65 Read from VARIABLE landing_gear.GJ >Global<
GJ(4)
66 Read from VARIABLE landing_gear.R21 >Global<
R21(4)
67 Read from VARIABLE landing_gear.R22 >Global<
R22(4)
68 Read from VARIABLE landing_gear.R23 >Global<
R23(4)
GJ(4)
72 Write to VARIABLE landing_gear.GJ >Global<
R21(4)
73 Write to VARIABLE landing_gear.R21 >Global<
R22(4)
74 Write to VARIABLE landing_gear.R22 >Global<
R23(4)
75 Write to VARIABLE landing_gear.R23 >Global<

1.1.2.3.4

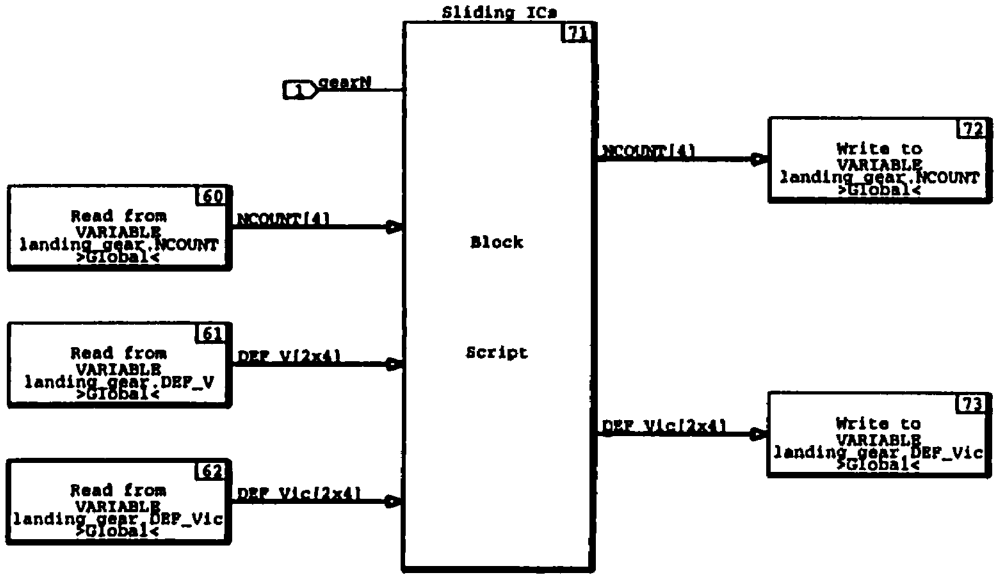
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Sliding Check	Standard	13	2


Sliding Check 0
61
Block
Script
1
gearN
2:9
L V(2x4)
10:13
NF(4)
60
Read from VARIABLE landing_gear.ISLIDE >Global<
ISLIDE(4)
2
Read from VARIABLE landing_gear.NCOUNT2 >Global<
NCOUNT2(4)
ISLIDE(4)
62
Write to VARIABLE landing_gear.ISLIDE >Global<
GOTO5
1
GOTO6
2
NCOUNT2(4)
1
Write to VARIABLE landing_gear.NCOUNT2 >Global<
INITVARS
11
Sliding Variable Init
63
CONDITION
BLOCK
NoDefault
1
gearN

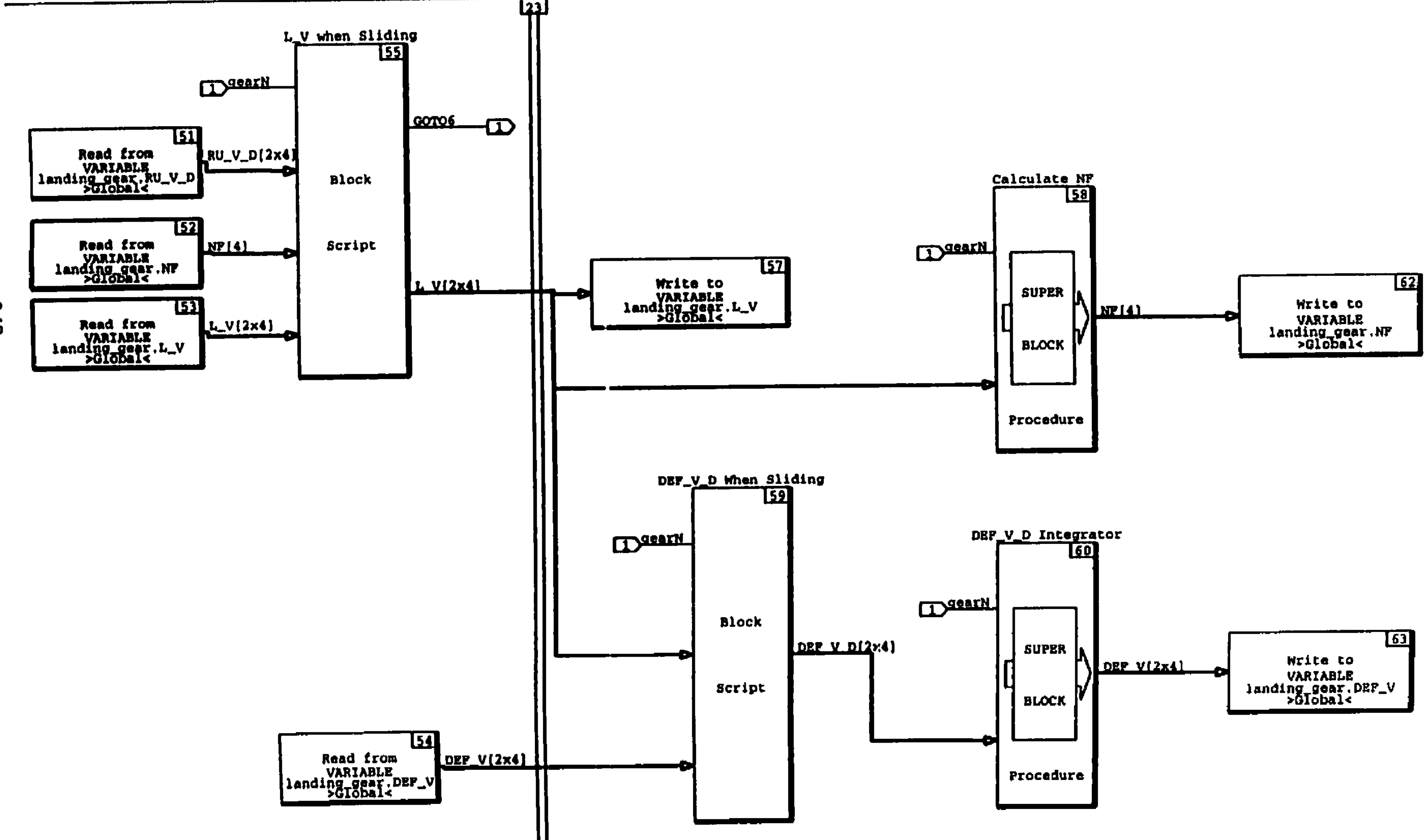
1.1.2.3.4.1

7-AUG-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Sliding Variable Init	Standard	1	0

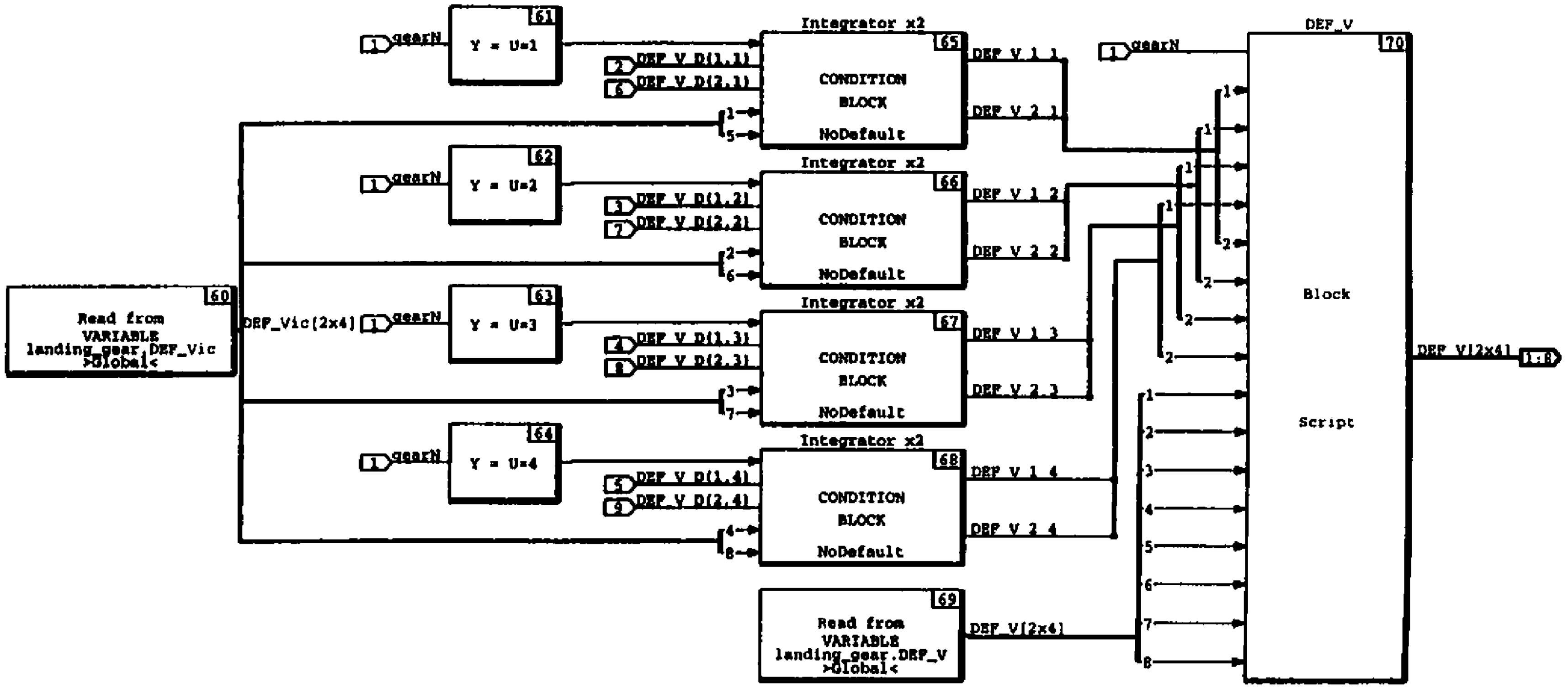

Sliding ICs
71
1
gearN
Block
Script
60
Read from VARIABLE landing_gear.NCOUNT >Global<
NCOUNT[4]
61
Read from VARIABLE landing_gear.DEF_V >Global<
DEF_V[2x4]
62
Read from VARIABLE landing_gear.DEF_Vic >Global<
DEF_Vic[2x4]
NCOUNT[4]
72
Write to VARIABLE landing_gear.NCOUNT >Global<
DEF_Vic[2x4]
73
Write to VARIABLE landing_gear.DEF_Vic >Global<

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Sliding Calculations	Standard	1	1



23
L_V when Sliding
55
Block
Script
gearN
GOTO6
1
51
Read from VARIABLE landing_gear.RU_V_D >Global<
RU_V_D(2x4)
52
Read from VARIABLE landing_gear.NF >Global<
NF(4)
53
Read from VARIABLE landing_gear.L_V >Global<
L_V(2x4)
L_V(2x4)
57
Write to VARIABLE landing_gear.L_V >Global<
Calculate NF
58
SUPER BLOCK
Procedure
NF(4)
62
Write to VARIABLE landing_gear.NF >Global<
DEF_V_D When Sliding
59
Block
Script
DEF_V_D(2x4)
54
Read from VARIABLE landing_gear.DEF_V >Global<
DEF_V(2x4)
DEF_V_D Integrator
60
SUPER BLOCK
Procedure
DEF_V(2x4)
63
Write to VARIABLE landing_gear.DEF_V >Global<

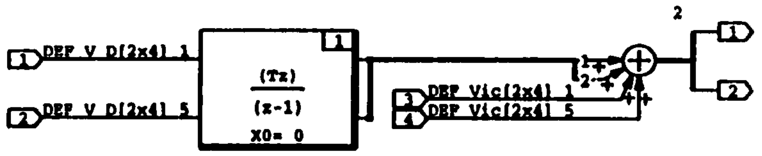
Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
DEF_V_D Integrator	Standard	9	8



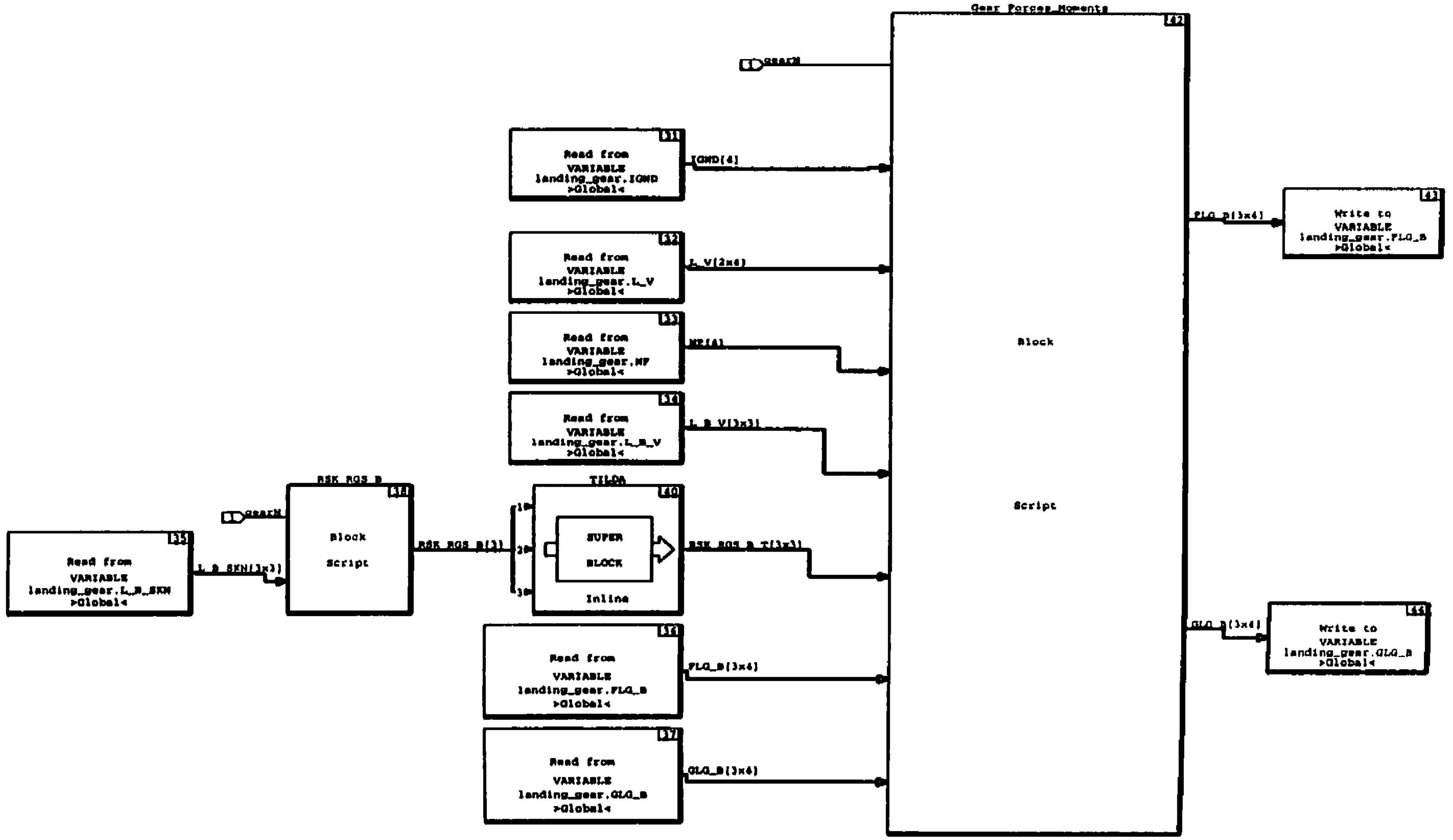
1.1.2.4.1.1

7-AUG-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Integrator x2	Standard	4	2



Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Gear Forces and Moments	Standard	1	0

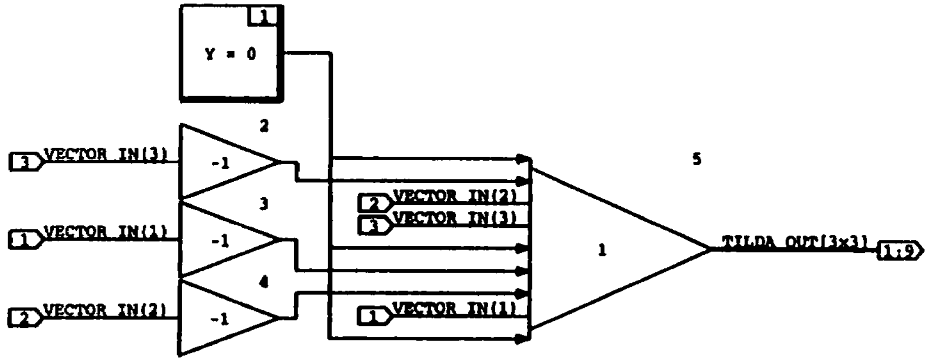


Gear_Forces_Moments
42
gearN
Read from VARIABLE landing_gear.IGND >Global<
31
IGND(4)
Read from VARIABLE landing_gear.L_V >Global<
32
L_V(2x4)
Read from VARIABLE landing_gear.NF >Global<
33
NF(4)
Read from VARIABLE landing_gear.L_B_V >Global<
34
L_B_V(3x3)
Block
Script
Write to VARIABLE landing_gear.FLG_B >Global<
43
FLG_B(3x4)
Write to VARIABLE landing_gear.GLG_B >Global<
44
GLG_B(3x4)
RSK_RGS_B
38
gearN
Block Script
Read from VARIABLE landing_gear.L_B_SKN >Global<
35
L_B_SKN(3x3)
RSK_RGS_B(3)
TILDA
40
SUPER BLOCK
Inline
RSK_RGS_B_T(3x3)
Read from VARIABLE landing_gear.FLG_B >Global<
36
FLG_B(3x4)
Read from VARIABLE landing_gear.GLG_B >Global<
37
GLG_B(3x4)

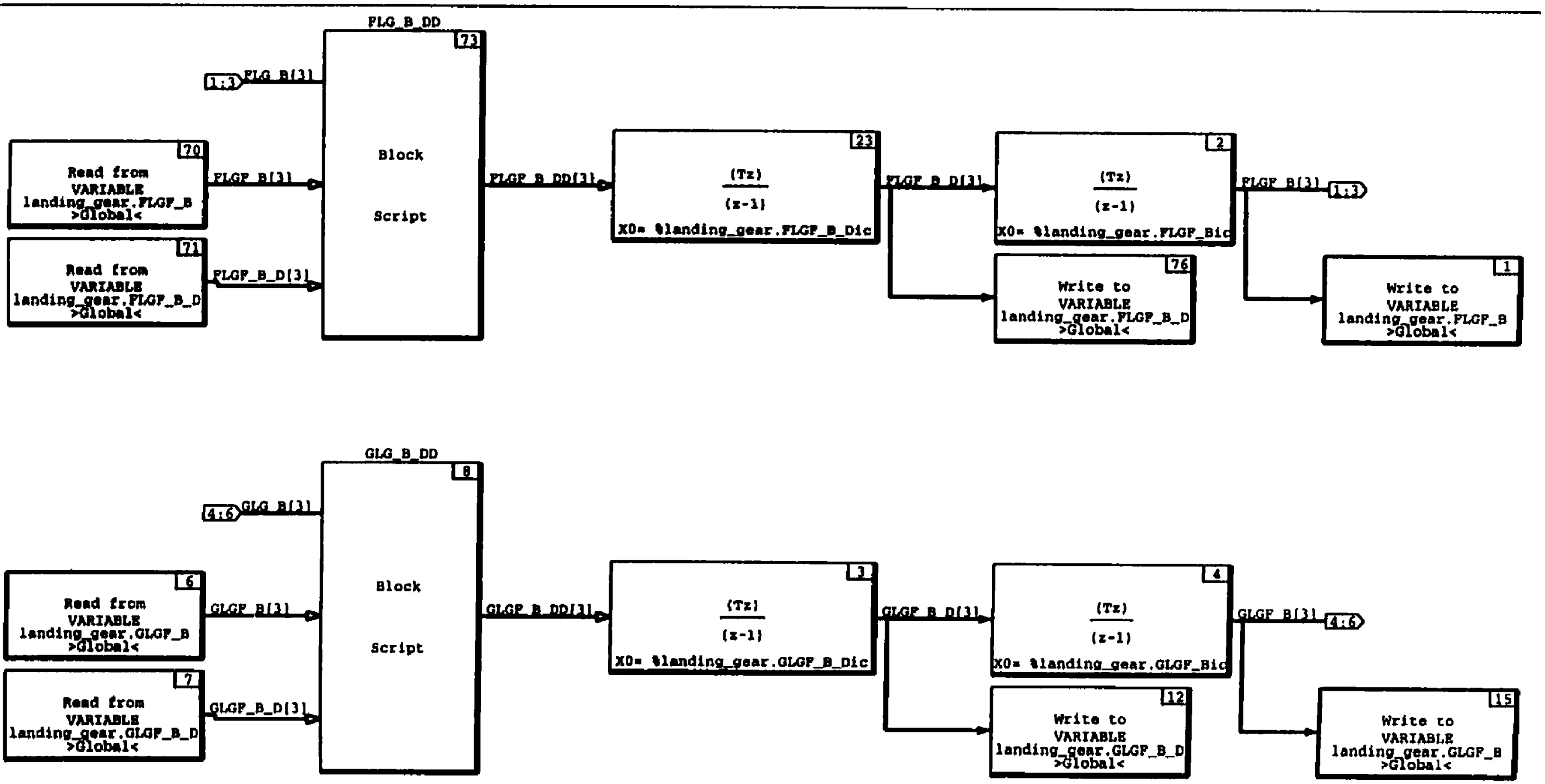
1.1.3.1

7-AUG-101

Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
TILDA	Inline	3	9



Discrete Procedure SuperBlock	Procedure Class	Inputs	Outputs
Filter FLG	Standard	6	6


FLG_B_DD
73
1:3
FLG_B[3]
Block
Script
70
Read from VARIABLE landing_gear.FLGF_B >Global<
FLGF_B[3]
71
Read from VARIABLE landing_gear.FLGF_B_D >Global<
FLGF_B_D[3]
FLGF_B_DD[3]
23
(Tz) / (z-1)
X0= %landing_gear.FLGF_B_Dic
FLGF_B_D[3]
2
(Tz) / (z-1)
X0= %landing_gear.FLGF_Bic
76
Write to VARIABLE landing_gear.FLGF_B_D >Global<
FLGF_B[3]
1:3
1
Write to VARIABLE landing_gear.FLGF_B >Global<
GLG_B_DD
8
4:6
GLG_B[3]
Block
Script
6
Read from VARIABLE landing_gear.GLGF_B >Global<
GLGF_B[3]
7
Read from VARIABLE landing_gear.GLGF_B_D >Global<
GLGF_B_D[3]
GLGF_B_DD[3]
3
(Tz) / (z-1)
X0= %landing_gear.GLGF_B_Dic
GLGF_B_D[3]
4
(Tz) / (z-1)
X0= %landing_gear.GLGF_Bic
12
Write to VARIABLE landing_gear.GLGF_B_D >Global<
GLGF_B[3]
4:6
15
Write to VARIABLE landing_gear.GLGF_B >Global<

E. Landing Gear Block Script

Table of Contents

PHISK_D

PHISK_Init

POSN_E

RSK RGS_B

RU_V_D

Sliding Check 0

Sliding Check 1

Sliding ICs

Sticking ICs

```
inputs: (gearN,PHISK,PHISKold);
outputs: BREAK;
parameters: (TOLERANCEPHISK,PHICOUNT);
environment: INIT;

integer gearN;
float PHISK(4),PHISKold(4);
integer BREAK;
float TOLERANCEPHISK;
integer PHICOUNT(4);

if INIT then
  for N = 1:4 do
    PHICOUNT(N) = 0;
  endfor;
endif;

if (abs(PHISK(gearN)-PHISKold(gearN)) < TOLERANCEPHISK) | (PHICOUNT(gearN) > 10) the
n
  BREAK = 1;
  PHICOUNT(gearN) = 0;
else
  BREAK = 0;
  PHICOUNT(gearN) = PHICOUNT(gearN) + 1;
endif;
```

Aug 7 2001 17:21:31 **DEF_V** Page 1

```
inputs: (gearN,DEF_V_Vnew,DEF_V_Vold);

outputs: DEF_V_V;

integer gearN;
float DEF_V_Vnew(8),DEF_V_Vold(8);
float DEF_V_V(8);

for K = 1:8 do
  if (K == gearN) then
    DEF_V_V(K) = DEF_V_Vnew(K);
  elseif (K == 4+gearN) then
    DEF_V_V(K) = DEF_V_Vnew(K);
  else
    DEF_V_V(K) = DEF_V_Vold(K);
  endif;
endfor;
```

```
inputs: (gearN,L_V_V,DEF_V_V);

outputs: DEF_V_D_V;

parameters: (COEF1,COEF2);

integer gearN;
float L_V_V(8),DEF_V_V(8);
float DEF_V_D_V(8);
float COEF1,COEF2;

DEF_V_D_V(gearN) = ( L_V_V(gearN) - COEF1*DEF_V_V(gearN) )/COEF2;
DEF_V_D_V(4+gearN) = ( L_V_V(4+gearN) - COEF1*DEF_V_V(4+gearN) )/COEF2;
```

```
inputs: (FLG_B,FLGF_B,FLGF_B_D);

outputs: FLGF_B_DD;

parameters: (OMEGALG2,ZETALG2OMEGALG);

float FLG_B(3),FLGF_B(3),FLGF_B_D(3);
float FLGF_B_DD(3);
float OMEGALG2,ZETALG2OMEGALG;

for K = 1:3 do
  FLGF_B_DD(K) = (OMEGALG2)*(FLG_B(K) - FLGF_B(K)) - ZETALG2OMEGALG*FLGF_B_D(K);
endfor;
```

Aug 7 2001 17:21:31 **Gear_Forces_Moments** Page 1

```
inputs: (gearN,IGND,L_V_V,NF,L_B_V_V,RSK_RGS_B_T_V,FLG_B_Vold,GLG_B_Vold);

outputs: (FLG_B_V,GLG_B_V);

integer gearN,IGND(4);
float L_V_V(8),NF(4),L_B_V_V(9),RSK_RGS_B_T_V(9),FLG_B_Vold(12),
      GLG_B_Vold(12);
float FLG_B_V(12),GLG_B_V(12);
float FLG_B(3,4),GLG_B(3,4),L_V(2,4),L_B_V(3,3),RSK_RGS_B_T(3,3),
      FLG_V(3,4);

Z = 0;
for K = 1:3 do
  for N = 1:4 do
    Z = Z + 1;
    FLG_B(K,N) = FLG_B_Vold(Z);
    GLG_B(K,N) = GLG_B_Vold(Z);
  endfor;
endfor;

for K = 1:3 do
  FLG_B(K,gearN) = 0;
  GLG_B(K,gearN) = 0;
endfor;

if ( IGND(gearN) == 1 ) then

  Z = 0;
  for M = 1:3 do
    for N = 1:3 do
      Z = Z + 1;
      L_B_V(M,N) = L_B_V_V(Z);
      RSK_RGS_B_T(M,N) = RSK_RGS_B_T_V(Z);
    endfor;
  endfor;

  FLG_V(1,gearN) = L_V_V(gearN);
  FLG_V(2,gearN) = L_V_V(4+gearN);
  FLG_V(3,gearN) = NF(gearN);

  for K = 1:3 do
    for M = 1:3 do
      FLG_B(K,gearN) = FLG_B(K,gearN) + L_B_V(K,M)*FLG_V(M,gearN);
    endfor;
  endfor;

  for K = 1:3 do
    for M = 1:3 do
      GLG_B(K,gearN) = GLG_B(K,gearN) + RSK_RGS_B_T(K,M)*FLG_B(M,gearN);
    endfor;
  endfor;

endif;

Z = 0;
for K = 1:3 do
  for N = 1:4 do
    Z = Z + 1;
    FLG_B_V(Z) = FLG_B(K,N);
    GLG_B_V(Z) = GLG_B(K,N);
  endfor;
endfor;
```

```
inputs: gearN;

outputs: (Break,gearNnew);

integer gearN;
integer Break, gearNnew;

if (gearN < 1) then
  Break = 0;
  gearNnew = 1;
elseif (gearN <= 3) then
  Break = 0;
  gearNnew = gearN + 1;
elseif (gearN >= 4) then
  Break = 1;
  gearNnew = 1;
endif;
```

```
inputs: (FLG_B_V,GLG_B_V);

outputs: (FLG_B,GLG_B);

float FLG_B_V(12),GLG_B_V(12);
float FLG_B(3),GLG_B(3);

for K = 1:3 do
  FLG_B(K) = 0;
  GLG_B(K) = 0;
  for N = 1:4 do
    FLG_B(K) = FLG_B(K) + FLG_B_V( 4*(K - 1) + N);
    GLG_B(K) = GLG_B(K) + GLG_B_V( 4*(K - 1) + N);
  endfor;
endfor;
```

```
inputs: (gearN,PHISK,PHISK_D,L_SKN_B_V,L_B_V_V,GJold,R21old,R22old,R23old);

outputs: (GJ,R21,R22,R23);

parameters: (KAPPA1,KAPPA2,PHISK0);

integer gearN;
float PHISK(4),PHISK_D(4),L_SKN_B_V(9),L_B_V_V(9),GJold(4),R21old(4),R22old(4),R23ol
d(4);
float GJ(4),R21(4),R22(4),R23(4);
float KAPPA1,KAPPA2,PHISK0;
float L_SKN_B(3,3),L_B_V(3,3),L_SKN_V(3,3);

for N = 1:4 do
  GJ(N) = GJold(N);
  R21(N) = R21old(N);
  R22(N) = R22old(N);
  R23(N) = R23old(N);
endfor;

Z = 0;
for L = 1:3 do
  for M = 1:3 do
    Z = Z + 1;
    L_SKN_B(L,M) = L_SKN_B_V(Z);
    L_B_V(L,M) = L_B_V_V(Z);
  endfor;
endfor;

if ( gearN <= 2 ) then
  GJ(gearN) = ( KAPPA1*(PHISK(gearN) + PHISK0) + KAPPA2*PHISK_D(gearN) );
elseif ( gearN >= 3 ) then
  GJ(gearN) = ( KAPPA1*(PHISK(gearN) - PHISK0) + KAPPA2*PHISK_D(gearN) );
endif;

for L = 1:3 do
  L_SKN_V(2,L) = L_SKN_B(2,2)*L_B_V(2,L) + L_SKN_B(2,3)*L_B_V(3,L);
endfor;

R21(gearN) = L_SKN_V(2,1);
R22(gearN) = L_SKN_V(2,2);
R23(gearN) = L_SKN_V(2,3);
```

Aug 7 2001 17:21:31 **GLG_B_DD** Page 1

```
inputs: (GLG_B,GLGF_B,GLGF_B_D);

outputs: GLGF_B_DD;

parameters: (OMEGALG2,ZETALG2OMEGALG);

float GLG_B(3),GLGF_B(3),GLGF_B_D(3);
float GLGF_B_DD(3);
float OMEGALG2,ZETALG2OMEGALG;

for K = 1;3 do
  GLGF_B_DD(K) = (OMEGALG2)*(GLG_B(K) - GLGF_B(K)) - ZETALG2OMEGALG*GLGF_B_D(K);
endfor;
```

Aug 7 2001 17:21:31 **GRADPOSN_E** Page 1

```
inputs: (gearN,PHISK,SIPHIB,COPHIB,COTHETB,GRADPOSN_E3old);
outputs: GRADPOSN_E3;
parameters: RGS;

integer gearN;
float PHISK(4),SIPHIB,COPHIB,COTHETB,GRADPOSN_E3old(4);
float GRADPOSN_E3(4);
float RGS(4);

for N = 1:4 do
  GRADPOSN_E3(N) = GRADPOSN_E3old(N);
endfor;

GRADPOSN_E3(gearN) = -RGS(gearN)*COTHETB*(COS( PHISK(gearN) )*SIPHIB +
        SIN( PHISK(gearN) )*COPHIB);
```

Aug 7 2001 17:21:31 **In_Air_PHISK** Page 1

```
inputs: ();
outputs: PHISK;
parameters: PHISK0;

float PHISK(4).
float PHISK0;

PHISK(1) = -PHISK0;
PHISK(2) = -PHISK0;
PHISK(3) = PHISK0;
PHISK(4) = PHISK0;
```

```
inputs: (gearN,ISLIDE);

outputs: (GOTO3,GOTO4FR2);

integer gearN,ISLIDE(4);
integer GOTO3,GOTO4FR2;

if ( ISLIDE(gearN) == 1 ) then
  GOTO3 = 1;
  GOTO4FR2 = 0;
else
  GOTO3 = 0;
  GOTO4FR2 = 1;
endif;
```

Aug 7 2001 17:21:31 **L_B_SKN** Page 1

```
inputs: (gearN,PHISK);

outputs: L_B_SKN_V;

integer gearN;
float PHISK(4);
float L_B_SKN_V(9);
float COPHISK(4),SIPHISK(4);

COPHISK(gearN) = cos( PHISK(gearN) );
SIPHISK(gearN) = sin( PHISK(gearN) );

L_B_SKN_V(1) =  1;
L_B_SKN_V(2) =  0;
L_B_SKN_V(3) =  0;
L_B_SKN_V(4) =  0;
L_B_SKN_V(5) =  COPHISK(gearN);
L_B_SKN_V(6) = -SIPHISK(gearN);
L_B_SKN_V(7) =  0;
L_B_SKN_V(8) =  SIPHISK(gearN);
L_B_SKN_V(9) =  COPHISK(gearN);
```

```
inputs: (SIPHIB,SITHETB,COPHIB,COTHETB);
outputs: L_B_V_V;

float SIPHIB,SITHETB,COPHIB,COTHETB;
float L_B_V_V(9);

L_B_V_V(1) =  COTHETB;
L_B_V_V(2) =  0;
L_B_V_V(3) = -SITHETB;
L_B_V_V(4) =  SIPHIB*SITHETB;
L_B_V_V(5) =  COPHIB;
L_B_V_V(6) =  SIPHIB*COTHETB;
L_B_V_V(7) =  COPHIB*SITHETB;
L_B_V_V(8) = -SIPHIB;
L_B_V_V(9) =  COPHIB*COTHETB;
```

```
inputs: (gearN,POSN_E_V,DEF_V_Vold,DEF_Vic_Vold,RD_E_Vold,ISLIDE);

outputs: (DEF_V_V,DEF_Vic_V,RD_E_V);

integer gearN,ISLIDE(4);
float POSN_E_V(12),DEF_V_Vold(8),DEF_Vic_Vold(8),RD_E_Vold(8);
float DEF_V_V(8),DEF_Vic_V(8),RD_E_V(8);
float POSN_E(3,4),DEF_V(2,4),DEF_Vic(2,4),RD_E(2,4);

Z = 0;
for M = 1:3 do
   for N = 1:4 do
      Z = Z + 1;
      POSN_E(M,N) = POSN_E_V(Z);
   endfor;
endfor;

Z = 0;
for K = 1:2 do
   for N = 1:4 do
      Z = Z + 1,
      DEF_V(K,N) = DEF_V_Vold(Z);
      DEF_Vic(K,N) = DEF_Vic_Vold(Z);
      RD_E(K,N) = RD_E_Vold(Z);
   endfor;
endfor;

for K = 1:2 do
   RD_E(K,gearN) = POSN_E(K,gearN);
   if (ISLIDE(gearN) = 0) then
      DEF_Vic(K,gearN) = DEF_Vic(K,gearN) - DEF_V(K,gearN);
   endif;
   DEF_V(K,gearN) = 0;
endfor;

Z = 0;
for K = 1:2 do
   for N = 1:4 do
      Z = Z + 1;
      RD_E_V(Z) = RD_E(K,N);
      DEF_V_V(Z) = DEF_V(K,N);
      DEF_Vic_V(Z) = DEF_Vic(K,N);
   endfor;
endfor;
```

```
inputs: (gearN,ISLIDEold,NCOUNTold);

outputs: (ISLIDE,NCOUNT);

integer gearN,ISLIDEold(4),NCOUNTold(4);
integer ISLIDE(4),NCOUNT(4);

for N = 1;4 do
  ISLIDE(N) = ISLIDEold(N);
  NCOUNT(N) = NCOUNTold(N);
endfor;

ISLIDE(gearN) = 1;
NCOUNT(gearN) = 0;
```

Aug 7 2001 17:21:31 **L_V_E_and_L_V_E_D** Page 1

```
inputs: (COPSIB,SIPSIB,PSIB_D);
outputs: (L_V_E_V,L_V_E_D_V);

float COPSIB,SIPSIB,PSIB_D;
float L_V_E_V(9),L_V_E_D_V(9);

L_V_E_V(1) = COPSIB;
L_V_E_V(2) = SIPSIB;
L_V_E_V(3) = 0.;
L_V_E_V(4) = -SIPSIB;
L_V_E_V(5) = COPSIB;
L_V_E_V(6) = 0.;
L_V_E_V(7) = 0.;
L_V_E_V(8) = 0.;
L_V_E_V(9) = 1.;

L_V_E_D_V(1) =  PSIB_D*L_V_E_V(4);
L_V_E_D_V(2) =  PSIB_D*L_V_E_V(5);
L_V_E_D_V(3) = 0.;
L_V_E_D_V(4) = -PSIB_D*L_V_E_V(1);
L_V_E_D_V(5) = -PSIB_D*L_V_E_V(2);
L_V_E_D_V(6) = 0.;
L_V_E_D_V(7) = 0.;
L_V_E_D_V(8) = 0.;
L_V_E_D_V(9) = 0.;
```

```
inputs: (gearN,RU_V_D_V,NF,L_V_Vold);

outputs: (GOTO6,L_V_V);

parameters: MUSL;

integer gearN;
float RU_V_D_V(8),NF(4),L_V_Vold(8);
integer GOTO6;
float L_V_V(8);
float MUSL;
float L_V(2,4),RU_V_D(2,4);

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    L_V(K,N) = L_V_Vold(Z);
    RU_V_D(K,N) = RU_V_D_V(Z);
  endfor;
endfor;

RUVD = ((RU_V_D(1,gearN))^2 + (RU_V_D(2,gearN))^2)^(0.5);

if (RUVD == 0) then
  L_V(1,gearN) = 0;
  L_V(2,gearN) = 0;
else
  L_V(1,gearN) = MUSL*NF(gearN)*RU_V_D(1,gearN)/RUVD;
  L_V(2,gearN) = MUSL*NF(gearN)*RU_V_D(2,gearN)/RUVD;
  if RUVD <= 0.05 then
    L_V(1,gearN) = L_V(1,gearN)*RUVD/0.05;
    L_V(2,gearN) = L_V(2,gearN)*RUVD/0.05;
  endif;
endif;

GOTO6 = 1;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    L_V_V(Z) = L_V(K,N);
  endfor;
endfor;
```

Aug 7 2001 17:21:31 **L_V_When_Sticking** Page 1

```
inputs: (gearN,L_V_E_V,L_V_E_D_V,RD_E_V,POSN_E_V,RU_V_D_V,L_V_Vold,DEF_V_Vold);

outputs: (L_V_V,DEF_V_V);

parameters: (COEF1,COEF2);

integer gearN;
float L_V_E_V(9),L_V_E_D_V(9),RD_E_V(8),POSN_E_V(12),RU_V_D_V(8),L_V_Vold(8),DEF_V_V
old(8);
float L_V_V(8),DEF_V_V(8);
float COEF1,COEF2;
float L_V_E(3,3),RD_E(2,4),POSN_E(3,4),RU_V_D(2,4);
float L_V_E_D(3,3),DEF_V(2,4),DEF_V_D(2,4),L_V(2,4);

Z = 0;
for K = 1:3 do
  for M = 1:3 do
    Z = Z + 1;
    L_V_E(K,M) = L_V_E_V(Z);
    L_V_E_D(K,M) = L_V_E_D_V(Z);
  endfor;
endfor;

Z = 0;
for K = 1:3 do
  for N = 1:4 do
    Z = Z + 1;
    POSN_E(K,N) = POSN_E_V(Z);
  endfor;
endfor;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    L_V(K,N) = L_V_Vold(Z);
    DEF_V(K,N) = DEF_V_Vold(Z);
    RD_E(K,N) = RD_E_V(Z);
    RU_V_D(K,N) = RU_V_D_V(Z);
  endfor;
endfor;

for K = 1:2 do
  DEF_V(K,gearN) = 0;
  for M = 1:2 do
    DEF_V(K,gearN) = DEF_V(K.gearN) + L_V_E(K,M)*( RD_E(M,gearN) - POSN_E(M,gearN) )
;
  endfor;
endfor;

for K = 1:2 do
  DEF_V_D(K,gearN) = - RU_V_D(K,gearN);
  for L = 1:2 do
    DEF_V_D(K,gearN) = DEF_V_D(K,gearN) + L_V_E_D(K,L)*RD_E(L,gearN);
  endfor;
endfor;

for K = 1:2 do
  L_V(K,gearN) = COEF1*DEF_V(K,gearN) + COEF2*DEF_V_D(K,gearN);
endfor;

Z = 0;
for K = 1:2 do
  for N = 1.4 do
    Z = Z + 1;
    L_V_V(Z) = L_V(K,N);
    DEF_V_V(Z) = DEF_V(K,N);
  endfor;
endfor;
```

```
inputs: (gearN,POSN_E_V,GRADPOSN_E3,PHISKold);

outputs: PHISK;

integer gearN;
float POSN_E_V(12),GRADPOSN_E3(4),PHISKold(4);
float PHISK(4);

for N = 1:4 do
  PHISK(N) = PHISKold(N);
endfor;

PHISK(gearN) = PHISKold(gearN) - ( POSN_E_V(8 + gearN)/GRADPOSN_E3(gearN) );
```

Aug 7 2001 17:21:31 NF Page 1

```
inputs: (gearN,GJ,R21,R22,R23,L_V_V,NFold);

outputs: NF;

parameters: RGS.

integer gearN;
float GJ(4),R21(4),R22(4),R23(4),L_V_V(8),NFold(4);
float NF(4);
float RGS(4);

for N = 1:4 do
  NF(N) = NFold(N);
endfor;

if (R23(gearN) <> 0.0) then
  NF(gearN) = -( GJ(gearN)/RGS(gearN) + R21(gearN)*L_V_V(gearN) +
                 R22(gearN)*L_V_V(4+gearN) )/ R23(gearN);
else
  NF(gearN) = 0.0;
endif;

if ( NF(gearN) > 0 ) then
  NF(gearN) = 0;
endif;
```

```
inputs: (gearN,GNDCHK,IGNDold,PHISKold);

outputs: (GOTO2,GOTO6,IGND,PHISK,INITVARS);

parameters: PHISK0;

integer gearN,IGNDold(4);
float GNDCHK(4),PHISKold(4);
integer GOTO2,GOTO6,IGND(4),INITVARS;
float PHISK(4);
float PHISK0;

for N = 1:4 do
  IGND(N) = IGNDold(N);
  PHISK(N) = PHISKold(N);
endfor;

if ( GNDCHK(gearN) >= 0 ) then
  IGND(gearN) = 1;
  if ( IGNDold(gearN) == 0 ) then
    INITVARS = 1;
  else
    INITVARS = 0;
  endif;
  GOTO2 = 1;
  GOTO6 = 0;
else
  IGND(gearN) = 0;
  INITVARS = 0;
  if ( IGNDold(gearN) == 1 ) then
    if ( gearN == 1 | gearN == 2 ) then
      PHISK(gearN) = -PHISK0;
    else
      PHISK(gearN) = PHISK0;
    endif;
  endif;
  GOTO2 = 0;
  GOTO6 = 1;
endif;
```

```
inputs: (gearN,L_B_SKN_V,L_E_B_V,OMEGAB_B_T_V,VB_E,GRADPOSN_E3,PHISK_Dold);

outputs: PHISK_D;

parameters: (RGS,RSK_B);

integer gearN,
float L_B_SKN_V(9),L_E_B_V(9),OMEGAB_B_T_V(9),VB_E(3),GRADPOSN_E3(4),PHISK_Dold(4);
float PHISK_D(4);
float RGS(4),RSK_B(3,4);
float L_B_SKN(3,3),L_E_B(3,3),OMEGAB_B_T(3,3),RGS_B(3,4),TEMP1(3),TEMP2;

for N = 1;4 do
  PHISK_D(N) = PHISK_Dold(N);
endfor;

Z = 0;
for K = 1;3 do
  for M = 1;3 do
    Z = Z + 1,
    L_B_SKN(K,M) = L_B_SKN_V(Z);
    L_E_B(K,M) = L_E_B_V(Z);
    OMEGAB_B_T(K,M) = OMEGAB_B_T_V(Z);
  endfor;
endfor;

for K = 1;3 do
    RGS_B(K,gearN) = L_B_SKN(K,3)*RGS(gearN);
endfor;

for M = 1;3 do
  TEMP1(M) = RSK_B(M,gearN) + RGS_B(M,gearN);
endfor;

TEMP2 = 0.;
for L = 1;3 do
  for M = 1:3 do
    TEMP2 = TEMP2 + L_E_B(3,L)*OMEGAB_B_T(L,M)*TEMP1(M);
  endfor;
endfor;

if (GRADPOSN_E3(gearN) <> 0) then
  PHISK_D(gearN) = -( VB_E(3) + TEMP2 )/GRADPOSN_E3(gearN);
else
  PHISK_D(gearN) = 0;
endif;
```

Aug 7 2001 17:21:31 **PHISK_Init** Page 1

```
inputs: (PHISKold,gearN,PHISKo);

outputs: PHISK;

integer gearN;
float PHISKold(4),PHISKo(4);
float PHISK(4);

for N = 1:4 do
  PHISK(N) = PHISKold(N);
endfor;

PHISK(gearN) = PHISKo(gearN);
```

```
inputs: (gearN,RB_E,L_E_B_V,RG_E3,L_V_E_V,PHISK,L_B_SKN_V,POSN_E_Vold);
outputs: (POSN_E_V);
parameters: (RSK_B,RGS);

integer gearN;
float RB_E(3),L_E_B_V(9),RG_E3,L_V_E_V(9),PHISK(4),L_B_SKN_V(9),POSN_E_Vold(12);
float POSN_E_V(12);
float RSK_B(3,4),RGS(4);
float L_E_B(3,3),L_V_E(3,3),L_B_SKN(3,3),RV_E(3),RGS_B(3,4),POSN_E(3,4);

Z = 0;
for K = 1:3 do
  for L = 1:3 do
    Z = Z + 1;
    L_E_B(K,L) = L_E_B_V(Z);
    L_V_E(K,L) = L_V_E_V(Z);
    L_B_SKN(K,L) = L_B_SKN_V(Z);
  endfor;
endfor;

Z = 0;
for K = 1:3 do
  for N = 1:4 do
    Z = Z + 1;
    POSN_E(K,N) = POSN_E_Vold(Z);
  endfor;
endfor;

RV_E(1) = 0;
RV_E(2) = 0;
RV_E(3) = RG_E3;

for L = 1:3 do
  RGS_B(L,gearN) = L_B_SKN(L,3)*RGS(gearN);
endfor;

for K = 1:3 do
  POSN_E(K,gearN) = RB_E(K) - RV_E(K);
  for L = 1:3 do
    POSN_E(K,gearN) = POSN_E(K,gearN) + L_E_B(K,L)*(RSK_B(L,gearN) + RGS_B(L,gearN))
;
  endfor;
endfor;

Z = 0;
for K = 1:3 do
  for N = 1:4 do
    Z = Z + 1;
    POSN_E_V(Z) = POSN_E(K,N);
  endfor;
endfor;
```

Aug 7 2001 17:21:31 **RSK_RGS_B** Page 1

```
inputs: (gearN,L_B_SKN_V);

outputs: RSK_RGS_B;

parameters: (RSK_B,RGS);

integer gearN;
float L_B_SKN_V(9);
float RSK_RGS_B(3);
float RSK_B(3,4),RGS(4);
float L_B_SKN(3,3),RGSN_B(3);

Z = 0;
for L = 1:3 do
   for M = 1:3 do
      Z = Z + 1;
      L_B_SKN(L,M) = L_B_SKN_V(Z);
   endfor;
endfor;

for L = 1:3 do
   RSK_RGS_B(L) = RSK_B(L,gearN) + L_B_SKN(L,3)*RGS(gearN);
endfor;
```

```
inputs: (gearN,L_B_SKN_V,OMEGAB_B_T_V,PHISK_D,L_E_B_V,VB_E,PSIB_D,
         L_V_E_V,RU_V_D_Vold);

outputs: RU_V_D_V;

parameters: (RGS,RSK_B);

integer gearN;
float L_B_SKN_V(9),OMEGAB_B_T_V(9),PHISK_D(4),L_E_B_V(9),VB_E(3),PSIB_D,
      L_V_E_V(9),POSN_E_V(12),RU_V_D_Vold(8);
float RU_V_D_V(8);
float RGS(4),RSK_B(3,4);
float L_B_SKN(3,3),OMEGAB_B_T(3,3),L_E_B(3,3),L_V_E(3,3),
      TEMP1(3),TEMP2(3),TEMP3(3),POSN_E_D(3,4),RU_V_D(2,4);

Z = 0;
for M = 1:3 do
  for L = 1:3 do
    Z = Z + 1;
    L_B_SKN(M,L) = L_B_SKN_V(Z);
    OMEGAB_B_T(M,L) = OMEGAB_B_T_V(Z);
    L_E_B(M,L) = L_E_B_V(Z);
    L_V_E(M,L) = L_V_E_V(Z);
  endfor;
endfor;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    RU_V_D(K,N) = RU_V_D_Vold(Z);
  endfor;
endfor;

TEMP1(1) = 0;
TEMP1(2) = L_B_SKN(2,2);
TEMP1(3) = L_B_SKN(3,2);

for L = 1:3 do
  TEMP2(L) = L_B_SKN(L,3)*RGS(gearN) + RSK_B(L,gearN);
endfor;

for M = 1:3 do
  TEMP3(M) = - RGS(gearN)*PHISK_D(gearN)*TEMP1(M);
  for L = 1:3 do
    TEMP3(M) = TEMP3(M) + OMEGAB_B_T(M,L)*TEMP2(L);
  endfor;
endfor;

for K = 1:2 do
  POSN_E_D(K,gearN) = VB_E(K);
  for M = 1:3 do
    POSN_E_D(K,gearN) = POSN_E_D(K,gearN) + L_E_B(K,M)*TEMP3(M);
  endfor;
endfor;
POSN_E_D(3,gearN) = 0;

for K = 1:2 do
  RU_V_D(K,gearN) = 0;
  for M = 1:3 do
    RU_V_D(K,gearN) = RU_V_D(K,gearN) + L_V_E(K,M)*POSN_E_D(M,gearN);
  endfor;
endfor;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    RU_V_D_V(Z) = RU_V_D(K,N);
  endfor;
endfor;
```

```
inputs: (gearN,L_V_V,NF,ISLIDEold,NCOUNT2old);

outputs: (ISLIDE,GOTO5,GOTO6,NCOUNT2,INITVARS);

parameters: MUST;

integer gearN,ISLIDEold(4),NCOUNT2old(4);
float L_V_V(8),NF(4);
integer ISLIDE(4),GOTO5,GOTO6,NCOUNT2(4),INITVARS;
float MUST;

for N = 1:4 do
  ISLIDE(N) = ISLIDEold(N);
  NCOUNT2(N) = NCOUNT2old(N);
endfor;

L_TOT = ( (L_V_V(gearN))^(2.) + (L_V_V(4+gearN))^(2.) )^(0.5);

if ( L_TOT > MUST*abs( NF(gearN) ) ) then
  NCOUNT2(gearN) = NCOUNT2(gearN) + 1;
else
  NCOUNT2(gearN) = 0;
endif;

if ( NCOUNT2(gearN) > 10 ) then
  ISLIDE(gearN) = 1;
  INITVARS = 1;
  GOTO5 = 1;
  GOTO6 = 0;
  NCOUNT2(gearN) = 0;
else
  ISLIDE(gearN) = 0;
  INITVARS = 0;
  GOTO5 = 0;
  GOTO6 = 1;
endif;
```

Aug 7 2001 17:21:31 **Sliding_Check_1** Page 1

```
inputs: (gearN,RU_V_D_Vold,RU_V_D_V,ISLIDEold,NCOUNTold);

outputs: (ISLIDE,NCOUNT,GOTO4FR3,GOTO5,INITVARS);

parameters: (VREF,NREF);

integer gearN,ISLIDEold(4),NCOUNTold(4);
float RU_V_D_Vold(8),RU_V_D_V(8);
integer ISLIDE(4),NCOUNT(4),GOTO4FR3,GOTO5,INITVARS;
integer NREF;
float VREF;
float RU_V_Dold(2,4),RU_V_D(2,4),RUVD,RUVDold;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    RU_V_Dold(K,N) = RU_V_D_Vold(Z);
    RU_V_D(K,N) = RU_V_D_V(Z);
  endfor;
endfor;

for N = 1:4 do
  ISLIDE(N) = ISLIDEold(N);
  NCOUNT(N) = NCOUNTold(N);
endfor;

RUVD = ((RU_V_D(1,gearN))^2 + (RU_V_D(2,gearN))^2)^(0.5);
RUVDold = ((RU_V_Dold(1,gearN))^2 + (RU_V_Dold(2,gearN))^2)^(0.5);

if ( RUVD > VREF) then
  ISLIDE(gearN) = 1;
elseif ( RUVD <= VREF ) & ( RUVDold <= VREF ) & ( NCOUNTold(gearN) <= NREF ) then
  ISLIDE(gearN) = 1;
else
  ISLIDE(gearN) = 0;
endif;

if ( ISLIDE(gearN) == 1 ) then
  GOTO4FR3 = 0;
  GOTO5 = 1;
  INITVARS = 0;
  NCOUNT(gearN) = NCOUNTold(gearN) + 1;
elseif ( ISLIDE(gearN) == 0 ) then
  GOTO4FR3 = 1;
  GOTO5 = 0;
  INITVARS = 1;
endif;
```

Aug 7 2001 17:21:31 **Sliding_ICs** Page 1

```
inputs: (gearN,NCOUNTold,DEF_V_V,DEF_Vic_Vold);

outputs: (NCOUNT,DEF_Vic_V);

integer gearN,NCOUNTold(4);
float DEF_V_V(8),DEF_Vic_Vold(8);
integer NCOUNT(4);
float DEF_Vic_V(8);
float DEF_V(2,4),DEF_Vic(2,4);

for N = 1:4 do
  NCOUNT(N) = NCOUNTold(N);
endfor;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    DEF_V(K,N) = DEF_V_V(Z);
    DEF_Vic(K,N) = DEF_Vic_Vold(Z);
  endfor;
endfor;

NCOUNT(gearN) = 0;

for K = 1:2 do
  DEF_Vic(K,gearN) = DEF_Vic(K,gearN) + DEF_V(K,gearN);
endfor;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    DEF_Vic_V(Z) = DEF_Vic(K,N);
  endfor;
endfor;
```

```
inputs: (gearN,POSN_E_V,L_E_V_V,DEF_V_V,RD_E_Vold,DEF_Vic_Vold);

outputs: (RD_E_V,DEF_Vic_V);

integer gearN;
float POSN_E_V(12),L_E_V_V(9),DEF_V_V(8),RD_E_Vold(8),DEF_Vic_Vold(8);
float RD_E_V(8),DEF_Vic_V(8);
float POSN_E(3,4),L_E_V(3,3),DEF_V(2,4),RD_E(2,4),DEF_Vic(2,4);

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    DEF_V(K,N) = DEF_V_V(Z);
    RD_E(K,N) = RD_E_Vold(Z);
    DEF_Vic(K,N) = DEF_Vic_Vold(Z);
  endfor;
endfor;

Z = 0;
for K = 1:3 do
  for N = 1:4 do
    Z = Z + 1;
    POSN_E(K,N) = POSN_E_V(Z);
  endfor;
endfor;

Z = 0;
for L = 1:3 do
  for M = 1:3 do
    Z = Z + 1;
    L_E_V(L,M) = L_E_V_V(Z);
  endfor;
endfor;

for K = 1:2 do

  RD_E(K,gearN) = POSN_E(K,gearN);
  for L = 1:2 do
    RD_E(K,gearN) = RD_E(K,gearN) + L_E_V(K,L)*DEF_V(L,gearN);
  endfor;

  DEF_Vic(K,gearN) = DEF_Vic(K,gearN) - DEF_V(K,gearN);

endfor;

Z = 0;
for K = 1:2 do
  for N = 1:4 do
    Z = Z + 1;
    RD_E_V(Z) = RD_E(K,N);
    DEF_Vic_V(Z) = DEF_Vic(K,N);
  endfor;
endfor;
```