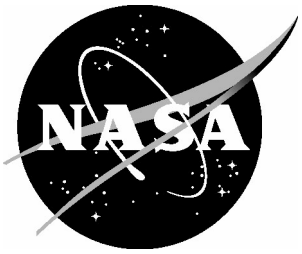
NASA/CR-2005-213746



Motion Cueing Algorithm Development: New Motion Cueing Program Implementation and Tuning

Robert J. Telban and Frank M. Cardullo
State University of New York, Binghamton, New York

Lon C. Kelly
Unisys Corporation, Hampton, Virginia

May 2005

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results ... even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

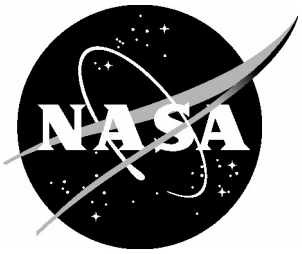
- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Phone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/CR-2005-213746



Motion Cueing Algorithm Development: New Motion Cueing Program Implementation and Tuning

Robert J. Telban and Frank M. Cardullo
State University of New York, Binghamton, New York

Lon C. Kelly
Unisys Corporation, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Purchase Order L70823D

May 2005

Acknowledgments

Jacob Houck of the Flight Simulation and Software Branch at the NASA Langley Research Center assisted in the preparation of this report.

Available from:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

Abstract

A computer program has been developed for the purpose of driving the NASA Langley Research Center Visual Motion Simulator (VMS). This program includes two new motion cueing algorithms, the optimal algorithm and the nonlinear algorithm. A general description of the program is given along with a description and flowcharts for each cueing algorithm, and also descriptions and flowcharts for subroutines used with the algorithms. Common block variable listings and a program listing are also provided.

The new cueing algorithms have a nonlinear gain algorithm implemented that scales each aircraft degree-of-freedom input with a third-order polynomial. A description of the nonlinear gain algorithm is given along with past tuning experience and procedures for tuning the gain coefficient sets for each degree-of-freedom to produce the desired piloted performance. This algorithm tuning will be needed when the nonlinear motion cueing algorithm is implemented on a new motion system in the Cockpit Motion Facility (CMF) at the NASA Langley Research Center.

This page intentionally left blank.

Table of Contents

Nomenclature

a	acceleration $\mathbf{a} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^{\mathbf{T}}$
A, B, C, D, H	matrices of the state-space model of a control system
$\mathbf{A}'$	system matrix of the standard form optimal control system
c_0, c_1, c_2, c_3	coefficients of the nonlinear scaling polynomial
f	specific force
Fr	reference frame
G	turbulence gust vector $\mathbf{G} = \begin{bmatrix} p_G & q_G & r_G & u_G & v_G & w_G \end{bmatrix}^{\mathbf{T}}$
g	acceleration due to gravity
H_G	augmented turbulence transfer function
J	system cost function
K	state feedback gain matrix
l_j	displacement of the j-th motion platform actuator
P	solution of the algebraic Riccati equation
Q, R, $\mathbf{R_d}$	weighting matrices in a cost function (tracking form)
$\mathbf{Q_2}$	weighting matrix for nonlinear algorithm control law
$\mathbf{R_1'}, \mathbf{R_2}, \mathbf{R_{12}}$	weighting matrices in a cost function (standard form)
R	radius vector
S	simulator centroid displacement
s	Laplace variable
s_0, s_1	slopes of the nonlinear gain polynomial
$\mathbf{T_S}$	transformation matrix from angular velocity to Euler angle rates

u	input to a control system
v	error output of neurocomputing solver
W(s)	optimal algorithm transfer function matrix
x	system state vector
y	desired state space system output
z	excitatory input signal for neurocomputing system
α	prescribed degree of nonlinearity for nonlinear algorithm
$\boldsymbol{\beta}$	Euler angles $\boldsymbol{\beta} = [\phi \quad \theta \quad \psi]^{\mathbf{T}}$
δ	pilot control input vector
μ	learning parameter for neurocomputing solver
$\boldsymbol{\omega}$	angular velocity about the body frame $\boldsymbol{\omega} = [p \quad q \quad r]^{\mathbf{T}}$

Subscripts

Subscripts indicate to what the main symbol is related

$( )_A$	aircraft
$( )_d$	simulator states included in the cost function
$( )_I$	inertial reference frame
$( )_j$	j-th actuator of the motion platform
$( )_S$	simulator
$( )_{SR}$	simulator rotation
$( )_{ST}$	simulator tilt coordination channel
$( )_{STL}$	simulator tilt coordination channel with limit
$( )_{x,y,z}$	x,y, or z component
$( )_\alpha$	relates to system with nonlinearity

Superscripts

Superscripts indicate which reference frame the main symbol is in

$( )^{A}$ in aircraft reference frame Fr_A

$( )^{I}$ in inertial reference frame Fr_I

$( )^{S}$ in simulator reference frame Fr_S

This page intentionally left blank.

1. Introduction

The flowchart for the new Langley Research Center Visual Motion Simulator (VMS) drive software is shown in Figure 2.1. Originally, three motion cueing algorithms were available: a modified version of the standard NASA adaptive algorithm (WCNTRL2), the optimal algorithm (WASHOPT3) and the nonlinear algorithm (NEWOPT4) discussed by Telban and Cardullo [1]. The modified adaptive algorithm is not operational. A one-time initialization of the cueing algorithm parameters and initial states is accomplished with a FORTRAN call to the subroutines WINIT2 and WINIT4. WINIT2 initializes variables exclusive to the adaptive and optimal algorithms, and variables common to both the optimal and nonlinear algorithms. WINIT4 initializes variables exclusive to the nonlinear algorithm, with symmetric matrices generated from initial vectors that contain the upper triangular elements.

The real time program sends a flag to the motion cueing algorithm that indicates the current "mode" of the simulation, i.e. "RESET", "HOLD", or "OPERATE". When the simulation is in the RESET mode, the motion base is driven slowly (in about 15 seconds) to its neutral position. During the HOLD mode, the motion base is driven slowly from the neutral position to the required trim orientation. In the OPERATE mode, the motion cueing algorithm uses the aircraft states to drive the motion base in real time.

The motion cueing algorithm produces the desired simulator displacement and attitude commands. These commands serve as input to the subroutine JACKDRVR, which performs a kinematic transformation to compute six actuator extensions and also activates a braking algorithm in the event any of the actuators reaches its extension limit.

The subroutine INVPLF performs an inverse kinematic transformation [2] to obtain the platform positions and attitudes from the computed leg extensions.

The executive cueing algorithm subroutines WASHOPT3 and NEWOPT4 also incorporate special cueing effects. The augmented turbulence cue for the vertical mode developed by Telban and Cardullo [1] is implemented in each subroutine. In order to increase the sensation of the touchdown and ground effects, an approach proposed by Parrish and Martin [3] was implemented in each subroutine. This approach uses increased nonlinear gains for the vertical mode that are instantaneously activated upon touchdown, and runway roughness amplitude that is gradually added to the simulator z-axis displacement after touchdown.

Table 1.1 provides a list of all the FORTRAN subroutines and common blocks used in the new software. The subroutines and common blocks used exclusively with the optimal algorithm are noted with an asterisk. All of the files have the extension .f unless otherwise noted. Listings of the variables for each common block used in the programs are given in Section 7.

Table 1.1. FORTRAN Files for New Drive Software.

gainopt3*	liba	ofil3*	winit2*	comint2.com
gainopt4	newopt4	resetc2	winit4	matrix1c.com
integ3*	nfilr	simq	wtrim3	nopt4.com
integ4	nfilx	state4		optint3.com*
invplf	nfily	vmult		wopt3.com
jackdrvr	nfilz	washopt3*		

Table 1.2 provides a list of the aircraft model inputs required for the motion cueing algorithms. Given are the cueing algorithm FORTRAN variables.

Table 1.2. Motion Cueing Algorithm Inputs.

	Optimal (IWASH=2)	Nonlinear (IWASH=3)
Aircraft Accel. at Motion Base Centroid	ACA(3)	ACA(3)
Aircraft Angular Velocity	WAA(3)	WAA(3)
Aircraft Euler Angles	BETAA(3)	BETAA(3)
Time Step Size	DT	DT
In-Air/On-Ground Flag	SQWASHI	SQWASHI
Aircraft Ground Speed	VGSPD	VGSPD
Vertical Gust	WGUST	WGUST

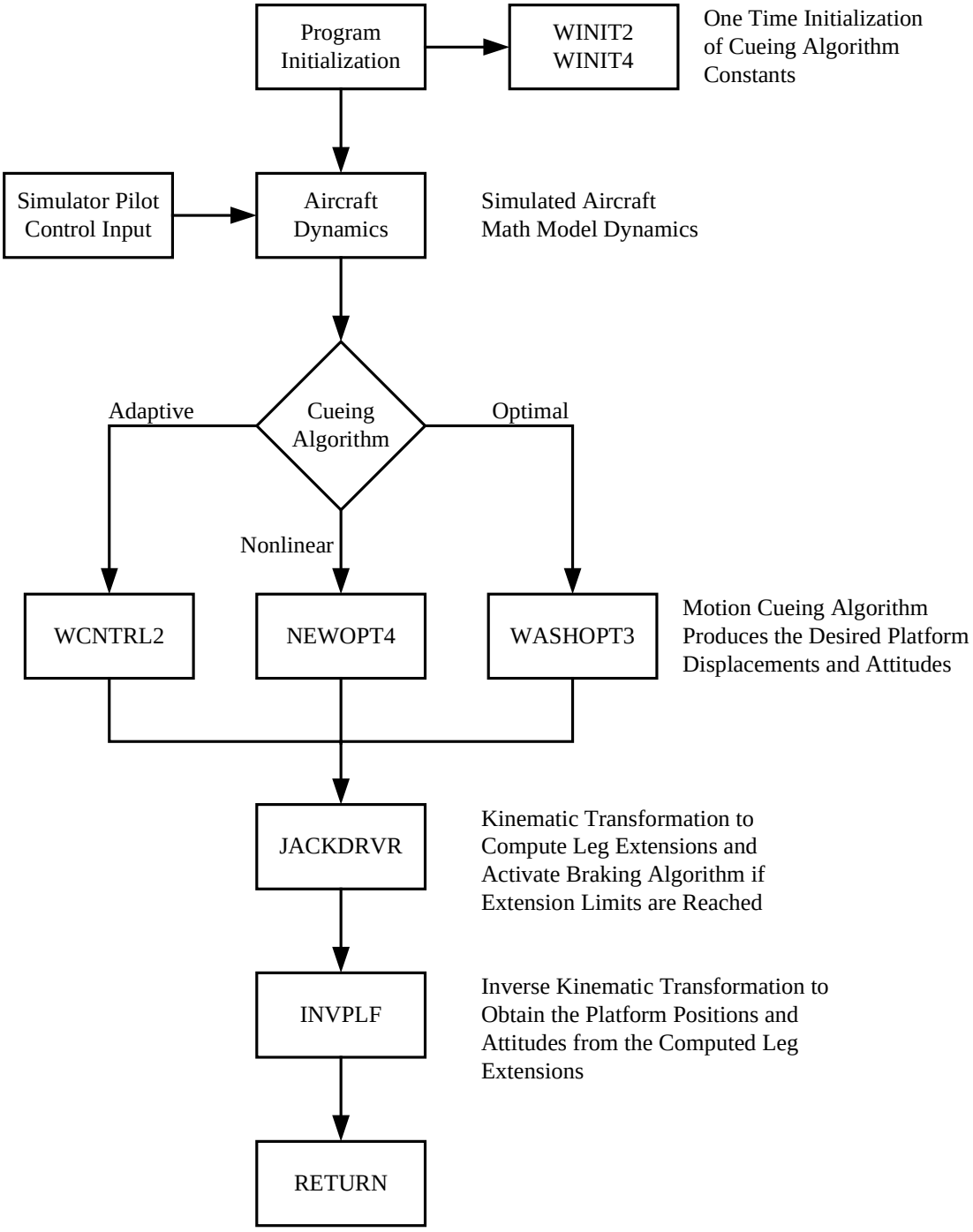


Figure 1.1. Langley Visual Motion Simulator (VMS) Drive Software Flow Chart.

2. Background Information

2.1. NASA Langley Visual Motion Simulator (VMS)

The NASA Langley Visual Motion Simulator (VMS), shown in Figure 2.1, is a general-purpose flight simulator consisting of a two-crewmember cockpit mounted on a 60-inch stroke six-degree-of-freedom synergistic motion base [4], [5].

Figure 2.1. NASA Langley Visual Motion Simulator (VMS). NASA Langley Research Center, Hampton, Virginia.

Motion cues are provided in the simulator by the extension or retraction of the six hydraulic actuators of the motion base relative to the simulator neutral position. The NASA adaptive algorithm and the new optimal and nonlinear algorithms were used to drive the motion base during the tuning of the new algorithms and the piloted test evaluation.

The cockpit of the VMS, shown in Figure 2.2, is designed to accommodate a generic transport aircraft configuration on the left side and a generic fighter or rotorcraft configuration on the right side. Both sides of the cockpit are outfitted with three heads-down CRT displays (primary flight display, navigation/map display, and engine display), a number of small standard electromechanical circular instruments and a landing gear handle mounted in the instrument panel. The left side contains a two-axis side stick

control loader, and the right side contains a control loaded two-axis center stick. Both sides contain control loaded rudder systems. The center aisle stand is outfitted with a control display unit, a four-lever throttle quadrant, a flap handle, a speed brake handle, and a slats handle. The cockpit is outfitted with four collimated window display systems to provide an out-the-window visual scene. During the piloted evaluations, the test subject flew from the left seat of the cockpit, while an observer/test conductor rode in the right seat.

Figure 2.2. Visual Motion Simulator Cockpit. NASA Langley Research Center, Hampton, Virginia.

The simulator includes a high fidelity, highly nonlinear mathematical model of a Boeing 757-200 aircraft, complete with landing gear dynamics, gust and wind models, flight management systems, and flight control computer systems. For this study, the test subjects flew the simulated aircraft in the manual control mode (without the autopilot), and with manual throttle control (without the autothrottle).

The out-the window visual scene is driven by an Evans and Sutherland ESIG 3000/GT computer generated image system. The visual database represented the Dallas/Fort Worth airport and its surrounding terrain. The study utilized runways 18L and 18R for approach maneuvers and runway 18R for takeoff maneuvers. The runways

were equipped with approach lights, precision approach path indicator lights, runway markings, and signage. The database included all runways and taxiways, and all airport structures and buildings. All tests were conducted in a daylight environment with full visibility.

2.2. Reference Frames

A series of reference frames are used in the definition of the motion cueing algorithms. These reference frames are defined below and are shown in Figure 2.3.

2.2.1. Aircraft Center of Gravity

The aircraft center of gravity reference frame Fr_{CG} has its origin at the center of gravity of the aircraft. Frame Fr_{CG} has an orientation for X_{CG}, Y_{CG}, and Z_{CG} that is parallel to reference frames Fr_S and $Fr_{A.}$

2.2.2. Simulator

The simulator reference frame Fr_S has its origin at the centroid of the simulator payload platform, i.e. the centroid of the upper bearing attachment points. The origin is fixed with respect to the simulator payload platform. X_S points forward and Z_S points downward with respect to the simulator cockpit, and Y_S points toward the pilot's right hand side. The x-y plane is parallel to the floor of the cockpit.

2.2.3. Aircraft

The aircraft reference frame Fr_A has its origin at the same relative cockpit location as the simulator reference frame Fr_S. Fr_A has the same orientation for X_A, Y_A, and Z_A with respect to the cockpit as the simulator frame Fr_S.

2.2.4. Inertial

The inertial reference frame Fr_I is earth-fixed with Z_I aligned with the gravity vector **g**. Its origin is located at the center of the fixed platform motion base. X_I points forward and Y_I points to the right hand side with respect to the simulator pilot.

2.2.5. Reference Frame Locations

In Figure 2.3 are four vectors that define the relative location of the reference frames. R_I defines the location of Fr_S with respect to Fr_I. R_S defines the location of Fr_{PS} with respect to Fr_S. Similarly, R_A defines the location of Fr_{PA} with respect to $Fr_{A,}$ where $R_A = R_S$. R_{CG} defines the location of Fr_A with respect to Fr_{CG}.

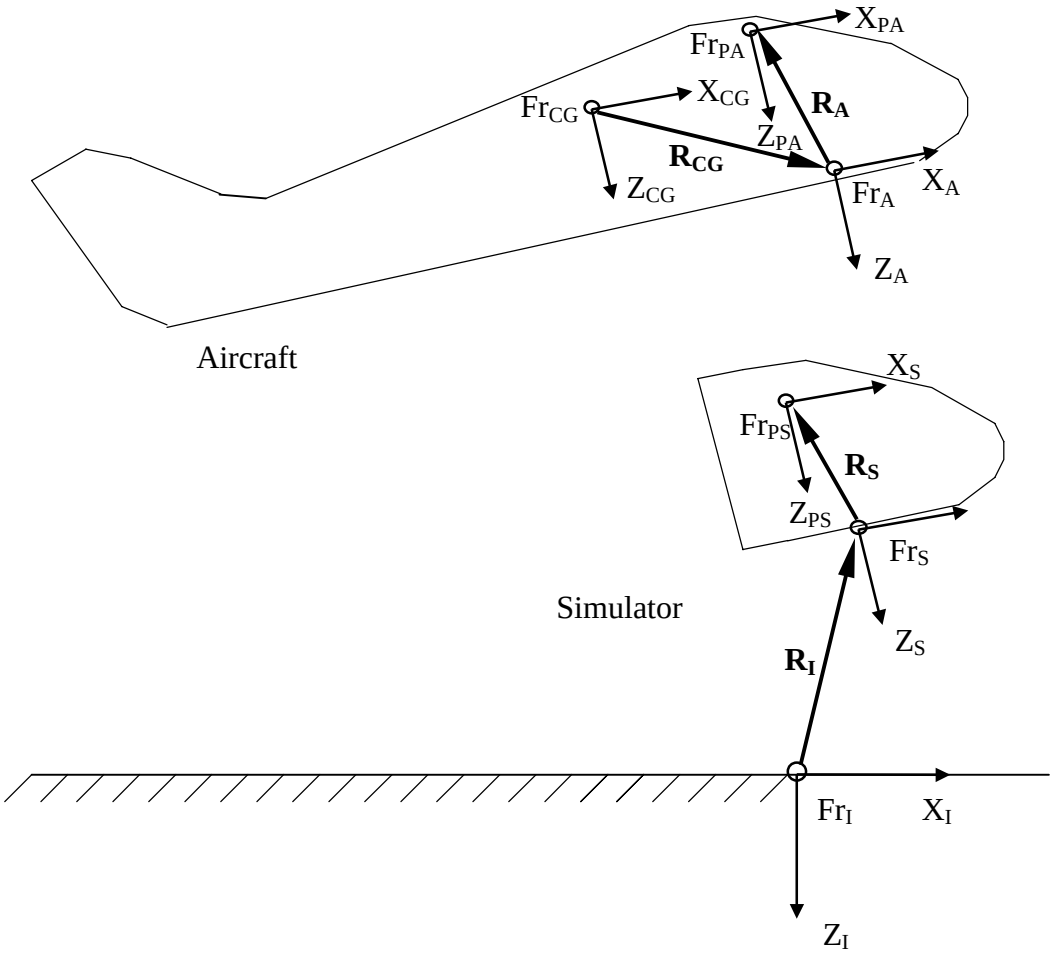


Figure 2.3. Reference Frame Locations. Adapted from Wu [6].

2.3. Coordinate Transformations

The orientation between the body-fixed simulator reference frame Fr_S and the inertial reference frame Fr_I can be specified by three Euler angles: $\boldsymbol{\beta} = [\phi \;\; \theta \;\; \psi]^T$ that define a sequence of rotations that carry Fr_S into Fr_I. A vector $\mathbf{V}$ expressed in the two frames can be related by the transformation matrix $\mathbf{L_{IS}}$ (Fr_I to Fr_S) or $\mathbf{L_{SI}}$ (Fr_S to Fr_I), with $\mathbf{V^S = L_{IS} V^I}$ and $\mathbf{V^I = L_{SI} V^S}$, where $\mathbf{L_{IS} = L_{SI}^{-1} = L_{SI}^T}$, and

$$\mathbf{L_{SI}} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}. \qquad (2.1)$$

The angular velocity of Fr_S with respect to Fr_I can be related to the Euler angle rates $\dot{\boldsymbol{\beta}}$ by the following expression. Let $\boldsymbol{\omega}_\mathbf{A}^\mathbf{A}$ represent the components of this angular velocity in frame Fr_S, then $\dot{\boldsymbol{\beta}} = \mathbf{T_S}\, \boldsymbol{\omega}_\mathbf{A}^\mathbf{A}$, where

$$\mathbf{T_S} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix}. \qquad (2.2)$$

Note that in this example, the body-fixed aircraft reference frame Fr_A can replace the body-fixed simulator reference frame Fr_S.

The subroutine LIBA computes the coordinate transformations of Eqs. (2.1) and (2.2) as shown in the flowchart given in Figure 2.4.

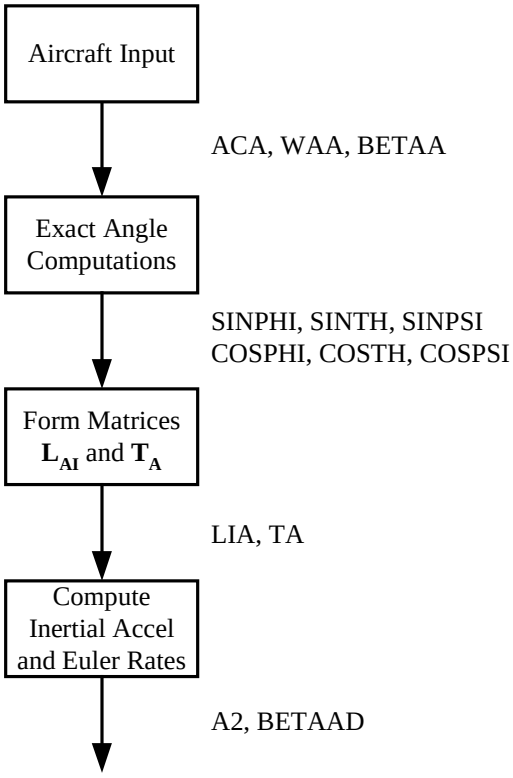


Figure 2.4. LIBA Subroutine Flowchart.

2.4. Nonlinear Input Scaling

Limiting and scaling are applied to both aircraft translational input signals $\mathbf{a}_{\mathbf{A}}^{\mathbf{A}}$ and rotational input signals $\boldsymbol{\omega}_{\mathbf{A}}^{\mathbf{A}}$. Limiting and scaling modify the amplitude of the input uniformly across all frequencies. Limiting is a nonlinear process that clips the signal so that it is limited to be less than a given magnitude. Limiting and scaling can be used to reduce the motion response of a flight simulator. A third-order polynomial scaling was developed [6] and has been implemented in the new simulator motion cueing algorithms.

When the magnitude of the input to the simulator motion system is small, the gain is desired to be relatively high, or the output will be below the pilot's perception threshold. When the magnitude of input is high, the gain is desired to be relatively low or

the simulator may attempt to go beyond the hardware limits. Let us define the input as x and the output as y. Now define $x_{\max}$ as the expected maximum input and $y_{\max}$ as the maximum output, and s_0 and s_1 as the slopes at $x = 0$ and $x = x_{\max}$ respectively. Four desired characteristics for the nonlinear scaling are expressed as:

(1) $x = 0 \Rightarrow y = 0,$
(2) $x = x_{\max} \Rightarrow y = y_{\max},$
(3) $y'\big|_{x=0} = s_0,$
(4) $y'\big|_{x=x_{\max}} = s_1,$

A third-order polynomial is then employed to provide functions with all the desired characteristics. This polynomial will be of the form

$$y = c_3 x^3 + c_2 x^2 + c_1 x + c_0 \tag{2.3}$$

where

$$c_0 = 0,$$
$$c_1 = s_0,$$
$$c_2 = x_{\max}^{-2}\left(3y_{\max} - 2s_0 x_{\max} - s_1 x_{\max}\right),$$
$$c_3 = x_{\max}^{-3}\left(s_0 x_{\max} - 2y_{\max} + s_1 x_{\max}\right).$$

One example of this polynomial gain is shown in Figure 2.5, with parameters set as $x_{\max} = 10,\ y_{\max} = 6,\ s_0 = 1.0,\ s_1 = 0.1.$

The subroutine GAINOPT3 is used to compute the polynomial scaling coefficients for each aircraft degree-of-freedom input for the optimal algorithm. The flowchart for GAINOPT3 is given in Figure 2.6. The subroutine GAINOPT4 computes the coefficients for the nonlinear algorithm, and follows the same variable flow as shown in Figure 2.6. Each subroutine contains scaling coefficients specific to either the optimal algorithm (GAINOPT3) or the nonlinear algorithm (GAINOPT4).

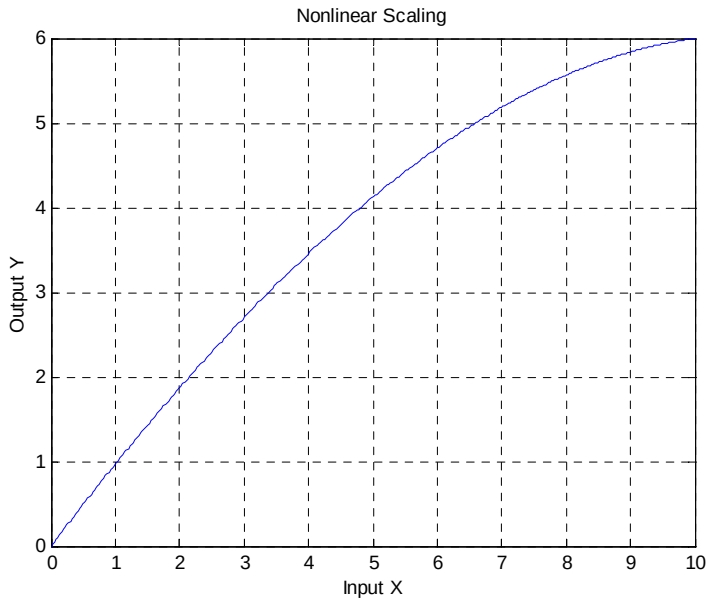


Figure 2.5. Nonlinear Input Scaling.

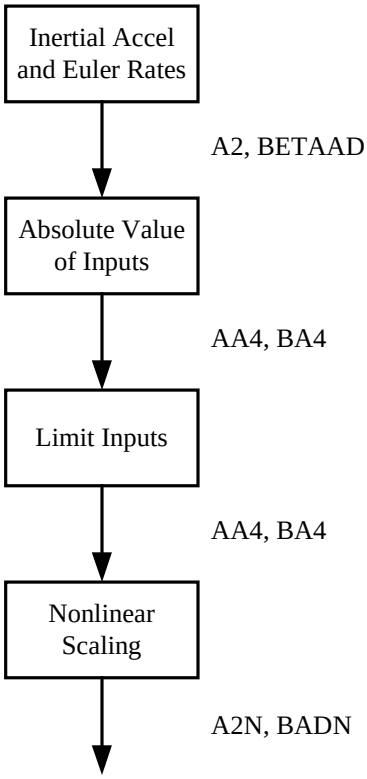


Figure 2.6. GAINOPT3/GAINOPT4 Subroutine Flowchart.

2.5. Actuator Geometry

The geometry of a six-degree-of-freedom synergistic motion system is given in Figure 2.7. The relevant vectors relating the locations of the upper and lower bearings of the j-th actuator are shown below in Figure 2.8.

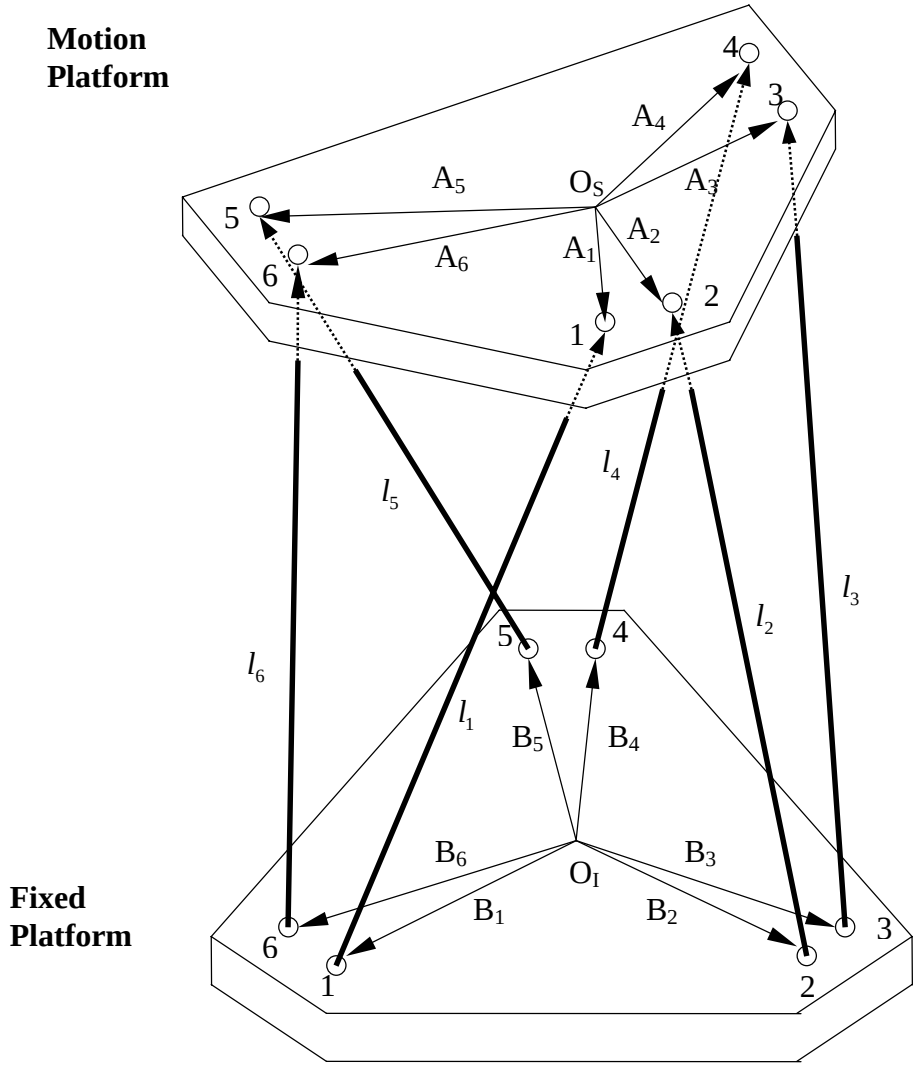


Figure 2.7. Geometry of a Six-Degree-of-Freedom Motion System. Adapted from Wu (1997).

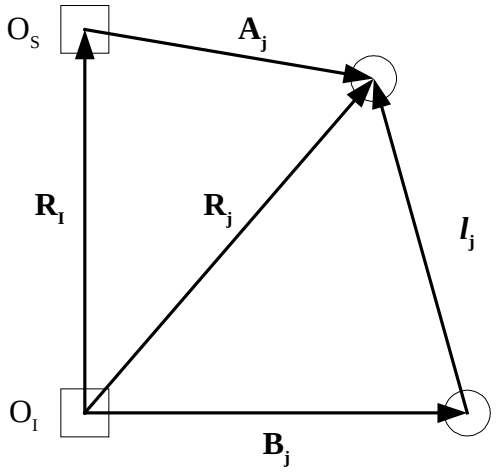


Figure 2.8. Vectors for the j-th Actuator.

In Figure 2.8, O_S and O_I are the centroids of the motion platform and fixed platform respectively, and are also respectively the origins for Fr_S and Fr_I. It can be seen that the relation among those vectors is

$$\mathbf{R_I} + \mathbf{A_j^I} = \mathbf{R_j} = \mathbf{B_j^I} + l_j. \tag{2.4}$$

The actuator length vector can then be found from

$$l_j = \mathbf{A_j^I} + \mathbf{R_I} - \mathbf{B_j^I}. \tag{2.5}$$

The expression of l_j in the inertial reference frame Fr_I is desired:

$$\begin{aligned} l_j^I &= \mathbf{A_j^I} + \mathbf{R_I} - \mathbf{B_j^I} \\ &= \mathbf{L_{IS}}\mathbf{A_j^S} + \mathbf{R_I} - \mathbf{B_j^I}, \end{aligned} \tag{2.6}$$

where $\mathbf{A_j^S}$ are the coordinates of the upper bearing attachment point of the j-th actuator in Fr_S and $\mathbf{B_j^I}$ are the coordinates of the lower bearing attachment point of the j-th actuator in Fr_I. The actuator extension is computed from a neutral platform position, where $\mathbf{R_I} = \mathbf{0}$, therefore

$$\Delta l_j^I = \mathbf{L_{IS}}\mathbf{A_j^S} + \Delta\mathbf{R_I}. \tag{2.7}$$

The subroutine JACKDRVR is used to compute the actuator extension lengths that are derived from the motion platform displacements and attitudes in Eq. (2.7). The flowchart for JACKDRVR is given in Figure 2.9. The variable inputs and outputs for the subroutine INVPLF are also shown in Figure 2.9. The actuator extension limiting performed within JACKDRVR is discussed in Section 6 and Appendix A.

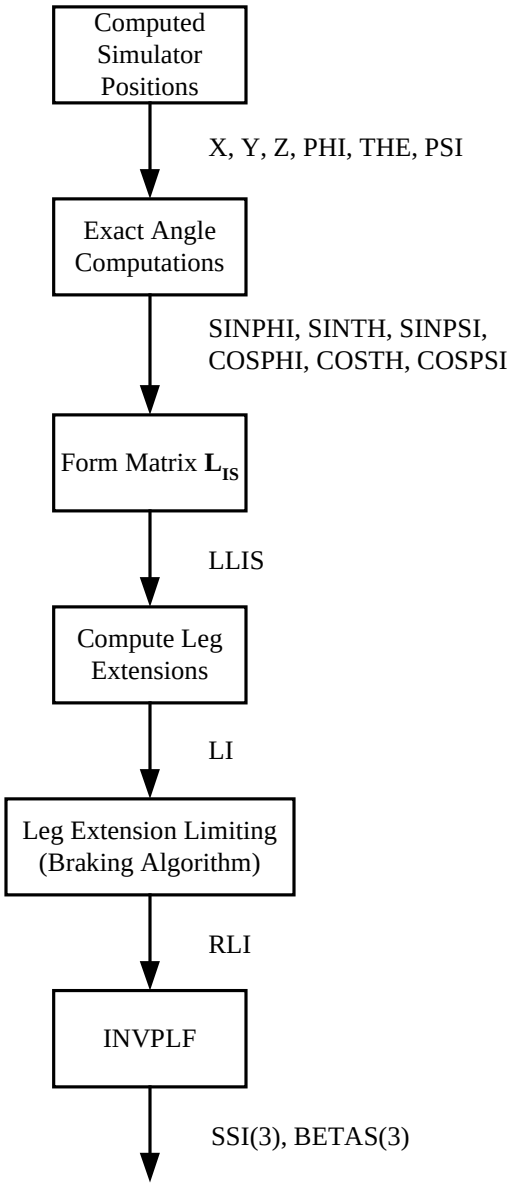


Figure 2.9. JACKDRVR/INVPLF Subroutine Flowchart.

2.6. Augmented Turbulence Cue

Reid and Robinson [7] first addressed the problem of producing acceptable motion cues to turbulence inputs. They noted that heave is the most critical cue in representing turbulence, but is also the most restricted cue when constraining motion within the platform geometry. To overcome this limitation, they developed an approach in which a second set of aircraft equations of motion driven only by the turbulence inputs is employed. The output from this augmented channel is then added to the output from the primary equations of motion, being driven by both turbulence and the pilot control inputs, before serving as input to the motion system. A similar approach to that developed by Reid and Robinson [7] has been implemented and is shown in Figure 2.10.

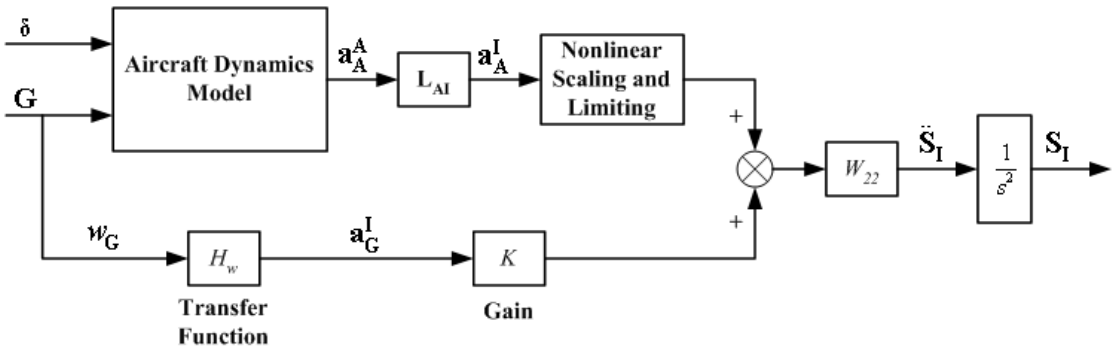


Figure 2.10. Optimal Algorithm Vertical Mode with Augmented Turbulence Channel.

The input to the augmented channel is the z-axis component w_G of the turbulence vector **G**. Reid and Robinson showed that w_G is the dominant turbulence component needed in producing vertical acceleration due to turbulence. The secondary flight equations can then be represented by a transfer function $H_G(s)$. The secondary acceleration $\mathbf{a}_G^I$ is then scaled with a constant gain K_G. Both the primary and secondary signals are then combined before input to the vertical motion cueing filter W_{22}.

From a simulated Being 757-200 aircraft test run, a system identification of aircraft vertical accelerations in response to turbulence was performed. The transfer function $H_G(s)$ was then created not only to represent the acceleration, but also incorporate some desired motion cueing characteristics, i.e., attenuated low-frequency content and increased high-frequency content. The following second-order transfer function was obtained for $H_G(s)$:

$$H_G(s) = 0.1\frac{(2.4s+1)(2.4s+1)}{(0.4s+1)(0.1s+1)}. \tag{2.8}$$

For the optimal algorithm, a gain of K_G equal to 0.8 was chosen to maximize the desired sensation of turbulence while sustaining the actuator extensions within the motion limits. A similar implementation to that shown in Figure 2.10 was applied for the nonlinear algorithm. In this approach, the linear cueing filter W_{22} was replaced with the nonlinear heave filter, with the gain K_G set equal to 1.2.

The transfer function given in Eq. (2.8) was implemented in state space form for the optimal and nonlinear algorithms in the subroutines WASHOPT3 and NEWOPT4 respectively. The transfer function coefficients and gain terms for each algorithm are initialized in the subroutines WINIT2 and WINIT4 respectively.

2.7. Trim Algorithm

Martin [8] reported that before going from the RESET mode to the OPERATE mode, several seconds are needed in the HOLD mode to produce pitch and roll angles that balance out any steady state longitudinal or lateral accelerations that may occur at time zero. In the subroutine WTRIM3 shown in Figure 2.11, a first-order washout filter is used to allow a smooth transition from the platform neutral state to the desired pitch angle:

$$\frac{\dot{\theta}(s)}{a_x(s)} = \frac{b_1 s}{s + a_0}. \tag{2.9}$$

Both b_1 and a_0 are chosen to produce the desired transition rate and attitude. As noted in Figure 2.11, Eq. (2.9) is integrated to produce the pitch attitude. Similar computations are performed to obtain the roll velocity $\dot{\phi}$, which is then integrated to produce the desired roll angle.

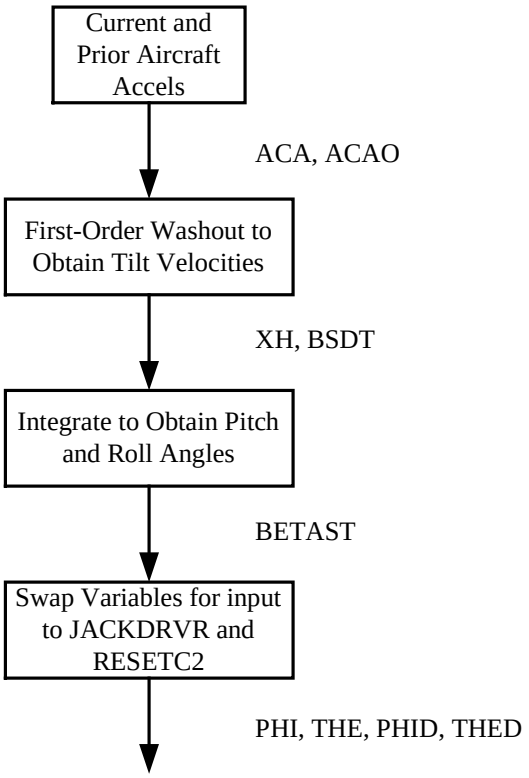


Figure 2.11. WTRIM3 Subroutine Flowchart.

3. Optimal Motion Cueing Algorithm

3.1. Algorithm Description

The theory and development of the optimal algorithm is well discussed by Telban and Cardullo [1]. The problem is to determine a transfer function matrix **W(s)** that relates the desired simulator motion input to the aircraft input such that a cost function constraining the pilot sensation error (between simulator and aircraft) is minimized. A mathematical model of the human vestibular system [1] is used in the filter development. The optimal algorithm uses an off-line program [1] to generate the desired transfer functions **W(s)** which are then implemented on-line. **W(s)** will relate the simulator commands to the aircraft states by $\mathbf{u_S} = \mathbf{W(s)} \times \mathbf{u_A}$. The block diagram for the on-line algorithm implementation is shown in Figure 3.1. Similar to the NASA adaptive algorithm [8], there are separate filtering channels for the translational and rotational degrees of freedom with the cross-feed path providing the tilt coordination cues.

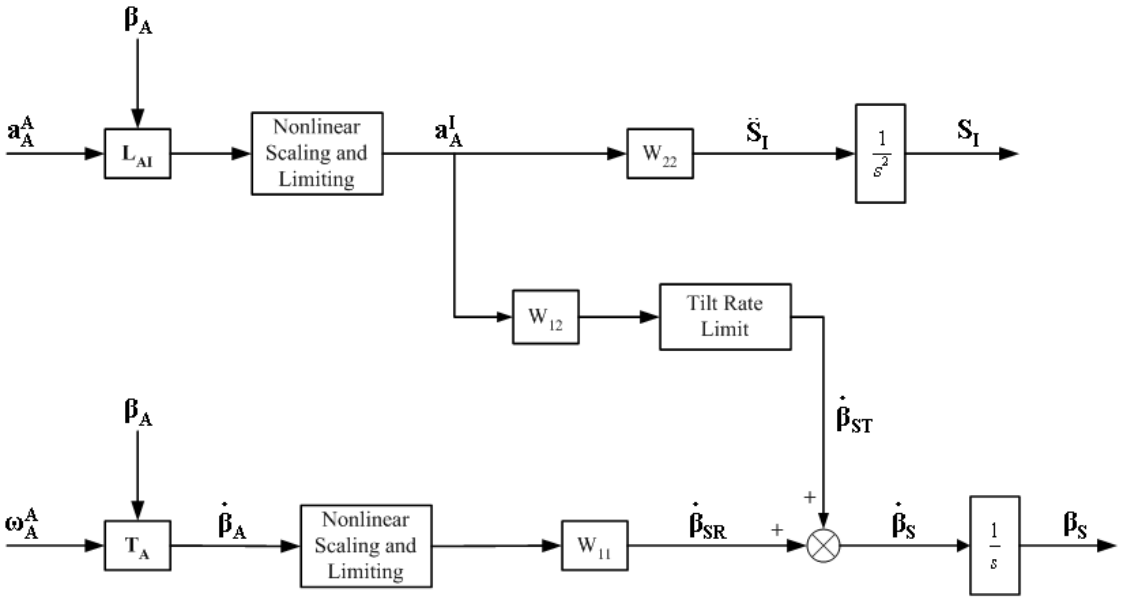


Figure 3.1. Optimal Algorithm Implementation.

The aircraft acceleration input vector is first transformed from the simulator body frame Fr_S to the inertial frame Fr_I as shown in Eq. (2.1). Nonlinear scaling in combination with limiting as described in Section 2.4 is then applied to the aircraft inputs. The scaled inertial acceleration $\mathbf{a}_{\mathbf{A}}^{\mathbf{I}}$ is then filtered through the translational filter W_{22} to produce a simulator translational acceleration command $\ddot{\mathbf{S}}_{\mathbf{I}}$. This acceleration is integrated twice to produce the simulator translational position command $\mathbf{S}_{\mathbf{I}}$.

The aircraft angular velocity input $\boldsymbol{\omega}_{\mathbf{A}}^{\mathbf{A}}$ is transformed to the Euler angular rate vector $\dot{\boldsymbol{\beta}}_{\mathbf{A}}$ as shown in Eq. (2.2), and is limited and scaled similar to the translational channel. This input is then passed through the rotational filter W_{11} to produce the vector $\dot{\boldsymbol{\beta}}_{\mathbf{SR}}$. The tilt coordination rate $\dot{\boldsymbol{\beta}}_{\mathbf{ST}}$ is formed from the acceleration $\mathbf{a}_{\mathbf{A}}^{\mathbf{I}}$ being passed through the tilt coordination filter W_{12}. The summation of $\dot{\boldsymbol{\beta}}_{\mathbf{ST}}$ and $\dot{\boldsymbol{\beta}}_{\mathbf{SR}}$ yields $\dot{\boldsymbol{\beta}}_{\mathbf{S}}$, which is then integrated to generate $\boldsymbol{\beta}_{\mathbf{S}}$, the simulator angular position command.

The simulator translational position $\mathbf{S}^{\mathbf{I}}$ and the angular position $\boldsymbol{\beta}_{\mathbf{S}}$ are then used to transform the simulator motion from degree-of-freedom space to actuator space as given in Eq. (2.7), generating the actuator commands required to achieve the desired platform motion.

The motion cueing filter *W(s)* is represented by a Laplace transform transfer function:

$$W(s) = \frac{b_n s^n + b_{n-1} s^{n-1} + \cdots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0}. \tag{3.1}$$

For numerical computation, Eq. (3.1) is realized in state space by the observable canonical form [9]:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \tag{3.2}$$

where the matrices **A** and **B** are

$$\mathbf{A} = \begin{bmatrix} -a_{n-1} & 1 & 0 & \cdots & 0 \\ -a_{n-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & 0 \\ -a_1 & 0 & 0 & \cdots & 1 \\ -a_0 & 0 & 0 & \cdots & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_{n-1} - a_{n-1}b_n \\ b_{n-2} - a_{n-2}b_n \\ \vdots \\ b_1 - a_1 b_n \\ b_0 - a_0 b_n \end{bmatrix},$$

and $\mathbf{y}(t) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}\mathbf{x}(t) + b_n\mathbf{u}(t)$.

The state equations of Eq. (3.2) are integrated with a 2nd-order Runge-Kutta method [10]. This ensures stable responses for sample rates of at least 32 Hz.

3.2. Program Implementation

The flowchart for the optimal algorithm subroutine WASHOPT3 is shown in Figure 3.2. The subroutine RESETC2 is similar to that described by Martin [8], providing a smooth return of the motion base to the neutral position in the reset mode. The subroutine WTRIM3 uses a set of first-order filters as shown in Section 2.7 to produce a smooth buildup of the motion platform pitch and roll tilt angles to the trim position based upon the aircraft trim states. The subroutine LIBA transforms the aircraft translational accelerations from the body to the inertial reference frame, and the aircraft angular velocities from the body to the Euler reference frame as discussed in Section 2.3 and shown in Figure 2.4. These transformed aircraft states then undergo a nonlinear scaling with the subroutine GAINOPT3 described in Section 2.4 and shown in Figure 2.6. The subroutine OFIL3 computes the responses to the linear filters for each mode. The subroutine INTEG4 then computes the desired simulator displacements and attitudes, taking the tilt coordination into account.

The flowchart for OFIL3 is given in Figure 3.3. Each filter is formulated in state space as shown in Eq. (3.2), and is integrated using the 2nd-order Runge-Kutta method. The outputs of OFIL3 are the desired motion cues that drive the simulator, i.e., three translational accelerations, three rotational angular velocities, and two tilt angular velocities that represent the tilt coordination cues.

The flowchart for INTEG3 is given in Figure 3.4. The simulator translational accelerations and angular velocities are first integrated to obtain the translational displacements and angular positions. The commanded pitch tilt position θ_{STL} is determined from both the "desired" tilt position θ_{ST} and the difference between the desired and commanded tilt positions:

$$\Delta\theta_{STL} = 0.005\left(\theta_{ST}^{(i)} - \theta_{STL}^{(i-1)}\right) + \left(\theta_{ST}^{(i)} - \theta_{ST}^{(i-1)}\right), \tag{3.3}$$

Where the superscripts *(i)* and *(i-1)* refer to the current and prior time steps. The commanded tilt position θ_{STL} is then computed, taking into account the tilt velocity limit $\dot{\theta}_{\lim} = 5$ deg/ sec and the time step Δt:

$$\theta_{STL}^{(i)} = \theta_{STL}^{(i)} + \max\left[-\dot{\theta}_{\lim}\Delta t, \min\left(\dot{\theta}_{\lim}\Delta t, \Delta\theta_{STL}\right)\right]. \tag{3.4}$$

The difference between the desired and commanded tilt angles is then used to generate additional translational response and achieve coordination between the translational and tilt channels:

$$x_{com} = x_{des} - R_{Sz}\left(\theta_{ST} - \theta_{STL}\right), \tag{3.5}$$

where R_{Sz} is the radius from the motion platform centroid to the pilot's head. Similar computations are performed to obtain the commanded roll tilt position $\phi_{STL}^{(i)}$ and the commanded displacement y_{com}.

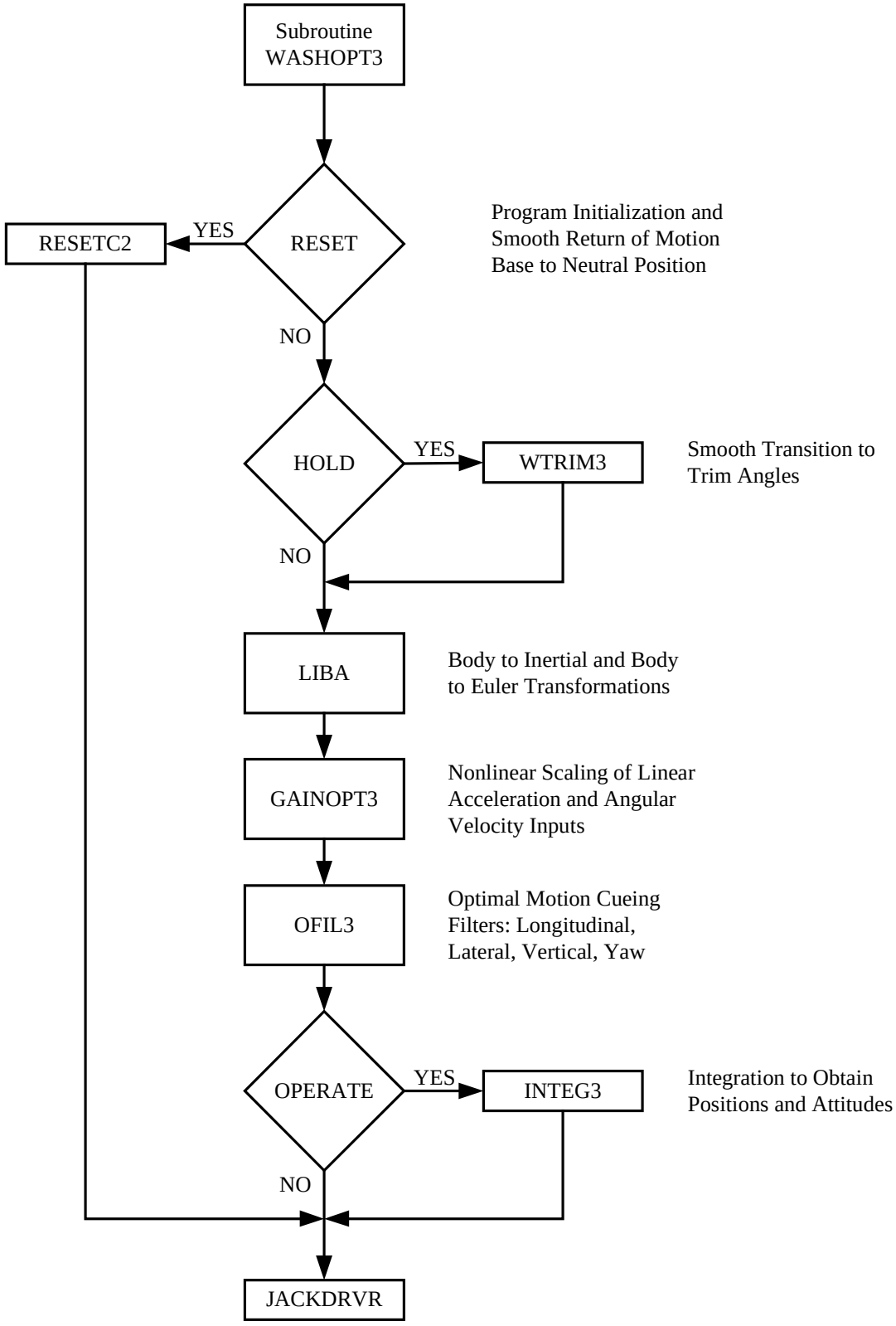


Figure 3.2. Optimal Algorithm Flowchart.

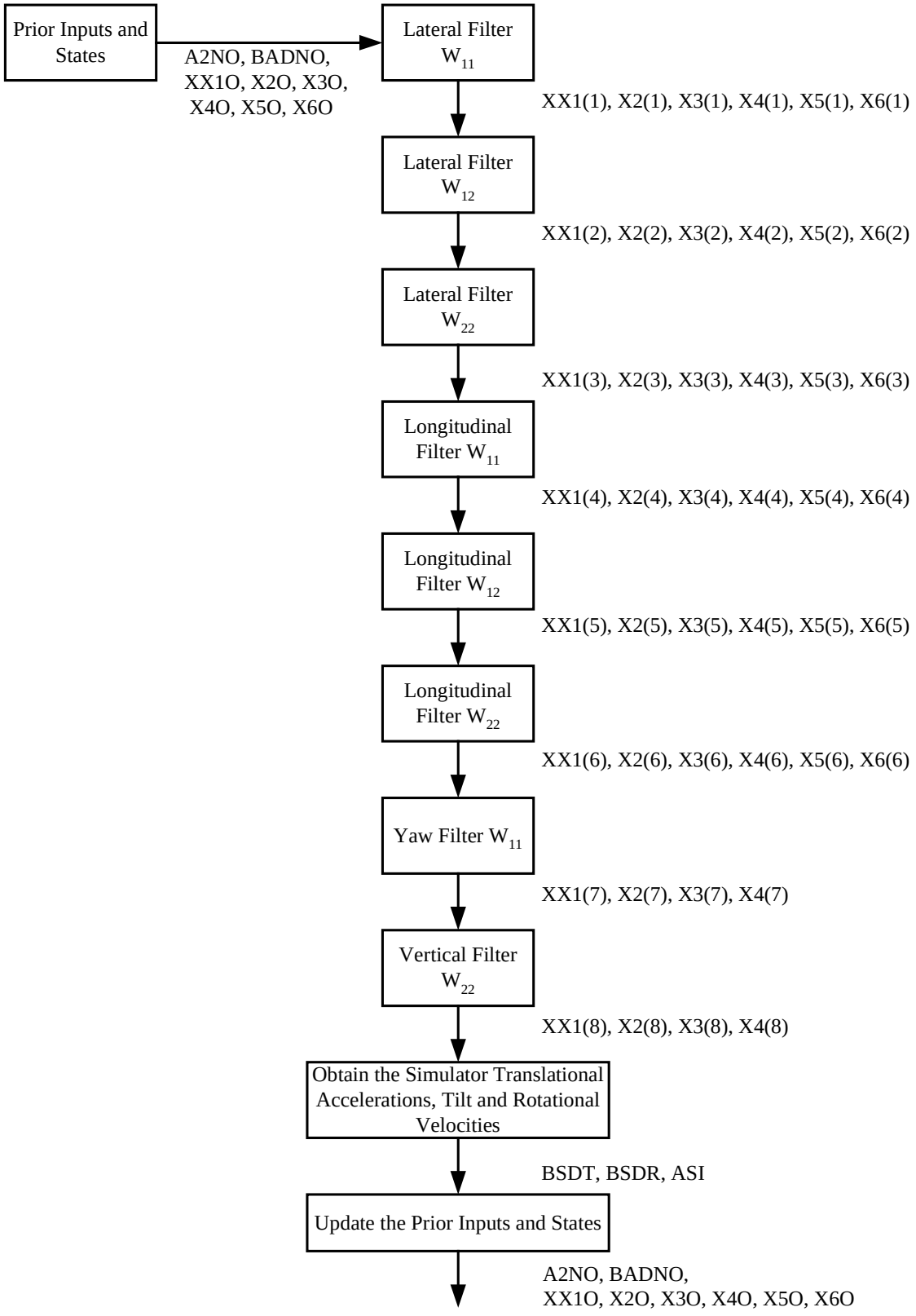


Figure 3.3. OFIL3 Subroutine Flowchart.

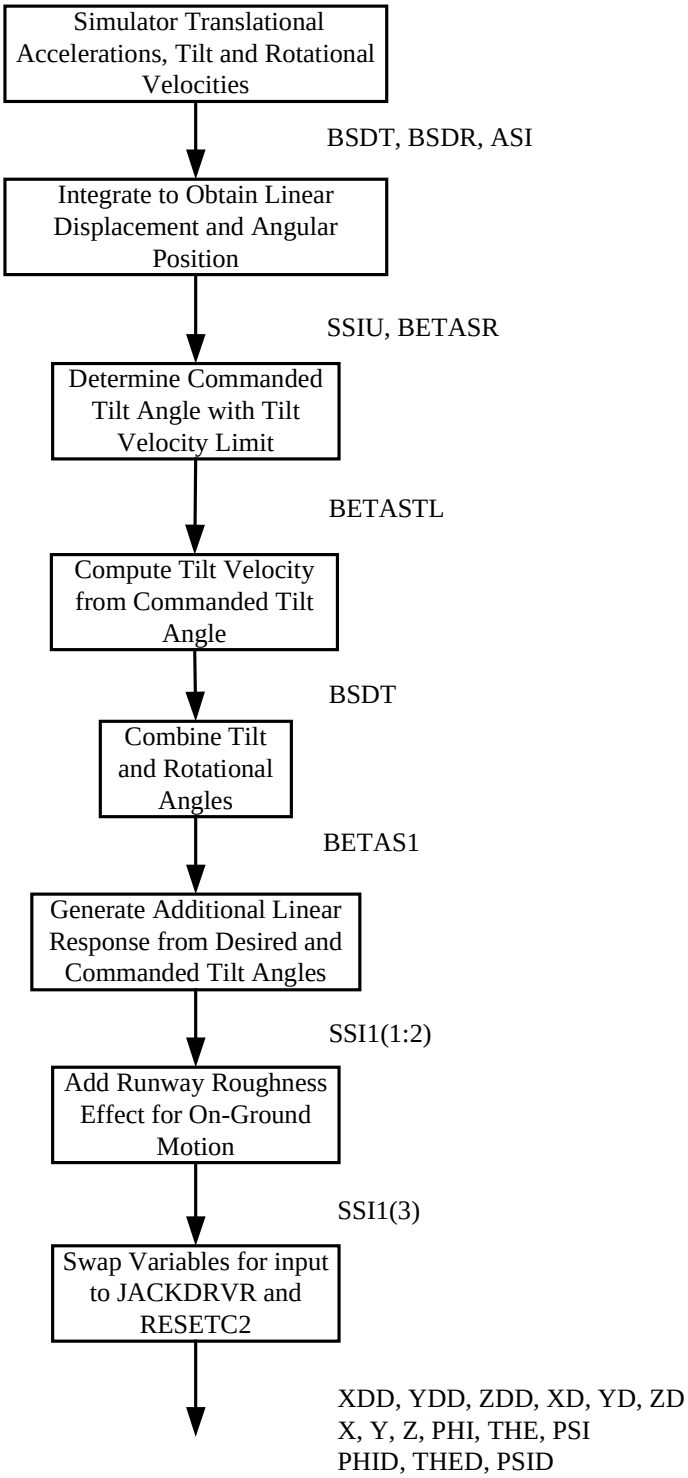


Figure 3.4. INTEG3 Subroutine Flowchart.

This page intentionally left blank.

4. Nonlinear Motion Cueing Algorithm

4.1. Algorithm Description

The theory and development of the nonlinear algorithm is well discussed by Telban and Cardullo [1]. The algorithm is formulated as a linear optimal control problem similar to the optimal algorithm, but is also updated in real time with a nonlinear control law. Furthermore, it incorporates models of the human vestibular sensation system along with an integrated visual-vestibular perception model [1]. The block diagram for the on-line implementation is shown in Figure 4.1. Similar to the optimal algorithm, there are separate filtering channels for the translational and rotational degrees of freedom with the cross-feed path providing the tilt coordination cues. Telban and Cardullo [1] reported that for the pitch and roll rotational channels, no benefit resulted from updating the Riccati equation in real time; thus the nonlinear filters are replaced with unity-gain filters.

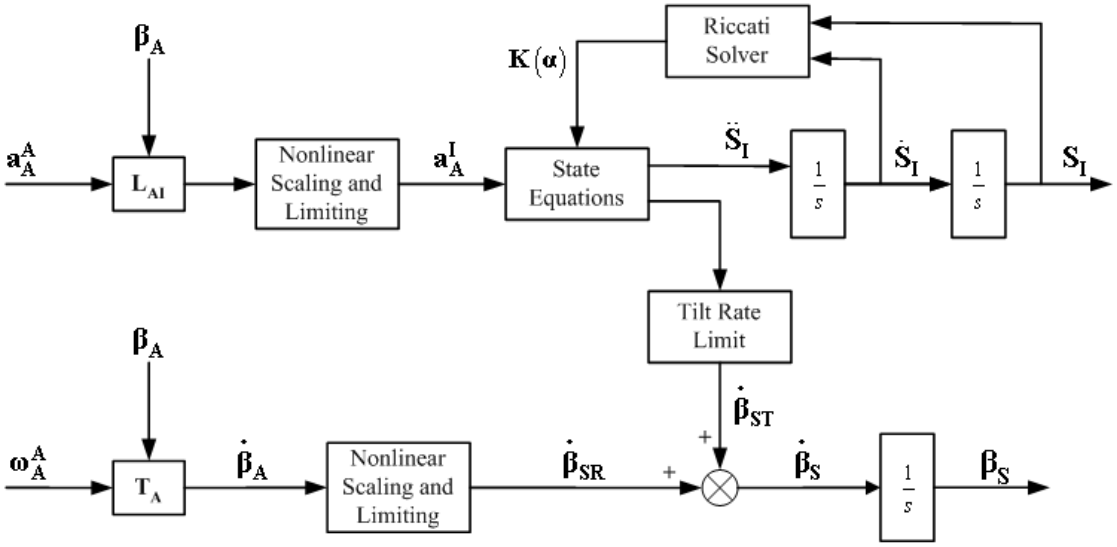


Figure 4.1. Nonlinear Algorithm Implementation with Unity-Gain Pitch Filter.

A nonlinear control law is implemented to generate a scalar coefficient α that is a function of the simulator motion system states:

$$\alpha = \mathbf{x}_\mathbf{d}^\mathbf{T}\mathbf{Q}_2\mathbf{x}_\mathbf{d}, \tag{4.1}$$

where $\mathbf{Q}_2$ is a weighting matrix that is at least positive semi-definite. As the computed system states increase in magnitude, i.e., with large commanded platform displacements and velocities, then α increases, resulting in faster control action to quickly wash out the platform to its neutral state. For small commands there will be limited control action, resulting in motion cues sustained for longer durations.

The solution of the algebraic Riccati equation is given as [1]

$$\left(\mathbf{A}' + \alpha\mathbf{I}\right)^\mathbf{T}\mathbf{P}\left(\alpha\right) + \mathbf{P}\left(\alpha\right)\left(\mathbf{A}' + \alpha\mathbf{I}\right) - \mathbf{P}\left(\alpha\right)\mathbf{B}\mathbf{R}_2^{-1}\mathbf{B}^\mathbf{T}\mathbf{P}\left(\alpha\right) + \mathbf{R}_1' = \mathbf{0}, \tag{4.2}$$

where $\mathbf{A}'$ and $\mathbf{B}$ are system matrices and $\mathbf{R}_1'$ and $\mathbf{R}_2$ are the standard optimal control weighting matrices defined by Telban and Cardullo [1] in the algorithm development. The system matrix $\mathbf{A}'$ is augmented with α times the identity matrix $\mathbf{I}$. The solution of Eq. (4.2) from the linear optimal algorithm that was computed off-line in MATLAB™ is used as the initial solution for the first time step. The Riccati equation of Eq. (4.2) is then updated in real time with a structured neural network developed by Ham and Collins [11] that is shown in Figure 4.2.

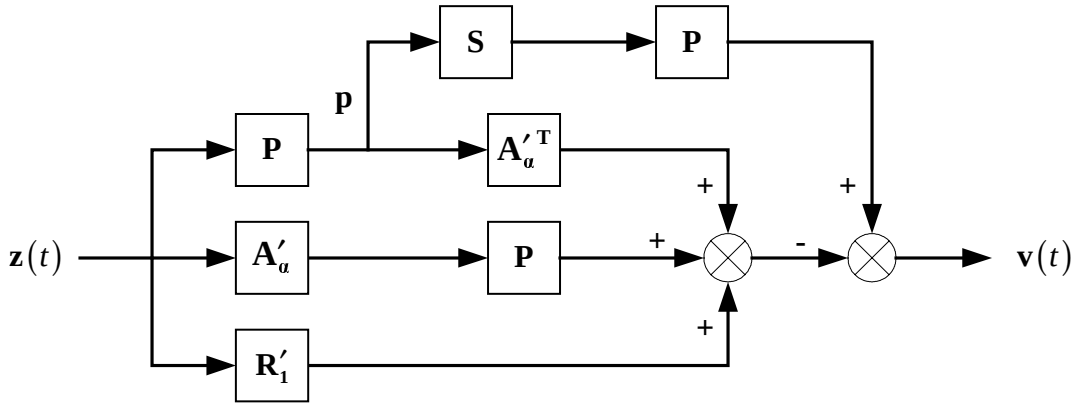


Figure 4.2. Structured Neural Network for Solving the Riccati Equation.

The error signal $\mathbf{v}(t)$ in Figure 4.2 is given as

$$\mathbf{v}(t) = \left[\mathbf{P}(t)\mathbf{S}\mathbf{P}(t) - \mathbf{A}_{\alpha}'^{\mathrm{T}}\mathbf{P}(t) - \mathbf{P}(t)\mathbf{A}_{\alpha}' - \mathbf{R}_1'\right]\mathbf{z}(t), \tag{4.3}$$

where $\mathbf{A}_{\alpha}' = \mathbf{A}' + \boldsymbol{\alpha}\mathbf{I}$ and $\mathbf{S} = \mathbf{B}^{\mathrm{T}}\mathbf{R}_2^{-1}\mathbf{B}$. The external excitatory vector input signals $\mathbf{z}(t)$ are a set of linearly independent bi-polar vectors given as

$$z^{(1)} = [1 \quad -1 \quad \cdots \quad -1],\ z^{(2)} = [-1 \quad 1 \quad \cdots \quad -1],\ z^{(n)} = [-1 \quad -1 \quad \cdots \quad 1], \tag{4.4}$$

where each vector $z^{(k)}$ is presented once to the neural network in an iteration, i.e., for one iteration there are a total of *n* presentations of the training step given in Eq. (4.4), with the solution $\mathbf{P}(k)$ updated with each training step. The update term $\Delta\mathbf{P}(k)$ is given as

$$\Delta\mathbf{P}(k) = \left[\mathbf{A}_{\alpha}'(k)\mathbf{v}(k)\mathbf{z}(k) + \mathbf{v}(k)\mathbf{z}^{\mathrm{T}}(k)\mathbf{A}_{\alpha}'(k) - \mathbf{v}(k)\mathbf{p}^{\mathrm{T}}(k)\mathbf{S}\right], \tag{4.5}$$

where $\mathbf{p}(t) = \mathbf{P}(t)\mathbf{z}(t)$ as shown in Figure 4.2. The updated Riccati equation solution $\mathbf{P}(k+1)$ is then computed:

$$\mathbf{P}(k+1) = \mathbf{P}(k) + \mu\Delta\mathbf{P}(k), \tag{4.6}$$

where $\mu > 0$ is the learning rate parameter.

The feedback matrix $\mathbf{K}(\alpha)$ is partitioned corresponding to the partition of the state vector **x** [1]:

$$\mathbf{u_S} = -\begin{bmatrix}\mathbf{K_1}(\alpha) & \mathbf{K_2}(\alpha)\end{bmatrix}\begin{bmatrix}\mathbf{x_e}\\ \mathbf{x_d}\end{bmatrix} - \mathbf{K_3}(\alpha)\mathbf{u_A}, \tag{4.7}$$

and the Riccati equation solution $\mathbf{P}(\alpha)$ can be partitioned as

$$\mathbf{P}(\alpha) = \begin{bmatrix}\mathbf{P_{11}}(\alpha) & \mathbf{P_{12}}(\alpha) & \mathbf{P_{13}}(\alpha)\\ \mathbf{P_{21}}(\alpha) & \mathbf{P_{22}}(\alpha) & \mathbf{P_{23}}(\alpha)\\ \mathbf{P_{31}}(\alpha) & \mathbf{P_{23}}(\alpha) & \mathbf{P_{33}}(\alpha)\end{bmatrix}, \tag{4.8}$$

where the partitions correspond to the partitions of the system matrix **A**. The partitioned feedback matrix is then given as

$$\begin{aligned}\mathbf{K_1}(\alpha) &= \mathbf{R_2^{-1}}\left[\mathbf{B_V^T P_{11}} + \mathbf{B_d^T P_{21}} + \mathbf{D_V^T Q C_V}\right]\\ \mathbf{K_2}(\alpha) &= \mathbf{R_2^{-1}}\left[\mathbf{B_V^T P_{12}} + \mathbf{B_d^T P_{22}}\right]\\ \mathbf{K_3}(\alpha) &= \mathbf{R_2^{-1}}\left[\mathbf{B_V^T P_{13}} + \mathbf{B_d^T P_{23}} - \mathbf{D_V^T Q C_V}\right],\end{aligned} \tag{4.9}$$

where the matrices $\mathbf{B_V}$, $\mathbf{B_d}$, and $\mathbf{D_V Q C_V}$ are defined by Telban and Cardullo [1], and by symmetry, $\mathbf{P_{12}} = \mathbf{P_{21}^T}$. The resulting state equations are then computed in real time:

$$\begin{bmatrix}\dot{\mathbf{x}}_\mathbf{e}\\ \dot{\mathbf{x}}_\mathbf{d}\end{bmatrix} = \begin{bmatrix}\mathbf{A_V} - \mathbf{B_V K_1}(\alpha) & -\mathbf{B_V K_2}(\alpha)\\ -\mathbf{B_d K_1}(\alpha) & \mathbf{A_d} - \mathbf{B_d K_2}(\alpha)\end{bmatrix}\begin{bmatrix}\mathbf{x_e}\\ \mathbf{x_d}\end{bmatrix} + \begin{bmatrix}-\mathbf{B_V}\left(\mathbf{I} + \mathbf{K_3}(\alpha)\right)\\ -\mathbf{B_d K_3}(\alpha)\end{bmatrix}\mathbf{u_A}, \tag{4.10}$$

with the matrices $\mathbf{A_V}$ and $\mathbf{A_d}$ defined by Telban and Cardullo [1]. These state equations are then integrated with a 2nd-order Runge-Kutta method [10].

4.2. Program Implementation

The nonlinear algorithm was developed to achieve the desired motion cues at an update rate of 60 Hz. Since the computer image generator, which provides the out-the-window visual imagery to the simulator pilot, also runs at 60 Hz, the motion cues would

be synchronous with the visual cues. However, because of the computer operating system, the real time operating system and the I/O system on the Langley real time computing system, the minimum interval must be an integer multiple of 125 μsec and a power of 2. Therefore, a time step of 16 msec or an update rate of 62.5 Hz was selected for the real time implementation and the pilot tests.

The flowchart for the nonlinear algorithm subroutine NEWOPT4 is shown in Figure 4.3. The RESET/HOLD modes and transformation subroutines are identical to the optimal algorithm discussed in the preceding section. The transformed aircraft states then undergo a nonlinear scaling with the subroutine GAINOPT4.

The subroutine NFILX accomplishes the task of solving the Riccati equation in real time for the longitudinal mode. Similarly, NFILY, NFILZ, and NFILR compute the Riccati equation solutions for the lateral, vertical, and yaw mode respectively. The feedback matrices and updated motion states for each mode are then computed in the subroutine STATE4. The subroutine INTEG4 computes the desired simulator displacements and attitudes, taking into account the tilt coordination limits. Note that STATE4 is computed four times when in the HOLD mode; this was done to produce faster convergence to the neutral state while eliminating discontinuities when transitioning from the HOLD to the OPERATE mode.

The flowchart for NFILX is given in Figure 4.4. The coefficient α is computed from the simulator motion states given in Eq. (4.1). For each training step, the corresponding input vector $\mathbf{z}$ given in Eq. (4.4) is referenced. The update term $\mathbf{\Delta P}$ given in Eq. (4.5) and its matrix transpose $\mathbf{\Delta P^T}$ are then computed and used to update the Riccati equation solution $\mathbf{P}(k+1)$ given in Eq. (4.6). The flowcharts for NFILY,

NFILZ, and NFILR are given in Figure 4.5, Figure 4.6, and Figure 4.7 respectively. The variable flow for each subroutine is identical to that shown for NFILX in Figure 4.4. Note that each subroutine uses the motion states specific to its mode for computing the coefficient α, with the system variables defined for each mode ending with the same letter as the subroutine name.

During implementation on the NASA Langley real time system, it was discovered that for a time step of 16 msec, the real time requirement for the nonlinear algorithm was not being met. The baseline software for the nonlinear algorithm, with all presentations of the excitatory vector $\mathbf{z}(t)$ given for each mode given in Eq. (4.4), i.e., n presentations for an n^{th}-order system, resulted in an average CPU time of 34 msec. This CPU time also includes the contributions of the aircraft model and control loader. In the subroutines NFILX, NFILY, NFILZ, and NFILR, matrix operations were optimized by taking advantage of the matrices $\mathbf{A}'$ and $\mathbf{S}$ being sparse, i.e., eliminating computations involving matrix elements of zero. In addition, the symmetry of the Riccati solution $\mathbf{P}$ and other matrices computed in the subroutines were also exploited to reduce computation. Optimizing matrix operations resulted in an average CPU time of 24 msec. Reducing the number of presentations of the vector $\mathbf{z}(t)$ for each mode to the first three vectors produced a CPU time of 12 msec, which meets the real time requirement. Telban and Cardullo [1] reported that this change resulted in imperceptible degradation of the motion cues.

The flowchart for STATE4 is given in Figure 4.8. The variable flow for computing the longitudinal mode simulator states is shown in detail. The partitions of the feedback matrix are computed as given in Eq. (4.9). The state equations are then updated

as given in Eq. (4.10) and integrated with a 2^{nd}-order Runge-Kutta method, resulting in the updated simulator motion states. The computation of the lateral, vertical, and yaw states follows the same variable flow as the longitudinal states, with the system variable names corresponding to the respective mode.

The flowchart for INTEG4 is given in Figure 4.9. The simulator translational accelerations, rotational angular velocities, and tilt angular velocities are computed as given in Eq. (4.7). The remaining computations are identical to those performed in the subroutine INTEG3 for the optimal algorithm as shown in Figure 3.4, and discussed in Section 3.2.

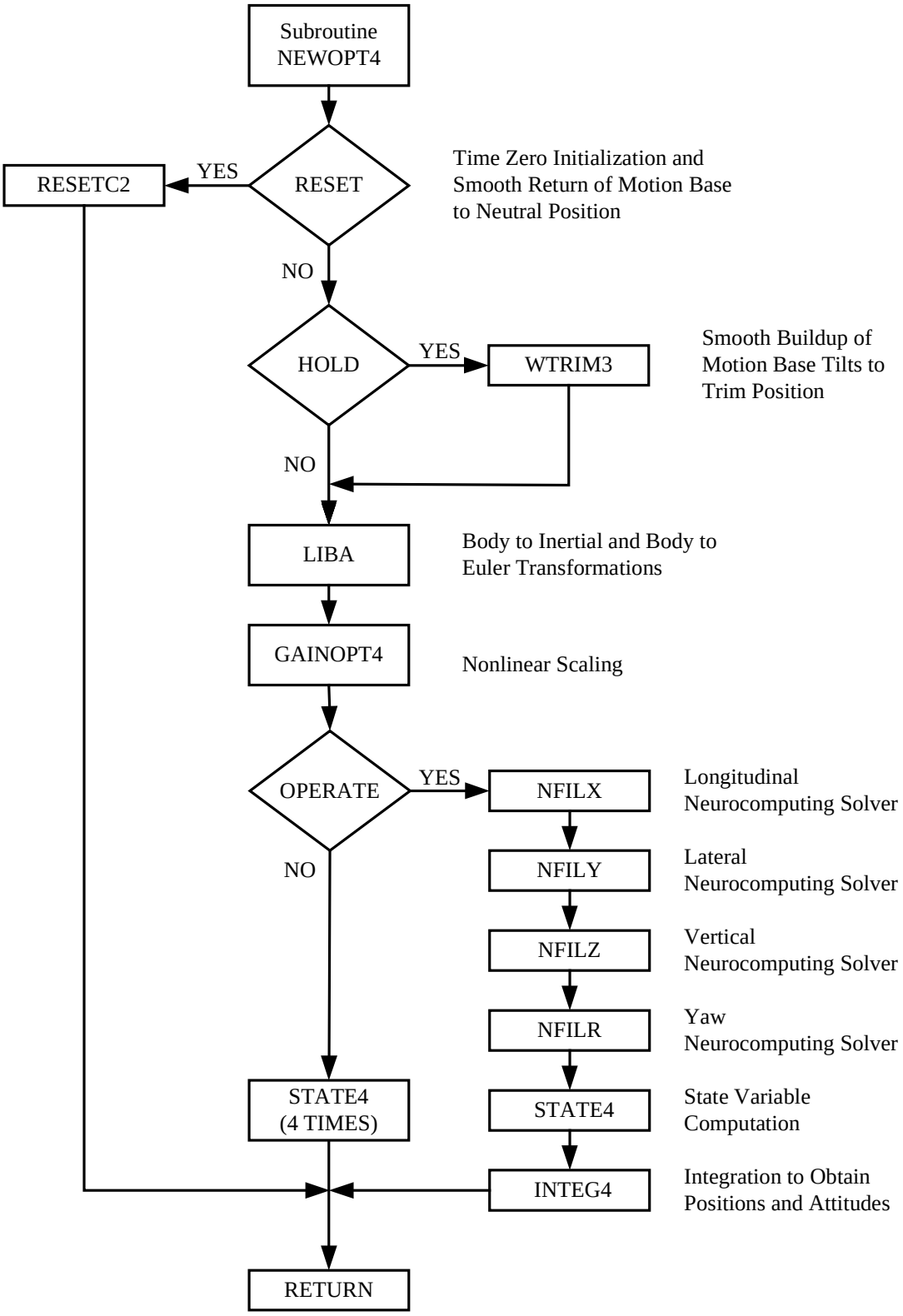


Figure 4.3. Nonlinear Algorithm Flowchart.

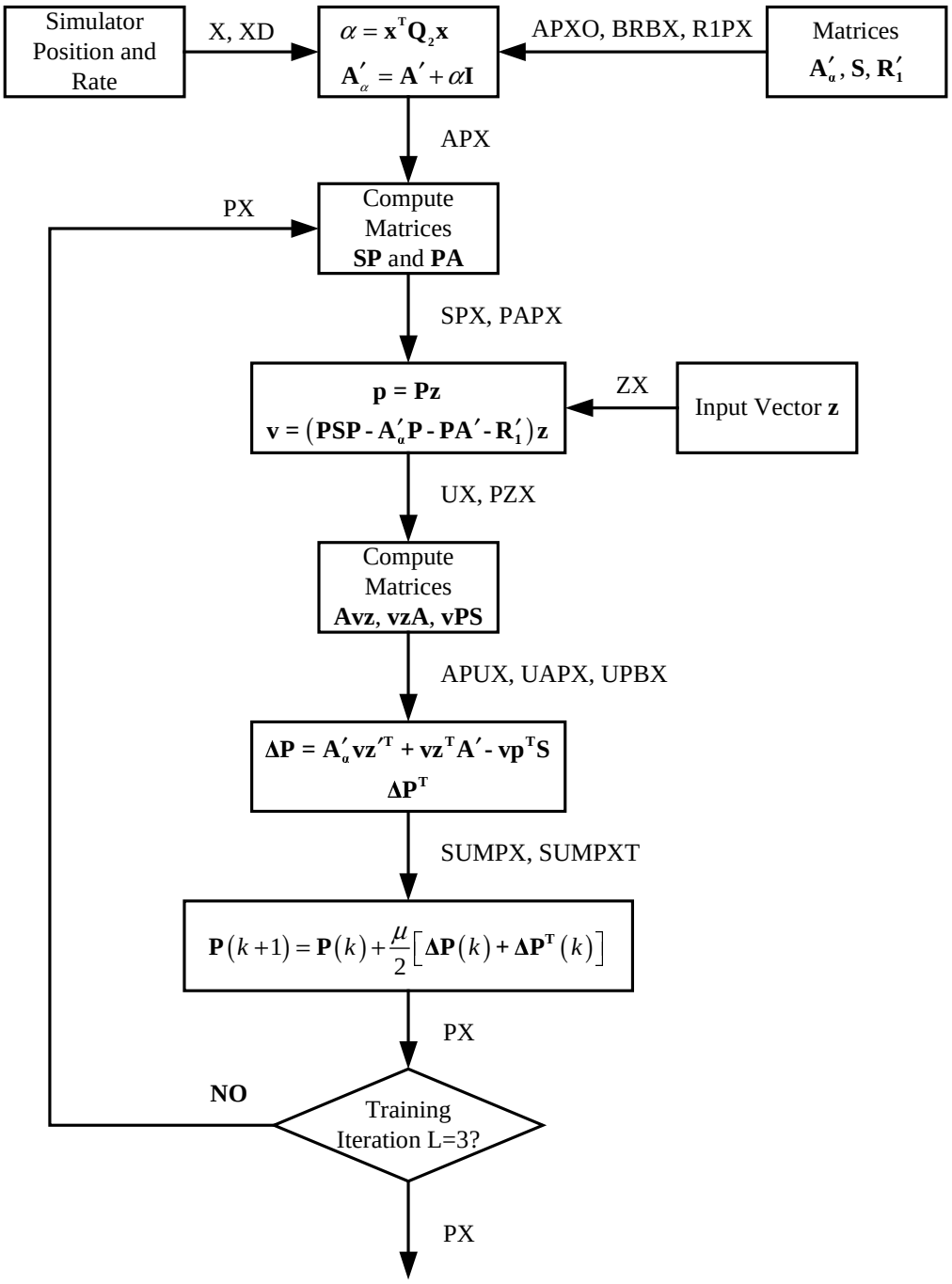


Figure 4.4. NFILX Subroutine Flowchart.

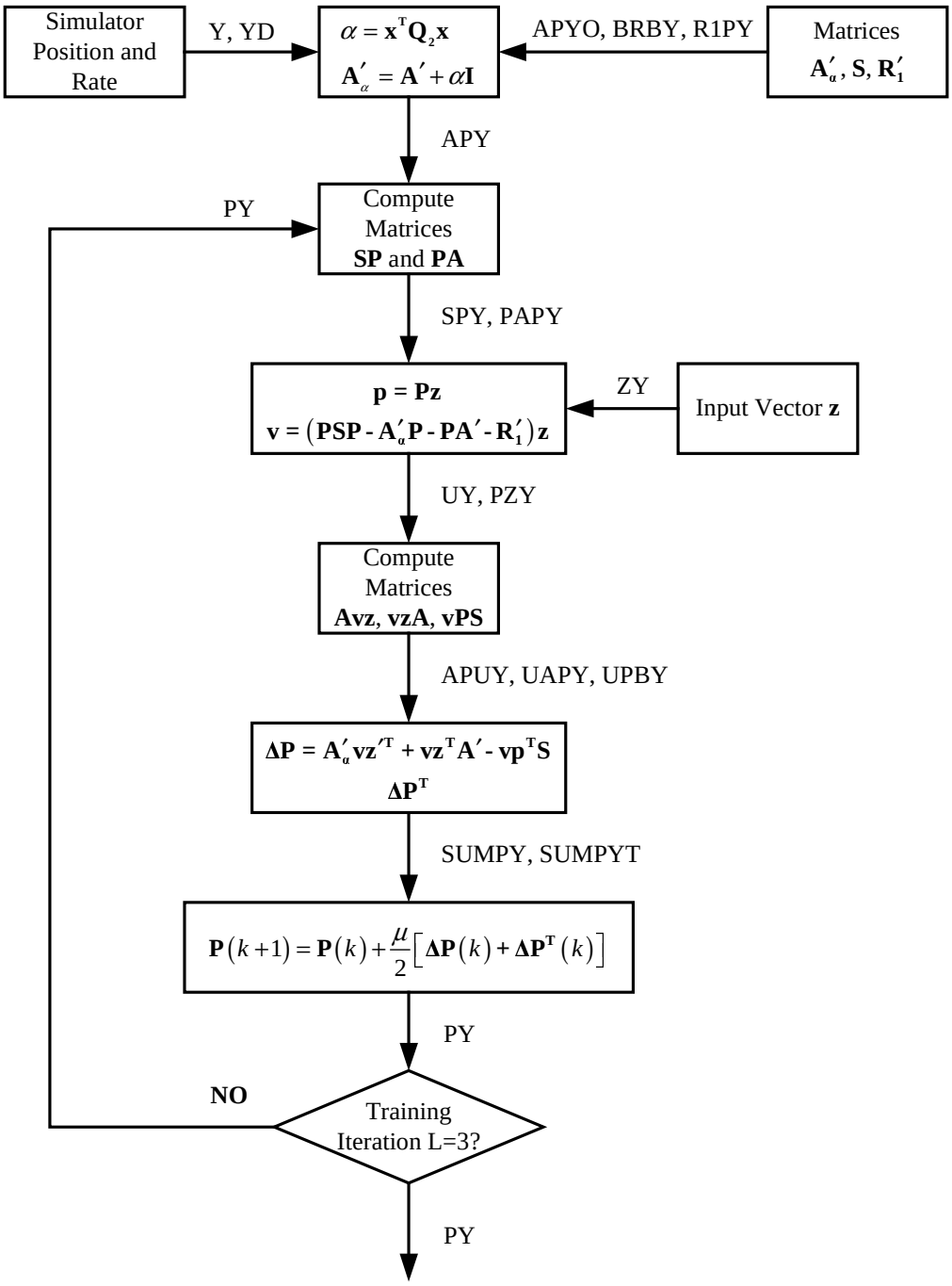


Figure 4.5. NFILY Subroutine Flowchart.

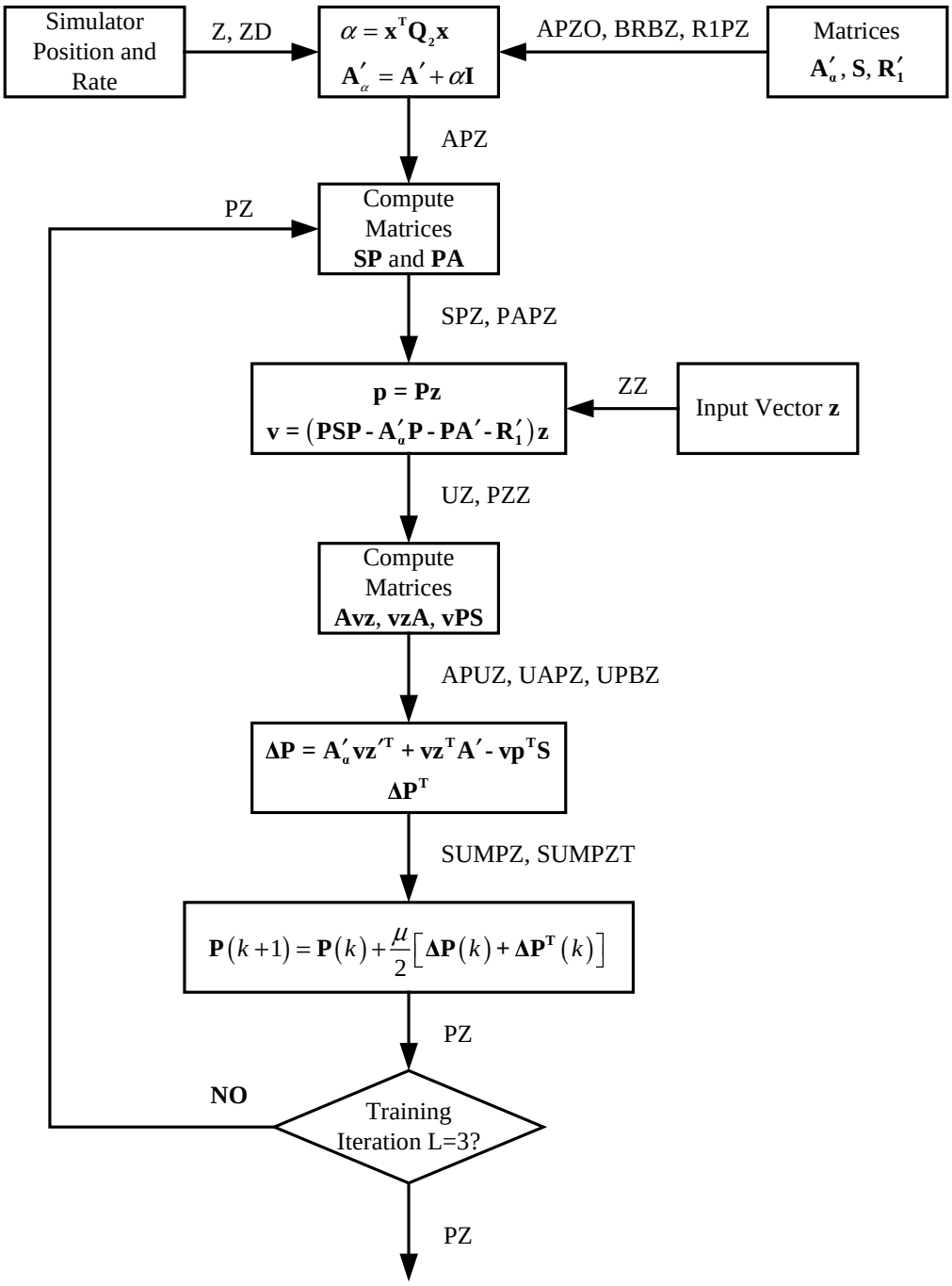


Figure 4.6. NFILZ Subroutine Flowchart.

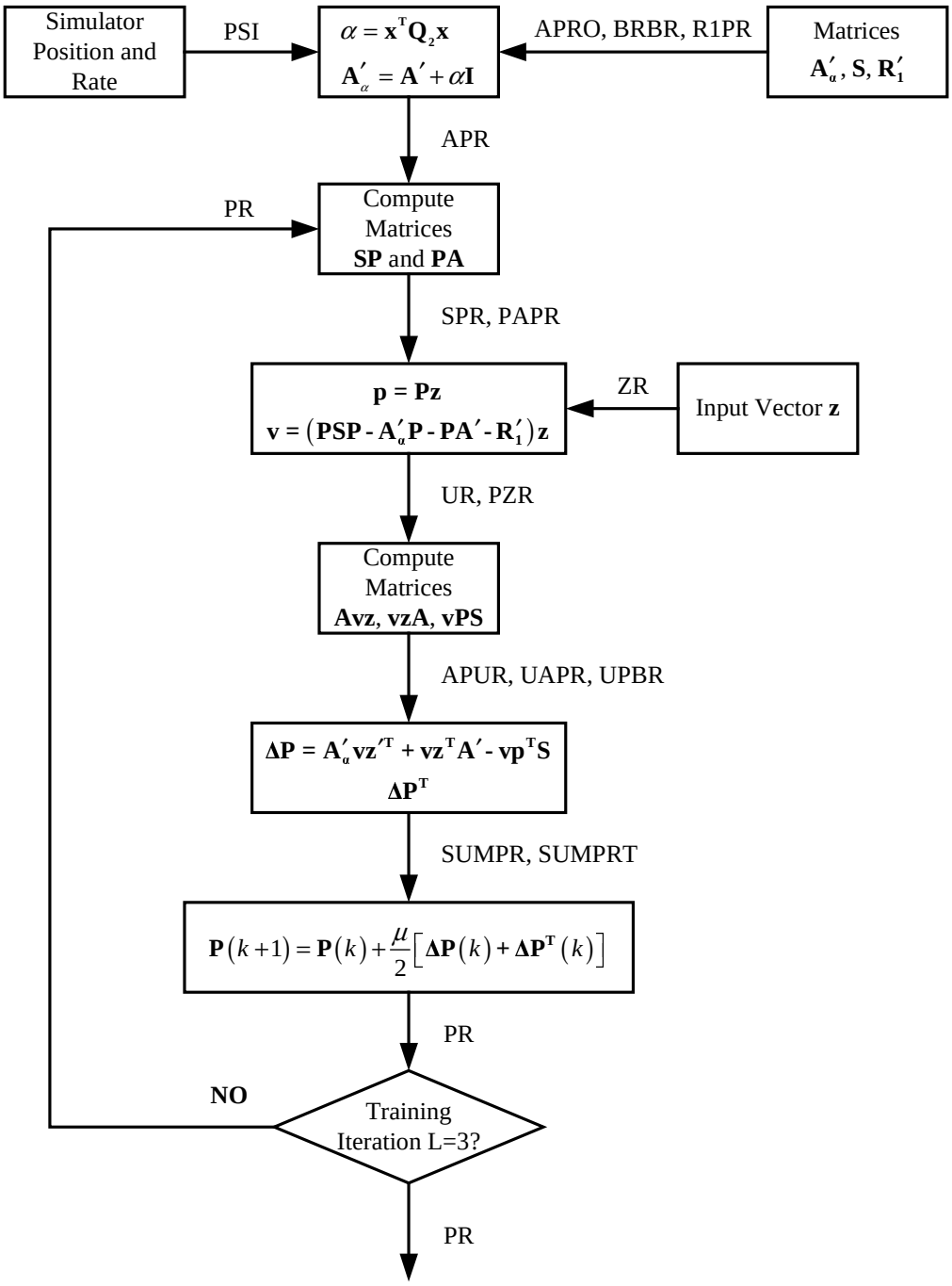


Figure 4.7. NFILR Subroutine Flowchart.

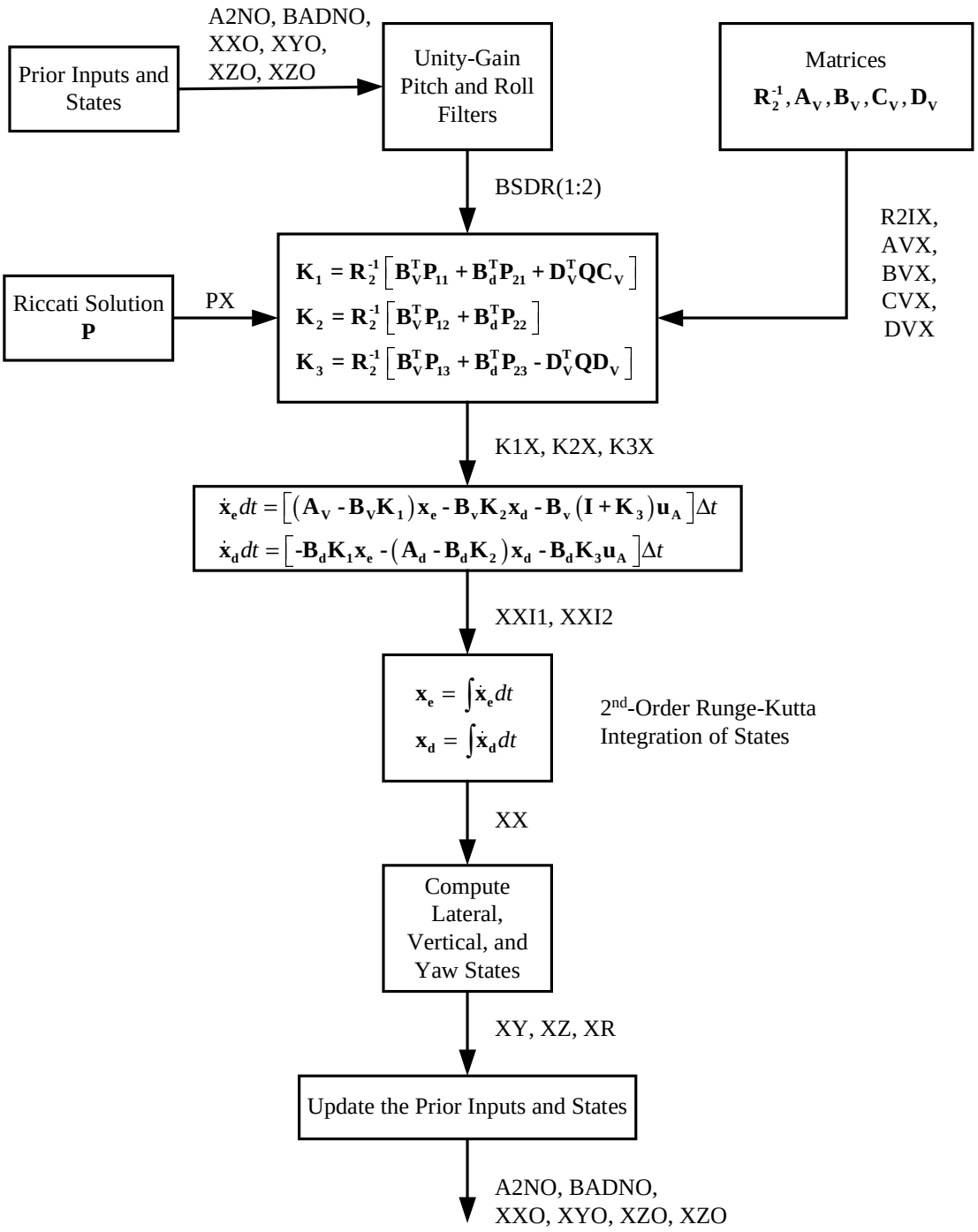


Figure 4.8. STATE4 Subroutine Flowchart.

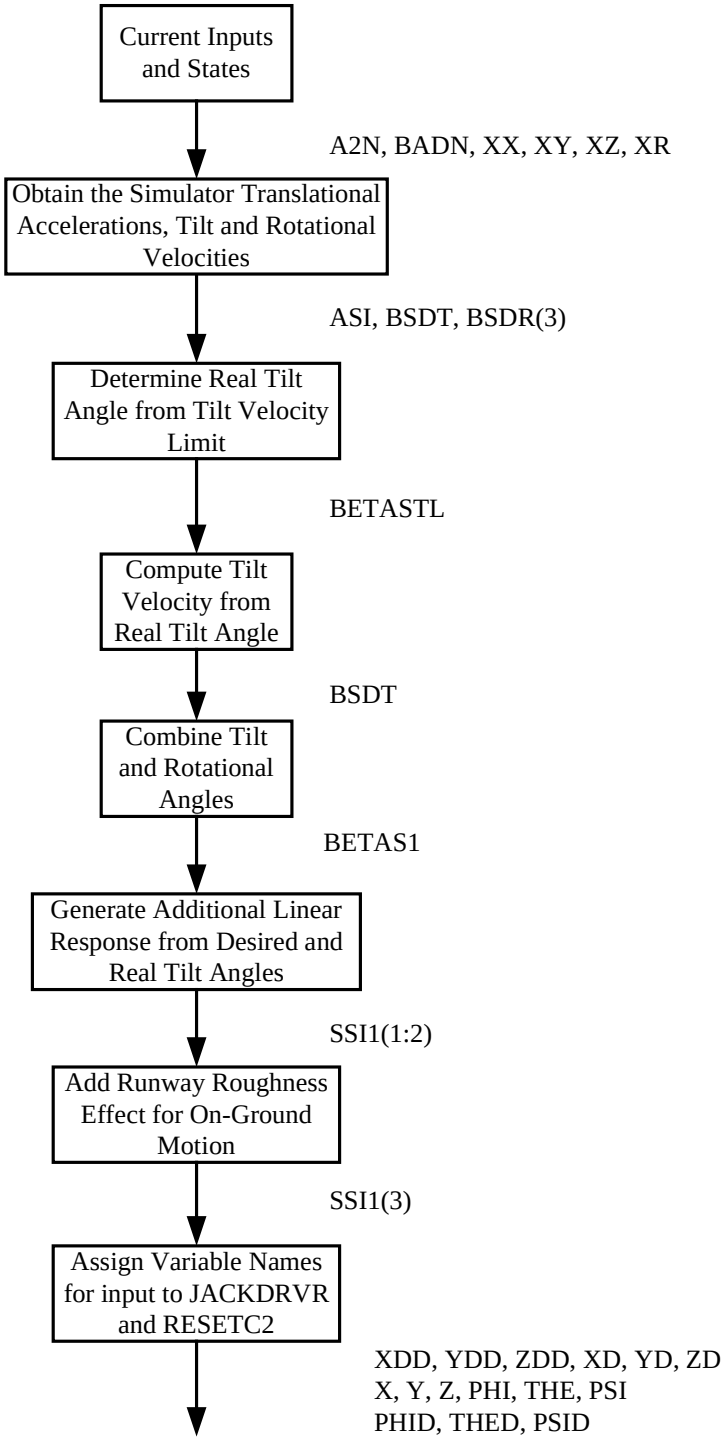


Figure 4.9. INTEG4 Subroutine Flowchart.

5. Nonlinear Algorithm Tuning

This section covers the procedures needed for tuning the nonlinear algorithm in order to produce the desired pilot performance. It is expected that the algorithm will require tuning with either a change of aircraft dynamics (the initial pilot tuning was done with the Boeing 757-200 aircraft model) or implementation on a new motion system such as the Cockpit Motion Facility (CMF).

The "off-line" development of the nonlinear algorithm mentioned in Section 4.1 and discussed in greater detail by Telban and Cardullo [1], resulted in a set of filters consisting of system matrices and an initial Riccati equation solution implemented for each mode. Each filter was optimized with single degree-of-freedom responses for synergistic six-degree-of-freedom motion systems similar to Figure 2.7. Except for special circumstances driven by suggested future research [1], these filters do not need to be re-synthesized. Only the nonlinear scaling coefficients for each degree-of-freedom, given in Eq. (2.3), will need to be adjusted to accommodate changes in aircraft dynamics or motion platform geometry.

It is expected that the optimal algorithm discussed in Section 3 will be used infrequently. However, in the event that future work with the optimal algorithm is desired, the same procedures described in this section for tuning the nonlinear algorithm gains can be applied.

5.1. Prior Tuning Experience

In order to determine the nonlinear scaling (gain) coefficients for each degree-of-freedom that resulted in the most desired pilot performance, a trained simulator pilot executed a series of pilot controlled maneuvers with the optimal algorithm on the Visual

Motion Simulator (VMS). A series of maneuvers were first executed with the coefficients determined prior to testing. Coefficients for each degree-of-freedom were then adjusted until the simulator pilot subjectively felt the desired perception and performance were reached, while ensuring that the simulator motion platform limits were not exceeded. The following maneuvers were executed for both algorithms:

Straight Approach and Landing (with varying wind from head to tail)
Offset Approach and Landing (with and without turbulence)
Pitch, Roll, and Yaw Doublets
Throttle Increase and Decrease
Coordinated Turn
Ground Maneuvers (taxiing, effect of aircraft brakes)
Takeoff from Full Stop.

The optimal algorithm resulted in motion cues with which the simulator pilot commented he had more control and confidence in comparison to the NASA adaptive algorithm. For both pitch and roll doublets, a fast response was observed when changing directions. On takeoffs, the optimal algorithm was found to be easier to pitch up to the desired attitude and control the aircraft. A noticeably large side force was observed with the coordinated turn maneuver. By reducing the gains for the roll degree-of-freedom, this side force was reduced to a minimal sensation. The pitch gains were decreased to reduce the likelihood of entering the braking region or exceeding the actuator limits. Reducing the gains for both roll and pitch degrees-of-freedom still yielded acceptable motion cues.

The severe turbulence effects that were included with the offset approach and landing maneuver were hardly noticeable. An increase of the vertical gain coefficients resulted in increased cues, but still less than satisfactory. This increase in the vertical gains (coupled with an increase of the surge gains) resulted in forward surge cues that are more coordinated with the pitch cues, and a larger aft surge cue (initially, the aft cue was

noticeably smaller than the forward cue). The unsatisfactory turbulence cues resulted in the inclusion of the augmented motion cue driven by the vertical gust.

A second pilot tuning evaluation was later performed on the VMS with both the optimal and nonlinear algorithms, with augmented turbulence cues implemented for both algorithms. A series of maneuvers were first executed with the polynomial gain coefficients determined prior to testing. Coefficients for each degree-of-freedom were then adjusted until the simulator pilot subjectively felt the desired perception and performance were reached, while ensuring that the simulator motion platform limits were not exceeded. The following maneuvers were executed for both algorithms:

Straight Approach and Landing (with varying wind from head to tail)
Offset Approach and Landing (with and without turbulence)
Takeoff from Full Stop (with and without engine failure)
Ground Maneuvers (taxiing, effect of aircraft brakes).

No additional tuning was needed for either the straight-in or offset approach maneuvers. However, both algorithms showed a tendency to exceed the actuator limits of the motion system with the takeoff maneuver. Reducing the surge gains for the optimal algorithm and both the surge and pitch gains for the nonlinear algorithm resulted in platform motion within the actuator limits during the takeoff maneuvers. The augmented turbulence gain terms for the optimal and nonlinear algorithms discussed in Section 2.6 were adjusted to produce the desired turbulence cues.

Table 5.1 lists the resulting nonlinear gains by degree-of-freedom implemented for each algorithm. From Eq. (2.3), the coefficients c_1, c_2, and c_3 are given for each degree-of-freedom.

Table 5.1. Nonlinear Gain Coefficients for the Cueing Algorithms.

	Optimal Algorithm			Nonlinear Algorithm		
Degree-of-Freedom	C1	C2	C3	C1	C2	C3
Surge (X)	0.6	-0.055	0.002	0.5	-0.05	0.002
Sway (Y)	0.5	-0.055	0.002	0.4	-0.035	0.001
In-Air (Z)	0.6	-0.082	0.0038	0.6	-0.082	0.0038
On-Ground (Z)	1.3	-0.0375	0.0003	2.0	-0.05	0.0
Roll (p)	0.3	-0.3	0.1	0.3	-0.3	0.1
Pitch (q)	0.4	-0.54	0.26	0.3	-0.3	0.1
Yaw (r)	1.1	-1.46	0.64	1.1	-1.46	0.64

5.2. Tuning Procedure

Nonlinear gain coefficients can be computed from Eq. (2.3) for each degree-of-freedom with the use of an Excel™ spreadsheet. Using this approach, a set of coefficients for each degree-of-freedom can be tested with piloted simulations, resulting in the desired nonlinear gains for the cueing algorithm. Table 5.2 provides five sets of coefficients computed for each in-air degree-of-freedom (excluding yaw) for the nonlinear algorithm, with an additional set of coefficients for the on-ground heave gain. The yaw coefficients are omitted since these gains remained unchanged with initial tuning of both algorithms.

Each degree-of-freedom consists of five sets of gain coefficients, which are labeled to identify both the degree-of-freedom and the coefficient set, e.g., the roll degree-of-freedom coefficient sets are labeled from NP1 to NP5. The nonlinear gain coefficients for the nonlinear algorithm are initialized in data statements in the subroutine WINIT4. These data statements can be edited as individual degree-of-freedom coefficient sets are updated during the tuning process.

Table 5.2. Nonlinear Gain Coefficient Pilot Test Spreadsheet.

ROLL

	S0	S1	XMAX	YMAX	GP4(1)	GP4(2)	GP4(3)
NP1	0.26	0	1	0.1	0.26	-0.22	0.06
NP2	0.28	0	1	0.1	0.28	-0.26	0.08
NP3	0.3	0	1	0.1	0.3	-0.3	0.1
NP4	0.32	0	1	0.1	0.32	-0.34	0.12
NP5	0.34	0	1	0.1	0.34	-0.38	0.14

PITCH

	S0	S1	XMAX	YMAX	GQ4(1)	GQ4(2)	GQ4(3)
NQ1	0.3	0.1	1	0.1	0.3	-0.4	0.2
NQ2	0.4	0.1	1	0.14	0.4	-0.48	0.22
NQ3	0.4	0.1	1	0.18	0.4	-0.36	0.14
NQ4	0.5	0.1	1	0.18	0.5	-0.56	0.24
NQ5	0.5	0.1	1	0.21	0.5	-0.47	0.18

SURGE

	S0	S1	XMAX	YMAX	GX4(1)	GX4(2)	GX4(3)
NX1	0.5	0.1	10	2	0.5	-0.05	0.002
NX2	0.5	0.1	10	2.5	0.5	-0.035	0.001
NX3	0.6	0.1	10	2.5	0.6	-0.055	0.002
NX4	0.7	0.1	10	2.5	0.7	-0.075	0.003
NX5	0.7	0.1	10	3	0.7	-0.06	0.002

SWAY

	S0	S1	XMAX	YMAX	GY4(1)	GY4(2)	GY4(3)
NY1	0.3	0	10	1.2	0.3	-0.024	0.0006
NY2	0.4	0	10	1.2	0.4	-0.044	0.0016
NY3	0.4	0	10	1.5	0.4	-0.035	0.001
NY4	0.5	0	10	1.5	0.5	-0.055	0.002
NY5	0.5	0	10	1.8	0.5	-0.046	0.0014

HEAVE (IN-AIR)

	S0	S1	XMAX	YMAX	GZ40(1)	GZ40(2)	GZ40(3)
NZ1	0.6	0.1	10	1.6	0.6	-0.082	0.0038
NZ2	0.6	0.1	10	2	0.6	-0.07	0.003
NZ3	0.7	0.1	10	2	0.7	-0.09	0.004
NZ4	0.7	0.1	10	2.4	0.7	-0.078	0.0032
NZ5	0.8	0.1	10	2.4	0.8	-0.098	0.0042

HEAVE (ON-GROUND)

	S0	S1	XMAX	YMAX	GZ4S(1)	GZ4S(2)	GZ4S(3)
NL1	1.6	0	20	20	1.6	-0.010	-0.0010
NL2	1.8	0	20	20	1.8	-0.030	-0.0005
NL3	2	0	20	20	2	-0.050	0.0000
NL4	2.2	0	20	20	2.2	-0.070	0.0005
NL5	2.4	0	20	20	2.4	-0.090	0.0010

Prior to adjustment of individual degree-of-freedom gain coefficient sets, it is suggested that an experienced simulator pilot execute a series of maneuvers to determine the baseline pilot performance for the current gain set, noting deficiencies for each degree-of-freedom that may arise from specific maneuvers. From prior piloted tuning tests, the following maneuvers are recommended for large transport aircraft:

Pitch Stick, Roll Stick, and Rudder Pedal Doublets
Throttle Increase and Decrease
Takeoff from Full Stop
Straight Approach and Landing (with and without turbulence)
Coordinated Turn
Offset Approach and Landing
Ground Maneuvers (taxiing, effect of aircraft brakes).

Table 5.3 lists the maneuvers, the degrees of freedom that require tuning for the maneuver, and the desired outcomes that result from the tuning. The maneuvers should be executed in the order listed in Table 5.3. Adjustment of gains for individual degrees of freedom should be done repeatedly for each maneuver. In general, individual degree-of-freedom gain coefficient sets should be adjusted until the desired motion cues and pilot performance result for each maneuver, while ensuring that the platform does not exceed the actuator extension limits. Following execution of the maneuvers with tuning of the degree-of-freedom nonlinear gain sets, the simulator pilot should repeat execution of all maneuvers, noting any final deficiencies in the motion cues and piloted performance that may persist. If necessary, additional fine tuning of any degree-of-freedom can then be performed.

The augmented turbulence gain K_G given in Figure 2.10 can be increased or decreased to produce the desired turbulence cues within the capabilities of the motion platform. An adjustment of K_G may also be sufficient with a change of the aircraft

dynamics; a large transport aircraft similar to a Boeing 757-200 should produce similar turbulence cues. The transfer function time constants given in Eq. (2.8) can also be adjusted to produce the desired motion cues for greater changes in aircraft dynamics. The lead time constants in the numerator can be increased to increase the magnitude of the low-frequency cues. Decreasing the time constants in the denominator will increase the magnitude of the high-frequency cues, but these terms should be large enough so that the transfer function can be integrated within the simulation time step.

Table 5.3. Maneuvers with Degree-of-Freedom Gains.

Maneuver	Degree-of-Freedom	Comments
Pitch Stick Doublet	Pitch	Adjust gains to obtain desired cues
Roll Stick Doublet	Roll/Yaw	Adjust gains to obtain desired cues
Rudder Pedal Doublet	Roll/Yaw	Adjust gains to obtain desired cues
Throttle Increase/Decrease	Surge/Heave	Adjust gains to obtain desired cues
Takeoff	Pitch/Surge/Heave	Ease in control of takeoff to desired pitch attitude, forward surge cues should be coordinated with pitch cues, vertical cues noticeable, ensure pitch and heave gains do not exceed platform limits
Straight-In Approach	Pitch/Surge/Heave	Ease in control of approach, forward surge cues should be coordinated with pitch cues, vertical cues noticeable
Coordinated Turn	Roll/Sway/Yaw	Minimize side force by reducing roll and/or increasing sway cues, ensure roll and sway gains do not exceed platform limits
Offset Approach	Roll/Sway/Yaw	Ease in control of approach, forward surge cues should be coordinated with pitch cues, minimal side forces
Turbulence	Heave	Adjust augmented turbulence gains to obtain desired turbulence sensation
Landing Touchdown	On-Ground Heave	Increase on-ground heave gain to obtain desired touchdown sensation, ensure motion platform limits are not exceeded
Ground Maneuvers	On-Ground Heave	Increase on-ground heave gain to obtain desired ground motion sensation, ensure motion platform limits are not exceeded

This page intentionally left blank.

6. Suggested Future Implementation

Reducing the nonlinear gains for the optimal algorithm (surge) and the nonlinear algorithm (surge and pitch) was necessary so that the takeoff maneuver could be flown within the 60-inch actuator extension limits and low bandwidth (2-Hz) of the Langley Visual Motion Simulator (VMS). However, gain reductions contributed to degradation in pilot performance that was observed most frequently with the straight-in approach. These results were observed during piloted performance tests conducted with the optimal and nonlinear motion cueing algorithms [12]. Implementation of the optimal and nonlinear algorithms on a platform with increased actuator extensions would allow for increased gains, thus resulting in improved pilot performance. One such motion platform is located in the Cockpit Motion Facility (CMF) [13], shown in Figure 6.1, presently being erected at the NASA Langley Research Center.

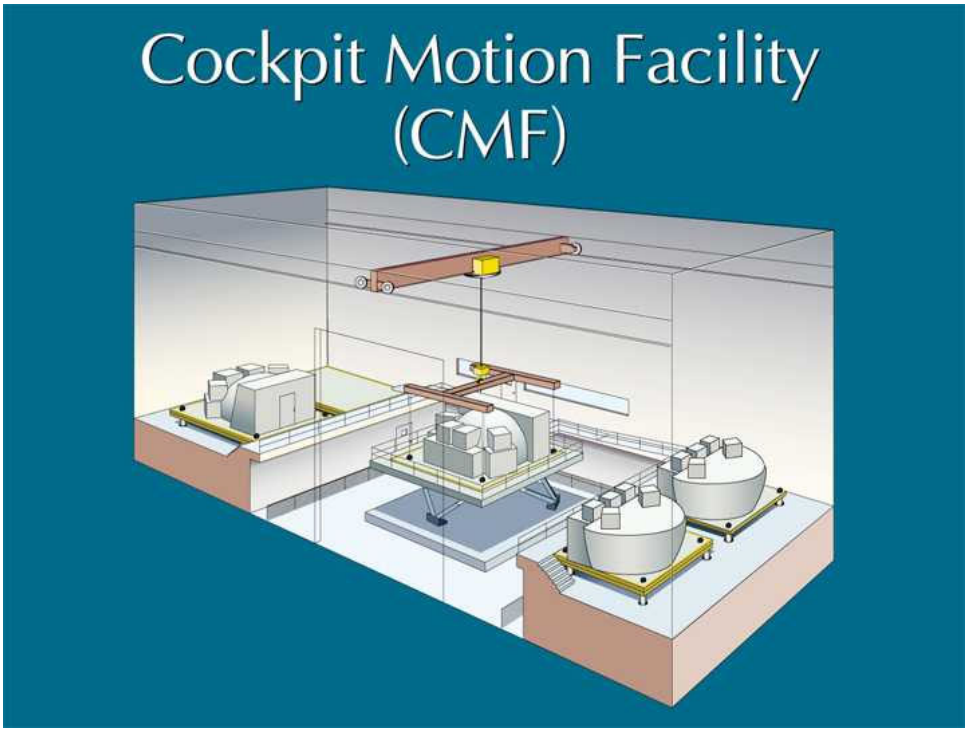


Figure 6.1. NASA Langley Cockpit Motion Facility (CMF).

The Cockpit Motion Facility is made up of one motion system site and four fixed-base sites. The motion system site contains a six-degree-of-freedom state-of-the-art synergistic motion base with 76-inch actuator extensions. The four fixed-base sites allow the simulator cockpits to be used in fixed-base mode when they are not resident on the motion system. Each cockpit has its own visual display system and all cockpits share Evans and Sutherland ESIG 3000 image generators.

Pilot tests with the optimal algorithm [12] revealed a number of cases in which the braking algorithm arrested the motion, but with the braking algorithm being unable to recover the motion. In one case, an experienced military aircraft pilot exceeded the motion system actuator limits on several test runs due to aggressive liftoff rotations that resulted in a very high rate of climb and large heave displacement. Similar tests revealed that the nonlinear algorithm was less likely to enter the braking region, due to the fact that the algorithm progressively reduces the platform displacements for large inputs; thus resulting in a larger motion envelope than the optimal algorithm. However, similar problems persisted with the braking algorithm when simulator motion was restrained, with the braking algorithm failing to recover to normal motion. Based upon these results, the braking algorithm is not performing its desired function and future improvements need to be investigated. An improved algorithm that is effective in both restraining large excursions and resuming regular simulator motion would allow increased nonlinear gains and improve motion cueing performance. One suggested approach is the algorithm developed by McFarland [14] for NASA Ames.

The nonlinear algorithm will ultimately be implemented on the CMF. Pilot tuning of the nonlinear gains, similar to that previously done for the new algorithms on

the VMS as discussed in Section 5.1, will be performed. The tuning procedure given in Section 5.2 can be followed. The gain coefficient sets given in Table 5.2 were set up for tuning the CMF with increased actuator extensions in comparison to the VMS. It is expected that the CMF will accommodate increased gains for the pitch, surge, and possibly the heave degrees of freedom. Due to the algorithm producing faster washout with large motion cues, the necessity for a braking algorithm to address large excursions may be minimal. It is suggested that the nonlinear algorithm be first implemented and tuned without a braking algorithm, with a new braking algorithm to be developed if needed based on that performance.

This page intentionally left blank.

7. Common Block Variable Listings

7.1. comint2.com (Variables Used in Optimal and Nonlinear Algorithms)

Variable	Description
X, Y, Z	Desired Platform Displacements (m)
XD, YD, ZD	Desired Platform Velocities (m/s)
XDD, YDD, ZDD	Desired Platform Accelerations (m/s/s)
PHI, THE, PSI	Desired Platform Attitudes (rad)
PHID, THED, PSID	Desired Platform Angular Velocities (rad/s)

7.2. optint3.com

Variable	Description
DT	Time Step (Sec)
XX1(8), X1O(8)	Filter State Variable 1 (Current and Prior Time)
X2(8), X2O(8)	Filter State Variable 2 (Current and Prior Time)
X3(8), X3O(8)	Filter State Variable 3 (Current and Prior Time)
X4(8), X4O(8)	Filter State Variable 4 (Current and Prior Time)
X5(6), X5O(6)	Filter State Variable 5 (Current and Prior Time)
X6(6), X6O(6)	Filter State Variable 6 (Current and Prior Time)
ACA(3), ACAO(3)	Aircraft Body Acceleration Vector (m/s/s)
BETAA(3)	Aircraft Euler Angle Vector (rad)
BETAAD(3)	Aircraft Euler Rate Vector (rad/sec)
A2(3)	Aircraft Inertial Acceleration Vector (m/s/s)
A2N(3), A2NO(3)	Scaled Aircraft Inertial Acceleration Vector (m/s/s)
BADN(3), BADNO(3)	Scaled Aircraft Euler Rate Vector (rad/s)
ASI(3), ASIO(3)	Desired Platform Acceleration Cue (m/s/s)
VSI(3), VSIO(3)	Desired Platform Velocity (m/s)
SSIU(3), SSI1(3), SSIO(3)	Desired Platform Displacement (m)
WAA(3)	Aircraft Body Velocity Vector (rad/sec)
BSDT(2), BSDTO(2), BSDTL(2)	Platform Tilt Cue (Current, Prior, and Limited) (rad/sec)
BETAS1(3)	Desired Platform Angular Position (rad)
BDLIM	Platform Tilt Cue Limit (rad/sec)
BSDR(3), BSDRO(3)	Platform Rotational Cue (Current and Prior) (rad/sec)
BETASR(3), BETASRO(3)	Platform Rotational Angle (Current and Prior) (rad/sec)
BETAST(2), BETASTO(2)	Platform Tilt Angle (Current and Prior) (rad)
BETASTL(2), BETASTLO(2)	Platform Tilt Angle with Limit (Current and Prior) (rad)
DIF(2)	Difference between Current and Limited Tilt Angles
XH(2), XHO(2)	Trim Filter State Variables
T1N1, T1DO, T2N1, T2DO	Trim Filter Coefficients
XT, XTO, XT2, XT2O	Augmented Turbulence Filter State Variables
ACZT	Augmented Turbulence Acceleration
WGUST, WGUSTO	Z-Axis Gust Velocity (m/sec)
G1D0, G1D1	Augmented Turbulence Filter Coefficients (Denominator)
G1N0, G1N1, G1N2	Augmented Turbulence Filter Coefficients (Numerator)

7.3. wopt3.com

Variable	Description
R1N6,R1N5,R1N4,R1N3,R1N2,R1N1,R1N0	Lateral Filter W_{11} Numerator Coefficients
R2N6,R2N5,R2N4,R2N3,R2N2,R2N1,R2N0	Lateral Filter W_{21} Numerator Coefficients
R4N6,R4N5,R4N4,R4N3,R4N2,R4N1,R4N0	Lateral Filter W_{22} Numerator Coefficients
R1D5,R1D4,R1D3,R1D2,R1D1,R1D0	Lateral Filter Denominator Coefficients
P1N6,P1N5,P1N4,P1N3,P1N2,P1N1,P1N0	Longitudinal Filter W_{11} Numerator Coefficients
P2N6,P2N5,P2N4,P2N3,P2N2,P2N1,P2N0	Longitudinal Filter W_{21} Numerator Coefficients
P4N6,P4N5,P4N4,P4N3,P4N2,P4N1,P4N0	Longitudinal Filter W_{22} Numerator Coefficients
P1D5,P1D4,P1D3,P1D2,P1D1,P1D0	Longitudinal Filter Denominator Coefficients
Y1N4,Y1N3,Y1N2,Y1N1,Y1N0	Yaw Filter Numerator Coefficients
Y1D3,Y1D2,Y1D1,Y1D0	Yaw Filter Denominator Coefficients
H1N4,H1N3,H1N2,H1N1,H1N0	Vertical Filter Numerator Coefficients
H1D3,H1D2,H1D1,H1D0	Vertical Filter Denominator Coefficients
GX3,GY3,GZ3,GP3,GQ3,GR3	In-Flight Polynomial Scaling Coefficients
AMX3,BMX3	In-Flight Translational and Rotational Limits
GZ30,GZ3S,AMX30,AMX3S	On-Ground Scaling Coefficients and Limits
GT2	Augmented Turbulence Acceleration Gain

7.4. matrix1c.com

Variable	Description
AAIS(3,6)	Motion Base Actuator Coordinates (m)
BBII(3,6)	Fixed Base Actuator Coordinates (m)
RRS(3)	Vector from Motion Base Centroid to Pilot Head (m)
LENEUT	Actuator Neutral Length (m)
LI(6)	Desired Actuator Extensions (m)
RLI(6), RLIO(6)	Actual Actuator Extensions (m)
LEGC(6)	Actual Actuator Extensions (in)
SSI(3)	Actual Simulator Displacements (m)
BETAS(3)	Actual Simulator Attitudes (rad)
BRAKE(6)	Actuator Braking Region Value
RATIO(6)	Actuator Braking Ratio
LAVAIL	Actuator Available Length (m)
RLID(6), RLIDO(6)	Actuator Velocity (m/s)
RLIDD(6)	Actuator Acceleration (m/s/s)
FLAG(6)	Braking Region Flag (0 or 1)
FLAG2	Braking Region Flag (0 or 1)
IT2, NC2	Braking Recovery Indices
SUMFLAG	Sum of FLAG(6) Values

7.5. nopt4.com

Variable	Description
Longitudinal Mode	
ALPX, ALPXMAX	Prescribed Nonlinearity α
APX(11,11), APXO(11)	State Space System Matrix $\mathbf{A}'$
BRBX(11,11), R1PX(11,11), R2IX(2)	System Weighting Matrix $\mathbf{S}, \mathbf{R}_1', \mathbf{R}_2$
AVX(6,6), BVX(6,2), DQCX(6)	Vestibular State Space Matrices $\mathbf{A}_V, \mathbf{B}_V, \mathbf{D}_V\mathbf{QC}_V$
K1X(2,6), K2X(2,3), K3X(2)	Feedback Matrices $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3$
ZX(11,11)	Neurocomputing Solver Bi-Polar Vectors **Z**
PX(11,11), PXVEC(66)	Riccati Equation Solution **P**
XX(9), XXO(9)	State Vector **x**
Lateral Mode	
ALPY, ALPYMAX	Prescribed Nonlinearity α
APY(11,11), APYO(11)	State Space System Matrix $\mathbf{A}'$
BRBY(11,11), R1PX(11,11), R2IY(2)	System Weighting Matrix $\mathbf{S}, \mathbf{R}_1', \mathbf{R}_2$
AVY(6,6), BVY(6,2), DQCY(6)	Vestibular State Space Matrices $\mathbf{A}_V, \mathbf{B}_V, \mathbf{D}_V\mathbf{QC}_V$
K1Y(2,6), K2Y(2,3), K3Y(2)	Feedback Matrices $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3$
ZY(11,11)	Neurocomputing Solver Bi-Polar Vectors **Z**
PY(11,11), PYVEC(66)	Riccati Equation Solution **P**
XY(9), XYO(9)	State Vector **x**
Yaw Mode	
ALPR, ALPRMAX	Prescribed Nonlinearity α
APR(5,5), APRO(5)	State Space System Matrix $\mathbf{A}'$
BRBR(5,5), R1PR(5,5), R2IR	System Weighting Matrix $\mathbf{S}, \mathbf{R}_1', \mathbf{R}_2$
AVY(3,3), BVR(3), DQCR(3), DQDR	Vestibular State Space Matrices $\mathbf{A}_V, \mathbf{B}_V, \mathbf{D}_V\mathbf{QC}_V, \mathbf{D}_V\mathbf{QD}_V$
K1R(3), K2R, K3R	Feedback Matrices $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3$
ZR(5,5)	Neurocomputing Solver Bi-Polar Vectors **Z**
PR(5,5), PRVEC(15)	Riccati Equation Solution **P**
XR(4), XRO(4)	State Vector **x**
Heave Mode	
ALPZ, ALPZMAX	Prescribed Nonlinearity α
APZ(6,6), APZO(6)	State Space System Matrix $\mathbf{A}'$
BRBZ(6,6), R1PZ(6,6)	System Weighting Matrix $\mathbf{S}, \mathbf{R}_1'$
AVZ(2,2), BVZ(2)	Vestibular State Space Matrices $\mathbf{A}_V, \mathbf{B}_V$
K1Z(2), K2Z(2), K3Z	Feedback Matrices $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3$
ZZ(6,6)	Neurocomputing Solver Bi-Polar Vectors **Z**
PZ(6,6), PZVEC(21)	Riccati Equation Solution **P**
XZ(5), XZO(5)	State Vector **x**
Nonlinear Gains and Turbulence	
GX4,GY4,GZ4,GP4,GQ4,GR4	In-Flight Polynomial Scaling Coefficients
AMX4,BMX4	In-Flight Translational and Rotational Limits
GZ40,GZ4S,AMX40,AMX4S	On-Ground Scaling Coefficients and Limits
GT4	Augmented Turbulence Acceleration Gain
G2D0, G2D1	Augmented Turbulence Filter Coefficients (Denominator)
G2N0, G2N1, G2N2	Augmented Turbulence Filter Coefficients (Numerator)

This page intentionally left blank.

8. Program Listing

8.1. gainopt3.f

```
C************************************************************
C  OPTIMAL ALGORITHM NONLINEAR GAIN SUBROUTINE.
C  THE INPUT IS FIRST LIMITED AND THEN SCALED BY A POLYNOMIAL.
C************************************************************
C
      SUBROUTINE GAINOPT3

      INCLUDE 'optint3.com'

      INCLUDE 'wopt3.com'

      REAL AA(3),BA(3)
C
C     Take Absolute Value of Input
C
      AA(1)=ABS(A2(1))
      AA(2)=ABS(A2(2))
      AA(3)=ABS(A2(3))
      BA(1)=ABS(BETAAD(1))
      BA(2)=ABS(BETAAD(2))
      BA(3)=ABS(BETAAD(3))
C
C     Limit Translational and Rotational Inputs
C
      AAM=MAX(AA(1),AA(2),AA(3))
      BAM=MAX(BA(1),BA(2),BA(3))
      IF(AAM.GT.AMX3) THEN
          RATIO=AMX3/AAM
          AA(1)=AA(1)*RATIO
          AA(2)=AA(2)*RATIO
          AA(3)=AA(3)*RATIO
      END IF
      IF(BAM.GT.BMX3) THEN
          RATIO=BMX3/BAM
          BA(1)=BA(1)*RATIO
          BA(2)=BA(2)*RATIO
          BA(3)=BA(3)*RATIO
      END IF
C
C     Perform Nonlinear Scaling of Inputs
C
      A2N(1)=(GX3(1)*AA(1)+GX3(2)*AA(1)**2.+GX3(3)*AA(1)**3.)
     +*SIGN(1.,A2(1))
      A2N(2)=(GY3(1)*AA(2)+GY3(2)*AA(2)**2.+GY3(3)*AA(2)**3.)
     +*SIGN(1.,A2(2))
      A2N(3)=(GZ3(1)*AA(3)+GZ3(2)*AA(3)**2.+GZ3(3)*AA(3)**3.)
     +*SIGN(1.,A2(3))
      BADN(1)=(GP3(1)*BA(1)+GP3(2)*BA(1)**2.+GP3(3)*BA(1)**3.)
     +*SIGN(1.,BETAAD(1))
      BADN(2)=(GQ3(1)*BA(2)+GQ3(2)*BA(2)**2.+GQ3(3)*BA(2)**3.)
     +*SIGN(1.,BETAAD(2))
      BADN(3)=(GR3(1)*BA(3)+GR3(2)*BA(3)**2.+GR3(3)*BA(3)**3.)
```

```
+*SIGN(1.,BETAAD(3))

 RETURN
 END
```

8.2. gainopt4.f

```
C************************************************************
C  NONLINEAR ALGORITHM NONLINEAR GAIN SUBROUTINE.
C  THE INPUT IS FIRST LIMITED AND THEN SCALED BY A POLYNOMIAL.
C************************************************************
C
      SUBROUTINE GAINOPT4

      INCLUDE 'optint3.com'

      INCLUDE 'nopt4.com'

      REAL AA4(3),BA4(3)
C
C  Take Absolute Value of Input
C
      AA4(1)=ABS(A2(1))
      AA4(2)=ABS(A2(2))
      AA4(3)=ABS(A2(3))
      BA4(1)=ABS(BETAAD(1))
      BA4(2)=ABS(BETAAD(2))
      BA4(3)=ABS(BETAAD(3))
C
C  Limit Translational and Rotational Inputs
C
      AAM=MAX(AA4(1),AA4(2),AA4(3))
      BAM=MAX(BA4(1),BA4(2),BA4(3))
      IF(AAM.GT.AMX4) THEN
         RATIO=AMX4/AAM
         AA4(1)=AA4(1)*RATIO
         AA4(2)=AA4(2)*RATIO
         AA4(3)=AA4(3)*RATIO
      END IF
      IF(BAM.GT.BMX4) THEN
         RATIO=BMX4/BAM
         BA4(1)=BA4(1)*RATIO
         BA4(2)=BA4(2)*RATIO
         BA4(3)=BA4(3)*RATIO
      END IF
C
C     Perform Nonlinear Scaling of Inputs
C
      A2N(1)=(GX4(1)*AA4(1)+GX4(2)*AA4(1)**2.+GX4(3)*AA4(1)**3.)
     +*SIGN(1.,A2(1))
      A2N(2)=(GY4(1)*AA4(2)+GY4(2)*AA4(2)**2.+GY4(3)*AA4(2)**3.)
     +*SIGN(1.,A2(2))
      A2N(3)=(GZ4(1)*AA4(3)+GZ4(2)*AA4(3)**2.+GZ4(3)*AA4(3)**3.)
     +*SIGN(1.,A2(3))
      BADN(1)=(GP4(1)*BA4(1)+GP4(2)*BA4(1)**2.+GP4(3)*BA4(1)**3.)
     +*SIGN(1.,BETAAD(1))
      BADN(2)=(GQ4(1)*BA4(2)+GQ4(2)*BA4(2)**2.+GQ4(3)*BA4(2)**3.)
     +*SIGN(1.,BETAAD(2))
      BADN(3)=(GR4(1)*BA4(3)+GR4(2)*BA4(3)**2.+GR4(3)*BA4(3)**3.)
     +*SIGN(1.,BETAAD(3))
```

```
      RETURN
      END
C
```

8.3. integ3.f

```
      SUBROUTINE INTEG3

      INCLUDE 'optint3.com'

      INCLUDE 'comint2.com'

      INCLUDE 'wcom2.com'

      INCLUDE 'matrix1c.com'

C
C     INTEGRATE:
C     SIMULATOR LINEAR ACCEL TO LINEAR DISPL
C     SIMULATOR ANGULAR VEL TO ANGULAR DISPL
C
      DO K=1,3
          VSI(K)=VSIO(K)+DT*0.5*(ASI(K)+ASIO(K))
          SSIU(K)=SSIO(K)+DT*0.5*(VSI(K)+VSIO(K))
          BETASR(K)=BETASRO(K)+DT*0.5*(BSDR(K)+BSDRO(K))

          ASIO(K)=ASI(K)
          VSIO(K)=VSI(K)
          SSIO(K)=SSIU(K)
          BSDRO(K)=BSDR(K)
          BETASRO(K)=BETASR(K)
      END DO
C
C     LIMIT THE ANGULAR RATE IN THE CROSS-OVER TILT CHANNEL.
C     THE REAL TILT POSITION WILL BE DETERMINED BY BOTH THE DESIRED
C     POSITION AND THE DIFFERENCE BETWEEN THE DESIRED AND REAL
C     TILT POSITION.
C
      DO K=1,2
         BETAST(K)=BETASTO(K)+DT*0.5*(BSDT(K)+BSDTO(K))
         DIF(K)=0.005*(BETAST(K)-BETASTLO(K))+(BETAST(K)-BETASTO(K))
         BETASTL(K)=BETASTLO(K)+MAX(-BDLIM*DT,MIN(BDLIM*DT,DIF(K)))
C
C     COMPUTE THE TILT ANGULAR VELOCITY
C
         BSDTL(K)=(BETASTL(K)-BETASTLO(K))/DT

         BETASTO(K)=BETAST(K)
         BETASTLO(K)=BETASTL(K)
         BSDTO(K)=BSDT(K)
      END DO
C
C     COMBINE THE TILT AND ROTATIONAL CHANNELS TO OBTAIN
C     THE DESIRED ANGULAR POSITION
C
      BETAS1(1)=BETASTL(1)+BETASR(1)
      BETAS1(2)=BETASTL(2)+BETASR(2)
      BETAS1(3)=BETASR(3)
C
C     USE DIFFERENCE BETWEEN DESIRED BETAST AND REAL BETAST
```

```
C     TO GENERATE ADDITIONAL LINEAR RESPONSE AND ACHIEVE
C     COORDINATION BETWEEN THE LINEAR AND TILT CHANNELS
C
      SSI1(1)=SSIU(1)+RRS(3)*(BETAST(2)-BETASTL(2))
      SSI1(2)=SSIU(2)-RRS(3)*(BETAST(1)-BETASTL(1))
C
C     FOR ON-GROUND MOTION, ADD RUNWAY ROUGHNESS EFFECT
C     AMPLITUDE IS FAIRED UPON TOUCHDOWN OR TAKEOFF
C
      SSI1(3)=SSIU(3)+XKA*SIN((WB+XKG*VGSPD)*T)
C
C     Swap Variables To Match Modified Algorithm
C     For Input to JACKDRVR
C
      XDD = ASI(1)
      YDD = ASI(2)
      ZDD = ASI(3)
      XD = VSI(1)
      YD = VSI(2)
      ZD = VSI(3)
      X = SSI1(1)
      Y = SSI1(2)
      Z = SSI1(3)
      PHI = BETAS1(1)
      THE = BETAS1(2)
      PSI = BETAS1(3)
      PHID = BSDTL(1)+BSDR(1)
      THED = BSDTL(2)+BSDR(2)
      PSID = BSDR(3)

      RETURN
      END
```

8.4. integ4.f

```
      SUBROUTINE INTEG4

      INCLUDE 'optint3.com'

      INCLUDE 'comint2.com'

      INCLUDE 'wcom2.com'

      INCLUDE 'matrix1c.com'

      INCLUDE 'nopt4.com'
C
C     SURGE FILTER OUTPUT ASI(1) & BSDT(2)
C
      ASI(1)=-K1X(2,1)*XX(1)-K1X(2,2)*XX(2)-K1X(2,3)*XX(3)
     x       -K1X(2,4)*XX(4)-K1X(2,5)*XX(5)-K1X(2,6)*XX(6)
     x       -K2X(2,1)*XX(7)-K2X(2,2)*XX(8)-K2X(2,3)*XX(9)
     x       -K3X(2)*A2N(1)

      BSDT(2)=-K1X(1,1)*XX(1)-K1X(1,2)*XX(2)-K1X(1,3)*XX(3)
     x        -K1X(1,4)*XX(4)-K1X(1,5)*XX(5)-K1X(1,6)*XX(6)
     x        -K2X(1,1)*XX(7)-K2X(1,2)*XX(8)-K2X(1,3)*XX(9)
     x        -K3X(1)*A2N(1)
C
C     SWAY FILTER OUTPUT ASI(2) & BSDT(1)
C
      ASI(2)=-K1Y(2,1)*XY(1)-K1Y(2,2)*XY(2)-K1Y(2,3)*XY(3)
     x       -K1Y(2,4)*XY(4)-K1Y(2,5)*XY(5)-K1Y(2,6)*XY(6)
     x       -K2Y(2,1)*XY(7)-K2Y(2,2)*XY(8)-K2Y(2,3)*XY(9)
     x       -K3Y(2)*A2N(2)

      BSDT(1)=-K1Y(1,1)*XY(1)-K1Y(1,2)*XY(2)-K1Y(1,3)*XY(3)
     x        -K1Y(1,4)*XY(4)-K1Y(1,5)*XY(5)-K1Y(1,6)*XY(6)
     x        -K2Y(1,1)*XY(7)-K2Y(1,2)*XY(8)-K2Y(1,3)*XY(9)
     x        -K3Y(1)*A2N(2)
C
C     HEAVE FILTER OUTPUT ASI(3)
C
      ASI(3)=(-K1Z(1)*XZ(1)-K1Z(2)*XZ(2)-K2Z(1)*XZ(3)
     x        -K2Z(2)*XZ(4)-K2Z(3)*XZ(5)-K3Z*A2N(3))
C
C     YAW FILTER OUTPUT BSDR(3)
C
      BSDR(3)=-K1R(1)*XR(1)-K1R(2)*XR(2)-K1R(3)*XR(3)
     x        -K2R*XR(4)-K3R*BADN(3)
C
C     LIMIT THE ANGULAR RATE IN THE CROSS-OVER TILT CHANNEL.
C     THE REAL TILT POSITION WILL BE DETERMINED BY BOTH THE DESIRED
C     POSITION AND THE DIFFERENCE BETWEEN THE DESIRED AND REAL
C     TILT POSITION.
C
      DO K=1,2
         BETASR(K)=BETASRO(K)+DT*BSDRO(K)
         BETAST(K)=BETASTO(K)+DT*BSDTO(K)
```

```
         DIF(K)=0.005*(BETAST(K)-BETASTLO(K))+(BETAST(K)-BETASTO(K))
         BETASTL(K)=BETASTLO(K)+MAX(-BDLIM*DT,MIN(BDLIM*DT,DIF(K)))
C
C     COMPUTE THE TILT ANGULAR VELOCITY
C
         BSDTL(K)=(BETASTL(K)-BETASTLO(K))/DT
         BSDRO(K) = BSDR(K)
         BETASRO(K)=BETASR(K)
         BETASTO(K)=BETAST(K)
         BETASTLO(K)=BETASTL(K)
         BSDTO(K)=BSDT(K)
      END DO
C
C     COMBINE THE TILT AND ROTATIONAL CHANNELS TO OBTAIN
C     THE DESIRED ANGULAR POSITION
C
      BETAS1(1)=BETASTL(1)+BETASR(1)
      BETAS1(2)=BETASTL(2)+BETASR(2)
      BETAS1(3)=XR(4)
C
C  USE DIFFERENCE BETWEEN DESIRED BETAST AND REAL BETAST
C  TO GENERATE ADDITIONAL LINEAR RESPONSE AND ACHIEVE
C  COORDINATION BETWEEN THE LINEAR AND TILT CHANNELS
C
      SSI1(1)=XX(8)+RRS(3)*(BETAST(2)-BETASTL(2))
      SSI1(2)=XY(8)-RRS(3)*(BETAST(1)-BETASTL(1))
C
C     FOR ON-GROUND MOTION, ADD RUNWAY ROUGHNESS EFFECT
C     AMPLITUDE IS FAIRED UPON TOUCHDOWN OR TAKEOFF
C
      SSI1(3)=XZ(4)+XKA*SIN((WB+XKG*VGSPD)*T)
C
C     Swap Variables To Match Modified Algorithm
C     For Input to JACKDRVR
C
      XDD = ASI(1)
      YDD = ASI(2)
      ZDD = ASI(3)
      XD = XX(9)
      YD = XY(9)
      ZD = XZ(5)
      X = SSI1(1)
      Y = SSI1(2)
      Z = SSI1(3)
      PHI = BETAS1(1)
      THE = BETAS1(2)
      PSI = BETAS1(3)
      PHID = BSDTL(1)+BSDR(1)
      THED = BSDTL(2)+BSDR(2)
      PSID = BSDR(3)

      RETURN
      END
```

8.5. invplf.f

```
C  THIS SUBROUTINE WILL UNDERTAKE AN INVERSE TRANSFORMATION DEVELOPED
C  BY THE NEWTON-RAPHSON TECHNIQUE. LEG EXTENSIONS WILL BE TRANSFORMED
C  TO THE DEGREES OF FREEDOM.  THIS INVERSE TRANSFORMATION IS PERFORMED
C  BY AN ITERATIVE METHOD DENOTED AS NEWTON-RAPHSON TECHNIQUE.
C  ITERATIONS ARE TERMINATED WHEN THE DIFFERENCE BETWEEN TWO SUBSEQUENT
C  ITERATIONS IS LESS THAN SOME ERROR CRITERION.

C  RLI IS THE LEG EXTENSIONS.
C  SSI IS THE TRANSLATIONAL DISPLACEMENT OF THE PLATFORM.
C  BETAS IS THE ANGULAR DISPLACEMENT OF THE PLATFORM.
C  XS,YS,ZS: COORDINATES OF THE FIXED ENDS OF THE LEGS.
C  XM,YM,ZM: COORDINATES OF THE MOVING ENDS OF THE LEGS.
C  AAIS,BBII: GEOMETRY OF THE MOTION SYSTEM.
C  SSIIN: INITIAL TRANSLATIONAL DISPLACEMENT.
C  RRS: VECTOR NOT USED BY THIS SUBROUTINE.
C  LENEUT: LENGTH OF LEGS IN NEUTRAL POSITION.
C
SUBROUTINE INVPLF
      REAL RAML(6),XS(6),YS(6),ZS(6),XM(6),YM(6),ZM(6)
      REAL F(6),PFX(6),PFY(6),PFZ(6),PFS(6),PFT(6),PFP(6),
     + A(3,3),ZEBRA(36),P

      INCLUDE 'matrix1c.com'

      DATA IFLAG/0/
C*********************************************
C INITIALIZE SIMULATOR POSITION.
      DATA X/0./,Y/0./,Z/0./,P/0./,T/0./,S/0./
C*********************************************
C
      SAVE

      IF(IFLAG.EQ.0) THEN
          DO JACK=1,6
              XM(JACK)=AAIS(1,JACK)
              YM(JACK)=AAIS(2,JACK)
              ZM(JACK)=AAIS(3,JACK)
              XS(JACK)=BBII(1,JACK)
              YS(JACK)=BBII(2,JACK)
              ZS(JACK)=BBII(3,JACK)
          END DO
          IFLAG=1
      END IF
C**********************
C      X=SSI(1)+SSIIN(1)
C      Y=SSI(2)+SSIIN(2)
C      Z=SSI(3)+SSIIN(3)
C      P=BETAS(1)
C      T=BETAS(2)
C      S=BETAS(3)
C**********************
      DO JACK=1,6
          RAML(JACK)=RLI(JACK)+LENEUT
      END DO
```

```
      IT=0
9     CONTINUE
      A(1,1)=COS(S)*COS(T)
      A(1,2)=SIN(S)*COS(T)
      A(1,3)=-SIN(T)
      A(2,1)=COS(S)*SIN(T)*SIN(P)-SIN(S)*COS(P)
      A(2,2)=SIN(S)*SIN(T)*SIN(P)+COS(S)*COS(P)
      A(2,3)=COS(T)*SIN(P)
      A(3,1)=COS(S)*SIN(T)*COS(P)+SIN(S)*SIN(P)
      A(3,2)=SIN(S)*SIN(T)*COS(P)-COS(S)*SIN(P)
      A(3,3)=COS(T)*COS(P)
      DO 17 I=1,6
        F(I)=XM(I)**2.+YM(I)**2.+ZM(I)**2.+XS(I)**2.+YS(I)**2.+
     +    ZS(I)**2.+X**2.+Y**2.+Z**2.-RAML(I)**2.
     +    +2.*(X-XS(I))*(XM(I)*A(1,1)+YM(I)*A(2,1)+ZM(I)*A(3,1))
     +    +2.*(Y-YS(I))*(XM(I)*A(1,2)+YM(I)*A(2,2)+ZM(I)*A(3,2))
     +    +2.*(Z-ZS(I))*(XM(I)*A(1,3)+YM(I)*A(2,3)+ZM(I)*A(3,3))
     +    -2.*(X*XS(I)+Y*YS(I)+Z*ZS(I))
        PFX(I)=2.*(X+XM(I)*A(1,1)+YM(I)*A(2,1)+ZM(I)*A(3,1)-XS(I))
        PFY(I)=2.*(Y+XM(I)*A(1,2)+YM(I)*A(2,2)+ZM(I)*A(3,2)-YS(I))
        PFZ(I)=2.*(Z+XM(I)*A(1,3)+YM(I)*A(2,3)+ZM(I)*A(3,3)-ZS(I))
        PFS(I)=-2.*(X-XS(I))*(XM(I)*A(1,2)+YM(I)*A(2,2)+ZM(I)*A(3,2))
     +         +2.*(Y-YS(I))*(XM(I)*A(1,1)+YM(I)*A(2,1)+ZM(I)*A(3,1))
        PFT(I)= 2.*(X-XS(I))*(-XM(I)*SIN(T)*COS(S)+YM(I)*SIN(P)*COS(T)*
     +    COS(S)+ZM(I)*COS(P)*COS(T)*COS(S))+2.*(Y-YS(I))*(-XM(I)*
     +    SIN(T)*SIN(S)+YM(I)*SIN(P)*COS(T)*SIN(S)+ZM(I)*COS(P)*COS(T)
     +    *SIN(S))-2.*(Z-ZS(I))*(XM(I)*COS(T)+YM(I)*SIN(P)*SIN(T)
     +    +ZM(I)*COS(P)* SIN(T))
        PFP(I)=2.*(X-XS(I))*(YM(I)*A(3,1)-ZM(I)*A(2,1))
     +         +2.*(Y-YS(I))*(YM(I)*A(3,2)-ZM(I)*A(2,2))
     +         +2.*(Z-ZS(I))*(YM(I)*A(3,3)-ZM(I)*A(2,3))
17    CONTINUE
      DO 1 N=1,6
          ZEBRA(N)   =PFX(N)
          ZEBRA(N+6) =PFY(N)
          ZEBRA(N+12)=PFZ(N)
          ZEBRA(N+18)=PFS(N)
          ZEBRA(N+24)=PFT(N)
          ZEBRA(N+30)=PFP(N)
1     CONTINUE
      N=6
      CALL SIMQ(ZEBRA,F,N,KS)
      IF(KS.EQ.1) THEN
          WRITE(*,*) ' 1 MATRIX IS SINGULAR'
          GOTO 22
      END IF
      IT=IT+1
      IF(IT.EQ.51) GO TO 22
      X=X-F(1)
      Y=Y-F(2)
      Z=Z-F(3)
      S=S-F(4)
      T=T-F(5)
      P=P-F(6)
      ZLIM1=0.01
      ZLIM2=0.1/57.296
      IF(MAX(ABS(F(1)),ABS(F(2)),ABS(F(3))).GT.ZLIM1) GO TO 9
```

```
      IF(MAX(ABS(F(4)),ABS(F(5)),ABS(F(6))).GT.ZLIM2) GO TO 9
 22   SSI(1)=X
      SSI(2)=Y
      SSI(3)=Z
      BETAS(1)=P
      BETAS(2)=T
      BETAS(3)=S
      RETURN
      END
C
```

8.6. jackdrvr.f

```
C  COMPUTE THE ACTUATOR EXTENSION COMMANDS BASED ON POSITION
C  IN INERTIAL FRAME.
C
      SUBROUTINE JACKDRVR

      INCLUDE 'matrix1c.com'

      INCLUDE 'comint2.com'

      INCLUDE 'optint3.com'

      REAL DUMMY31A(3,1),LLIS(3,3),L1(6),L2(6),L3(6),LENGTHTOT(6)

      REAL LLIMH,LLIML

C
C********** Langley VMS motion system geometry **********
      DATA LENEUT/3.2649/
      DATA RRS/0.0254,-0.635,-2.1946/
      DATA AAIS/2.1117179,   0.0762, 0.0,
     x           2.1117179,  -0.0762, 0.0,
     x          -0.98986594, -1.8669, 0.0,
     x          -1.12184942, -1.7907, 0.0,
     x          -1.12184942,  1.7907, 0.0,
     x          -0.98986594,  1.8669, 0.0/
      DATA BBII/  1.5021179,   1.9812,     2.58064,
     x            1.5021179,  -1.9812,     2.58064,
     x            0.96471232, -2.29147116, 2.58064,
     x           -2.46682768, -0.31027116, 2.58064,
     x           -2.46682768,  0.31027116, 2.58064,
     x            0.96471232,  2.29147116, 2.58064/
      DATA RLI/6*0./, SSI/3*0.0/, BETAS/3*0.0/
C*******************************************************
C
C     McFadden Actuator Stroke Limit
C     (When Used Replace LLIMH/LLIML with LLIM)
C      DATA LLIM/0.92075/
C
C     Langley VMS Actuator Stroke Limits
      DATA LLIMH/0.7864/,LLIML/0.6487/
C
      DATA ACMAX/0.7/
      DATA FLAG/6*0/,FLAG2/0/,IT2/400/,NC2/400/,SUMFLAG/0/
      DATA RLID/6*0./,RLIDD/6*0./,RLIO/6*0./,RLIDO/6*0./

C
C     EXACT ANGLE COMPUTATIONS
C
      SINPHI = SIN(PHI)
      SINTH  = SIN(THE)
      SINPSI = SIN(PSI)
      COSPHI = COS(PHI)
      COSTH  = COS(THE)
      COSPSI = COS(PSI)
```

```
C
C     FORM LLIS TRANSFORMATION MATRIX
C
      LLIS(1,1) = COSPSI*COSTH
      LLIS(2,1) = SINPSI*COSTH
      LLIS(3,1) = -SINTH
      LLIS(1,2) = COSPSI*SINTH*SINPHI - SINPSI*COSPHI
      LLIS(2,2) = SINPSI*SINTH*SINPHI + COSPSI*COSPHI
      LLIS(3,2) = COSTH*SINPHI
      LLIS(1,3) = COSPSI*SINTH*COSPHI + SINPSI*SINPHI
      LLIS(2,3) = SINPSI*SINTH*COSPHI - COSPSI*SINPHI
      LLIS(3,3) = COSTH*COSPHI
C
C     Compute Leg Extensions
C
      DO JACK = 1,6
         CALL VMULT(LLIS,AAIS(1,JACK),DUMMY31A,3,3,1)
         L1(JACK) = DUMMY31A(1,1) + X - BBII(1,JACK)
         L2(JACK) = DUMMY31A(2,1) + Y - BBII(2,JACK)

         L3(JACK) = DUMMY31A(3,1) + Z - BBII(3,JACK)
         LENGTHTOT(JACK) = SQRT(L1(JACK)**2+L2(JACK)**2+L3(JACK)**2)
         LI(JACK)=LENGTHTOT(JACK) - LENEUT
      END DO

C
C****************** JACK EXTENSION LIMITING **********************
      DO JACK=1,6
         IF(FLAG(JACK).EQ.1) GOTO 5
C
C     Avail. Length for Same +/- Extension Limits
C         LAVAIL=LLIM-RLI(JACK)*SIGN(1.,RLID(JACK))
C
C     Avail. Length for Different +/- Extension Limits
C
          IF (RLI(JACK).GT.0.) THEN
              LAVAIL=LLIMH-RLI(JACK)*SIGN(1.,RLID(JACK))
          ELSE
              LAVAIL=LLIML-RLI(JACK)*SIGN(1.,RLID(JACK))
          END IF
C
          BRAKE(JACK)=ABS(RLID(JACK))**2-1.98*ACMAX*LAVAIL
          IF(BRAKE(JACK).LT.0.) GOTO 5
          FLAG(JACK)=1
          VLEAD=RLID(JACK)
          FLAG2=1
          DO JK=1,6
             IF(FLAG(JK).EQ.0) THEN
                 RATIO(JK)=ABS(RLID(JK)/VLEAD)
             ELSE
                 RATIO(JK)=1.
             END IF
          END DO
5     END DO
      SUMFLAG=FLAG(1)+FLAG(2)+FLAG(3)+FLAG(4)+FLAG(5)+FLAG(6)
C
C
```

```
C     When brake is set, determine if brake should be released
C
      DXX=ABS(X)-ABS(SSI(1))
      DYY=ABS(Y)-ABS(SSI(2))
      DZZ=ABS(Z)-ABS(SSI(3))
      DPHI=ABS(PHI)-ABS(BETAS(1))
      DTHE=ABS(THE)-ABS(BETAS(2))
      DPSI=ABS(PSI)-ABS(BETAS(3))
      DMAX=MAX(DXX,DYY,DZZ,DPHI,DTHE,DPSI)
      DMIN=MIN(DXX,DYY,DZZ,DPHI,DTHE,DPSI)
      IF((DMAX.LE.1.E-5).AND.(DMIN.LE.-0.01)) THEN
         ID=0
      ELSE
         ID=1
      END IF

      DO JACK=1,6
          IF(FLAG(JACK).EQ.0) GOTO 20
          RLIDD(JACK)=-ACMAX*SIGN(1.,RLID(JACK))
          RLID(JACK) = RLID(JACK)+RLIDD(JACK)*H
          GOTO 30
20        IF(SUMFLAG.EQ.0) GOTO 50
          IF(ABS(RLID(JACK)).GT.0.001) THEN
              RLIDD(JACK)=-ACMAX*SIGN(1.,RLID(JACK))*RATIO(JACK)
              RLID(JACK) = RLID(JACK)+RLIDD(JACK)*H
          END IF
30        IF(ABS(RLID(JACK)).GT.(ACMAX*H)) THEN
              RLI(JACK)=RLI(JACK)+RLID(JACK)*H
          ELSE
              RLID(JACK)=0.
              FLAG(JACK)=0
          END IF
          GOTO 70
50        IF(FLAG2.EQ.0) GOTO 60
          IF(ID.NE.0) GOTO 70
          FLAG2=0
          IT2=0
60        RLI(JACK)  =RLI(JACK) +
     +      (1.-COS(3.1416/2.*IT2/NC2))*(LI(JACK)-RLI(JACK))
          IF((JACK.EQ.6).AND.(IT2.LT.NC2)) IT2=IT2+1
C
70    END DO
C
      DO JACK=1,6
C
C     Same +/- Extension limits
C         RLI(JACK)=MIN(0.999*LLIM,MAX(-0.999*LLIM,RLI(JACK)))
C
C     Different +/- Extension Limits
          RLI(JACK)=MIN(0.999*LLIMH,MAX(-0.999*LLIML,RLI(JACK)))
C
          RLID(JACK)=(RLI(JACK)-RLIO(JACK))/H
          RLIDD(JACK)=(RLID(JACK)-RLIDO(JACK))/H
C
          RLIO(JACK)=RLI(JACK)
          RLIDO(JACK)=RLID(JACK)
      END DO
```

```
RETURN
END
```

8.7. liba.f

```
C
C
C  COMPUTE THE TRANSFORMATION MATRICES LIA AND TS
C
      SUBROUTINE LIBA

      INCLUDE 'optint3.com'

      REAL LIA(3,3), TA(3,3)

      DATA TA/1.,3*0.0,1.,3*0.0,1./

C
C     EXACT ANGLE COMPUTATIONS
C
      SINPHI = SIN(BETAA(1))
      SINTH  = SIN(BETAA(2))
      SINPSI = SIN(BETAA(3))
      COSPHI = COS(BETAA(1))
      COSTH  = COS(BETAA(2))
      COSPSI = COS(BETAA(3))
      TANTH  = SINTH/COSTH
C
C     FORM LIA TRANSFORMATION MATRIX
C
      LIA(1,1) = COSPSI*COSTH
      LIA(2,1) = SINPSI*COSTH
      LIA(3,1) = -SINTH
      LIA(1,2) = COSPSI*SINTH*SINPHI - SINPSI*COSPHI
      LIA(2,2) = SINPSI*SINTH*SINPHI + COSPSI*COSPHI
      LIA(3,2) = COSTH*SINPHI
      LIA(1,3) = COSPSI*SINTH*COSPHI + SINPSI*SINPHI
      LIA(2,3) = SINPSI*SINTH*COSPHI - COSPSI*SINPHI
      LIA(3,3) = COSTH*COSPHI
C
C     FORM TA TRANSFORMATION MATRIX
C
      TA(1,2)=SINPHI*TANTH
      TA(1,3)=COSPHI*TANTH
      TA(2,2)=COSPHI
      TA(2,3)=-SINPHI
      TA(3,2)=SINPHI/COSTH
      TA(3,3)=COSPHI/COSTH
C
C     Compute Inertial Accleration A2
C
      CALL VMULT(LIA,ACA,A2,3,3,1)
C
C     Compute euler Rates BETAAD
C
      CALL VMULT(TA,WAA,BETAAD,3,3,1)

      RETURN
      END
```

8.8. newopt4.f

```
C***** SUBROUTINE NEWOPT4.F ******
C
C  NONLINEAR WASHOUT ALGORITHM: ANG. VELOCITY DEVELOPMENT.
C  GIVEN  A/C ACCELS ACA AND EULER RATES BETAAD,
C  COMPUTE SIMULATOR INERTIAL DISPLACEMENT AND EULER ANGLES.
C
   SUBROUTINE NEWOPT4(MODE)

      INCLUDE 'comint2.com'

      INCLUDE 'wcom2.com'

      INCLUDE 'optint3.com'

      INCLUDE 'nopt4.com'

      INCLUDE 'matrix1c.com'

C     DATA IRESET/0/,IHOLD/0/

C
C     Compute Fairing Parameters
C
      SQWASH=SQWASHP*EA+SQWASHI*(1.0-EA)
      DELSQ=MAX(MIN(SQWASH-SQWASHP,1.0),-1.0)
      SQWASHP=SQWASH+DELSQ

      A=1.0-SQWASHP
      AA=1.0-SQWASHI
C
C     Fairing of Heave Nonlinear Gain and Limit
C
      GZ4(1)=AA*GZ40(1)+SQWASHI*GZ4S(1)
      GZ4(2)=AA*GZ40(2)+SQWASHI*GZ4S(2)
      GZ4(3)=AA*GZ40(3)+SQWASHI*GZ4S(3)
      AMX4=AA*AMX40+SQWASHI*AMX4S
C
C     Fairing of Runway Roughness Amplitude
C
      XKA=A*XKA0+SQWASHP*XKAS

      IF(MODE.EQ.1) THEN
         H = DT
C
C     Set "old" variables for future use in HOLD and OPERATE modes
C
         DO I=1,3
            A2NO(I)=0.
            BADNO(I)=0.
         END DO

         DO I=1,9
            XXO(I)=0.
            XYO(I)=0.
```

```
            END DO

            DO J=1,11
               DO I=1,11
                  IF (I.GE.J) THEN
                     PX(I,J)=PXVEC((J-1)*11-J*(J-1)/2+I)
                     PX(J,I)=PX(I,J)
                     PY(I,J)=PYVEC((J-1)*11-J*(J-1)/2+I)
                     PY(J,I)=PY(I,J)
                  END IF
               END DO
            END DO

            DO I=1,4
               XRO(I)=0.
            END DO

            DO J=1,5
               DO I=1,5
                  IF (I.GE.J) THEN
                     PR(I,J)=PRVEC((J-1)*5-J*(J-1)/2+I)
                     PR(J,I)=PR(I,J)
                  END IF
               END DO
            END DO

            DO I=1,5
               XZO(I)=0.
            END DO

            DO J=1,6
               DO I=1,6
                  IF (I.GE.J) THEN
                     PZ(I,J)=PZVEC((J-1)*6-J*(J-1)/2+I)
                     PZ(J,I)=PZ(I,J)
                  END IF
               END DO
            END DO

            CALL RESETC2
C
C     Set "old" variables for future use in HOLD and OPERATE modes
C
            BETASRO(1)=PHI
            BETASRO(2)=THE
            BSDRO(1)=PHID
            BSDRO(2)=THED

            DO I=1,2
               XHO(I)=0.
               ACAO(I)=0.
               BSDTO(I)=0.
               BETASTO(I)=0.
               BETASTLO(I)=0.
               XTO=0.
               XT2O=0.
               WGUSTO=0.
```

```
            ACZT=0.
         END DO

         GO TO 1
      END IF

      IF(MODE.EQ.2) THEN
         CALL WTRIM3
      END IF
C
C     Compute Augmented Acceleration from W-Gust
C
      IF(MODE.EQ.3) THEN
         WGAV=0.5*(WGUST+WGUSTO)
         XT=XTO+DT*(-G2D1*XTO+XT2O+(G2N1-G2D1*G2N2)*WGAV)
         XTAV=0.5*(XT+XTO)
         XT2=XT2O+DT*(-G2D0*XTAV+(G2N0-G2D0*G2N2)*WGAV)
         ACZT=XT+G2N2*WGUST
         XTO=XT
         XT2O=XT2
         WGUSTO=WGUST
C
C     (first-order turbulence model no longer used)
C
C        XT=XTO+DT*(-G1D0*XTO+(G1N0-G1D0*G1N1)*WGUSTO)
C        XTO=XT
C        WGUSTO=WGUST
C        ACZT=XT+G1N1*WGUST
      END IF

      CALL LIBA
      CALL GAINOPT4
         A2N(3)=A2N(3)+GT4*ACZT

      IF(MODE.EQ.3) THEN
         CALL NFILX
         CALL NFILY
         CALL NFILZ
         CALL NFILR
         CALL STATE4
         ELSE IF(MODE.EQ.2) THEN
            CALL STATE4
            CALL STATE4
            CALL STATE4
            CALL STATE4
      END IF


      IF(MODE.EQ.3) CALL INTEG4

    1 RETURN
      END
C
C
```

8.9. nfilr.f

```
C
C     Yaw Filter Neurocomputing Solver for the Riccati Equation
C
      SUBROUTINE NFILR

      INCLUDE 'comint2.com'

      INCLUDE 'nopt4.com'

      REAL SPR(5,5),SUMER,SUMPR,SUMPRT
      REAL PAPR(5,5),UPBR(5,5),UAPR(5,5),APUR(5,5)
      REAL EUR(5,5),UR(5),PZR(5)
C
C     Compute Prescribed Nonlinearity Alpha
C
      ALPR=PSI*Q2R*PSI
      IF(ALPR.GE.ALPRMAX) ALPR=ALPRMAX
      DO I=1,5
         APR(I,I)=APRO(I)+ALPR
      END DO
C
C     Start Training Iterations & Initialize Variables
C
      DO L=1,3

         DO I=1,5
            DO J=1,5
               SPR(I,J)=0.
               UPBR(I,J)=0.
               PAPR(I,J)=0.
               UAPR(I,J)=0.
               APUR(I,J)=0.
            END DO
         END DO
C
C     Compute Matrix Products SP and PA
C
         DO I=1,5
            DO J=1,5
               DO K=1,5
                  SPR(I,J)=SPR(I,J)+BRBR(I,K)*PR(K,J)
               END DO
            END DO
            DO K=1,5
               PAPR(I,1)=PAPR(I,1)+PR(I,K)*APR(K,1)
               PAPR(I,3)=PAPR(I,3)+PR(I,K)*APR(K,3)
               PAPR(I,5)=PAPR(I,5)+PR(I,K)*APR(K,5)
            END DO
            PAPR(I,2)=PR(I,1)*APR(1,2)+PR(I,2)*APR(2,2)
            PAPR(I,4)=PR(I,4)*APR(4,4)
         END DO
C
C     Compute Error Signal v, Vector p and Matrix Product pz
C     Note: Matrix Product A'P is transpose of PA by symmetry of P
```

```
C
         DO I=1,5
            UR(I)=0.
            PZR(I)=0.
            DO J=1,5
               IF(I.LE.J) THEN
                  SUMER=0.
                  DO K=1,5
                     SUMER=SUMER+PR(I,K)*SPR(K,J)
                  END DO
                  EUR(I,J)=SUMER-PAPR(I,J)-PAPR(J,I)-R1PR(I,J)
                  EUR(J,I)=EUR(I,J)
               END IF
               UR(I)=UR(I)+EUR(I,J)*ZR(L,J)
               PZR(I)=PZR(I)+PR(I,J)*ZR(L,J)
            END DO
         END DO
C
C     Compute Matrix Product Avz
C
         DO I=1,2
            DO J=1,5
               DO K=1,3
                  APUR(I,J)=APUR(I,J)+APR(I,K)*UR(K)*ZR(L,J)
               END DO
                  APUR(I,J)=APUR(I,J)+APR(I,5)*UR(5)*ZR(L,J)
            END DO
         END DO
         DO I=3,4
            DO J=1,5
                  APUR(I,J)=APUR(I,J)+APR(I,1)*UR(1)*ZR(L,J)
               DO K=3,5
                  APUR(I,J)=APUR(I,J)+APR(I,K)*UR(K)*ZR(L,J)
               END DO
            END DO
         END DO
            DO J=1,5
               APUR(5,J)=APR(5,5)*UR(5)*ZR(L,J)
            END DO
C
C     Compute Matrix Product vzA
C
         DO I=1,5
            DO K=1,5
               UAPR(I,1)=UAPR(I,1)+UR(I)*ZR(L,K)*APR(K,1)
               UAPR(I,3)=UAPR(I,3)+UR(I)*ZR(L,K)*APR(K,3)
               UAPR(I,5)=UAPR(I,5)+UR(I)*ZR(L,K)*APR(K,5)
            END DO
            UAPR(I,2)=UR(I)*ZR(L,1)*APR(1,2)
     x         +UR(I)*ZR(L,2)*APR(2,2)
            UAPR(I,4)=UR(I)*ZR(L,4)*APR(4,4)
         END DO
C
C     Compute Matrix Product vp'S
C
         DO I=1,5
            DO J=1,4
```

```
                  DO K=1,4
                     UPBR(I,J)=UPBR(I,J)+UR(I)*PZR(K)*BRBR(K,J)
                  END DO
               END DO
            END DO
C
C     Sum Avz + vzA + vp'S and integrate to update Riccati Solution P
C
            DO I=1,5
               DO J=1,5
                  IF(I.LE.J) THEN
                     SUMPR=APUR(I,J)+UAPR(I,J)-UPBR(I,J)
                     IF(I.NE.J) THEN
                        SUMPRT=APUR(J,I)+UAPR(J,I)-UPBR(J,I)
                     ELSE
                        SUMPRT=SUMPR
                     END IF
                     PR(I,J)=PR(I,J)+MUR*(SUMPR+SUMPRT)
                     PR(J,I)=PR(I,J)
                  END IF
               END DO
            END DO

         END DO

         RETURN
         END
```

8.10. nfilx.f

```
C
C     Surge Filter Neurocomputing Solver for the Riccati Equation
C
      SUBROUTINE NFILX

      INCLUDE 'comint2.com'

      INCLUDE 'nopt4.com'

      REAL SPX(11,11),SUMEX,SUMPX,SUMPXT
      REAL PAPX(11,11),UPBX(11,11),UAPX(11,11),APUX(11,11)
      REAL EUX(11,11),UX(11),PZX(11)

C
C     Compute Prescribed Nonlinearity Alpha
C
      ALPX=X*Q2X(1)*X+XD*Q2X(2)*XD
      IF(ALPX.GT.ALPXMAX) ALPX=ALPXMAX
      DO I=1,11
         APX(I,I)=APXO(I)+ALPX
      END DO
C
C     Start Training Iterations & Initialize Variables
C
      DO L=1,3

         DO I=1,11
            DO J=1,11
               SPX(I,J)=0.0
               PAPX(I,J)=0.0
               UPBX(I,J)=0.0
               UAPX(I,J)=0.0
               APUX(I,J)=0.0
            END DO
         END DO
C
C     Compute Matrix Product SP
C
         DO I=1,6
            DO J=1,11
               DO K=1,6
                  SPX(I,J)=SPX(I,J)+BRBX(I,K)*PX(K,J)
               END DO
                  SPX(I,J)=SPX(I,J)+BRBX(I,9)*PX(9,J)
            END DO
         END DO
            DO J=1,11
               DO K=1,6
                  SPX(9,J)=SPX(9,J)+BRBX(9,K)*PX(K,J)
               END DO
                  SPX(9,J)=SPX(9,J)+BRBX(9,9)*PX(9,J)
            END DO
C
C     Compute Matrix Product PA
```

```
C
         DO I=1,11
            DO J=1,6
               DO K=1,6
                  PAPX(I,J)=PAPX(I,J)+PX(I,K)*APX(K,J)
               END DO
            END DO
            PAPX(I,7)=PX(I,7)*APX(7,7)
            PAPX(I,8)=PX(I,7)*APX(7,8)+PX(I,8)*APX(8,8)
            PAPX(I,9)=PX(I,8)*APX(8,9)+PX(I,9)*APX(9,9)
               DO K=1,6
                  PAPX(I,10)=PAPX(I,10)+PX(I,K)*APX(K,10)
               END DO
                  PAPX(I,10)=PAPX(I,10)+PX(I,10)*APX(10,10)
               DO K=1,6
                  PAPX(I,11)=PAPX(I,11)+PX(I,K)*APX(K,11)
               END DO
                  PAPX(I,11)=PAPX(I,11)+PX(I,11)*APX(11,11)
         END DO
C
C     Compute Error Signal v, Vector p and Matrix Product vz
C     Note: Matrix Product A'P is transpose of PA by symmetry of P
C
         DO I=1,11
            UX(I)=0.
            PZX(I)=0.
            DO J=1,11
               IF(I.LE.J) THEN
                  SUMEX=0.
                  DO K=1,11
                     SUMEX=SUMEX+PX(I,K)*SPX(K,J)
                  END DO
                  EUX(I,J)=SUMEX-PAPX(J,I)-PAPX(I,J)-R1PX(I,J)
                  EUX(J,I)=EUX(I,J)
               END IF
               UX(I)=UX(I)+EUX(I,J)*ZX(L,J)
               PZX(I)=PZX(I)+PX(I,J)*ZX(L,J)
            END DO
         END DO
C
C     Compute Matrix Product Avz
C
         DO I=1,6
            DO J=1,11
               DO K=1,6
                  APUX(I,J)=APUX(I,J)+APX(I,K)*UX(K)*ZX(L,J)
               END DO
                  APUX(I,J)=APUX(I,J)+APX(I,10)*UX(10)*ZX(L,J)
     x            +APX(I,11)*UX(11)*ZX(L,J)
            END DO
         END DO
            DO J=1,11
               APUX(7,J)=APX(7,7)*UX(7)*ZX(L,J)
     x         +APX(7,8)*UX(8)*ZX(L,J)
               APUX(8,J)=APX(8,8)*UX(8)*ZX(L,J)
     x         +APX(8,9)*UX(9)*ZX(L,J)
               APUX(9,J)=APX(9,9)*UX(9)*ZX(L,J)
```

```
                APUX(10,J)=APX(10,10)*UX(10)*ZX(L,J)
                APUX(11,J)=APX(11,11)*UX(11)*ZX(L,J)
             END DO
C
C     Compute Matrix Product vzA
C
          DO I=1,11
             DO J=1,6
                DO K=1,6
                   UAPX(I,J)=UAPX(I,J)+UX(I)*ZX(L,K)*APX(K,J)
                END DO
             END DO
             UAPX(I,7)=UX(I)*ZX(L,7)*APX(7,7)
             UAPX(I,8)=UX(I)*ZX(L,7)*APX(7,8)
     x             +UX(I)*ZX(L,8)*APX(8,8)
             UAPX(I,9)=UX(I)*ZX(L,8)*APX(8,9)
     x             +UX(I)*ZX(L,9)*APX(9,9)
             DO K=1,6
                 UAPX(I,10)=UAPX(I,10)+UX(I)*ZX(L,K)*APX(K,10)
             END DO
                 UAPX(I,10)=UAPX(I,10)+UX(I)*ZX(L,10)*APX(10,10)
             DO K=1,6
                 UAPX(I,11)=UAPX(I,11)+UX(I)*ZX(L,K)*APX(K,11)
             END DO
                 UAPX(I,11)=UAPX(I,11)+UX(I)*ZX(L,11)*APX(11,11)
          END DO
C
C     Compute Matrix Product vp'S
C
          DO I=1,11
             DO J=1,6
                DO K=1,6
                    UPBX(I,J)=UPBX(I,J)+UX(I)*PZX(K)*BRBX(K,J)
                END DO
                    UPBX(I,J)=UPBX(I,J)+UX(I)*PZX(9)*BRBX(9,J)
             END DO
                DO K=1,6
                    UPBX(I,9)=UPBX(I,9)+UX(I)*PZX(K)*BRBX(K,9)
                END DO
                    UPBX(I,9)=UPBX(I,9)+UX(I)*PZX(9)*BRBX(9,9)
          END DO
C
C     Sum Avz + vzA + vp'S and integrate to update Riccati Solution P
C
          DO I=1,11
             DO J=1,11
                 IF(I.LE.J) THEN
                   SUMPX=APUX(I,J)+UAPX(I,J)-UPBX(I,J)
                   IF(I.NE.J) THEN
                   SUMPXT=APUX(J,I)+UAPX(J,I)-UPBX(J,I)
                   ELSE
                   SUMPXT=SUMPX
                   END IF
                   PX(I,J)=PX(I,J)+MUX*(SUMPX+SUMPXT)
                   PX(J,I)=PX(I,J)
                 END IF
             END DO
```

```
   END DO
END DO

RETURN
END
```

8.11. nfily.f

```
C
C     Sway Filter Neurocomputing Solver for the Riccati Equation
C
      SUBROUTINE NFILY

      INCLUDE 'comint2.com'

      INCLUDE 'nopt4.com'

      REAL SPY(11,11),SUMEY,SUMPY,SUMPYT
      REAL PAPY(11,11),UPBY(11,11),UAPY(11,11),APUY(11,11)
      REAL EUY(11,11),UY(11),PZY(11)

C
C     Compute Prescribed Nonlinearity Alpha
C
      ALPY=Y*Q2Y(1)*Y+YD*Q2Y(2)*YD
      IF(ALPY.GT.ALPYMAX) ALPY=ALPYMAX
      DO I=1,11
         APY(I,I)=APYO(I)+ALPY
      END DO
C
C     Start Training Iterations & Initialize Variables
C
      DO L=1,3

         DO I=1,11
            DO J=1,11
               SPY(I,J)=0.0
               PAPY(I,J)=0.0
               UPBY(I,J)=0.0
               UAPY(I,J)=0.0
               APUY(I,J)=0.0
            END DO
         END DO
C
C     Compute Matrix Product SP
C
         DO I=1,6
            DO J=1,11
               DO K=1,6
                  SPY(I,J)=SPY(I,J)+BRBY(I,K)*PY(K,J)
               END DO
                  SPY(I,J)=SPY(I,J)+BRBY(I,9)*PY(9,J)
            END DO
         END DO
            DO J=1,11
               DO K=1,6
                  SPY(9,J)=SPY(9,J)+BRBY(9,K)*PY(K,J)
               END DO
                  SPY(9,J)=SPY(9,J)+BRBY(9,9)*PY(9,J)
            END DO
C
C     Compute Matrix Product PA
```

```
C
         DO I=1,11
            DO J=1,6
               DO K=1,6
                  PAPY(I,J)=PAPY(I,J)+PY(I,K)*APY(K,J)
               END DO
            END DO
            PAPY(I,7)=PY(I,7)*APY(7,7)
            PAPY(I,8)=PY(I,7)*APY(7,8)+PY(I,8)*APY(8,8)
            PAPY(I,9)=PY(I,8)*APY(8,9)+PY(I,9)*APY(9,9)
               DO K=1,6
                  PAPY(I,10)=PAPY(I,10)+PY(I,K)*APY(K,10)
               END DO
                  PAPY(I,10)=PAPY(I,10)+PY(I,10)*APY(10,10)
               DO K=1,6
                  PAPY(I,11)=PAPY(I,11)+PY(I,K)*APY(K,11)
               END DO
                  PAPY(I,11)=PAPY(I,11)+PY(I,11)*APY(11,11)
         END DO
C
C     Compute Error Signal v, Vector p and Matrix Product vz
C     Note: Matrix Product A'P is transpose of PA by symmetry of P
C
         DO I=1,11
            UY(I)=0.
            PZY(I)=0.
            DO J=1,11
               IF(I.LE.J) THEN
                  SUMEY=0.
                  DO K=1,11
                     SUMEY=SUMEY+PY(I,K)*SPY(K,J)
                  END DO
                  EUY(I,J)=SUMEY-PAPY(J,I)-PAPY(I,J)-R1PY(I,J)
                  EUY(J,I)=EUY(I,J)
               END IF
               UY(I)=UY(I)+EUY(I,J)*ZY(L,J)
               PZY(I)=PZY(I)+PY(I,J)*ZY(L,J)
            END DO
         END DO

C
C     Compute Matrix Product Avz
C
         DO I=1,6
            DO J=1,11
               DO K=1,6
                  APUY(I,J)=APUY(I,J)+APY(I,K)*UY(K)*ZY(L,J)
               END DO
                  APUY(I,J)=APUY(I,J)+APY(I,10)*UY(10)*ZY(L,J)
     x               +APY(I,11)*UY(11)*ZY(L,J)
            END DO
         END DO
            DO J=1,11
               APUY(7,J)=APY(7,7)*UY(7)*ZY(L,J)
     x         +APY(7,8)*UY(8)*ZY(L,J)
               APUY(8,J)=APY(8,8)*UY(8)*ZY(L,J)
     x         +APY(8,9)*UY(9)*ZY(L,J)
```

```
               APUY(9,J)=APY(9,9)*UY(9)*ZY(L,J)
               APUY(10,J)=APUY(10,J)+APY(10,10)*UY(10)*ZY(L,J)
               APUY(11,J)=APUY(11,J)+APY(11,11)*UY(11)*ZY(L,J)
            END DO
C
C     Compute Matrix Product vzA
C
         DO I=1,11
            DO J=1,6
               DO K=1,6
                  UAPY(I,J)=UAPY(I,J)+UY(I)*ZY(L,K)*APY(K,J)
               END DO
            END DO
            UAPY(I,7)=UY(I)*ZY(L,7)*APY(7,7)
            UAPY(I,8)=UY(I)*ZY(L,7)*APY(7,8)
      x              +UY(I)*ZY(L,8)*APY(8,8)
            UAPY(I,9)=UY(I)*ZY(L,8)*APY(8,9)
      x              +UY(I)*ZY(L,9)*APY(9,9)
            DO K=1,6
                UAPY(I,10)=UAPY(I,10)+UY(I)*ZY(L,K)*APY(K,10)
            END DO
                UAPY(I,10)=UAPY(I,10)+UY(I)*ZY(L,10)*APY(10,10)
               DO K=1,6
                  UAPY(I,11)=UAPY(I,11)+UY(I)*ZY(L,K)*APY(K,11)
               END DO
                  UAPY(I,11)=UAPY(I,11)+UY(I)*ZY(L,11)*APY(11,11)
         END DO
C
C     Compute Matrix Product vp'S
C
         DO I=1,11
            DO J=1,6
               DO K=1,6
                  UPBY(I,J)=UPBY(I,J)+UY(I)*PZY(K)*BRBY(K,J)
               END DO
                  UPBY(I,J)=UPBY(I,J)+UY(I)*PZY(9)*BRBY(9,J)
            END DO
               DO K=1,6
                  UPBY(I,9)=UPBY(I,9)+UY(I)*PZY(K)*BRBY(K,9)
               END DO
                  UPBY(I,9)=UPBY(I,9)+UY(I)*PZY(9)*BRBY(9,9)
         END DO
C
C     Sum Avz + vzA + vp'S and integrate to update Riccati Solution P
C
         DO I=1,11
            DO J=1,11
               IF(I.LE.J) THEN
                  SUMPY=APUY(I,J)+UAPY(I,J)-UPBY(I,J)
                  IF(I.NE.J) THEN
                     SUMPYT=APUY(J,I)+UAPY(J,I)-UPBY(J,I)
                  ELSE
                     SUMPYT=SUMPY
                  END IF
                  PY(I,J)=PY(I,J)+MUY*(SUMPY+SUMPYT)
                  PY(J,I)=PY(I,J)
               END IF
```

```
      END DO
   END DO

END DO

RETURN
END
```

8.12. nfilz.f

```
C
C     Heave Filter Neurocomputing Solver for the Riccati Equation
C
      SUBROUTINE NFILZ

      INCLUDE 'comint2.com'

      INCLUDE 'nopt4.com'

      REAL SPZ(6,6),SUMEZ,SUMPZ,SUMPZT
      REAL PAPZ(6,6),UPBZ(6,6),UAPZ(6,6),APUZ(6,6)
      REAL EUZ(6,6),UZ(6),PZZ(6)

C
C     Compute Prescribed Nonlinearity Alpha
C
      ALPZ=Z*Q2Z(1)*Z+ZD*Q2Z(2)*ZD
      IF(ALPZ.GT.ALPZMAX) ALPZ=ALPZMAX
      DO I=1,6
         APZ(I,I)=APZO(I)+ALPZ
      END DO
C
C     Start Training Iterations & Initialize Variables
C
      DO L=1,3

         DO I=1,6
            DO J=1,6
               SPZ(I,J)=0.
               PAPZ(I,J)=0.
               UPBZ(I,J)=0.
               UAPZ(I,J)=0.
               APUZ(I,J)=0.
            END DO
         END DO
C
C     Compute Matrix Product SP
C
         DO I=1,2
            DO J=1,6
               SPZ(I,J)=BRBZ(I,1)*PZ(1,J)
     x                 +BRBZ(I,2)*PZ(2,J)+BRBZ(I,5)*PZ(5,J)
            END DO
         END DO
            DO J=1,6
               SPZ(5,J)=BRBZ(5,1)*PZ(1,J)
     x                 +BRBZ(5,2)*PZ(2,J)+BRBZ(5,5)*PZ(5,J)
            END DO
C
C     Compute Matrix Product PA
C

         DO I=1,6
            PAPZ(I,1)=PZ(I,1)*APZ(1,1)+PZ(I,2)*APZ(2,1)
```

```
            PAPZ(I,2)=PZ(I,1)*APZ(1,2)+PZ(I,2)*APZ(2,2)
            PAPZ(I,3)=PZ(I,3)*APZ(3,3)
            PAPZ(I,4)=PZ(I,3)*APZ(3,4)+PZ(I,4)*APZ(4,4)
            PAPZ(I,5)=PZ(I,4)*APZ(4,5)+PZ(I,5)*APZ(5,5)
            PAPZ(I,6)=PZ(I,1)*APZ(1,6)+PZ(I,2)*APZ(2,6)
     x              +PZ(I,6)*APZ(6,6)
         END DO
C
C     Compute Error Signal v, Vector p and Matrix Product vz
C     Note: Matrix Product A'P is transpose of PA by symmetry of P
C
         DO I=1,6
            UZ(I)=0.
            PZZ(I)=0.
            DO J=1,6
               IF(I.LE.J) THEN

                  SUMEZ=0.
                  DO K=1,6
                     SUMEZ=SUMEZ+PZ(I,K)*SPZ(K,J)
                  END DO
                  EUZ(I,J)=SUMEZ-PAPZ(J,I)-PAPZ(I,J)-R1PZ(I,J)
                  EUZ(J,I)=EUZ(I,J)
               END IF
               UZ(I)=UZ(I)+EUZ(I,J)*ZZ(L,J)
               PZZ(I)=PZZ(I)+PZ(I,J)*ZZ(L,J)
            END DO
         END DO
C
C     Compute Matrix Product Avz
C
         DO I=1,2
            DO J=1,6
               APUZ(I,J)=APZ(I,1)*UZ(1)*ZZ(L,J)
     x             +APZ(I,2)*UZ(2)*ZZ(L,J)+APZ(I,6)*UZ(6)*ZZ(L,J)
            END DO
         END DO
            DO J=1,6
               APUZ(3,J)=APZ(3,3)*UZ(3)*ZZ(L,J)
     x             +APZ(3,4)*UZ(4)*ZZ(L,J)
               APUZ(4,J)=APZ(4,4)*UZ(4)*ZZ(L,J)
     x             +APZ(4,5)*UZ(5)*ZZ(L,J)
               APUZ(5,J)=APZ(5,5)*UZ(5)*ZZ(L,J)
               APUZ(6,J)=APZ(6,6)*UZ(6)*ZZ(L,J)
            END DO
C
C     Compute Matrix Product vzA
C
         DO I=1,6
            UAPZ(I,1)=UZ(I)*ZZ(L,1)*APZ(1,1)+UZ(I)*ZZ(L,2)*APZ(2,1)
            UAPZ(I,2)=UZ(I)*ZZ(L,1)*APZ(1,2)+UZ(I)*ZZ(L,2)*APZ(2,2)
            UAPZ(I,3)=UZ(I)*ZZ(L,3)*APZ(3,3)
            UAPZ(I,4)=UZ(I)*ZZ(L,3)*APZ(3,4)+UZ(I)*ZZ(L,4)*APZ(4,4)
            UAPZ(I,5)=UZ(I)*ZZ(L,4)*APZ(4,5)+UZ(I)*ZZ(L,5)*APZ(5,5)
            UAPZ(I,6)=UZ(I)*ZZ(L,1)*APZ(1,6)
     x              +UZ(I)*ZZ(L,2)*APZ(2,6)+UZ(I)*ZZ(L,6)*APZ(6,6)
```

```
         END DO
C
C     Compute Matrix Product vp'S
C
         DO I=1,6
            DO J=1,2
               UPBZ(I,J)=UZ(I)*PZZ(1)*BRBZ(1,J)
     x             +UZ(I)*PZZ(2)*BRBZ(2,J)+UZ(I)*PZZ(5)*BRBZ(5,J)
            END DO
               UPBZ(I,5)=UZ(I)*PZZ(1)*BRBZ(1,5)
     x             +UZ(I)*PZZ(2)*BRBZ(2,5)+UZ(I)*PZZ(5)*BRBZ(5,5)

         END DO
C
C     Sum Avz + vzA + vp'S and integrate to update Riccati Solution P
C
         DO I=1,6
            DO J=1,6
               IF(I.LE.J) THEN
                  SUMPZ=APUZ(I,J)+UAPZ(I,J)-UPBZ(I,J)
                  IF(I.NE.J) THEN
                     SUMPZT=APUZ(J,I)+UAPZ(J,I)-UPBZ(J,I)
                  ELSE
                     SUMPZT=SUMPZ
                  END IF
                  PZ(I,J)=PZ(I,J)+MUZ*(SUMPZ+SUMPZT)
                  PZ(J,I)=PZ(I,J)
               END IF
            END DO
         END DO

      END DO

      RETURN
      END
```

8.13. ofil3.f

```
C     OPTIMAL FILTERS (ALL IN INERTIAL FRAME):
C
C         6TH ORDER PHIAD   TO PHISD
C         6TH ORDER A2(2)  TO PHISD
C         6TH ORDER A2(2)  TO AYSI
C         6TH ORDER THETAAD TO THETASD
C         6TH ORDER A2(1)  TO THETASD
C         6TH ORDER A2(1)  TO AXSI
C         4TH ORDER PSIA   TO PSIS
C         4TH ORDER A2(3)  TO AZSI
C

      SUBROUTINE OFIL3

      REAL B1(8),C1(8),D1(8),E1(8),F1(6),G1(6)
      REAL B2(8),C2(8),D2(8),E2(8),F2(6),G2(6)

      INCLUDE 'matrix1c.com'

      INCLUDE 'optint3.com'

      INCLUDE 'wopt3.com'

C
C********************** BLOCK NO. C1 ****************************
C  FILTER FROM PHIAD TO PHISD:
C
      B1(1)=DT*(-R1D5*X1O(1)+X2O(1)+(R1N5-R1D5*R1N6)*BADNO(1))
      C1(1)=DT*(-R1D4*X1O(1)+X3O(1)+(R1N4-R1D4*R1N6)*BADNO(1))
      D1(1)=DT*(-R1D3*X1O(1)+X4O(1)+(R1N3-R1D3*R1N6)*BADNO(1))
      E1(1)=DT*(-R1D2*X1O(1)+X5O(1)+(R1N2-R1D2*R1N6)*BADNO(1))
      F1(1)=DT*(-R1D1*X1O(1)+X6O(1)+(R1N1-R1D1*R1N6)*BADNO(1))
      G1(1)=DT*(-R1D0*X1O(1)       +(R1N0-R1D0*R1N6)*BADNO(1))

      B2(1)=DT*
     x   (-R1D5*(X1O(1)+B1(1))+X2O(1)+C1(1)+(R1N5-R1D5*R1N6)*BADN(1))
   C2(1)=DT*
     x   (-R1D4*(X1O(1)+B1(1))+X3O(1)+D1(1)+(R1N4-R1D4*R1N6)*BADN(1))
   D2(1)=DT*
     x   (-R1D3*(X1O(1)+B1(1))+X4O(1)+E1(1)+(R1N3-R1D3*R1N6)*BADN(1))
   E2(1)=DT*
     x   (-R1D2*(X1O(1)+B1(1))+X5O(1)+F1(1)+(R1N2-R1D2*R1N6)*BADN(1))
   F2(1)=DT*
     x   (-R1D1*(X1O(1)+B1(1))+X6O(1)+G1(1)+(R1N1-R1D1*R1N6)*BADN(1))
   G2(1)=DT*
     x   (-R1D0*(X1O(1)+B1(1))+(R1N0-R1D0*R1N6)*BADN(1))

      XX1(1)=X1O(1)+0.5*(B1(1)+B2(1))
      X2(1)=X2O(1)+0.5*(C1(1)+C2(1))
      X3(1)=X3O(1)+0.5*(D1(1)+D2(1))
      X4(1)=X4O(1)+0.5*(E1(1)+E2(1))
      X5(1)=X5O(1)+0.5*(F1(1)+F2(1))
      X6(1)=X6O(1)+0.5*(G1(1)+G2(1))
C
```

```
C************************ BLOCK NO. C2 ****************************
C   FILTER FROM A2(2) TO PHISD:
C
      B1(2)=DT*(-R1D5*X1O(2)+X2O(2)+(R2N5-R1D5*R2N6)*A2NO(2))
      C1(2)=DT*(-R1D4*X1O(2)+X3O(2)+(R2N4-R1D4*R2N6)*A2NO(2))
      D1(2)=DT*(-R1D3*X1O(2)+X4O(2)+(R2N3-R1D3*R2N6)*A2NO(2))
      E1(2)=DT*(-R1D2*X1O(2)+X5O(2)+(R2N2-R1D2*P2N6)*A2NO(2))
      F1(2)=DT*(-R1D1*X1O(2)+X6O(2)+(R2N1-R1D1*P2N6)*A2NO(2))
      G1(2)=DT*(-R1D0*X1O(2)       +(R2N0-R1D0*P2N6)*A2NO(2))

      B2(2)=DT*
     x   (-R1D5*(X1O(2)+B1(2))+X2O(2)+C1(2)+(R2N5-R1D5*R2N6)*A2N(2))
      C2(2)=DT*
     x   (-R1D4*(X1O(2)+B1(2))+X3O(2)+D1(2)+(R2N4-R1D4*R2N6)*A2N(2))
      D2(2)=DT*
     x   (-R1D3*(X1O(2)+B1(2))+X4O(2)+E1(2)+(R2N3-R1D3*R2N6)*A2N(2))
      E2(2)=DT*
     x   (-R1D2*(X1O(2)+B1(2))+X5O(2)+F1(2)+(R2N2-R1D2*R2N6)*A2N(2))
      F2(2)=DT*
     x   (-R1D1*(X1O(2)+B1(2))+X6O(2)+G1(2)+(R2N1-R1D1*R2N6)*A2N(2))
      G2(2)=DT*
     x   (-R1D0*(X1O(2)+B1(2))+(R2N0-R1D0*R2N6)*A2N(2))

      XX1(2)=X1O(2)+0.5*(B1(2)+B2(2))
      X2(2)=X2O(2)+0.5*(C1(2)+C2(2))
      X3(2)=X3O(2)+0.5*(D1(2)+D2(2))
      X4(2)=X4O(2)+0.5*(E1(2)+E2(2))
      X5(2)=X5O(2)+0.5*(F1(2)+F2(2))
      X6(2)=X6O(2)+0.5*(G1(2)+G2(2))
C
C************************ BLOCK NO. C3 ****************************
C   FILTER FROM A2(2) TO AYSI:
C
      B1(3)=DT*(-R1D5*X1O(3)+X2O(3)+(R4N5-R1D5*R4N6)*A2NO(2))
      C1(3)=DT*(-R1D4*X1O(3)+X3O(3)+(R4N4-R1D4*R4N6)*A2NO(2))
      D1(3)=DT*(-R1D3*X1O(3)+X4O(3)+(R4N3-R1D3*R4N6)*A2NO(2))
      E1(3)=DT*(-R1D2*X1O(3)+X5O(3)+(R4N2-R1D2*P4N6)*A2NO(2))
      F1(3)=DT*(-R1D1*X1O(3)+X6O(3)+(R4N1-R1D1*P4N6)*A2NO(2))
      G1(3)=DT*(-R1D0*X1O(3)       +(R4N0-R1D0*P4N6)*A2NO(2))

      B2(3)=DT*
  x   (-R1D5*(X1O(3)+B1(3))+X2O(3)+C1(3)+(R4N5-R1D5*R4N6)*A2N(2))
      C2(3)=DT*
  x   (-R1D4*(X1O(3)+B1(3))+X3O(3)+D1(3)+(R4N4-R1D4*R4N6)*A2N(2))
      D2(3)=DT*
  x   (-R1D3*(X1O(3)+B1(3))+X4O(3)+E1(3)+(R4N3-R1D3*R4N6)*A2N(2))
      E2(3)=DT*
  x   (-R1D2*(X1O(3)+B1(3))+X5O(3)+F1(3)+(R4N2-R1D2*R4N6)*A2N(2))
      F2(3)=DT*
  x   (-R1D1*(X1O(3)+B1(3))+X6O(3)+G1(3)+(R4N1-R1D1*R4N6)*A2N(2))
      G2(3)=DT*
  x   (-R1D0*(X1O(3)+B1(3))+(R4N0-R1D0*R4N6)*A2N(2))

      XX1(3)=X1O(3)+0.5*(B1(3)+B2(3))
      X2(3)=X2O(3)+0.5*(C1(3)+C2(3))
      X3(3)=X3O(3)+0.5*(D1(3)+D2(3))
      X4(3)=X4O(3)+0.5*(E1(3)+E2(3))
```

```
      X5(3)=X5O(3)+0.5*(F1(3)+F2(3))
      X6(3)=X6O(3)+0.5*(G1(3)+G2(3))
C
C********************** BLOCK NO. C4 ***************************
C  FILTER FROM THETAAD TO THETASD:
C
      B1(4)=DT*(-P1D5*X1O(4)+X2O(4)+(P1N5-P1D5*P1N6)*BADNO(2))
      C1(4)=DT*(-P1D4*X1O(4)+X3O(4)+(P1N4-P1D4*P1N6)*BADNO(2))
      D1(4)=DT*(-P1D3*X1O(4)+X4O(4)+(P1N3-P1D3*P1N6)*BADNO(2))
      E1(4)=DT*(-P1D2*X1O(4)+X5O(4)+(P1N2-P1D2*P1N6)*BADNO(2))
      F1(4)=DT*(-P1D1*X1O(4)+X6O(4)+(P1N1-P1D1*P1N6)*BADNO(2))
      G1(4)=DT*(-P1D0*X1O(4)       +(P1N0-P1D0*P1N6)*BADNO(2))

      B2(4)=DT*
     x   (-P1D5*(X1O(4)+B1(4))+X2O(4)+C1(4)+(P1N5-P1D5*P1N6)*BADN(2))
      C2(4)=DT*
     x   (-P1D4*(X1O(4)+B1(4))+X3O(4)+D1(4)+(P1N4-P1D4*P1N6)*BADN(2))
      D2(4)=DT*
     x   (-P1D3*(X1O(4)+B1(4))+X4O(4)+E1(4)+(P1N3-P1D3*P1N6)*BADN(2))
      E2(4)=DT*
     x   (-P1D2*(X1O(4)+B1(4))+X5O(4)+F1(4)+(P1N2-P1D2*P1N6)*BADN(2))
      F2(4)=DT*
     x   (-P1D1*(X1O(4)+B1(4))+X6O(4)+G1(4)+(P1N1-P1D1*P1N6)*BADN(2))
      G2(4)=DT*
     x   (-P1D0*(X1O(4)+B1(4))+(P1N0-P1D0*P1N6)*BADN(2))

      XX1(4)=X1O(4)+0.5*(B1(4)+B2(4))
      X2(4)=X2O(4)+0.5*(C1(4)+C2(4))
      X3(4)=X3O(4)+0.5*(D1(4)+D2(4))
      X4(4)=X4O(4)+0.5*(E1(4)+E2(4))
      X5(4)=X5O(4)+0.5*(F1(4)+F2(4))
      X6(4)=X6O(4)+0.5*(G1(4)+G2(4))
C
C********************** BLOCK NO. C5 ***************************
C   FILTER FROM A2(1) TO THETASD:
C
      B1(5)=DT*(-P1D5*X1O(5)+X2O(5)+(P2N5-P1D5*P2N6)*A2NO(1))
      C1(5)=DT*(-P1D4*X1O(5)+X3O(5)+(P2N4-P1D4*P2N6)*A2NO(1))
      D1(5)=DT*(-P1D3*X1O(5)+X4O(5)+(P2N3-P1D3*P2N6)*A2NO(1))
      E1(5)=DT*(-P1D2*X1O(5)+X5O(5)+(P2N2-P1D2*P2N6)*A2NO(1))
      F1(5)=DT*(-P1D1*X1O(5)+X6O(5)+(P2N1-P1D1*P2N6)*A2NO(1))
      G1(5)=DT*(-P1D0*X1O(5)       +(P2N0-P1D0*P2N6)*A2NO(1))

      B2(5)=DT*
     x   (-P1D5*(X1O(5)+B1(5))+X2O(5)+C1(5)+(P2N5-P1D5*P2N6)*A2N(1))
      C2(5)=DT*
     x   (-P1D4*(X1O(5)+B1(5))+X3O(5)+D1(5)+(P2N4-P1D4*P2N6)*A2N(1))
      D2(5)=DT*
     x   (-P1D3*(X1O(5)+B1(5))+X4O(5)+E1(5)+(P2N3-P1D3*P2N6)*A2N(1))
      E2(5)=DT*
     x   (-P1D2*(X1O(5)+B1(5))+X5O(5)+F1(5)+(P2N2-P1D2*P2N6)*A2N(1))
      F2(5)=DT*
     x   (-P1D1*(X1O(5)+B1(5))+X6O(5)+G1(5)+(P2N1-P1D1*P2N6)*A2N(1))
      G2(5)=DT*
     x   (-P1D0*(X1O(5)+B1(5))+(P2N0-P1D0*P2N6)*A2N(1))

      XX1(5)=X1O(5)+0.5*(B1(5)+B2(5))
```

```
      X2(5)=X2O(5)+0.5*(C1(5)+C2(5))
      X3(5)=X3O(5)+0.5*(D1(5)+D2(5))
      X4(5)=X4O(5)+0.5*(E1(5)+E2(5))
      X5(5)=X5O(5)+0.5*(F1(5)+F2(5))
      X6(5)=X6O(5)+0.5*(G1(5)+G2(5))
C
C********************** BLOCK NO. C6 ***************************
C   FILTER FROM A2(1) TO AXSI:
C
      B1(6)=DT*(-P1D5*X1O(6)+X2O(6)+(P4N5-P1D5*P4N6)*A2NO(1))
      C1(6)=DT*(-P1D4*X1O(6)+X3O(6)+(P4N4-P1D4*P4N6)*A2NO(1))
      D1(6)=DT*(-P1D3*X1O(6)+X4O(6)+(P4N3-P1D3*P4N6)*A2NO(1))
      E1(6)=DT*(-P1D2*X1O(6)+X5O(6)+(P4N2-P1D2*P4N6)*A2NO(1))
      F1(6)=DT*(-P1D1*X1O(6)+X6O(6)+(P4N1-P1D1*P4N6)*A2NO(1))
      G1(6)=DT*(-P1D0*X1O(6)       +(P4N0-P1D0*P4N6)*A2NO(1))

      B2(6)=DT*
     x       (-P1D5*(X1O(6)+B1(6))+X2O(6)+C1(6)+(P4N5-P1D5*P4N6)*A2N(1))
      C2(6)=DT*
     x       (-P1D4*(X1O(6)+B1(6))+X3O(6)+D1(6)+(P4N4-P1D4*P4N6)*A2N(1))
      D2(6)=DT*
     x       (-P1D3*(X1O(6)+B1(6))+X4O(6)+E1(6)+(P4N3-P1D3*P4N6)*A2N(1))
      E2(6)=DT*
     x       (-P1D2*(X1O(6)+B1(6))+X5O(6)+F1(6)+(P4N2-P1D2*P4N6)*A2N(1))
      F2(6)=DT*
     x       (-P1D1*(X1O(6)+B1(6))+X6O(6)+G1(6)+(P4N1-P1D1*P4N6)*A2N(1))
      G2(6)=DT*
     x       (-P1D0*(X1O(6)+B1(6))+(P4N0-P1D0*P4N6)*A2N(1))

      XX1(6)=X1O(6)+0.5*(B1(6)+B2(6))
      X2(6)=X2O(6)+0.5*(C1(6)+C2(6))
      X3(6)=X3O(6)+0.5*(D1(6)+D2(6))
      X4(6)=X4O(6)+0.5*(E1(6)+E2(6))
      X5(6)=X5O(6)+0.5*(F1(6)+F2(6))
      X6(6)=X6O(6)+0.5*(G1(6)+G2(6))
C
C********************** BLOCK NO. C7 ***************************
C  FILTER FROM PSIAD2 TO PSISD2:
C
      B1(7)=DT*(-Y1D3*X1O(7)+X2O(7)+(Y1N3-Y1D3*Y1N4)*BADNO(3))
      C1(7)=DT*(-Y1D2*X1O(7)+X3O(7)+(Y1N2-Y1D2*Y1N4)*BADNO(3))
      D1(7)=DT*(-Y1D1*X1O(7)+X4O(7)+(Y1N1-Y1D1*Y1N4)*BADNO(3))
      E1(7)=DT*(-Y1D0*X1O(7)+       (Y1N0-Y1D0*Y1N4)*BADNO(3))

      B2(7)=DT*
     x       (-Y1D3*(X1O(7)+B1(7))+X2O(7)+C1(7)+(Y1N3-Y1D3*Y1N4)*BADN(3))
      C2(7)=DT*
     x       (-Y1D2*(X1O(7)+B1(7))+X3O(7)+D1(7)+(Y1N2-Y1D2*Y1N4)*BADN(3))
      D2(7)=DT*
     x       (-Y1D1*(X1O(7)+B1(7))+X4O(7)+E1(7)+(Y1N1-Y1D1*Y1N4)*BADN(3))
      E2(7)=DT*
     x       (-Y1D0*(X1O(7)+B1(7))+(Y1N0-Y1D0*Y1N4)*BADN(3))

      XX1(7)=X1O(7)+0.5*(B1(7)+B2(7))
      X2(7)=X2O(7)+0.5*(C1(7)+C2(7))
      X3(7)=X3O(7)+0.5*(D1(7)+D2(7))
      X4(7)=X4O(7)+0.5*(E1(7)+E2(7))
```

```
C
C********************** BLOCK NO. C8 ***************************
C   FILTER FROM A2(3) TO AZSI:
C
      B1(8)=DT*(-H1D3*X1O(8)+X2O(8)+(H1N3-H1D3*H1N4)*A2NO(3))
      C1(8)=DT*(-H1D2*X1O(8)+X3O(8)+(H1N2-H1D2*H1N4)*A2NO(3))
      D1(8)=DT*(-H1D1*X1O(8)+X4O(8)+(H1N1-H1D1*H1N4)*A2NO(3))
      E1(8)=DT*(-H1D0*X1O(8)+       (H1N0-H1D0*H1N4)*A2NO(3))

      B2(8)=DT*
     x   (-H1D3*(X1O(8)+B1(8))+X2O(8)+C1(8)+(H1N3-H1D3*H1N4)*A2N(3))
      C2(8)=DT*
     x   (-H1D2*(X1O(8)+B1(8))+X3O(8)+D1(8)+(H1N2-H1D2*H1N4)*A2N(3))
      D2(8)=DT*
     x   (-H1D1*(X1O(8)+B1(8))+X4O(8)+E1(8)+(H1N1-H1D1*H1N4)*A2N(3))
      E2(8)=DT*
     x   (-H1D0*(X1O(8)+B1(8))+(H1N0-H1D0*H1N4)*A2N(3))

      XX1(8)=X1O(8)+0.5*(B1(8)+B2(8))
      X2(8)=X2O(8)+0.5*(C1(8)+C2(8))
      X3(8)=X3O(8)+0.5*(D1(8)+D2(8))
      X4(8)=X4O(8)+0.5*(E1(8)+E2(8))

C
C     COMPUTE STATE VARIABLES TO OBTAIN:
C     SIMULATOR TRANS ACCEL
C     SIMULATOR ANGULAR VEL (TILT AND PURE)
C
      BSDT(1)=XX1(2)+R2N6*A2N(2)
      BSDT(2)=XX1(5)+P2N6*A2N(1)
      BSDR(1)=XX1(1)+R1N6*BADN(1)
      BSDR(2)=XX1(4)+P1N6*BADN(2)
      BSDR(3)=XX1(7)+Y1N4*BADN(3)
      ASI(1)=XX1(6)+P4N6*A2N(1)
      ASI(2)=XX1(3)+R4N6*A2N(2)
      ASI(3)=XX1(8)+H1N4*A2N(3)

C
C     UPDATE ALL THE DUMMY VARIABLES:
C
      DO I=1,8
         X1O(I)=XX1(I)
         X2O(I)=X2(I)
         X3O(I)=X3(I)
         X4O(I)=X4(I)
      END DO
C
      DO I=1,6
         X5O(I)=X5(I)
         X6O(I)=X6(I)
      END DO
C
      DO IJ=1,3
         A2NO(IJ) = A2N(IJ)
         BADNO(IJ) = BADN(IJ)
      END DO
```

```
RETURN
END
```

8.14. resetc2.f

```
      SUBROUTINE RESETC2
C
C     THIS ROUTINE:
C
C    (1) DOES T=0 INITIALIZATION OF MOTION VARIABLES
C
C    (2) USES A SECOND ORDER SCHEME TO DRIVE TO THE NEUTRAL
C        POSITION. (OVERDAMPED OSCILLATOR; ZETA = 1.5, OMEGAN = 1.0 )
C
      INCLUDE 'comint2.com'

      INCLUDE 'wcom2.com'

      INCLUDE 'matrix1c.com'

C**********************************************************************
C
C      GAINS, ACCELERATION AND VELOCITY LIMITS FOR SECOND ORDER.
C
C      VALUES FOR ACCELERATION AND VELOCITY ARE METERS/SEC**2
C      AND METERS/SEC.  ROTATIONS AND ROTATIONAL VELOCITIES
C      ARE IN RADIANS AND RADIANS/SECOND.
C
C**********************************************************************
C
C      .03492 RADIANS/SEC**2 == 2.0  DEG/SEC**2
C      .294   METERS/SEC**2  ==  .03 G
C
      PARAMETER (A_ACCLIM = .03491)
      PARAMETER (T_ACCLIM = .294  )
C
C      THESE PARAMETERS ARE SET TO THE PERFORMANCE LIMIT OF THE BASE
C
C      .2617 RAD/SEC == 15 DEG/SEC
C      .610  METERS/SEC
C
      PARAMETER (A_VELLIM = .2617 )
      PARAMETER (T_VELLIM = .610  )
C
C      VALUES FOR X FILTER
C
      DATA XDDLIM /  T_ACCLIM /
      DATA XDLIM  /  T_VELLIM /
C
C     VALUES FOR Y FILTER
C
      DATA YDDLIM /  T_ACCLIM /
      DATA YDLIM  /  T_VELLIM /
C
C     VALUES FOR Z FILTER
C
      DATA ZDDLIM /  T_ACCLIM /
      DATA ZDLIM  /  T_VELLIM /
C
```

```
C      VALUES FOR PSI FILTER
C
       DATA PSIDDLIM /  A_ACCLIM  /
       DATA PSIDLIM  /  A_VELLIM  /
C
C      VALUES FOR THETA FILTER
C
       DATA THEDDLIM /  A_ACCLIM  /
       DATA THEDLIM  /  A_VELLIM  /
C
C      VALUES FOR PHI FILTER
C
       DATA PHIDDLIM /  A_ACCLIM  /
       DATA PHIDLIM  /  A_VELLIM  /
       DATA TWOZOMGN/ 3.0 /
C
C      local functions
       CLIMIT(X,XMIN,XMAX) = MIN( MAX( X, XMIN ), XMAX )
C
C      If last operate run ended braked, reset braking algorithm
C      to unbraked state
C
       IF (FLAG2.EQ.1) THEN
          X = SSI(1)
          Y = SSI(2)
          Z = SSI(3)
          PHI = BETAS(1)
          THE = BETAS(2)
          PSI = BETAS(3)
          DO IJ=1,6
             RLID(IJ)=0.
             RLIDO(IJ)=0.
          END DO
           IT2=0
           FLAG2=0
       END IF

C
C      T = 0 INITIALIZATION;
C
       T       = 0.
       INT     = 0
       DXDLX   = 0.
       DXDLXD  = 0.
       DXDDX   = 0.
       DXDDXD  = 0.
       DTHDDX  = 0.
       LAMX    = LAMX0
       DELX    = DELX0
       DYDLY   = 0.
       DYDLYD  = 0.
       DYDDY   = 0.
       DYDDYD  = 0.
       DPHDDY  = 0.
       LAMY    = LAMY0
       DELY    = DELY0
       DZDEZ   = 0.
```

```
      DZDEZD  = 0.
      ETAZ    = ETAZ0
      DPSDE   = 0.
      ETAPS   = ETAPS0

C
C      FADE TO THE NEUTRAL POSITION USING SECOND ORDER FILTER TO ROVIDE
C      VELOCITY AND ACCELERATION CONTROL.
C
C
C      DRIVE X TO NEUTRAL POSITION (X = 0)
C
      XDDF  = -X - TWOZOMGN*XD
      XDD   = CLIMIT(XDDF,-XDDLIM,XDDLIM)
      XDF   = XD + XDD*H
      XD    = CLIMIT(XDF,-XDLIM,XDLIM)
      X     = X + XD*H
C
C      DRIVE Y TO NEUTRAL POSITION (Y = 0)
C
      YDDF  = -Y - TWOZOMGN*YD
      YDD   = CLIMIT(YDDF,-YDDLIM,YDDLIM)
      YDF   = YD + YDD*H
      YD    = CLIMIT(YDF,-YDLIM,YDLIM)
      Y     = Y + YD*H
C
C        DRIVE Z TO NEUTRAL POSITION (Z = 0)
C
      ZDDF  = -Z - TWOZOMGN*ZD
      ZDD   = CLIMIT(ZDDF,-ZDDLIM,ZDDLIM)
      ZDF   = ZD + ZDD*H
      ZD    = CLIMIT(ZDF,-ZDLIM,ZDLIM)
      Z     = Z + ZD*H
C
C        DRIVE PSI TO NEUTRAL POSITION (PSI = 0)
C
      PSIDDF  = -PSI - TWOZOMGN*PSID
      PSIDDFL = CLIMIT(PSIDDF,-PSIDDLIM,PSIDDLIM)
      PSIDF   = PSID + PSIDDFL*H
      PSID    = CLIMIT(PSIDF,-PSIDLIM,PSIDLIM)
      PSI     = PSI + PSID*H
C
C        DRIVE THETA TO NEUTRAL POSITION (THETA = 0)
C
      THEDDF  = -THE - TWOZOMGN*THED
      THEDDFL = CLIMIT(THEDDF,-THEDDLIM,THEDDLIM)
      THEDF   = THED + THEDDFL*H
      THED    = CLIMIT(THEDF,-THEDLIM,THEDLIM)
      THE     = THE + THED*H
C
C        DRIVE PHI TO NEUTRAL POSITION (PHI = 0)
C
      PHIDDF  = -PHI - TWOZOMGN*PHID
      PHIDDFL = CLIMIT(PHIDDF,-PHIDDLIM,PHIDDLIM)
      PHIDF   = PHID + PHIDDFL*H
      PHID    = CLIMIT(PHIDF,-PHIDLIM,PHIDLIM)
      PHI     = PHI + PHID*H
```

```
C
C     DUMMY INTEGRATIONS
C
      XD1     = XD
      DXDLXD1 = DXDLXD
      DXDDXD1 = DXDDXD
      YD1     = YD
      DYDLYD1 = DYDLYD
      DYDDYD1 = DYDDYD
      ZD1     = ZD
      DZDEZD1 = DZDEZD
      RETURN
      END
```

8.15. simq.f

```
      SUBROUTINE SIMQ(A,B,N,KS)
      DIMENSION A(36),B(6)
      TOL=0.
      KS=0
      JJ=-N
      DO 65 J=1,N
          JY=J+1
          JJ=JJ+N+1
          BIGA=0.
          IT=JJ-J
          DO 30 I=J,N
              IJ=IT+I
              IF(ABS(BIGA)-ABS(A(IJ))) 20,30,30
 20           BIGA=A(IJ)
              IMAX=I
 30       CONTINUE
          IF(ABS(BIGA)-TOL) 35,35,40
 35       KS=1
          RETURN
 40       I1=J+N*(J-2)
          IT=IMAX-J
          DO 50 K=J,N
              I1=I1+N
              I2=I1+IT
              SAVE1=A(I1)
              A(I1)=A(I2)
              A(I2)=SAVE1
 50       A(I1)=A(I1)/BIGA
          SAVE1=B(IMAX)
          B(IMAX)=B(J)
          B(J)=SAVE1/BIGA
          IF(J-N) 55,70,55
 55       IQS=N*(J-1)
          DO 65 IX=JY,N
              IXJ=IQS+IX
              IT=J-IX
              DO 60 JX=JY,N
                  IXJX=N*(JX-1)+IX
                  JJX=IXJX+IT
 60           A(IXJX)=A(IXJX)-(A(IXJ)*A(JJX))
 65   B(IX)=B(IX)-(B(J)*A(IXJ))
 70   NY=N-1
      IT=N*N
      DO 80 J=1,NY
          IA=IT-J
          IB=N-J
          IC=N
          DO 80 K=1,J
              B(IB)=B(IB)-A(IA)*B(IC)
              IA=IA-N
 80   IC=IC-1
      RETURN
      END
C
```

8.16. state4.f

```
C
C     NONLINEAR STATE SPACE FILTERS (ALL IN INERTIAL FRAME):
C
      SUBROUTINE STATE4

      INCLUDE 'optint3.com'

      INCLUDE 'nopt4.com'

      REAL ABK1X(6,6),BK2X(6,3),BK3X(6),XXI1(9),XXI2(9)
      REAL ABK1Y(6,6),BK2Y(6,3),BK3Y(6),XYI1(9),XYI2(9)
      REAL XZI1(5),XZI2(5),XRI1(4),XRI2(4)
C
C     ROLL AND PITCH UNITY GAIN FILTERS
C
      BSDR(1)=BADNO(1)
      BSDR(2)=BADNO(2)
C
C     9TH ORDER SURGE/PITCH STATE SPACE FILTER
C
      DO I=1,6
         DO J=1,6
            K1X(1,J)=R2IX(1)*
     x      (BVX(1,1)*PX(1,J)+BVX(2,1)*PX(2,J)+BVX(3,1)*PX(3,J)
     x      +BVX(4,1)*PX(4,J)+BVX(5,1)*PX(5,J)+BVX(6,1)*PX(6,J)
     x      +DQCX(J))

            K1X(2,J)=R2IX(2)*
     x      (BVX(1,2)*PX(1,J)+BVX(2,2)*PX(2,J)+BVX(3,2)*PX(3,J)
     x      +BVX(4,2)*PX(4,J)+BVX(5,2)*PX(5,J)+BVX(6,2)*PX(6,J)
     x      +PX(9,J))

            ABK1X(I,J)=
     x      AVX(I,J)-BVX(I,1)*K1X(1,J)-BVX(I,2)*K1X(2,J)
         END DO
       END DO

       DO I=1,6
          DO J=1,3
             K2X(1,J)=R2IX(1)*
     x       (BVX(1,1)*PX(1,J+6)+BVX(2,1)*PX(2,J+6)+BVX(3,1)*PX(3,J+6)
     x       +BVX(4,1)*PX(4,J+6)+BVX(5,1)*PX(5,J+6)+BVX(6,1)*PX(6,J+6))

             K2X(2,J)=R2IX(2)*
     x       (BVX(1,2)*PX(1,J+6)+BVX(2,2)*PX(2,J+6)+BVX(3,2)*PX(3,J+6)
     x       +BVX(4,2)*PX(4,J+6)+BVX(5,2)*PX(5,J+6)+BVX(6,2)*PX(6,J+6)
     x       +PX(9,J+6))

             BK2X(I,J)=BVX(I,1)*K2X(1,J)+BVX(I,2)*K2X(2,J)
          END DO
       END DO

          K3X(1)=R2IX(1)*
     x    (BVX(1,1)*PX(1,11)+BVX(2,1)*PX(2,11)+BVX(3,1)*PX(3,11)
```

```
     x   +BVX(4,1)*PX(4,11)+BVX(5,1)*PX(5,11)+BVX(6,1)*PX(6,11))

      K3X(2)=R2IX(2)*
     x   (BVX(1,2)*PX(1,11)+BVX(2,2)*PX(2,11)+BVX(3,2)*PX(3,11)
     x   +BVX(4,2)*PX(4,11)+BVX(5,2)*PX(5,11)+BVX(6,2)*PX(6,11)
     x   +PX(9,11))

      DO I=1,6
         BK3X(I)=BVX(I,1)*K3X(1)+BVX(I,2)*(1.0+K3X(2))
      END DO

      DO I=1,6
         XXI1(I)=DT*
     x   (ABK1X(I,1)*XXO(1)+ABK1X(I,2)*XXO(2)+ABK1X(I,3)*XXO(3)
     x    +ABK1X(I,4)*XXO(4)+ABK1X(I,5)*XXO(5)+ABK1X(I,6)*XXO(6)
     x    -BK2X(I,1)*XXO(7)-BK2X(I,2)*XXO(8)-BK2X(I,3)*XXO(9)
     x    -BK3X(I)*A2NO(1))
      END DO
      XXI1(7)=DT*XXO(8)
      XXI1(8)=DT*XXO(9)
      XXI1(9)=DT*
     x   (-K1X(2,1)*XXO(1)-K1X(2,2)*XXO(2)-K1X(2,3)*XXO(3)
     x    -K1X(2,4)*XXO(4)-K1X(2,5)*XXO(5)-K1X(2,6)*XXO(6)
     x    -K2X(2,1)*XXO(7)-K2X(2,2)*XXO(8)-K2X(2,3)*XXO(9)
     x    -K3X(2)*A2NO(1))

      DO I=1,6
         XXI2(I)=DT*
     x   (ABK1X(I,1)*(XXO(1)+XXI1(1))+ABK1X(I,2)*(XXO(2)+XXI1(2))
     x   +ABK1X(I,3)*(XXO(3)+XXI1(3))+ABK1X(I,4)*(XXO(4)+XXI1(4))
     x   +ABK1X(I,5)*(XXO(5)+XXI1(5))+ABK1X(I,6)*(XXO(6)+XXI1(6))
     x   -BK2X(I,1)*(XXO(7)+XXI1(7))-BK2X(I,2)*(XXO(8)+XXI1(8))
     x   -BK2X(I,3)*(XXO(9)+XXI1(9))-BK3X(I)*A2N(1))
      END DO
      XXI2(7)=DT*(XXO(8)+XXI1(8))
      XXI2(8)=DT*(XXO(9)+XXI1(9))
      XXI2(9)=DT*(-K1X(2,1)*(XXO(1)+XXI1(1))
     x    -K1X(2,2)*(XXO(2)+XXI1(2))-K1X(2,3)*(XXO(3)+XXI1(3))
     x    -K1X(2,4)*(XXO(4)+XXI1(4))-K1X(2,5)*(XXO(5)+XXI1(5))
     x    -K1X(2,6)*(XXO(6)+XXI1(6))-K2X(2,1)*(XXO(7)+XXI1(7))
     x    -K2X(2,2)*(XXO(8)+XXI1(8))-K2X(2,3)*(XXO(9)+XXI1(9))
     x    -K3X(2)*A2N(1))

      DO I=1,9
         XX(I)=XXO(I)+0.5*(XXI1(I)+XXI2(I))
      END DO
C
C     9TH ORDER SWAY/ROLL STATE SPACE FILTER
C
      DO I=1,6
         DO J=1,6
            K1Y(1,J)=R2IY(1)*
     x      (BVY(1,1)*PY(1,J)+BVY(2,1)*PY(2,J)+BVY(3,1)*PY(3,J)
     x      +BVY(4,1)*PY(4,J)+BVY(5,1)*PY(5,J)+BVY(6,1)*PY(6,J)
     x      +DQCY(J))

            K1Y(2,J)=R2IY(2)*
```

```
x       (BVY(1,2)*PY(1,J)+BVY(2,2)*PY(2,J)+BVY(3,2)*PY(3,J)
x       +BVY(4,2)*PY(4,J)+BVY(5,2)*PY(5,J)+BVY(6,2)*PY(6,J)
x       +PY(9,J))

        ABK1Y(I,J)=
x       AVY(I,J)-BVY(I,1)*K1Y(1,J)-BVY(I,2)*K1Y(2,J)
     END DO
 END DO

 DO I=1,6
    DO J=1,3
       K2Y(1,J)=R2IY(1)*
x       (BVY(1,1)*PY(1,J+6)+BVY(2,1)*PY(2,J+6)+BVY(3,1)*PY(3,J+6)
x       +BVY(4,1)*PY(4,J+6)+BVY(5,1)*PY(5,J+6)+BVY(6,1)*PY(6,J+6))

       K2Y(2,J)=R2IY(2)*
x       (BVY(1,2)*PY(1,J+6)+BVY(2,2)*PY(2,J+6)+BVY(3,2)*PY(3,J+6)
x       +BVY(4,2)*PY(4,J+6)+BVY(5,2)*PY(5,J+6)+BVY(6,2)*PY(6,J+6)
x       +PY(9,J+6))

    BK2Y(I,J)=BVY(I,1)*K2Y(1,J)+BVY(I,2)*K2Y(2,J)
    END DO
 END DO

 K3Y(1)=R2IY(1)*
x       (BVY(1,1)*PY(1,11)+BVY(2,1)*PY(2,11)+BVY(3,1)*PY(3,11)
x       +BVY(4,1)*PY(4,11)+BVY(5,1)*PY(5,11)+BVY(6,1)*PY(6,11))

 K3Y(2)=R2IY(2)*
x       (BVY(1,2)*PY(1,11)+BVY(2,2)*PY(2,11)+BVY(3,2)*PY(3,11)
x       +BVY(4,2)*PY(4,11)+BVY(5,2)*PY(5,11)+BVY(6,2)*PY(6,11)
x       +PY(9,11))

 DO I=1,6
    BK3Y(I)=BVY(I,1)*K3Y(1)+BVY(I,2)*(1.0+K3Y(2))
 END DO

 DO I=1,6
    XYI1(I)=DT*
x   (ABK1Y(I,1)*XYO(1)+ABK1Y(I,2)*XYO(2)+ABK1Y(I,3)*XYO(3)
x   +ABK1Y(I,4)*XYO(4)+ABK1Y(I,5)*XYO(5)+ABK1Y(I,6)*XYO(6)
x   -BK2Y(I,1)*XYO(7)-BK2Y(I,2)*XYO(8)-BK2Y(I,3)*XYO(9)
x   -BK3Y(I)*A2NO(2))
 END DO
 XYI1(7)=DT*XYO(8)
 XYI1(8)=DT*XYO(9)
 XYI1(9)=DT*
x     (-K1Y(2,1)*XYO(1)-K1Y(2,2)*XYO(2)-K1Y(2,3)*XYO(3)
x      -K1Y(2,4)*XYO(4)-K1Y(2,5)*XYO(5)-K1Y(2,6)*XYO(6)
x      -K2Y(2,1)*XYO(7)-K2Y(2,2)*XYO(8)-K2Y(2,3)*XYO(9)
x      -K3Y(2)*A2NO(2))

 DO I=1,6
    XYI2(I)=DT*
x       (ABK1Y(I,1)*(XYO(1)+XYI1(1))+ABK1Y(I,2)*(XYO(2)+XYI1(2))
x       +ABK1Y(I,3)*(XYO(3)+XYI1(3))+ABK1Y(I,4)*(XYO(4)+XYI1(4))
x       +ABK1Y(I,5)*(XYO(5)+XYI1(5))+ABK1Y(I,6)*(XYO(6)+XYI1(6))
```

```
     x      -BK2Y(I,1)*(XYO(7)+XYI1(7))-BK2Y(I,2)*(XYO(8)+XYI1(8))
     x      -BK2Y(I,3)*(XYO(9)+XYI1(9))-BK3Y(I)*A2N(2))
   END DO
      XYI2(7)=DT*(XYO(8)+XYI1(8))
      XYI2(8)=DT*(XYO(9)+XYI1(9))
      XYI2(9)=DT*(-K1Y(2,1)*(XYO(1)+XYI1(1))
     x    -K1Y(2,2)*(XYO(2)+XYI1(2))-K1Y(2,3)*(XYO(3)+XYI1(3))
     x    -K1Y(2,4)*(XYO(4)+XYI1(4))-K1Y(2,5)*(XYO(5)+XYI1(5))
     x    -K1Y(2,6)*(XYO(6)+XYI1(6))-K2Y(2,1)*(XYO(7)+XYI1(7))
     x    -K2Y(2,2)*(XYO(8)+XYI1(8))-K2Y(2,3)*(XYO(9)+XYI1(9))
     x    -K3Y(2)*A2N(2))

      DO I=1,9
         XY(I)=XYO(I)+0.5*(XYI1(I)+XYI2(I))
      END DO
C
C     5TH ORDER HEAVE STATE SPACE FILTER
C
      K1Z(1)=BVZ(1)*PZ(1,1)+BVZ(2)*PZ(2,1)+PZ(5,1)
      K1Z(2)=BVZ(1)*PZ(1,2)+BVZ(2)*PZ(2,2)+PZ(5,2)
      K2Z(1)=BVZ(1)*PZ(1,3)+BVZ(2)*PZ(2,3)+PZ(5,3)
      K2Z(2)=BVZ(1)*PZ(1,4)+BVZ(2)*PZ(2,4)+PZ(5,4)
      K2Z(3)=BVZ(1)*PZ(1,5)+BVZ(2)*PZ(2,5)+PZ(5,5)
      K3Z=BVZ(1)*PZ(1,6)+BVZ(2)*PZ(2,6)+PZ(5,6)

      DO I=1,2
      XZI1(I)=DT*
     x   ((AVZ(I,1)-BVZ(I)*K1Z(1))*XZO(1)
     x   +(AVZ(I,2)-BVZ(I)*K1Z(2))*XZO(2)
     x   -BVZ(I)*(K2Z(1)*XZO(3)+K2Z(2)*XZO(4)+K2Z(3)*XZO(5))
     x   -BVZ(I)*(1+K3Z)*A2NO(3))
   END DO
      XZI1(3)=DT*XZO(4)
      XZI1(4)=DT*XZO(5)
      XZI1(5)=DT*(-K1Z(1)*XZO(1)-K1Z(2)*XZO(2)-K2Z(1)*XZO(3)
     x        -K2Z(2)*XZO(4)-K2Z(3)*XZO(5)-K3Z*A2NO(3))

      DO I=1,2
      XZI2(I)=DT*
     x   ((AVZ(I,1)-BVZ(I)*K1Z(1))*(XZO(1)+XZI1(1))
     x   +(AVZ(I,2)-BVZ(I)*K1Z(2))*(XZO(2)+XZI1(2))
     x   -BVZ(I)*(K2Z(1)*(XZO(3)+XZI1(3))+K2Z(2)*(XZO(4)+XZI1(4))
     x           +K2Z(3)*(XZO(5)+XZI1(5)))
     x   -BVZ(I)*(1+K3Z)*A2N(3))
   END DO
      XZI2(3)=DT*(XZO(4)+XZI1(4))
      XZI2(4)=DT*(XZO(5)+XZI1(5))
      XZI2(5)=DT*(-K1Z(1)*(XZO(1)+XZI1(1))
     x      -K1Z(2)*(XZO(2)+XZI1(2))
     x      -K2Z(1)*(XZO(3)+XZI1(3))
     x      -K2Z(2)*(XZO(4)+XZI1(4))
     x      -K2Z(3)*(XZO(5)+XZI1(5))-K3Z*A2N(3))

      DO I=1,5
         XZ(I)=XZO(I)+0.5*(XZI1(I)+XZI2(I))
      END DO
C
```

```
C     4TH ORDER YAW STATE SPACE FILTER
C
      DO I=1,3
         K1R(I)=R2IR*
     x   (BVR(1)*PR(1,I)+BVR(2)*PR(2,I)+BVR(3)*PR(3,I)+PR(4,I)+DQCR(I))
      END DO
      K2R=R2IR*
     x   (BVR(1)*PR(1,4)+BVR(2)*PR(2,4)+BVR(3)*PR(3,4)+PR(4,4))
      K3R=R2IR*
     x   (BVR(1)*PR(1,5)+BVR(2)*PR(2,5)+BVR(3)*PR(3,5)+PR(4,5)-DQDR)

      DO I=1,3
         XRI1(I)=DT*
     x   ((AVR(I,1)-BVR(I)*K1R(1))*XRO(1)
     x    +(AVR(I,2)-BVR(I)*K1R(2))*XRO(2)
     x    +(AVR(I,3)-BVR(I)*K1R(3))*XRO(3)
     x    -BVR(I)*K2R*XRO(4)
     x    -BVR(I)*(1+K3R)*BADNO(3))
      END DO
      XRI1(4)=DT*(-K1R(1)*XRO(1)-K1R(2)*XRO(2)
     x    -K1R(3)*XRO(3)-K2R*XRO(4)-K3R*BADNO(3))

      DO I=1,3
         XRI2(I)=DT*
     x   ((AVR(I,1)-BVR(I)*K1R(1))*(XRO(1)+XRI1(1))
     x    +(AVR(I,2)-BVR(I)*K1R(2))*(XRO(2)+XRI1(2))
     x    +(AVR(I,3)-BVR(I)*K1R(3))*(XRO(3)+XRI1(3))
     x    -BVR(I)*K2R*(XRO(4)+XRI1(4))
     x    -BVR(I)*(1+K3R)*BADN(3))
   END DO
      XRI2(4)=DT*(-K1R(1)*(XRO(1)+XRI1(1))
     x            -K1R(2)*(XRO(2)+XRI1(2))
     x            -K1R(3)*(XRO(3)+XRI1(3))
     x            -K2R*(XRO(4)+XRI1(4))-K3R*BADN(3))

      DO I=1,4
         XR(I)=XRO(I)+0.5*(XRI1(I)+XRI2(I))
      END DO
C
C************* UPDATE ALL THE DUMMY VARIABLES: ********************
C
      DO I=1,9
         XXO(I)=XX(I)
         XYO(I)=XY(I)
   END DO
C
      DO I=1,5
         XZO(I)=XZ(I)
      END DO

      DO I=1,4
         XRO(I)=XR(I)
   END DO

      A2NO(1) = A2N(1)
      A2NO(2) = A2N(2)
      A2NO(3) = A2N(3)
```

```
BADNO(1) = BADN(1)
BADNO(2) = BADN(2)
BADNO(3) = BADN(3)

RETURN
END
```

8.17. vmult.f

```
C
C*********************************************************************
C       Subroutine VMULT : MATRIX MULTIPLICATION.
C*********************************************************************
C
      SUBROUTINE VMULT(A,B,C,K,L,M)
      DIMENSION A(K,L),B(K,L),C(K,M)
      DO 20 KK = 1,K
          DO 20 MM = 1,M
              C(KK,MM) = 0.0
              DO 20 LL = 1,L
                  C(KK,MM) = C(KK,MM) + A(KK,LL)*B(LL,MM)
  20  CONTINUE
      RETURN
      END
```

8.18. washopt3.f

```
C***** SUBROUTINE WASHOPT3.FOR ******
C
C     OPTIMAL WASHOUT ALGORITHM: ANG. VELOCITY DEVELOPMENT.
C     GIVEN  A/C ACCELS ACA AND EULER RATES BETAAD,
C     COMPUTE SIMULATOR INERTIAL DISPLACEMENT AND EULER ANGLES.
C
      SUBROUTINE WASHOPT3(MODE)

      INCLUDE 'comint2.com'

      INCLUDE 'wcom2.com'

      INCLUDE 'optint3.com'

      INCLUDE 'wopt3.com'

      INCLUDE 'matrix1c.com'
C
C     Compute Fairing Parameters
C
      SQWASH=SQWASHP*EA+SQWASHI*(1.0-EA)
      DELSQ=MAX(MIN(SQWASH-SQWASHP,1.0),-1.0)
      SQWASHP=SQWASH+DELSQ

      A=1.0-SQWASHP
      AA=1.0-SQWASHI
C
C     Fairing of Heave Nonlinear Gain and Limit
C
      GZ3(1)=AA*GZ30(1)+SQWASHI*GZ3S(1)
      GZ3(2)=AA*GZ30(2)+SQWASHI*GZ3S(2)
      GZ3(3)=AA*GZ30(3)+SQWASHI*GZ3S(3)
      AMX3=AA*AMX30+SQWASHI*AMX3S
C
C     Fairing of Runway Roughness Amplitude
C
      XKA=A*XKA0+SQWASHP*XKAS

      IF(MODE.EQ.1) THEN
         H = DT
C
C     Set "old" variables for future use in OFIL3
C
          DO I=1,3
            A2NO(I)=0.
            BADNO(I)=0.
         END DO

         DO I=1,8
            X1O(I)=0.
            X2O(I)=0.
            X3O(I)=0.
            X4O(I)=0.
```

```
         END DO

         DO I=1,6
            X5O(I)=0.
            X6O(I)=0.
         END DO

         CALL RESETC2
C
C     Set "old" variables for future use in HOLD and OPERATE modes
C
         ASIO(1)=XDD
         ASIO(2)=YDD
         ASIO(3)=ZDD
         VSIO(1)=XD
         VSIO(2)=YD
         VSIO(3)=ZD
         SSIO(1)=X
         SSIO(2)=Y
         SSIO(3)=Z
         BETASRO(1)=PHI
         BETASRO(2)=THE
         BETASRO(3)=PSI
         BSDRO(1)=PHID
         BSDRO(2)=THED
         BSDRO(3)=PSID

         DO I=1,2
            XHO(I)=0.
            ACAO(I)=0.
            BSDTO(I)=0.
            BETASTO(I)=0.
            BETASTLO(I)=0.
            XTO=0.
            XT2O=0.
            WGUSTO=0.
            ACZT=0.
         END DO

         GO TO 1
      END IF

      IF(MODE.EQ.2) CALL WTRIM3
C
C     Compute Augmented Acceleration from W-Gust
C
      IF(MODE.EQ.3) THEN
         WGAV=0.5*(WGUST+WGUSTO)
         XT=XTO+DT*(-G1D1*XTO+XT2O+(G1N1-G1D1*G1N2)*WGAV)
         XTAV=0.5*(XT+XTO)
         XT2=XT2O+DT*(-G1D0*XTAV+(G1N0-G1D0*G1N2)*WGAV)
         ACZT=XT+G1N2*WGUST
         XTO=XT
         XT2O=XT2
         WGUSTO=WGUST
C
C     (first-order turbulence model no longer used)
```

```
C
C         XT=XTO+DT*(-G1D0*XTO+(G1N0-G1D0*G1N1)*WGUSTO)
C         ACZT=XT+G1N1*WGUSTO
C         XTO=XT
C         WGUSTO=WGUST
        END IF

        CALL LIBA
        CALL GAINOPT3
           A2N(3)=A2N(3)+GT2*ACZT

        CALL OFIL3

        IF (MODE.EQ.3) CALL INTEG3

      1 RETURN
        END
C
```

8.19. winit2.f

```
      SUBROUTINE WINIT2
C
C     THIS ROUTINE LOADS THE INITIAL VALUES INTO THE WASHOUT
C     PARAMETER ARRAYS.

      INCLUDE 'comint2.com'

      INCLUDE 'wcom2.com'

      INCLUDE 'optint3.com'

      INCLUDE 'wopt3.com'

      INCLUDE 'matrix1c.com'

      REAL      DERIV(64)
C
      EQUIVALENCE (DERIV  (1), THESDD  )

C
C     INITIALIZATION OF ADAPTIVE ALGORITHM DERIVATIVES
C
      DO 10 I = 1,64
         DERIV(I) = 0.
10    CONTINUE
C
C***************************** WU *********************************
C NOTE:  RX,RY,RZ HAVE DIFFERENT MEANING FROM THOSE IN UTIAS REPORT.
C        R IS THE VECTOR FROM C.G. TO CENTROID OF A/C .
C      RX = SSIIN(1)
C      RY = SSIIN(2)
C      RZ = SSIIN(3)
      DATA RX/0./, RY/0./, RZ/0./
*****************************************************************
C
C     FOR 737 MOTION BASE  RX,RY,RZ = 12.192,0.2286,1.7399
C
C     INTEGRATION CONSTANTS
C
C*******changed by wu************
      DATA T       /0./
      DATA INT     /0/
      DATA NEQ     /30/

C*************** CHANGED BY WU *********************
C**   GAINS ARE CHANGED FOR CONVENIENCE OF COMPARISON
C**   BETWEEN NASA & ADAPTIVE ALGORITHMS.
C    DATA X1      /3.6576/
C    DATA SX0     /0.500/
C    DATA Y1      /2.4384/
C    DATA SY0     /0.50/
C    DATA Z1      /1.2192/
C    DATA SZ0     /0.50/
      DATA X1      /3.6576/
```

```
      DATA SX0     /1.0/
      DATA Y1      /2.4384/
      DATA SY0     /1.0/
      DATA Z1      /1.2192/
      DATA SZ0     /1.0/
C*****************************************************
      DATA P1      /.18/
      DATA SP0     /1.0/
      DATA Q1      /.5/
      DATA SQ0     /1.0/
      DATA R1      /.15/
      DATA SR0     /1.0/
C
C     LONGITUDINAL PARAMETERS
C
      DATA WX      /0.00929/
      DATA BX      /0.01/
      DATA CX      /.2/
      DATA DX      /.707/
      DATA EX      /.25/
      DATA GAMX    /.1640419948/
      DATA KLX     /.3229173125/
      DATA KDX     /.0107639104/
      DATA KILX    /.02/
      DATA KIDX    /.5/
      DATA LAMXL   /-.1/
      DATA LAMXU   /1./
      DATA DELXL   /0./
      DATA DELXU   /1./
      DATA LAMXDL  /-.06/
      DATA DELXDL  /-1000./
      DATA LAMX0   /1.0/
      DATA DELX0   /0.5/
C
C    DATA WX      /10.0/
C    DATA BX      /80.0/
C    DATA CX      /50.0/
C    DATA DX      /4.5/
C    DATA EX      /2.25/
C    DATA GAMX    /.12/
C    DATA KLX     /3.5/
C    DATA KDX     /0.10/
C    DATA KILX    /.02/
C    DATA KIDX    /.5/
C    DATA LAMXL   /-.1/
C    DATA LAMXU   /1./
C    DATA DELXL   /0./
C    DATA DELXU   /1./
C    DATA LAMXDL  /-.06/
C    DATA DELXDL  /-1000./
C    DATA LAMX0   /1.0/
C    DATA DELX0   /1.0/

C
C     LATERAL PARAMETERS
C
      DATA WY      /0.00929/
```

```
      DATA BY      /0.1/
      DATA CY      /2.0/
      DATA DY      /1.2727/
      DATA EY      /0.81/
C*********** CHANGE BY WU ***********
C    DATA GAMY    /0.0328084/
      DATA GAMY    /.1640419948/
C************************************
      DATA KLY     /0.516668/
      DATA KDY     /0.269098/
      DATA KILY    /.05/
      DATA KIDY    /1.5/
      DATA LAMYL   /-.1/
      DATA LAMYU   /.8/
      DATA DELYL   /0./
      DATA DELYU   /.3/
      DATA LAMYDL  /-.06/
      DATA DELYDL  /-.2/
      DATA LAMY0   /0.8/
      DATA DELY0   /0.3/
C
C    DATA WY      /1500./
C    DATA BY      /400.0/
C    DATA CY      /100.0/
C    DATA DY      /12.80/
C    DATA EY      /10.24/
C    DATA GAMY    /0.05/
C    DATA KLY     /0.085/
C    DATA KDY     /0.15/
C    DATA KILY    /.05/
C    DATA KIDY    /1.5/
C    DATA LAMYL   /-.1/
C    DATA LAMYU   /.2/
C    DATA DELYL   /0./
C    DATA DELYU   /.3/
C    DATA LAMYDL  /-.06/
C    DATA DELYDL  /-.04/
C    DATA LAMY0   /0.2/
C    DATA DELY0   /0.3/
C
C   VERTICAL PARAMETERS
C
      DATA BZ      /.1/
      DATA CZ      /0.1/
      DATA DZ      /1.2727/
      DATA EZ      /.81/
      DATA KEZ     /.516669/
      DATA KIEZ    /.05/
      DATA ETAZL   /0./
      DATA ETAZU   /.25/
      DATA ETAZDL  /-.06/
      DATA ETAZ0   /.25/
C
C     YAW PARAMETERS
C
      DATA BPS     /1.0/
      DATA EPS     /0.3/
```

```
      DATA KEPS    /100.0/
      DATA KIEPS   /0.1/
      DATA ETAPSU  /1.0/
      DATA ETAPSL  /0./
      DATA ETAPSDL /-.4/
      DATA ETAPS0  /1./
C
C     TOUCHDOWN PARAMETERS
C
         DATA KLXS    /3.229173125/
      DATA KILXS   /.1/
      DATA LAMXDLS /-.1/
      DATA LAMX0S  /1./

      DATA WYS     /.009290304/
      DATA BYS     /.01/
      DATA CYS     /.2/
      DATA DYS     /.707/
      DATA EYS     /.25/
      DATA GAMYS   /.1640419948/
      DATA KLYS    /3.229173125/
      DATA KILYS   /.1/
      DATA LAMYLS  /-.1/
      DATA LAMYUS  /1./
      DATA LAMYDLS /-.1/
      DATA LAMY0S  /1.0/
      DATA ETAZUS  /.5/
      DATA ETAZ0S  /.5/
C
C     FLIGHT PARAMETERS
C
         KLX0    = KLX
      KILX0   = KILX
      LAMXDL0 = LAMXDL
      LAMX00  = LAMX0

      WY0     = WY
      BY0     = BY
      CY0     = CY
      DY0     = DY
      EY0     = EY
      GAMY0   = GAMY
      KLY0    = KLY
      KILY0   = KILY
      LAMYL0  = LAMYL
      LAMYU0  = LAMYU
      LAMYDL0 = LAMYDL
      LAMY00  = LAMY0
      ETAZU0  = ETAZU
      ETAZ00  = ETAZ0
C
C     Runway Roughness Parameters
C
         DATA XKA0/0./,XKAS/0.00027/,WB/30.0/,XKG/0.1/
C
C     Fairing Parameters
C
```

```
         DATA SQWASHI/0.0/,EA/0.939413/,SQWASHP/0.0/
C
C     COMPENSATION TERMS
C
      DATA CCXA    /.0069/
      DATA CCYA    /.0069/
      DATA CCZA    /.0069/
      DATA CCXV    /.15/
      DATA CCYV    /.15/
      DATA CCZV    /.133/
      DATA CCPS    /.12/
      DATA CCTH    /.12/
      DATA CCPH    /.12/
C
C     GRAVITATIONAL CONSTANT
C
      DATA G       /9.806178/
C
C     Initialization of Optimal Algorithm Inputs
C
      DATA SSI1/3*0.0/,VSI/3*0.0/,BETAS1/3*0.0/
C
C     Initialization of Dummy Variables
C
      DATA A2NO/3*0./,BADNO/3*0./
      DATA X1O/8*0./,X2O/8*0./,X3O/8*0./,X4O/8*0./
      DATA X5O/6*0./,X6O/6*0./
      DATA SSIO/3*0.0/,VSIO/3*0.0/,ASIO/3*0./
      DATA BSDTO/2*0.0/,BSDRO/3*0.0/
      DATA BETASRO/3*0.0/,BETASTO/2*0.0/,BETASTLO/2*0.0/
      DATA XHO/2*0.0/,ACAO/2*0.0/
      DATA T1N1,T1D0,T2N1,T2D0/-0.019,0.333333,0.019,0.333333/
C
C     Tilt Coordination Limit
C
      DATA BDLIM/0.0873/
C
C     Nonlinear Scaling Coefficients
C
      DATA GX/0.6,-0.046,0.0014/
      DATA GY/0.6,-0.064,0.0026/
      DATA GZ0/0.6,-0.082,0.0038/
      DATA GZS/0.6,-0.01,0.0/
      DATA GP/1.0,-1.35,0.6/
      DATA GQ/1.0,-0.5,0.0/
      DATA GR/1.1,-1.46,0.64/
C
C     Translational and Rotational Limits
C
      DATA AMX0/10./,BMX/1./,AMXS/50./
C
C  Parameters for optimal roll/sway channel filters.
C
         DATA R1N6,R1N5,R1N4,
     +       R1N3,R1N2,R1N1,R1N0,
     +       R2N6,R2N5,R2N4,
     +       R2N3,R2N2,R2N1,R2N0,
```

```
     +      R4N6,R4N5,R4N4,
     +      R4N3,R4N2,R4N1,R4N0,
     +      R1D5,R1D4,
     +      R1D3,R1D2,R1D1,R1D0/

     +    1.0006,   15.9733,   42.0572,
     +   56.8102,   43.5211,   16.2283,    1.2344,
     +   -0.0008,   -1.1663,   -4.7183,
     +   -3.5599,   -1.2765,   -0.0785,    0.0000,
     +    3.4188,   24.4014,   29.7868,
     +   -0.0048,    0.0000,    0.0000,    0.0000,
     +   16.3952,   43.7546,
     +   58.6718,   44.7900,   16.7041,    1.2927/
C
C  Parameters for optimal pitch/surge channel filters.
C
      DATA P1N6,P1N5,P1N4,
     +     P1N3,P1N2,P1N1,P1N0,
     +     P2N6,P2N5,P2N4,
     +     P2N3,P2N2,P2N1,P2N0,
     +     P4N6,P4N5,P4N4,
     +     P4N3,P4N2,P4N1,P4N0,
     +     P1D5,P1D4,
     +     P1D3,P1D2,P1D1,P1D0/

     +    1.0006,   15.9733,   42.0572,
     +   56.8102,   43.5210,   16.2282,    1.2344,
     +    0.0008,    1.1663,    4.7183,
     +    3.5599,    1.2765,    0.0785,   -0.0000,
     +    3.4188,   24.4014,   29.7868,
     +   -0.0048,    0.0000,    0.0000,    0.0000,
     +   16.3952,   43.7546,
     +   58.6718,   44.7900,   16.7041,    1.2927/
C
C  Parameters for optimal yaw channel filters.
C
      DATA Y1N4,Y1N3,Y1N2,Y1N1,Y1N0,
     +     Y1D3,Y1D2,Y1D1,Y1D0/

     +  0.9734, 11.7588,  1.6345,  0.0000,  0.0000,
     +          17.1168, 10.1205,  1.4510,  0.0232/
C
C  Parameters for optimal heave channel filters.
C
      DATA H1N4,H1N3,H1N2,H1N1,H1N0,
     +     H1D3,H1D2,H1D1,H1D0/

     +  0.2779,  0.0540,  0.0000,  0.0000,  0.0000,
     +           1.5321,  1.1124,  0.2736,  0.0209/

C
C     Nonlinear Scaling Coefficients
C
      DATA GX3/0.6,-0.055,0.002/
      DATA GY3/0.5,-0.055,0.002/
      DATA GZ30/0.6,-0.082,0.0038/
      DATA GZ3S/1.3,-0.0375,0.0003/
```

```
      DATA GP3/0.3,-0.3,0.1/
      DATA GQ3/0.4,-0.54,0.26/
      DATA GR3/1.1,-1.46,0.64/
C
C     Translational and Rotational Limits
C
      DATA AMX30/10./,BMX3/1./,AMX3S/20./
C
C     Augmented Turbulence Parameters
C
      DATA G1D0,G1D1,G1N0,G1N1,G1N2
     +   /25.0,12.5,2.5,12.0,14.4/
      DATA XTO,XT2O/2*0.0/,WGUSTO/0.0/,ACZT/0.0/
      DATA GT2/0.8/

      RETURN
      END
```

8.20. winit4.f

```
      SUBROUTINE WINIT4
C
C     THIS ROUTINE LOADS THE INITIAL VALUES INTO THE WASHOUT
c     PARAMETER ARRAYS.

      INCLUDE 'nopt4.com'

      REAL BRBXVEC(66),R1PXVEC(66),ZXVEC(66)
      REAL BRBYVEC(66),R1PYVEC(66),ZYVEC(66)
      REAL BRBRVEC(15),R1PRVEC(15),ZRVEC(15)
      REAL BRBZVEC(21),R1PZVEC(21),ZZVEC(21)

C
C     Initialization of Nonlinear Algorithm Inputs
C
      DATA XXO/9*0./XYO/9*0./,XRO/4*0./,XZO/5*0./
C
C  Parameters for nonlinear roll/sway channel filters.
C
      DATA ALPY /0.0/, ALPYMAX /1.0/, Q2Y /0.0,0.8/, MUY /4.0E-6 /

      DATA APY /
     + -0.48601433,  1.50095561,  0.43295640, -2.00166583,  2.30852675,
     +  0.93783312,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     + -0.22785440,  1.23627334,  0.43548518, -2.00719798,  2.32816934,
     +  0.92719057,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000, -0.50000000,  0.50000000,
     + -0.00000000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.17852148, -0.42864297,  0.00091112, -0.01882892,  0.19024153,
     +  0.74839400,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.84518815,  0.23802370,  0.00091112,  0.18117108, -0.00975847,
     +  0.08172733,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.47047268, -2.17119788, -0.42654860,  1.84882203, -2.39757925,
     + -1.63405743,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000,  0.00000000,  0.00000000,
     +  0.00000000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000,  0.00000000,  0.00000000,
     +  0.00000000,  1.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000,  0.00000000,  0.00000000,
     +  0.00000000,  0.0000,  1.0000,  0.0000,  0.0000,  0.0000,
     + -0.00007038,  0.00070080,  0.00016391, -0.00074143,  0.00089035,
     +  0.00036336,  0.0000,  0.0000,  0.0000, -1.000000,  0.0000,
     + -1.33333333, -1.33333333,  0.00000000, -0.20000000,  0.20000000,
     +  1.33333333,  0.0000,  0.0000,  0.0000,  0.0000, -3.14159265/
      DATA BRBYVEC /
     +  1.77789635,  1.77659704, -0.00027616,  0.26791584, -0.26816675,
     + -1.77838998,  0.0000,  0.0000,  1.33333333,  0.0000,  0.0000,
     +  1.78953524,  0.00274996,  0.25422767, -0.25172914,
     + -1.77168162,  0.0000,  0.0000,  1.33333333,  0.0000,  0.0000,
     +  0.00064319, -0.00290936,  0.00349374,
     +  0.00142583,  0.0000,  0.0000, -0.00000000,  0.0000,  0.0000,
     +  0.05316003, -0.05580338,
     + -0.27311620,  0.0000,  0.0000,  0.20000000,  0.0000,  0.0000,
     +  0.05897769,
```

```
+  0.27441166,  0.0000,  0.0000, -0.20000000,  0.0000,  0.0000,
+  1.78093859,  0.0000,  0.0000, -1.33333333,  0.0000,  0.0000,
+  0.0000,  0.0000,  0.00000000,  0.0000,  0.0000,
+  0.0000,  0.00000000,  0.0000,  0.0000,
+  1.00000000,  0.0000,  0.0000,
+  0.0000,  0.0000,
+  0.0000/
 DATA R1PYVEC /
+  8.63477261,  7.08082902,  0.00000000, -0.55988506, -0.55988506,
+-12.57237449,  0.0000,  0.0000,  0.0000, -0.11033547,  0.0000,
+  5.80695939,  0.00000000, -0.45897456, -0.45897456,
+-10.30873990,  0.0000,  0.0000,  0.0000, -0.11097991,  0.0000,
+  0.00000000, -0.00000000, -0.00000000,
+ -0.00000000,  0.0000,  0.0000,  0.0000,  0.00000000,  0.0000,
+  0.03635800,  0.03635800,
+  0.81558635,  0.0000,  0.0000,  0.0000, -0.00023219,  0.0000,
+  0.03635800,
+  0.81558635,  0.0000,  0.0000,  0.0000, -0.00023219,  0.0000,
+ 18.30829171,  0.0000,  0.0000,  0.0000,  0.10870250,  0.0000,
+  8.00000000,  0.0000,  0.0000,  0.0000,  0.0000,
+  4.00000000,  0.0000,  0.0000,  0.0000,
+  1.00000000,  0.0000,  0.0000,
+  1.0000,  0.00000000,
+  0.0000/
 DATA R2IY /4.17714546E-005,  1.00000000/
 DATA AVY /
+ -0.3001182, -0.3501544, -0.0, -0.0432552, -0.0432552,-0.0219541,
+ -0.0408725, -0.6256486, -0.0, -0.0373488, -0.0373488,-0.0382025,
+  0.0000000,  0.0000000,  0.0, -0.5000000,  0.5000000,-0.0000000,
+  0.1789127, -0.4325385,  0.0, -0.0147076,  0.1852924, 0.7463742,
+  0.8455793,  0.2341282, -0.0,  0.1852924, -0.0147076, 0.0797075,
+  0.2873278, -0.3474845,  0.0, -0.0806039, -0.0806039,-0.6884752/
 DATA BVY /
+  1.684826,-16.777107, -3.924000, 17.749601,-21.314831,-8.698809,
+  1.333333,  1.333333, -0.000000,  0.200000, -0.200000,-1.333333/
 DATA DQCY /
+   2641.408443,   2656.836237,     -0.000000,
+      5.558626,      5.558626,  -2602.315320/
 DATA ZYVEC /
+       1.0,      -1.0,      -1.0,      -1.0,      -1.0,
+      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
+       1.0,      -1.0,      -1.0,      -1.0,
+      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
+       1.0,      -1.0,      -1.0,
+      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
+       1.0,      -1.0,
+      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
+       1.0,
+      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
+       1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
+       1.0,      -1.0,      -1.0,      -1.0,      -1.0,
+       1.0,      -1.0,      -1.0,      -1.0,
+       1.0,      -1.0,      -1.0,
+       1.0,      -1.0,
+       1.0/
 DATA PYVEC /
+  8.78919914,
```

```
     +  8.64675152,  2.04454782, -5.18310331,  3.53876667, -7.88399752,
     + -8.38739821,-21.02141997,-25.10394095,  0.01775090, -6.40258907,
     +  9.07316301,  2.21477279, -4.18934594,  4.64570745, -7.69883299,
     + -8.39486263,-21.05959150,-25.21567898,  0.01774509, -6.43447599,
     +  0.71635788, -1.20003026,  0.98991751, -1.32889642,
     + -1.74100084, -4.45442250, -5.59861393,  0.02613645, -1.44219625,
     +  5.44431336,  0.18951117,  3.77151434,
     +  4.13378917, 10.88195613, 13.90068610, -0.02161095,  3.56872382,
     +  4.49770413, -3.84403097,
     + -4.01767838,-10.33185268,-12.56415459,  0.02169120, -3.19600160,
     +  8.54553179,
     +  8.27250466, 20.46798345, 23.69772741, -0.01783242,  6.00808010,
     + 23.81195147, 33.64457162, 29.33323432, -0.00748064,  6.63816656,
     + 71.85128707, 71.56090170, -0.03103290, 16.77905545,
     + 82.30634518, -0.06573714, 19.98961531,
     +  0.49999450, -0.01797396,
     +  5.06794969/

      DO J=1,11
         APYO(J)=APY(J,J)
         DO I=1,11
            IF(I.GE.J) THEN
               BRBY(I,J)=BRBYVEC((J-1)*11-J*(J-1)/2+I)
               BRBY(J,I)=BRBY(I,J)
               R1PY(I,J)=R1PYVEC((J-1)*11-J*(J-1)/2+I)
               R1PY(J,I)=R1PY(I,J)
               PY(I,J)=PYVEC((J-1)*11-J*(J-1)/2+I)
               PY(J,I)=PY(I,J)
               ZY(I,J)=ZYVEC((J-1)*11-J*(J-1)/2+I)
               ZY(J,I)=ZY(I,J)
            END IF
         END DO
      END DO
C
C  Parameters for nonlinear pitch/surge channel filters.
C
      DATA ALPX /0.0/, ALPXMAX /1.0/, Q2X /0.0,0.6/, MUX /4.0E-6/

      DATA APX /
     +  0.35214308, -1.72273247,  0.43037039, -2.38099014,  1.90345806,
     + -1.44348410,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.60755620, -1.99016155,  0.42784160, -2.36134755,  1.89792591,
     + -1.45137984,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000, -0.50000000,  0.50000000,
     + -0.00000000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.17753181, -0.42963264, -0.00091112, -0.00975847,  0.18117108,
     +  0.74938368,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.84419847,  0.23703402, -0.00091112,  0.19024153, -0.01882892,
     +  0.08271701,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     + -0.37464499,  1.04552994, -0.43677819,  2.29193764, -2.05630186,
     +  0.75422005,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000,  0.00000000,  0.00000000,
     +  0.00000000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000,  0.00000000,  0.00000000,
     +  0.00000000,  1.0000,  0.0000,  0.0000,  0.0000,  0.0000,
     +  0.00000000,  0.00000000,  0.00000000,  0.00000000,  0.00000000,
     +  0.00000000,  0.0000,  1.0000,  0.0000,  0.0000,  0.0000,
```

```
+ -0.00024842,  0.00052276, -0.00016391,  0.00089035, -0.00074143,
+  0.00054140,  0.0000,  0.0000,  0.0000, -1.000000,  0.0000,
+ -1.33333333, -1.33333333,  0.00000000, -0.20000000,  0.20000000,
+  1.33333333,  0.0000,  0.0000,  0.0000,  0.0000, -3.14159265/
 DATA BRBXVEC /
+  1.77925517,  1.77466884,  0.00097480,  0.26137162, -0.26225730,
+ -1.78099759,  0.0000,  0.0000,  1.33333333,  0.0000,  0.0000,
+  1.78432002, -0.00205131,  0.27780923, -0.27594547,
+ -1.77100221,  0.0000,  0.0000,  1.33333333,  0.0000,  0.0000,
+  0.00064319, -0.00349374,  0.00290936,
+ -0.00212447,  0.0000,  0.0000, -0.00000000,  0.0000,  0.0000,
+  0.05897769, -0.05580338,
+ -0.25512671,  0.0000,  0.0000,  0.20000000,  0.0000,  0.0000,
+  0.05316003,
+  0.25705694,  0.0000,  0.0000, -0.20000000,  0.0000,  0.0000,
+  1.78479499,  0.0000,  0.0000, -1.33333333,  0.0000,  0.0000,
+  0.0000,  0.0000,  0.00000000,  0.0000,  0.0000,
+  0.0000,  0.00000000,  0.0000,  0.0000,
+  1.00000000,  0.0000,  0.0000,
+  0.0000,  0.0000,
+  0.0000/
 DATA R1PXVEC /
+ 12.10587580, 13.94623075, -0.00000000,  0.66307892,  0.66307892,
+ -7.44252436,  0.0000,  0.0000,  0.0000,  0.10967645,  0.0000,
+ 16.06665966, -0.00000000,  0.76398942,  0.76398942,
+ -8.57318831,  0.0000,  0.0000,  0.0000,  0.10903201,  0.0000,
+  0.00000000, -0.00000000, -0.00000000,
+  0.00000000,  0.0000,  0.0000,  0.0000,  0.00000000,  0.0000,
+  0.03635800,  0.03635800,
+ -0.40737763,  0.0000,  0.0000,  0.0000, -0.00023219,  0.0000,
+  0.03635800,
+ -0.40737763,  0.0000,  0.0000,  0.0000, -0.00023219,  0.0000,
+  4.57748825,  0.0000,  0.0000,  0.0000, -0.11130943,  0.0000,
+  8.00000000,  0.0000,  0.0000,  0.0000,  0.0000,
+  4.00000000,  0.0000,  0.0000,  0.0000,
+  1.00000000,  0.0000,  0.0000,
+  1.0000,  0.00000000,
+  0.0000/
 DATA R2IX /4.17714546E-005,  1.00000000/
 DATA AVX /
+ -0.3001182, -0.3501544, -0.0, -0.0432552, -0.0432552,-0.0219541,
+ -0.0408725, -0.6256486, -0.0, -0.0373488, -0.0373488,-0.0382025,
+  0.0000000,  0.0000000,  0.0, -0.5000000,  0.5000000,-0.0000000,
+  0.1789127, -0.4325385,  0.0, -0.0147076,  0.1852924, 0.7463742,
+  0.8455793,  0.2341282, -0.0,  0.1852924, -0.0147076, 0.0797075,
+  0.2873278, -0.3474845,  0.0, -0.0806039, -0.0806039,-0.6884752/
 DATA BVX /
+  5.947141,-12.514793,  3.924000,-21.314831,17.749601,-12.961123,
+  1.333333,  1.333333, -0.000000,  0.200000, -0.200000,-1.333333/
 DATA DQCX /
+  -2625.631557,  -2610.203763,     -0.000000,
+      5.558626,      5.558626,   2664.724680/
 DATA ZXVEC /
+       1.0,      -1.0,      -1.0,      -1.0,      -1.0,
+      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
+       1.0,      -1.0,      -1.0,      -1.0,
+      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
```

```
     +       1.0,      -1.0,      -1.0,
     +      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
     +       1.0,      -1.0,
     +      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
     +       1.0,
     +      -1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
     +       1.0,      -1.0,      -1.0,      -1.0,      -1.0,      1.0,
     +       1.0,      -1.0,      -1.0,      -1.0,      -1.0,
     +       1.0,      -1.0,      -1.0,      -1.0,
     +       1.0,      -1.0,      -1.0,
     +       1.0,      -1.0,
     +       1.0/
      DATA PXVEC /
     +  8.30351624,
     +  8.09520238,  1.68093020, -4.48023672,  3.57953908, -8.19270520,
     + -8.31577879,-20.67790997,-24.24095727, -0.01780137, -6.16084104,
     +  8.45574766,  1.51070523, -3.37329594,  4.57329645, -7.94167444,
     + -8.30831437,-20.63973845,-24.12921925, -0.01780718, -6.12895412,
     +  0.71635788, -0.98991751,  1.20003026, -2.39658160,
     + -1.74100084, -4.45442250, -5.59861393, -0.02613645, -1.44219625,
     +  4.49770413,  0.18951117,  4.17497242,
     +  4.01767838, 10.33185268, 12.56415459,  0.02169120,  3.19600160,
     +  5.44431336, -4.99112805,
     + -4.13378917,-10.88195613,-13.90068610, -0.02161095, -3.56872382,
     +  9.64863006,
     +  8.43067234, 21.23134650, 25.64717081,  0.01771985,  6.55535001,
     + 23.81195147, 33.64457162, 29.33323432,  0.00748064,  6.63816656,
     + 71.85128707, 71.56090170,  0.03103290, 16.77905545,
     + 82.30634518,  0.06573714, 19.98961531,
     +  0.49999450,  0.01797396,
     +  5.06794969/

      DO J=1,11
         APXO(J)=APX(J,J)
         DO I=1,11
            IF(I.GE.J) THEN
               BRBX(I,J)=BRBXVEC((J-1)*11-J*(J-1)/2+I)
               BRBX(J,I)=BRBX(I,J)
               R1PX(I,J)=R1PXVEC((J-1)*11-J*(J-1)/2+I)
               R1PX(J,I)=R1PX(I,J)
               PX(I,J)=PXVEC((J-1)*11-J*(J-1)/2+I)
               PX(J,I)=PX(I,J)
               ZX(I,J)=ZXVEC((J-1)*11-J*(J-1)/2+I)
               ZX(J,I)=ZX(I,J)
            END IF
         END DO
      END DO
C
C  Parameters for nonlinear yaw channel filters.
C
      DATA ALPR /0.0/, ALPRMAX /1.0/, Q2R /120.0/, MUR /2.0E-6 /

      DATA APR /
     +-1.45728460E-004,-1.69985373E-006,-4.89593935E-004,
     +-2.79035083E-002,   0.0,
     +   1.0,   0.0,   0.0,   0.0,   0.0,
     +1.86874341E-001,2.17980102E-003,-4.89593935E-004,
```

```
     +-2.79035083E-002,0.00000000E+000,
     +   0.0,   0.0,   0.0,   0.0,   0.0,
     +5.21851607E-003,6.08715277E-005,1.75322913E-002,
     +9.99220787E-001,  -1.0/
      DATA BRBRVEC /
     +3.49492524E-002,4.07666539E-004,1.17416612E-001,-5.21851607E-003,
     +   0.0,
     +4.75523782E-006,1.36960937E-003,-6.08715277E-005,   0.0,
     +3.94476555E-001,-1.75322913E-002,   0.0,
     +7.79212948E-004,   0.0,
     +   0.0/
      DATA R1PRVEC /
     +7.79212948E-004,   0.0,7.79212948E-004,   0.0,-2.79035083E-002,
     +0.00000000E+000,   0.0,0.00000000E+000,   0.0,
     +7.79212948E-004,   0.0,-2.79035083E-002,
     +2.00000000E+002,   0.0,
     +9.99220787E-001/
      DATA R2IR /7.79212948E-004/
      DATA AVR /
     +-1.87020070E-001,-2.18150087E-003,-6.28318531E-001,
     +1.00000000E+000,0.00000000E+000,0.00000000E+000,
     +0.00000000E+000,0.00000000E+000,-6.28318531E-001/
      DATA BVR /-6.69716293E+000,-7.81192456E-002,-2.25000000E+001/
      DATA DQCR / 35.80986220,  0.00000000, 35.80986220/
      DATA DQDR /1.28234623E+003/
      DATA ZRVEC /
     +   1.0,  -1.0,  -1.0,  -1.0,  -1.0,
     +   1.0,  -1.0,  -1.0,  -1.0,
     +   1.0,  -1.0,  -1.0,
     +   1.0,  -1.0,
     +   1.0/
      DATA PRVEC /
     +  0.52254103,  0.13331932,  0.53598671, -7.30258353, -6.33348013,
     +  5.01432503,  1.07135777,  6.89175648,  5.57001008,
     +  0.73962239, -5.97893648, -5.25519323,
     + 323.73106725, 186.47187109,
     + 139.87165376/

      DO J=1,5
         APRO(J)=APR(J,J)
         DO I=1,5
            IF(I.GE.J) THEN
               BRBR(I,J)=BRBRVEC((J-1)*5-J*(J-1)/2+I)
               BRBR(J,I)=BRBR(I,J)
               R1PR(I,J)=R1PRVEC((J-1)*5-J*(J-1)/2+I)
               R1PR(J,I)=R1PR(I,J)
               PR(I,J)=PRVEC((J-1)*5-J*(J-1)/2+I)
               PR(J,I)=PR(I,J)
               ZR(I,J)=ZRVEC((J-1)*5-J*(J-1)/2+I)
               ZR(J,I)=ZR(I,J)
            END IF
         END DO
      END DO

c
c  Parameters for nonlinear heave channel filters.
c
```

```
      DATA ALPZ /0.0/, ALPZMAX /0.2/, Q2Z /1.0,2.0/, MUZ /1.0E-7 /

      DATA APZ /
     +   -0.060606,    0.139394,  0.0000,  0.0000,  0.0000,  0.000000,
     +   -0.567713,   -0.767713,  0.0000,  0.0000,  0.0000,  0.000000,
     +    0.000000,    0.000000,  0.0000,  0.0000,  0.0000,  0.000000,
     +    0.000000,    0.000000,  1.0000,  0.0000,  0.0000,  0.000000,
     +    0.000000,    0.000000,  0.0000,  1.0000,  0.0000,  0.000000,
     +   -1.717157,   -2.282843,  0.0000,  0.0000,  0.0000, -62.831853/
      DATA BRBZVEC /
     +    2.948629,    3.920000,  0.0000,  0.0000,  1.717157, 0.0000,
     +    5.211371,  0.0000,  0.0000,    2.282843,  0.0000,
     +    0.000000,  0.0000,  0.0000,  0.0000,
     +    0.000000,  0.0000,  0.0000,
     +    1.000000,  0.0000,
     +    0.000000/
      DATA R1PZVEC /
     +  200.0000,  200.0000,    0.0000,    0.0000,    0.0000,   0.0000,
     +  200.0000,    0.0000,    0.0000,    0.0000,    0.0000,
     +   40.0000,    0.0000,    0.0000,    0.0000,
     +  400.0000,    0.0000,    0.0000,
     +   40.0000,    0.0000,
     +    0.0000/
      DATA AVZ /   -0.060606,    0.139394,   -0.567713,   -0.767713/
      DATA BVZ /    1.717157,    2.282843/
      DATA ZZVEC /
     +   1.0,  -1.0,  -1.0,  -1.0,  -1.0,  -1.0,
     +   1.0,  -1.0,  -1.0,  -1.0,  -1.0,
     +   1.0,  -1.0,  -1.0,  -1.0,
     +   1.0,  -1.0,  -1.0,
     +   1.0,  -1.0,
     +   1.0/
      DATA PZVEC /
     +  910.190970, -315.435226,   -1.160668,  -79.558688, -844.169063,
     +  -13.404542,  327.670260,  -60.259856, -315.797740, -198.948877,
     +   -3.146942,  193.813146,  269.544193,  145.881379,    2.202307,
     + 1160.148818,  888.175899,   13.591950, 1946.358732,   30.388540,
     +    0.480397/

      DO J=1,6
         APZO(J)=APZ(J,J)
         DO I=1,6
            IF(I.GE.J) THEN
               BRBZ(I,J)=BRBZVEC((J-1)*6-J*(J-1)/2+I)
               BRBZ(J,I)=BRBZ(I,J)
               R1PZ(I,J)=R1PZVEC((J-1)*6-J*(J-1)/2+I)
               R1PZ(J,I)=R1PZ(I,J)
               PZ(I,J)=PZVEC((J-1)*6-J*(J-1)/2+I)
               PZ(J,I)=PZ(I,J)
               ZZ(I,J)=ZZVEC((J-1)*6-J*(J-1)/2+I)
               ZZ(J,I)=ZZ(I,J)
            END IF
         END DO
      END DO
c
c     Nonlinear Scaling Coefficients
c
```

```
      DATA GX4/0.5,-0.05,0.002/
      DATA GY4/0.4,-0.035,0.001/
      DATA GZ40/0.6,-0.082,0.0038/
      DATA GZ4S/2.0,-0.05, 0.0/
      DATA GP4/0.3,-0.3,0.1/
      DATA GQ4/0.3,-0.3,0.1/
      DATA GR4/1.1,-1.46,0.64/
C
C     Translational and Rotational Limits
C
      DATA AMX40/10./,BMX4/1./,AMX4S/20./
C
C     Augmented Turbulence Parameters
C
      DATA G2D0,G2D1,G2N0,G2N1,G2N2
     +   /25.0,12.5,2.5,12.0,14.4/
      DATA GT4/1.2/

      RETURN
      END
```

8.21. wtrim3.f

```
      SUBROUTINE WTRIM3
C
C     THIS ROUTINE USES TILT ANGLES TO TRIM OUT ANY INITIAL
C     STEADY-STATE LONGITUDINAL OR LATERAL ACCELERATIONS
C
      INCLUDE 'optint3.com'

      INCLUDE 'comint2.com'
C
C     First-Order Washout Filters to Obtain Angular Rates
C
      XH(1)=XHO(1)+DT*(-T1D0*(XHO(1)+T1N1*ACAO(2)))
      BSDT(1)=XH(1)+T1N1*ACA(2)
      XH(2)=XHO(2)+DT*(-T2D0*(XHO(2)+T2N1*ACAO(1)))
      BSDT(2)=XH(2)+T2N1*ACA(1)
C
C     Integrate Rates to Obtain Angular Displacements
C
      DO K=1,2
         BETAST(K)=BETASTO(K)+DT*0.5*(BSDT(K)+BSDTO(K))
         BSDTO(K)=BSDT(K)
         BETASTO(K)=BETAST(K)
         BETASTLO(K)=BETASTO(K)
         XHO(K)=XH(K)
         ACAO(K)=ACA(K)
      END DO
C
C     Assign Variable Names To Match Modified Algorithm
C     For Input to JACKDRVR
C
      PHI = BETAST(1)
      THE = BETAST(2)
      PHID = BSDT(1)
      THED = BSDT(2)

      RETURN
      END
```

Appendix A. Actuator Extension Limiting

There exists the possibility that, without restriction, the output of the cueing algorithm may drive the motion platform beyond its hardware limits. This raises the requirement that the simulation software should be able to handle the situation that the motion platform may need to be arrested before encountering the hardware limits. A braking algorithm was developed [6] to perform this task. When approaching the hardware limits, the braking algorithm takes over the control of the motion platform from the motion cueing algorithm. It is also necessary that the braking algorithm return control to the cueing algorithm when the cueing algorithm begins to drive the platform toward smaller excursions. In other words, the braking algorithm should release the motion platform to gradually resume normal function.

The logic of the braking algorithm is shown in Figure A.1. The algorithm makes a series of decisions to determine when to arrest, or "brake" the platform and while "braked", to determine when to release the brake. The first decision is based on an evaluation of the expression $2b_0 a_b l_{avail} - \dot{l}^2$ for each actuator at each simulation cycle, where $\dot{l}$ is the velocity of the actuator, l_{avail} is the available actuator length, i.e. the difference between the maximum and computed actuator extensions, a_b is the deceleration by which the braking algorithm will slow down (and stop) the actuator, and b_0 is a coefficient set less than or equal to 1 that is described below. When one actuator reaches the braking region defined by

$$2b_0 a_b l_{avail} - \dot{l}^2 \le 0\,, \tag{A.1}$$

it will be decelerated by a_b. At the same time, all the other actuators will be decelerated proportionally to their respective velocities.

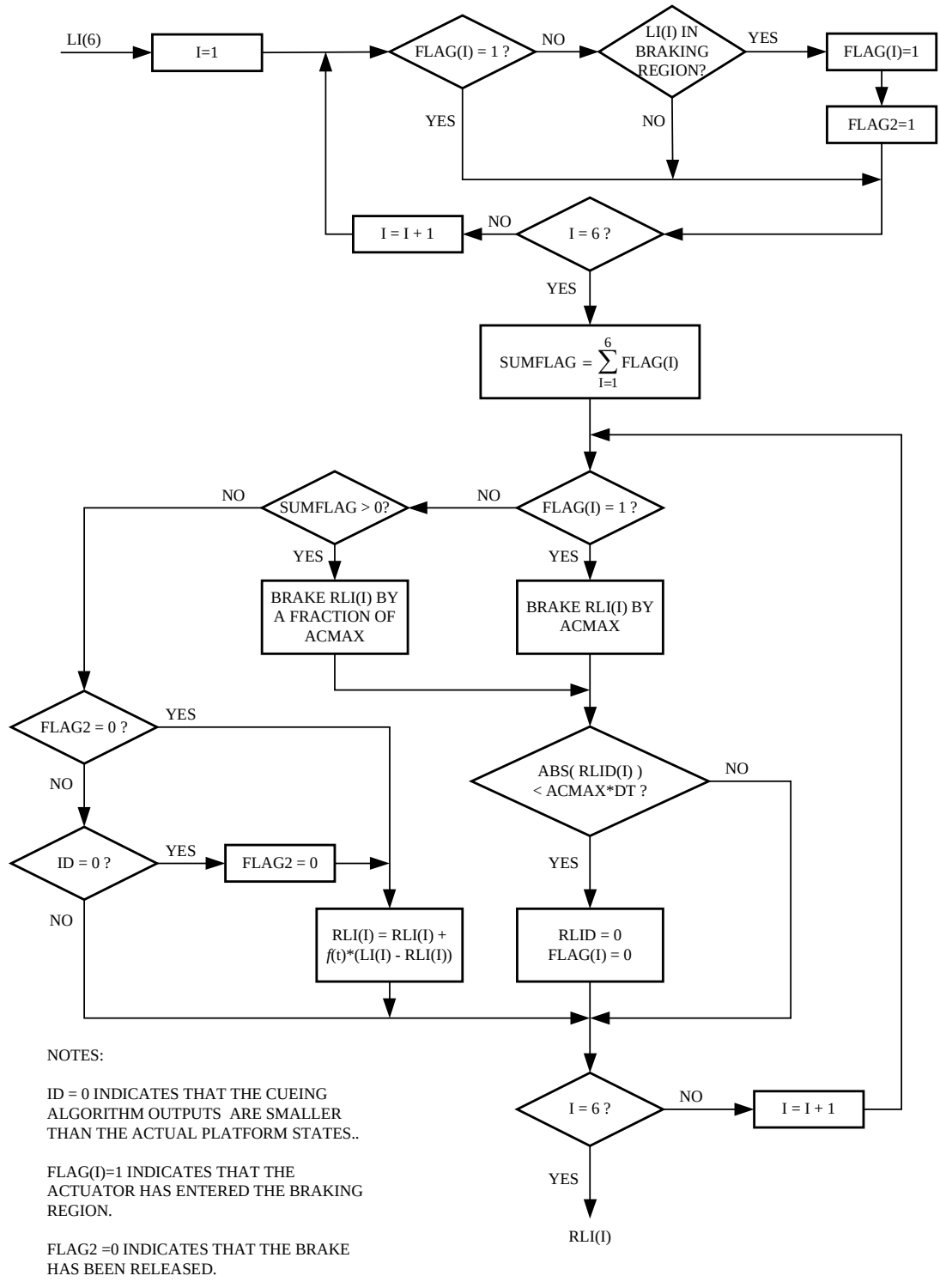


Figure A.1. Braking Algorithm in the JACKDRVR Subroutine.

The braking deceleration a_b, noted by the variable ACMAX in Figure A.1, should be set much smaller than the maximum actuator acceleration so that the perceptual effect of entering the braking region and stopping the motion platform is minimal. The braking algorithm sets a software limit position that is slightly before the simulator hardware limit position. When $b_0 = 1$, the actuator will be stopped exactly at the software limit position; when $b_0 < 1$, the actuator will be stopped before the software limit position.

From Eq. A.1, the available length l_{avail} can be found as a function of the actuator velocity $\dot{l}$. Figure A.2 shows the available length for various combinations of the coefficient b_0 and deceleration a_b. Note that a reduction in the deceleration results in an increase in the available actuator length when entering the braking region. For smaller values of deceleration, larger values of b_0 are used to reduce the available length.

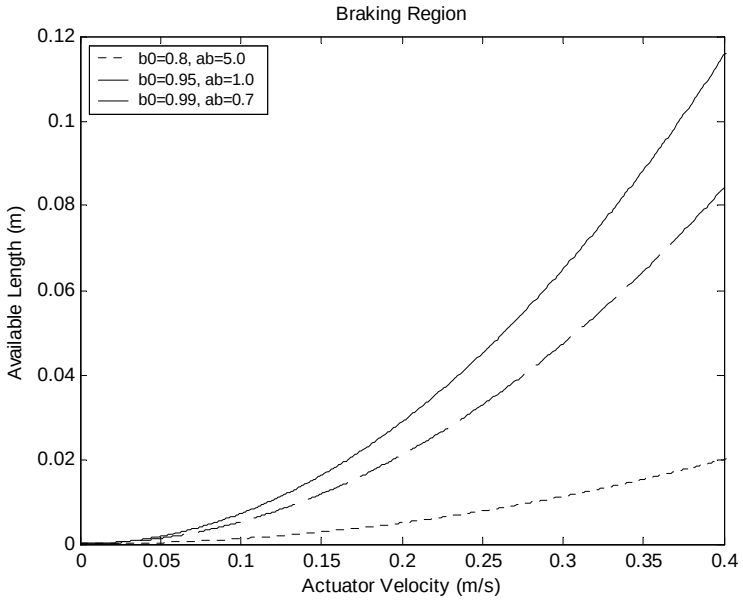


Figure A.2. Available Actuator Length upon Entering the Braking Region.

In order to "release" the brake, a decision is based on a comparison between the actual motion platform states and the cueing algorithm output. This comparison begins

when the platform has been completely stopped by the braking algorithm. When the cueing algorithm output states become smaller than the corresponding actual platform states, the braking algorithm will "release" the brake.

When the brake is released, the cueing algorithm output may have large velocities while the simulator has small velocities at that instant. To avoid the normal cueing algorithm output being resumed with any excessively large acceleration, an algorithm that allows the simulator to gradually follow the washout algorithm output was developed. The algorithm expresses the actuator extension command to the motion platform as

$$l_{com} = l_{act} + f(t)(l_{des} - l_{act}), \tag{A.2}$$

where l_{com} is the commanded actuator extension that drives the motion platform, l_{act} is the actual extension of the actuator (limited by the braking algorithm), l_{des} is the desired actuator extension generated by the cueing algorithm, and $f(t)$ is a function of time. During normal operation $f(t)$ is set equal to one. After the brake is released $f(t)$ is set to zero and is gradually increased to one.

References

[1] Telban, R. J., and Cardullo, F. M., *Motion Cueing Algorithm Development: Human-Centered Linear and Nonlinear Algorithms*. 2005, NASA CR-2005-213747, NASA Langley Research Center, Hampton, VA.

[2] Dieudonne, J. E., Parrish, R. V., and Bardusch, R. E., *An Actuator Extension Transformation for a Motion Simulator and an Inverse Transformation Applying Newton-Raphson's Method*. 1972, NASA TN D-7067, Langley Research Center, Hampton, VA.

[3] Parrish, R. V., and Martin, D. J., Jr., *Application of Nonlinear Adaptive Motion Washout to Transport Ground-Handling Simulation*. 1983, NASA Technical Memorandum 84568, NASA Langley Research Center, Hampton, VA.

[4] Parrish, R. V., Dieudonne, J. E., and Martin, D. J., Jr., *Motion Software for a Synergistic Six-Degree-of-Freedom Motion Base*. 1973, NASA TN D-7350, NASA Langley Research Center, Hampton, VA.

[5] Telban, R. J., Cardullo, F. M., and Houck, J. A., *Developments in Human Centered Cueing Algorithms for Control of Flight Simulator Motion Systems*. *AIAA Modeling and Simulation Technologies Conference*. 1999. Portland. OR.

[6] Wu, W., *Development of Cueing Algorithm for the Control of Simulator Motion Systems*, 1997, M.S. Thesis, State University of New York at Binghamton, Binghamton, NY.

[7] Reid, L. D., and Robinson, P. A., *Augmenting Flight Simulator Motion Response to Turbulence.* Journal of Aircraft, 1989. **27**(4): p. 306-311.

[8] Martin, D. J., Jr., *A Digital Program for Motion Washout on Langley's Six-degree-of-freedom Motion Simulator*. 1977, NASA CR-145219, NASA Langley Research Center, Hampton, VA.

[9] Brogan, W.L., *Modern Control Theory*. 3rd ed. 1991, Upper Saddle River, NJ: Prentice-Hall.

[10] Reid, L. D., and Nahon, M. A, *Flight Simulation Motion-Base Drive Algorithms: Part 1 - Developing and Testing the Equations*. 1985, UTIAS Report No. 296, CN ISSN 0082-5255, Institute for Aerospace Studies, University of Toronto, Toronto, Canada.

[11] Ham, F. M., and Collins, E. G., *A Neurocomputing Approach for Solving the Algebraic Matrix Riccati Equation*. *IEEE International Conference on Neural Networks*. 1996.

[12] Telban, R. J., Cardullo, F. M., and Kelly, L. C., *Motion Cueing Algorithm Development: Piloted Performance Testing of the Cueing Algorithms*. 2005, NASA CR-2005-213748, NASA Langley Research Center, Hampton, VA.

[13] Smith, R. M., *A Description of the Cockpit Motion Facility and the Research Flight Deck Simulator*. *AIAA Modeling and Simulation Technologies Conference*. 2000. Denver, CO.

[14] McFarland, R. E., *Adjustable Limiting Algorithms for Robust Motion Simulation*. *AIAA Modeling and Simulation Technologies Conference*. 2001. Montreal, Canada.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 05 - 2005	Contractor Report	

4. TITLE AND SUBTITLE

Motion Cueing Algorithm Development: New Motion Cueing Program Implementation and Tuning

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Telban, Robert J.; Cardullo, Frank M.; and Kelly, Lon C.

5d. PROJECT NUMBER

L70823D

5e. TASK NUMBER

5f. WORK UNIT NUMBER

23-090-70-10

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

State University of New York - Binghamton
Binghamton, NY 13902-6000
Unisys Corporation
Hampton, VA 23666

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-2199

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/CR-2005-213746

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 54
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

Prepared by the State University of New York - Binghamton for NASA Langley Research Center under subcontract Unisys Corporation. NASA Langley Research Center Technical Monitor: Jacob A. Houck.
An electronic version can be found at http://ntrs.nasa.gov

14. ABSTRACT

A computer program has been developed for the purpose of driving the NASA Langley Research Center Visual Motion Simulator (VMS). This program includes two new motion cueing algorithms, the optimal algorithm and the nonlinear algorithm. A general description of the program is given along with a description and flowcharts for each cueing algorithm, and also descriptions and flowcharts for subroutines used with the algorithms. Common block variable listings and a program listing are also provided. The new cueing algorithms have a nonlinear gain algorithm implemented that scales each aircraft degree-of-freedom input with a third-order polynomial. A description of the nonlinear gain algorithm is given along with past tuning experience and procedures for tuning the gain coefficient sets for each degree-of-freedom to produce the desired piloted performance. This algorithm tuning will be needed when the nonlinear motion cueing algorithm is implemented on a new motion system in the Cockpit Motion Facility (CMF) at the NASA Langley Research Center.

15. SUBJECT TERMS

Flight Simulation, Simulators, Motion Systems, Cueing Algorithms, Motion Perception

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	144	(301) 621-0390