This is to certify that the

thesis entitled

The Development of a Remote-Controlled Aircraft Flight Simulation

presented by

David Arnold McClaughry

has been accepted towards fulfillment
of the requirements for

Master of Science degree in **Mechanical Engineering**

Major professor

Erik Goodman

Date 5/8/85

THE DEVELOPMENT OF A REMOTE-CONTROLLED AIRCRAFT FLIGHT SIMULATION

By

David Arnold McClaughry

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Mechanical Engineering

1985

ABSTRACT

THE DEVELOPMENT OF A
REMOTE-CONTROLLED AIRCRAFT
FLIGHT SIMULATION

By

David Arnold McClaughry

This thesis presents a structured approach for calculating, displaying and simulating the motion of a remote-controlled aircraft. A PRIME 750 is used to compute and sum the forces generated by the engine, fuselage and aerodynamic bodies, yielding the equations of motion. These are then integrated over time to yield the aircraft's velocities. The velocities are downloaded to an Evans and Sutherland PS300 which performs an integration over time to find the aircraft's position. The solutions obtained provide a description of the flight path of the aircraft which is used to generate the PS300's visual display. Various strategies to partition the computational load between the PRIME and the PS300 were investigated, with the goal of providing smooth visualization while preserving rapid control response. The pilot is able to interactively implement aircraft controls via the PS300 control dials.

to my family

ACKNOWLEDGEMENTS

I wish to express my gratitude to my advisors, Dr. Erik Goodman and Dr. Clark Radcliffe for their guidance and assistance throughout my research and graduate study.

Many thanks to my family for their never-ending love and support.

A special thank you goes to Marci for her love, understanding and help through the difficult times.

Finally, a thanks to my friends for all they have done for me.

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

A - acceleration, projected frontal area

b - airfoil span

c - airfoil chord

C_D - airfoil drag coefficient

C_{Df} - fuselage drag coefficient

C_L - airfoil lift coefficient

C_M - airfoil moment coefficient

C_T - propeller thrust coefficient

D - airfoil drag force, propeller diameter

D_f - fuselage drag force

D_i - airfoil induced drag force

F - resultant external force vector

F_{el} - force at an individual force-generating element

G - resultant moment vector about the center of mass

G_{el} - moment about an individual force-generating element

I - moment of inertia matrix

I_{xx} - moment of inertia about x axis

I_{yy} - moment of inertia about y axis

I_{zz} - moment of inertia about z axis

I_{xz} - product of inertia $\int xz\ dm$

J - advance ratio of propeller

K_i - Runge-Kutta intermediate velocity evaluation

L - airfoil lift force

m - mass of the aircraft

M - airfoil moment

n - angular velocity of propeller

P,Q,R - angular velocity about the center of mass

P',Q',R' - transformed angular velocity about the center of mass

r - line of action of an individual force-generating element

q - dynamic pressure

s - airfoil area

T - thrust generated by the propeller

U,V,W - linear velocity of the center of mass

U',V',W' - transformed linear velocity of the center of mass

V - velocity vector of the center of mass

V_o - previous velocity vector of the center of mass

X - position vector of the aircraft

Δt - time interval for integration

α - angle of attack of the airfoil, angular acceleration vector about the center of mass

λ_i - ith eigenvalue of a system of equations

λ_r - real root of an eigenvalue

ϕ,θ,γ - angles for coordinate transformations

ρ - air density

ω - angular velocity vector about the center of mass

NOTE: Throughout this paper the author refers to the Evans and Sutherland PS300 system as the E-S.

CHAPTER 1

Introduction

With the recent advances in the computer industry, real-time flight simulators can be developed on mini- and micro-computers with a high degree of accuracy and realism. These advances allow computer flight simulators to be used in a variety of applications.

Commercial airlines, the military and NASA use computer-driven flight simulators for safely training pilots in a wide range of flight situations in order to improve their piloting abilities. These simulators are very complex machines, often having one or more computers dedicated solely to the simulation task. Realizing the force feedback in the controls, providing the sensation of flight forces and motion and

displaying intricate instrumentation and advanced graphics are some of the features of these very complex simulation systems.

In a second application, research simulators are of great aid to aircraft designers. Preliminary testing of aircraft designs can be confidently performed without actually building a plane. It is possible to evaluate the stresses which aircraft parts will be subjected to in order to aid in design optimization. The effects of parameter changes on aircraft stability and maneuverability can be determined before implementation.

A third type of flight simulator is the recreational or hobby simulator. Only recently have home computers gained the capabilities of modeling a complex dynamic problem such as flight simulation in near real-time. However, the realism of these simulators is limited. It is difficult to gain a feel for controls implemented with keystrokes on a keyboard. Also, the graphics are often "jumpy" and do not provide a realistic display. All of the above simulation systems rely heavily upon extensive flight testing data and/or several very complex aerodynamic theories.

The goal of this thesis is to outline a method for generating simple but realistic equations of motion, and implementing these in a flight simulation. According to this goal, the simulator can be categorized as a hobby-type simulator. Note that exact engineering data are not expected from this research -- only an approximation of the

flight path of a vehicle of the type modeled. However, some of the negative factors of the prevailing hobby-type simulator are addressed and solved.

Before embarking on the dynamic modeling of an aircraft, it is valuable to examine the constraints on the simulation, and thus define a foundation for the flight simulator. The computing systems available were a multiple-user PRIME 750 which can handle the computationally intensive equations of motion for the flight simulation, and an Evans and Sutherland PS300 that lends itself well to the graphical display tasks. The rate of communication between the two systems and the effects of a multiple-user system on the simulation provide additional constraints on the simulation performance. These effects -- in addition to each computer's architecture -- determined the simulation task assignment.

The flight situation selected for simulation was the remote piloting of an aircraft using visual feedback from the point of view of a fixed ground observer. This situation was well-suited to the available resources, and presented a graphical challenge. To model the flight, visual feedback of the simulator should correspond to an abstraction of the actual flight scene. Similarly, the controls should resemble those of the real aircraft. Immediately, it was recognized that the force feedback present in aircraft controls would be difficult to model. Therefore, simulation of a flight scheme which does not rely on control feedback was selected. Finally, the idea of a simple dynamic

model became important. The computing time period and power for real-time computation on the time-sharing mini-computer is limited. It is assumed that the aircraft modeled flies at relatively low airspeed and altitude, implying that the fluid properties of air are constant. This simplifies the calculation of the aerodynamic forces. Therefore, simpler calculations yielding a quicker response time resulted in a more realistic real-time simulation.

Clearly, a simulator of this type can serve a useful purpose in training and practicing the techniques necessary to fly a remote-controlled aircraft. An aircraft of this type requires only visual feedback between the pilot and the plane. No instrumentation and certainly no force feedback controls are present to aid in the flying of the plane. The crash or loss of a remote-controlled aircraft from inadequate practice and simulation would be costly in terms of time and money spent in designing and building the model.

CHAPTER 2

Dynamic Model

The first step in the development of a flight simulator is the construction of a dynamic mathematical model of the aircraft which can be implemented on the computer.

The key to the dynamic modeling is simplification. A fast and computationally simple model of the aircraft is needed for implementation as a real-time simulation. This will maximize the simulator's control response. Conversely, a certain amount of detail must be present in the model to attain the flying qualities of an aircraft. The guideline: Any factors which do not influence the general flying characteristic of the plane will be neglected.

The aircraft is best modeled under these conditions as a body with various forces and moments acting upon it. As a further extension of the simplification idea, the aircraft structure is assumed to be rigid during flight. Initially, the forces and moments acting on the plane are very general and can be located anywhere on the aircraft. These forces and moments are then summed vectorially at the center of mass of the aircraft (C.M.). Newton's Second Law, $\underline{F} = m\underline{A}$, can be applied and the equations of motion defined.

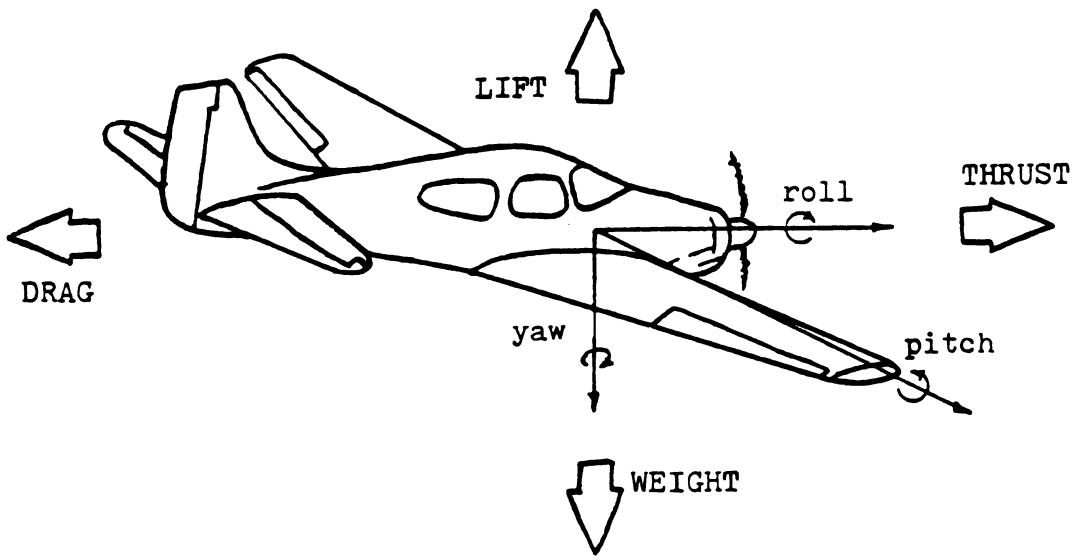


Figure 1: Primary and secondary forces and moments of an aircraft

There are four primary forces acting on the plane: Lift, thrust, drag, and weight [7]. These forces are shown in Figure 1. Figure 1 also identifies a set of secondary moments [7]. These are the control moments; i.e., the rolling moment, the pitching moment and the yaw moment. These moments provide handling and directional control of the

aircraft. The primary force and moment generating elements are identified in Figure 2 -- the main wing, the horizontal tail, the vertical tail, the fuselage and the engine/propeller combination. The control elements also identified in Figure 2 -- the aileron, the elevator, the rudder and the throttle -- produce the directional control and performance forces and moments. There are many more components that

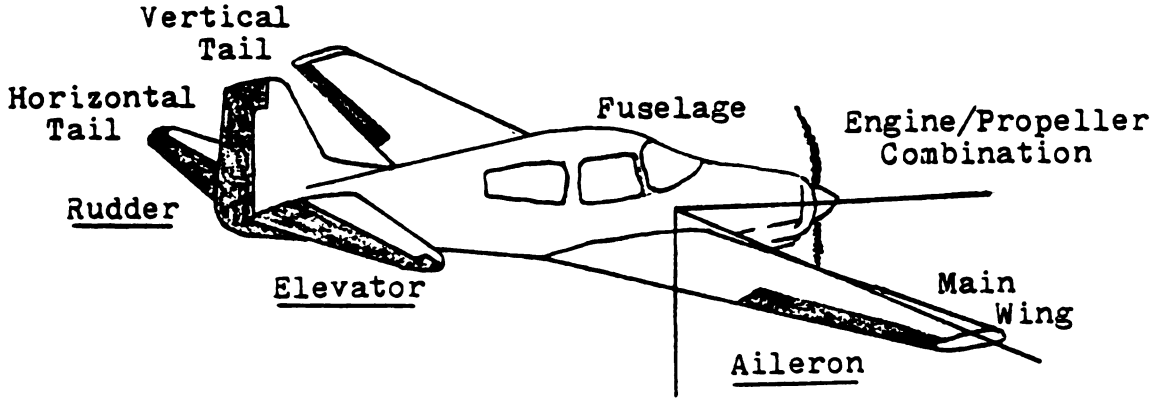


Figure 2: Force-generating elements and control surfaces of an aircraft

contribute to the flight forces and moments of a real plane. However, these main elements may be used to define a mathematically simple model that describes the aircraft flying characteristics adequately for the purposes at hand.

Based on the mode of generation of these forces and moments, the elements are divided into three unique categories: 1) the airfoil-shaped sections, which include the main wing, the horizontal

tail and the vertical tail, 2) the fuselage, and 3) the engine/propeller combination. The evaluation of the forces and moments of these elements will be addressed individually. The control forces and moments are a subset of the primary forces and moments and will be discussed later.

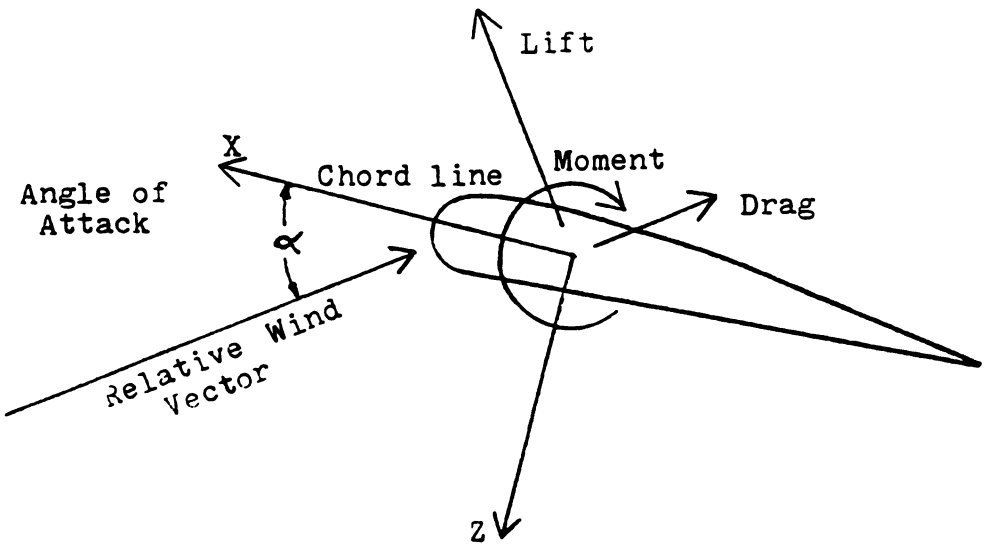


Figure 3: Forces and moments generated by an airfoil-shaped body

Airfoil Sections

When an airfoil moves through a fluid, two forces and a moment are generated. These forces, shown in Figure 3, are lift, which is perpendicular to the relative wind vector, and drag, which is parallel to the relative wind vector[1]. The moment produced is about the y axis

of the airfoil. The above forces and moments are evaluated using wind tunnel testing. The National Advisory Committee on Aeronautics (NACA) and later the National Aeronautics and Space Administration (NASA) have published wing section data that gives the lift, drag, and moment coefficients for a given wing type [1]. The airfoil selected was one which is suitable for a light commercial aircraft; the classification number is NACA 63-412.

Thus, the force and moment calculations are based upon the wind tunnel test data for the selected airfoil. These data are presented in the form of a force or moment coefficient versus the angle of attack of the airfoil (α). The angle of attack is defined as the angle included between the relative wind vector and the chord line of the airfoil, as shown in Figure 3. The relative wind of the airfoil must be calculated to determine this angle. Three main factors define the relative wind of an airfoil: 1) the linear velocity of the C.M. and the angular velocity about the C.M., 2) the location of the airfoil relative to the C.M. of the plane, and 3) the orientation of the airfoil axis to the plane axis. The first two factors determine the fluid velocity at the airfoil. The third factor transforms the fluid velocity into the coordinate system defined by the dihedral, washout and sweep angles of the airfoil. These angles are fixed for a given surface of the aircraft. (The development of the transformation of coordinates appears in Appendix A.) At this point the airfoil is assumed to be of infinite length and therefore two-dimensional. This assumption neglects the downwash of the wings. The correction for this assumption will be

addressed later. The angle of attack is then defined by the relative horizontal and vertical velocities. The forces and moments generated are calculated using the wind tunnel test data (shown in Appendix B) and the characteristic dimensions of the airfoil:

$$L = C_L \; q \; s \tag{1}$$
$$D = C_D \; q \; s \tag{2}$$
$$M = C_M \; q \; s \; c \tag{3}$$

where b - airfoil span
c - airfoil chord
C_D - drag coefficient
C_L - lift coefficient
C_M - moment coefficient
D - airfoil drag force
L - airfoil lift force
M - airfoil moment
q - dynamic pressure, $\frac{1}{2}\rho V^2$
s - wing section area, b.c

At this point it is important to discuss the implication of using a finite aspect ratio wing. The primary result when downwash is included is an increase in the airfoil drag [9]. The additional drag effect is known as induced drag, and current aerodynamic theory predicts the induced drag to be:

$$D_i = L^2 \; / \; qb^2\pi \tag{4}$$

where D_i - airfoil induced drag force

Therefore, once the lift is known the induced drag effects can be incorporated into the force and moment summation. In view of the need for simplification of this model, other secondary effects of downwash have been omitted.

The procedure described above is used repeatedly for all of the airfoil bodies on the aircraft. The same wind tunnel test data is used in each case with the exception of the rudder. The rudder is assumed to be a symmetric airfoil. This results in no moment or perpendicular force being produced when the airfoil angle of attack is zero. Therefore in the lift and moment calculation an offset has been subtracted to satisfy the symmetric airfoil condition.

Fuselage

The main purpose of the fuselage is to house the passengers and freight that the aircraft is to carry. It is designed to have minimum effect on the flying forces and moments of the aircraft. It is assumed that the fuselage never experiences large angles of attack. Therefore, moments produced by the drag on the fuselage are negligible. However, there is a substantial contribution in the drag or performance forces. For this reason the fuselage is modeled as a pure drag-producing element. Three major parts of the fuselage are examined -- the fuselage body, the engine/nacelle and the landing gears/wheels. Aerodynamic theory shows that this parasitic drag is proportional to the square of the velocity [4]. Therefore, a relationship between the aircraft's velocity and projected frontal area and the fuselage drag can be defined:

$$D_f = C_{Df} V^2 A \qquad (5)$$

where A - projected frontal area of the aircraft
C_{Df} - fuselage drag coefficient
D_f - fuselage drag force
V - velocity

Once again the evaluation of the drag coefficients come from testing performed by the NACA. Because of the small angle of attack assumption the drag coefficient is assumed to be constant. The data for the three identified fuselage parts are summed and the result identified as the force and moment contributions of the fuselage.

Engine/Propeller Combination

The main purpose of the engine/propeller combination is to furnish the thrust to overcome the drag forces of the plane, as well as provide a means for longitudinal acceleration of the plane. The engine model is assumed to be an ideal engine in the sense of maintaining a constant engine speed for a given throttle setting. The propeller is analyzed as a rotating airfoil which creates a lift along the x axis of the plane, namely thrust.

A simple analytical expression is difficult to define for the thrust of a propeller. However, the basic characteristics can be examined by considering a typical blade and applying blade element theory [6]. With this analysis a relationship between the advance rate of the propeller and the thrust coefficient of the propeller can be developed. Having defined the propeller characteristics and evaluated

the advance ratio

$$J = V/nD \tag{6}$$

where J - advance ratio
n - rotational speed of the propeller
D - propeller diameter

the thrust can be determined:

$$T = C_T n^2 \rho D^4 \tag{7}$$

where C_T - thrust coefficient
T - thrust

(The thrust coefficient graph is shown in Appendix B.)

The secondary effects of the aircraft power plant have been disregarded for the sake of simplistic modeling, even though these effects may require some pilot control responses. The effects of the rotating flow interacting with the flying surfaces behind the propeller were neglected. The effects of precession during attitude changes, and asymmetric loading of the propeller during constant angle of attack conditions -- such as climb and dive -- were also neglected.

By using the ideas described up to this point, the primary forces and moments of the aircraft were defined and evaluated. It is important to be able to implement the necessary controls of the plane. Analyzing the effects of the aircraft controls will give an insight to the simplest method of modeling them.

Throttle and Control Surfaces

Throttle: Since the engine is assumed to be ideal, a throttle variation is simply an increase or decrease in the engine speed. This will be implemented in a 0-100 percent range.

Aileron, Elevator, and Rudder: Each control deflection produces a moment about an aircraft axis which acts to place the main force-generating elements in a different flight configuration. The key is to produce a moment about a given axis. The three control surfaces are implemented in the same manner but yield vastly different effects. The desired effect can be achieved by changing the angle of attack of the chord line of a given primary force-generating element. By changing the angle of attack of the horizontal tail, a moment about the lateral axis is generated, and the plane pitches up or down. This simulates the effects of an elevator. To induce a moment about the longitudinal axis of the plane, the angle of attack of the outer portion of the main wings is varied. Unlike the elevator, the right and left wing sections are displaced in opposite directions. This results in an unequal lift distribution and therefore a roll moment, simulating the effects of the aileron. The rudder acts in a manner similar to the elevator. The angle of attack of the vertical tail is changed, thus inducing a sideways force in one direction which transforms to a yaw moment at the plane's C.M. These changes in angle of attack are very slight, with a maximum effect being 0.6 percent of the mean angle of attack calculated from the relative wind. The moments induced by these small changes in

angle of attack of the force-generating elements are approximately equal to the moments generated by the control surfaces of a real aircraft. They are present to provide a means of generating the necessary control moments. This method of modeling the controls does allow for many of the considerations of a real aircraft, such as trimming the aircraft at cruise speed and adverse yaw in a roll.

With the evaluation of the forces and moments of the plane complete, the summation process can be performed. The axis system used is attached to the plane, with the origin coincident to the plane's C.M. The characteristic moment arms are defined using the layout of a typical light aircraft and estimating the line of action of the forces of each element. (These dimensions are shown in Appendix C.) All of the forces and moments can now be transformed to the aircraft's C.M. using the following relationship:

$$\underset{\sim}{F} = \sum \underset{\sim}{F}_{el} \tag{8}$$

$$\underset{\sim}{G} = \sum (\underset{\sim}{G}_{el} + \underset{\sim}{F}_{el} \times \underset{\sim}{r}) \tag{9}$$

where F is the force at the C.M.
G is the moment at the C.M.
F_{el} is the force at the element
G_{el} is the moment at the element
r is the distance to the line of action

This results in a new set of forces and moments acting through the C.M.

To derive the equations of motion it is necessary to apply Newton's Second Law to the rigid body aircraft. This task is most easily performed in the frame of reference in which the forces were summed,

fixed to the plane. This eliminates the occurrence of cross-products and derivatives of the moments of inertia. However, it will require additional terms. The equations of motion for a rotating reference frame are:

$$\underset{\sim}{F} = m\underset{\sim}{A} + m\underset{\sim}{\omega} \times \underset{\sim}{V} \tag{10}$$

$$\underset{\sim}{G} = \underline{\underline{I}}\underset{\sim}{\alpha} + \underline{\underline{I}}\underset{\sim}{\omega} \times \underset{\sim}{\omega} \tag{11}$$

where F - total forces summed at C.M.
G - total moments summed at C.M.
A - linear acceleration of C.M.
α - angular acceleration of C.M.
V - linear velocity of C.M.
ω - angular velocity of C.M.
I - moments of inertia of aircraft
m - mass of the aircraft

By rearranging the terms and placing the accelerations on the left, the result is:

$$\underset{\sim}{A} = \underset{\sim}{F}/m - \underset{\sim}{\omega} \times \underset{\sim}{V} \tag{12}$$

$$\underset{\sim}{\alpha} = (\underset{\sim}{G} - \underline{\underline{I}}\underset{\sim}{\omega} \times \underset{\sim}{\omega})/\underline{I} \tag{13}$$

These are the accelerations of the C.M. of the plane and are integrated twice over time to describe the flight path of the aircraft. These equations were analyzed using DIFFEQ, a simulation package for solving non-linear differential equations, in order to verify the behavior of the equations [13]. (The results of this verification can be seen in Appendix D.)

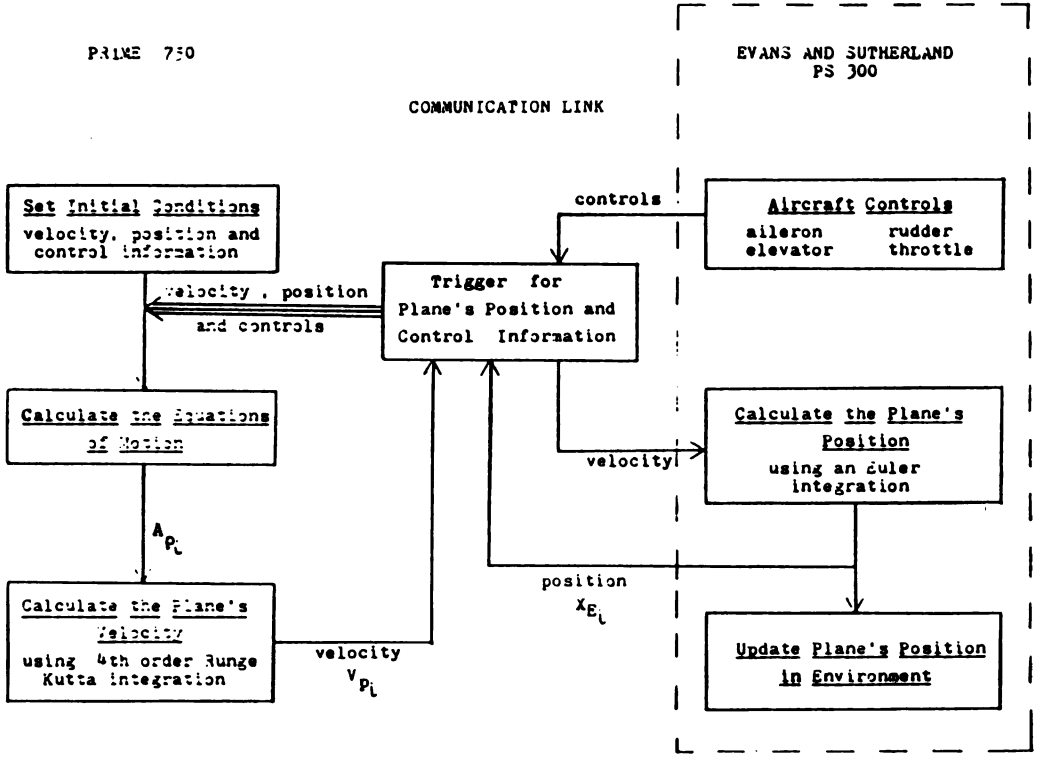
CHAPTER 3

Simulation Implementation

With a usable mathematical model of the aircraft constructed, the next step in the development of a flight simulator is to utilize the model together with an integration scheme to define the aircraft's flight path. Once the plane's position is calculated, the flight path can be visualized on the computer. With the resources of two computers, many methods for implementing the simulation are available. A desirable solution is one which updates aircraft position and control inputs as quickly as possible. Therefore, the simulation tasks must be matched to the abilities of each system.

There are four basic steps in the simulation: Evaluation of the equations of motion, velocity integration, position integration and flight path display. It is beneficial to identify the computer system and necessary communication associated with each task and then to discuss the specifics of each task individually. The evaluation of the equations of motion is performed on the PRIME; it requires the velocity and position of the plane, as well as the control settings. The PRIME integrates these results over time to yield the velocity of the plane. The new velocities are sent down to the E-S once this integration is completed. With this velocity information the E-S can perform an integration over time to obtain the flight path data. The position is frequently recalculated on E-S at a rate determined by the system's display refresh rate. Finally, the flight path data is updated graphically on the E-S. Figure 4 gives a schematic representation of the task division in addition to the points of communication between the two systems. Equations 14-15, in a general sense, identify the time steps used for the integration process. Figure 5 outlines the order and timing of the various tasks in the simulation.

The evaluation of the equations of motion is performed using the model described in the previous section. The key point in implementing the model is the update of the control information and plane position. This information is provided by a triggering mechanism on the E-S that collects the current aircraft status and uploads it when the PRIME is ready to evaluate the equations of motion again.



$$V_{p_i} = \int A_{p_i} \cdot \Delta t_p \qquad (14)$$

$$X_{E_i} = X_{E_{i-1}} + \sum_{j=1}^{n} V_{p_{i-1}} \cdot \Delta t_E \qquad (15)$$

A_p - the acceleration calculated on the PRIME

V_p - the velocity calculated on the PRIME, and down loaded to the E&S

X_E - the position calculated on the E&S, and up loaded to the PRIME

Δt_p - the time step for the PRIME to complete the velocity integration

Δt_E - the time step for the E&S to complete the position integration

n - the number of position integrations per one velocity integration, $INT(\Delta t_p / \Delta t_E)$

Figure 4: Schematic representation of the simulation task division and comunication points

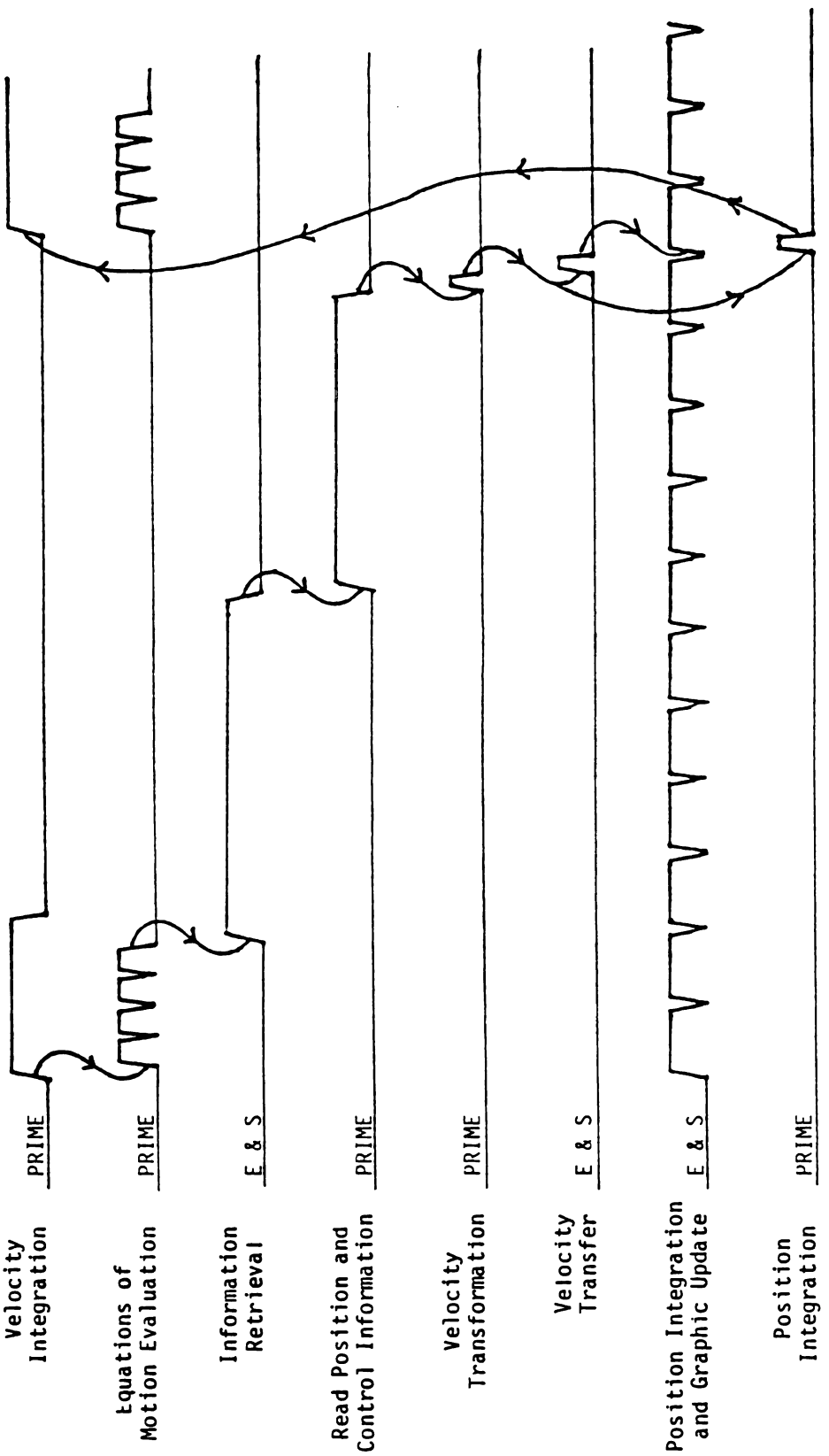


Figure 5: Timing and triggering mechanisms of simulation tasks

The purpose of the velocity integration is to calculate the plane's velocity from the equations of motion. The most suitable integration routine is determined by the constraints on the problem. The simulation is working in a real-time environment and there is a relatively large time step involved; instabilities due to numerical integration technique are quite possible. Also, the time step over which the equations of motion are evaluated is not fixed, due to the multiple user configuration of the PRIME. Therefore a high-order, variable time step routine is necessary. The integration method which is most easily implemented in this situation is a fourth order Runge-Kutta:

$$V = V_0 + \tfrac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \qquad (16)$$

$$K_1 = \Delta t \cdot f(V_0) \qquad (17)$$

$$K_2 = \Delta t \cdot f(V_0 + \tfrac{1}{2}K_1) \qquad (18)$$

$$K_3 = \Delta t \cdot f(V_0 + \tfrac{1}{2}K_2) \qquad (19)$$

$$K_4 = \Delta t \cdot f(V_0 + K_3) \qquad (20)$$

where V - plane's velocity
K_i - intermediate velocity evaluation
Δt - time interval

A problem arises with the use of a variable time step routine. The time step, Δt, is not known until the integration is complete, yet it is needed to complete the integration. To resolve this problem, the time step from the previous velocity evaluation is used. This will induce some variation between real time and the velocity calculation. However, it is assumed that the time steps are fairly consistent, and this effect is negligible. The PRIME's internal clock, with a rate of 3

centiseconds, is used as a real-time clock for the velocity integration.

Once the velocities are calculated, this information is made available for the evaluation of the equations of motion at the next time step and for the position integration, as shown in Figure 4. Before the velocity information is sent down to the E-S, it must be transformed into a fixed inertial reference frame. (This transformation is discussed in Appendix A.) At this point the transformed velocities are downloaded to the E-S for position integration. Recall that a communication between the two computers during the velocity evaluation invokes the uploading of the current control information and plane position from the E-S. By using this point for the triggering mechanism the simulation is kept in synchronization; the most current information is always used and the looping time reduced.

The objective of the position integration is to generate the flight path of the plane from the velocity information at a rate quick enough to make the display appear to move continuously. To achieve this goal the position integration is performed entirely on the E-S, independent of the user load on the PRIME. With this independent configuration the time step for position integration is relatively small. It is sufficient to use an Euler method of integration to calculate the plane's position:

$$X_i = X_{i-1} + V_j \cdot \Delta t \qquad (21)$$

where X - plane's position

By examining equation 21, it is seen that there is still a dependence on the PRIME for velocity information in the position calculation. To allow for the small time step necessary for display continuity, the position integration must not wait for a new velocity. It will, instead, use the velocity information of the previous time step. As soon as the new velocities are calculated they are utilized in the position integration -- a procedure which can also be seen in Figure 4 and equation 15. By isolating the position integration on the E-S, the display is updated in a graphically smooth manner while the velocity update keeps the dynamic model, velocity integration and position integration in synchronization. A clock function on the E-S, with a rate of 5 centiseconds, is used as the real time clock for the position integration.

The final task in the simulation is the display of the flight path. Before the position integration information can be utilized it is necessary to describe the environment that the plane will fly in and identify the viewpoint of the pilot. The "simulation world" consists of a horizon, some points of reference for the pilot, and the aircraft itself. All of these objects are defined as vector lists on the E-S. Each of them is fixed in position with the exception of the plane, which is able to translate and rotate in all directions. Figure 6 shows this environment as it is depicted on the E-S. By the definition of a remote-controlled simulation, the pilot's viewpoint is placed slightly above the origin of the environment. The line of sight is defined by the vector between the pilot's eyes and the aircraft's C.M. as shown in

Figure 7. As the plane flies through the environment, the pilot rotates to track the aircraft's C.M., always keeping the aircraft in the center of the pilot's field of view, i.e., the center of the screen.

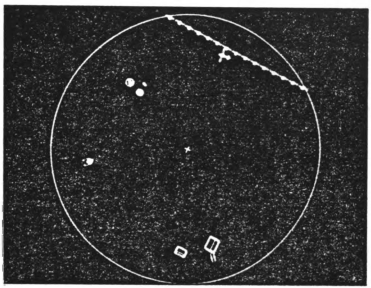


Figure 6: Depiction of the simulation environment

This will present a display in which the environment appears to translate while the plane remains fixed. The plane also changes size with its distance from the pilot and rotates about its own axes as it rolls, pitches and yaws.

One of the great advantages of using the Evans and Sutherland PS300 system is the firmware functions that are provided. These functions enable a graphic transformation to be performed extremely rapidly. Use of these functions by the flight simulator allows position information from the integration to be used directly to update the graphic display. (A schematic representation of these functions and how they are

incorporated into the simulator are provided in Appendix E.)

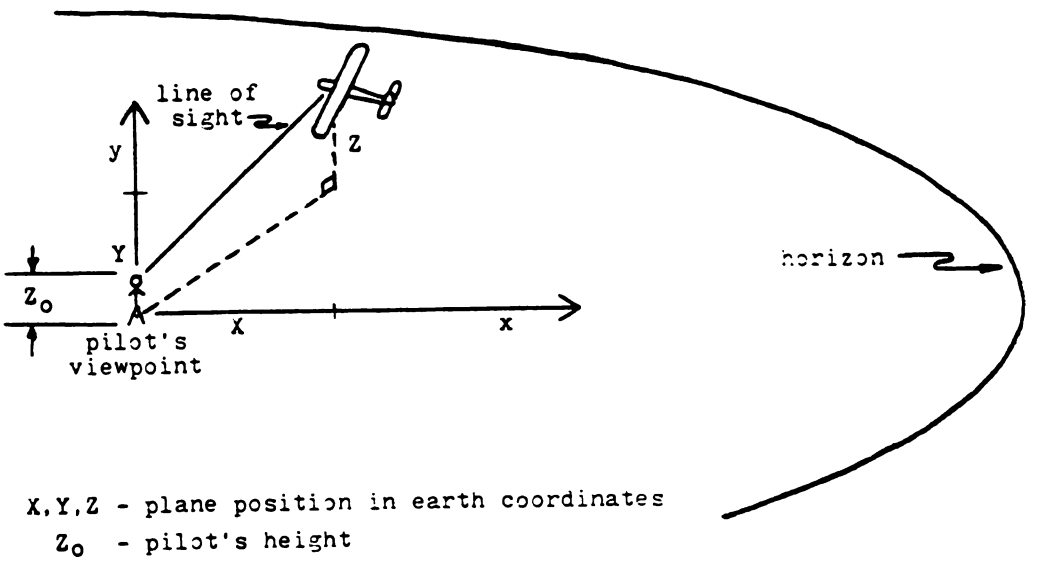


Figure 7: Definition of the remote-controlled pilot's line of sight

CHAPTER 4

Real-time Considerations

A stable set of differential equations, when solved numerically, can suffer from numerical instabilities and result in incorrect behavior. Therefore, it is necessary to discuss the magnitude of the time step required for the simulation's numerical technique. Assuming that the local truncation error and the error due to computer round-off are negligible, the important factor in numerical stability is the use of an appropiate time step.

It is necessary to evaluate the characteristic response of the aircraft when addressing this time step problem. An insight into the aircraft's response is gained by solving the eigenvalue problem for the

equations of motion. These equations which define the dynamic model of the aircraft are, in general, non-linear. Therefore, it is necessary to use linearizing techniques about an operating point on these equations to find the eigenvalues [11]. The eigenvalue results are as follows:

$$\begin{aligned} \lambda_1 &= -0.14 \\ \lambda_{2,3} &= -1.06 \pm 5.31\ j \\ \lambda_{4,5} &= -6.25 \pm 8.57\ j \\ \lambda_6 &= -26.46 \end{aligned}$$

The eigenvalue problem was evaluated at the steady-state flight operating point with small and large pertubation, as well as at an operating point other than steady-state flight. In each of the three cases, the eigenvalues remained approximately constant.

The characteristic response of the system described by the eigenvalues is related to the time step required for numerical stability in the integration of these equations. For a Runge-Kutta method the stability requirement on the time step is [5]:

$$\Delta t < 2.7/\max|\lambda_r| \qquad (22)$$

where Δt - time step
λ_r - real part of the eigenvalue

This requires the time step for the simulation to be:

$$\Delta t < .10 \text{ sec}$$

A time evaluation of the simulation is needed to determine whether the time step requirement can be met. The simulation process is timed

by dividing the task into three unique functions. Each of these functions is timed individually using dummy functions in place of the two processes not being analyzed. A breakdown of the time analysis is:

PRIME OPERATIONS --	simulation calculation	~ 0.23 sec
--	I/O with E-S	~ 0.42 sec
E-S OPERATIONS --	graphic display and I/O with PRIME	~ 0.47 sec
	TOTAL	1.12 sec

The simulation calculation includes the evaluation of the equations of motion, integration to velocity and coordinate transformation to earth-fixed axis. The PRIME input/output consists of the time used to send information to the E-S and the time spent reading the information sent to the screen from the E-S. The time used by the E-S includes the update of the graphics, as well as the collecting and sending of position and control information to the PRIME.

From this time evaluation of the simulator, it is seen that the time step required for the velocity integration cannot be met with the current simulation system. The problem lies in the architecture of the computer systems. The PRIME with its multiple-user configuration and large overhead operating system does not perform efficiently in real-time execution. A solution to the numerical instability problem follows one of two ideas: To reduce the overall time step, or to utilize an integration technique that will be stable at larger time steps.

The time step can be reduced by increasing the simulation's priority on the operating system. This will decrease the calculation time necessary for the PRIME. However, this time reduction will only result in a small decrease in the total time step (the PRIME calculation makes up 20 percent of the total time step).

By slightly modifying the simulator's architecture, more velocity integrations can be performed in the simulation loop. As shown in Figure 5, the PRIME waits for approximately 0.5 seconds for the information retrieval on the E-S. By allowing the PRIME to perform two more velocity integrations during this information retrieval, the critical time step is reduced by a factor of three. This idea is demonstrated in Figure 8.

The simulation can be slowed down so that the plane no longer flies at real time. This will give the effect of a plane with a heavily-damped response. The simulation pilot will still retain the control capacities of the aircraft. The major drawback is a limited sense of realism with the slowed time step.

The loop time was reduced to a satisfactory range by incorporating the above changes into the flight simulation. Perhaps a different integration technique could be employed to yield a stable integration process without the slow-motion factor. An in-depth real-time analysis/ numerical stability study would be required to optimize the order and time step of the integration.

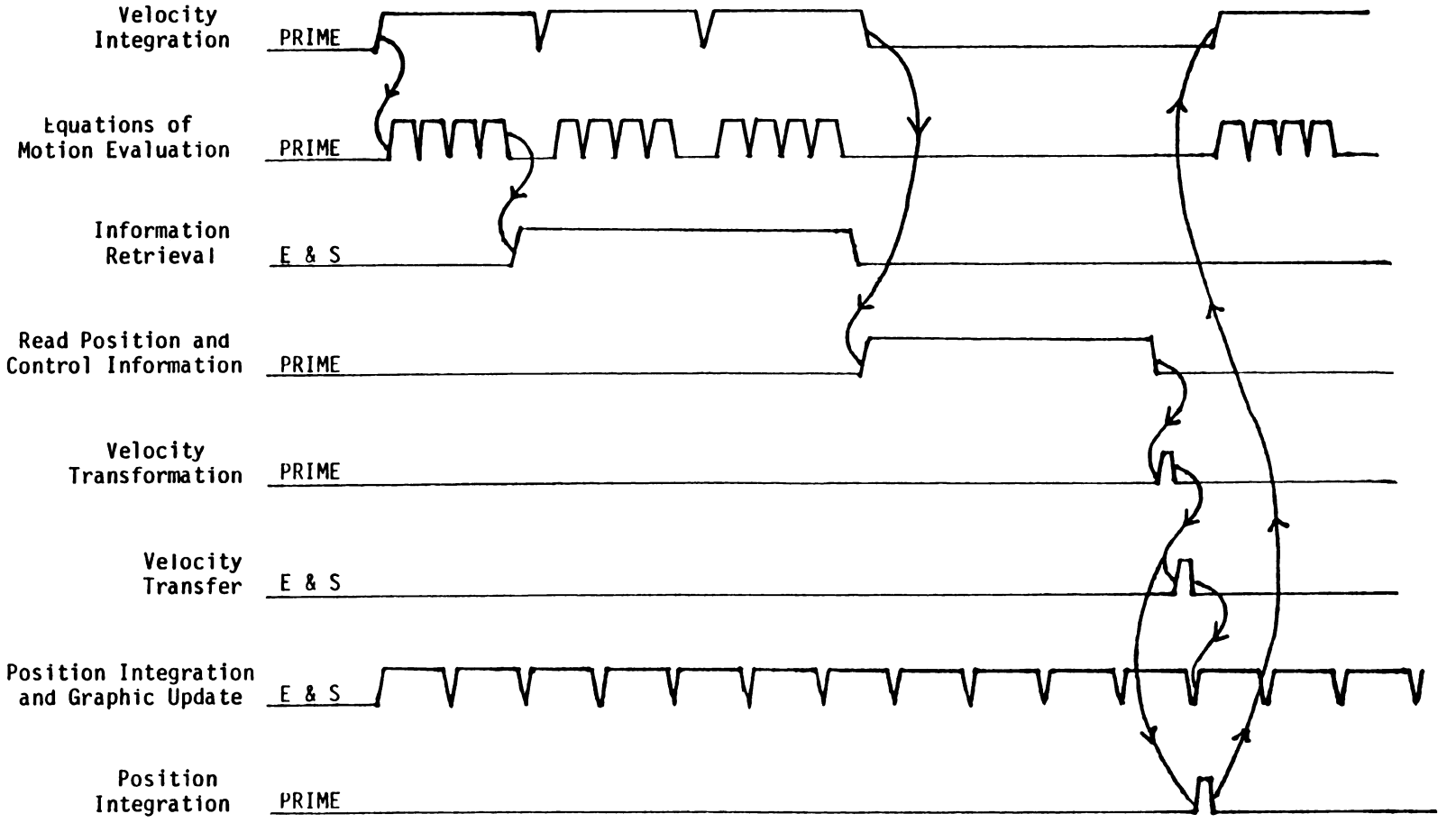


Figure 8: Revised timing and triggering mechanisms of simulation tasks

CHAPTER 5

Conclusions

The dynamics of an aircraft were modeled mathematically using the fundamentals of aerodynamics and mechanics. This model incorporated four main aircraft controls -- throttle, aileron, elevator and rudder -- to provide directional control.

A simulation algorithm which utilizes the dynamic model was developed to visualize the flight path from the point of view of a fixed ground observer. The current looping time of approximately 1.2 seconds, resulting in an unstable numerical evaluation of the equations of motion, prevented the simulation from operating at real time. This long looping time was attributed to the multiple-user configuration of the

PRIME and the communication link between the E-S and the PRIME.

An evaluation of the characteristic response of the equations of motion suggests that a looping time of 0.1 seconds is required for a stable integration using the present technique. The simulator was modified to utilize increased priority on the operating system, multiple integrations per simulation loop and a slower-than-real-time operation to correct for the large looping time difficulities.

Future work on refining the integration technique and/or the speed of communication between the two systems could increase the speed of response of the simulation. This would result in the real-time operation of the remote-controlled flight simulator.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Chow, Chuen-Yen, and Kuethe, Arnold. Foundations of Aerodynamics, 3rd ed., John Wiley and Son, Inc., 1973.

[2] Etkin, Bernard. Dynamics of Flight: Stablilty and Control, John Wiley and Son, Inc., 1959.

[3] Etkin, Bernard. Dynamics of Atmospheric Flight, John Wiley and Son, Inc., 1972.

[4] Foss, John, and Potter, Merle. Fluid Mechanics, John Wiley and Son, Inc., 1975.

[5] Fox, L., and Mayers, D.F. Computing Methods for Scientist and Engineers, Clarendon Press, 1968.

[6] Glauert, H. The Elements of Airfoil and Airscrew Theory, Cambridge Press, 1930.

[7] Kershner, William K. The Advanced Pilot's Flight Manual, 3rd ed., Iowa State University Press, 1970.

[8] Kolk, W. Richard. Modern Flight Dynamics, Prentice-Hall, 1961.

[9] Laitone, E.V. "ME 164: Engineering Aerodynamics Class Notes," University of California - Berkley, 1976.

[10] Miele, Angelo. Flight Mechanics - Theory of Flight Paths, Addison-Wesley Publishing Co., 1962.

[11] Ogata, Katsuhiko. Modern Control Engineering, Prentice-Hall Inc., 1970.

[12] Roskam, Jan. Airplane Flight Dynamics and Automatic Flight Control, Roskam Aviation and Engineering Co., 1979.

[13] Michigan State University, College of Engineering, Case Center User's Guide, Michigan State University Press, 1984.

APPENDICES

APPENDIX A

Coordinate Transformations

Coordinate transformations play an important role in the development of a flight simulator. There are three instances when a coordinate transformation occurs. They are 1) to align the relative wind with the force element axes, 2) to transform the linear velocities from body coordinates to earth coordinates, and 3) to transform the angular velocities from body coordinates to earth coordinates. The first two situations are actually the same transformation using different angles of rotation. For the relative wind transformation, the rotation angles are defined by the dihedral, washout and sweep of the airfoil. Once the plane is defined, this transformation remains the same. For the second case the rotation angles are defined by the roll,

pitch and yaw angles of the plane relative to the earth. These angles are always changing during the plane's flight. This type of a coordinate transformation requires an ordered rotation about 3 axes [2]. The initial orientation is $Cx_1y_1z_1$ and it is desired to end in Cxyz. To perform this transformation the following rotations are applied.

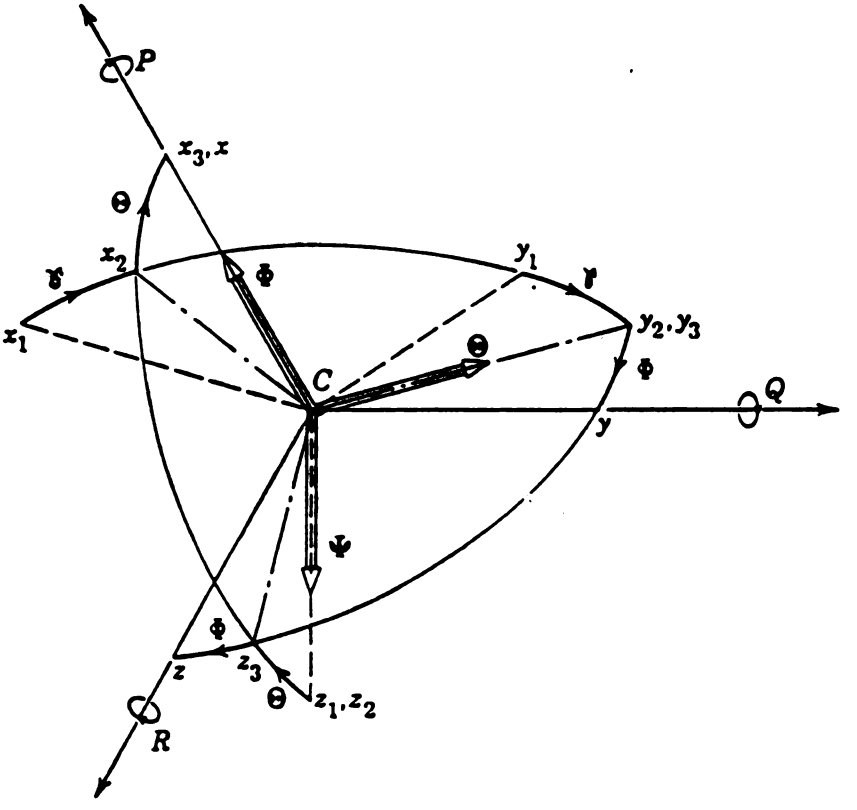


Figure A1: Axis systems used for coordinate transformations

1) a rotation γ about Oz_1 carrying the axes to $Cx_2y_2z_2$
2) a rotation θ about Oy_2 carrying the axes to $Cx_3y_3z_3$
3) a rotation ϕ about Ox_3 carrying yhe axes to Cxyz

In matrix form this transformation is as follows:

$$\begin{bmatrix} U' \\ V' \\ W' \end{bmatrix} = \begin{bmatrix} T(1,1) & T(1,2) & T(1,3) \\ T(2,1) & T(2,2) & T(2,3) \\ T(3,1) & T(3,2) & T(3,3) \end{bmatrix} \begin{bmatrix} U \\ V \\ W \end{bmatrix} \qquad \text{(A1)}$$

where
$$T(1,1) = \cos(\theta)\cdot\cos(\gamma)$$
$$T(2,1) = \cos(\theta)\cdot\sin(\gamma)$$
$$T(3,1) = -\sin(\gamma)$$
$$T(1,2) = \sin(\phi)\cdot\sin(\theta)\cdot\cos(\gamma) - \cos(\phi)\cdot\sin(\gamma)$$
$$T(2,2) = \sin(\phi)\cdot\sin(\theta)\cdot\sin(\gamma) + \cos(\phi)\cdot\cos(\gamma)$$
$$T(3,2) = \sin(\phi)\cdot\cos(\theta)$$
$$T(1,3) = \cos(\phi)\cdot\sin(\theta)\cdot\cos(\gamma) + \sin(\phi)\cdot\sin(\gamma)$$
$$T(2,3) = \cos(\phi)\cdot\sin(\theta)\cdot\sin(\gamma) - \sin(\phi)\cdot\cos(\gamma)$$
$$T(3,3) = \cos(\phi)\cdot\cos(\theta)$$

For the relative wind calculation:

ϕ = dihedral angle
θ = washout angle
γ = sweep angle

For the linear velocity transformation the Euler angle would be used:

ϕ = roll angle
θ = pitch angle
γ = yaw angle

For the rotational velocity transformation the angular velocity components (P,Q,R) of the plane must be represented in terms of the Euler angles (ϕ,θ,γ) and their derivatives $(\dot{\phi},\dot{\theta},\dot{\gamma})$ [2]. Let (i,j,k) be unit vectors with subscripts 1,2,3 denoting direction. The angular velocity can be defined as

$$\omega = i\dot{\phi} + j_2\dot{\theta} + k_2\dot{\theta} \qquad \text{(A2)}$$

By projecting this onto Cxyz and remembering that

$$\omega = iP + jQ + kR \tag{A3}$$

The angular velocities are defined:

$$P = \dot{\phi} - \dot{\gamma}\cdot SIN(\theta) \tag{A4}$$
$$Q = \dot{\theta}\cdot COS(\phi) + \dot{\gamma}\cdot COS(\theta)\cdot SIN(\phi) \tag{A5}$$
$$R = \dot{\gamma}\cdot COS(\theta)\cdot COS(\phi) - \dot{\theta}\cdot SIN(\phi) \tag{A6}$$

Solving the above equations for the derivatives of the Euler angles yields:

$$\dot{\phi} = P + Q\cdot SIN(\phi)\cdot TAN(\theta) + R\cdot COS(\phi)\cdot TAN(\theta) \tag{A7}$$
$$\dot{\theta} = Q\cdot COS(\phi) - R\cdot SIN(\phi) \tag{A8}$$
$$\dot{\gamma} = (Q\cdot SIN(\phi) + R\cdot COS(\phi))/COS(\theta) \tag{A9}$$

In matrix form this transformation is as follows:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} 1 & SIN(\phi)\cdot TAN(\theta) & COS(\phi)\cdot TAN(\theta) \\ 0 & COS(\phi) & -SIN(\phi) \\ 0 & SIN(\phi)/COS(\theta) & COS(\phi)/COS(\theta) \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \tag{A10}$$

APPENDIX B

Force Generation Tables

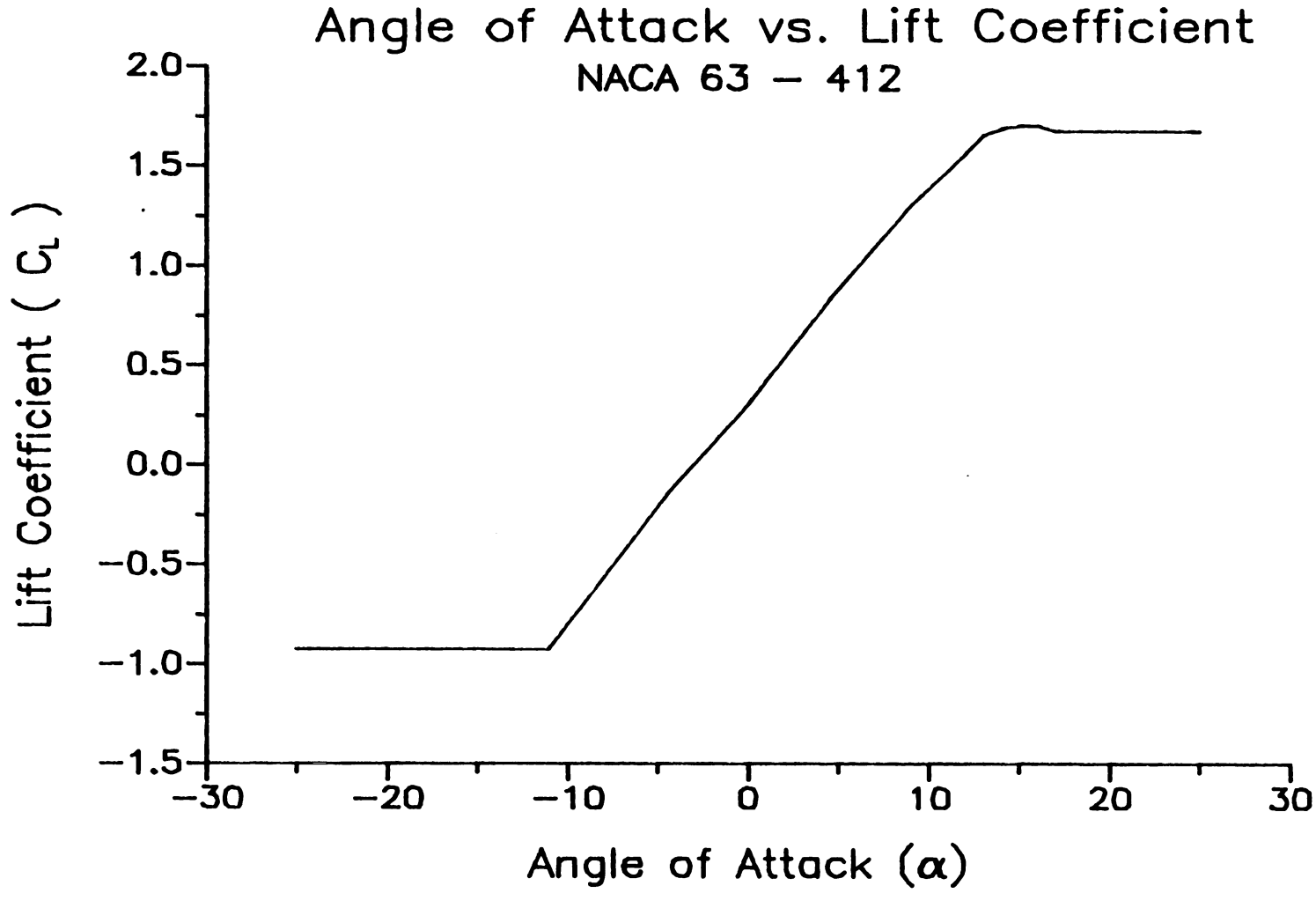


Figure B1. Angle of Attack vs. Lift Coefficient Curve

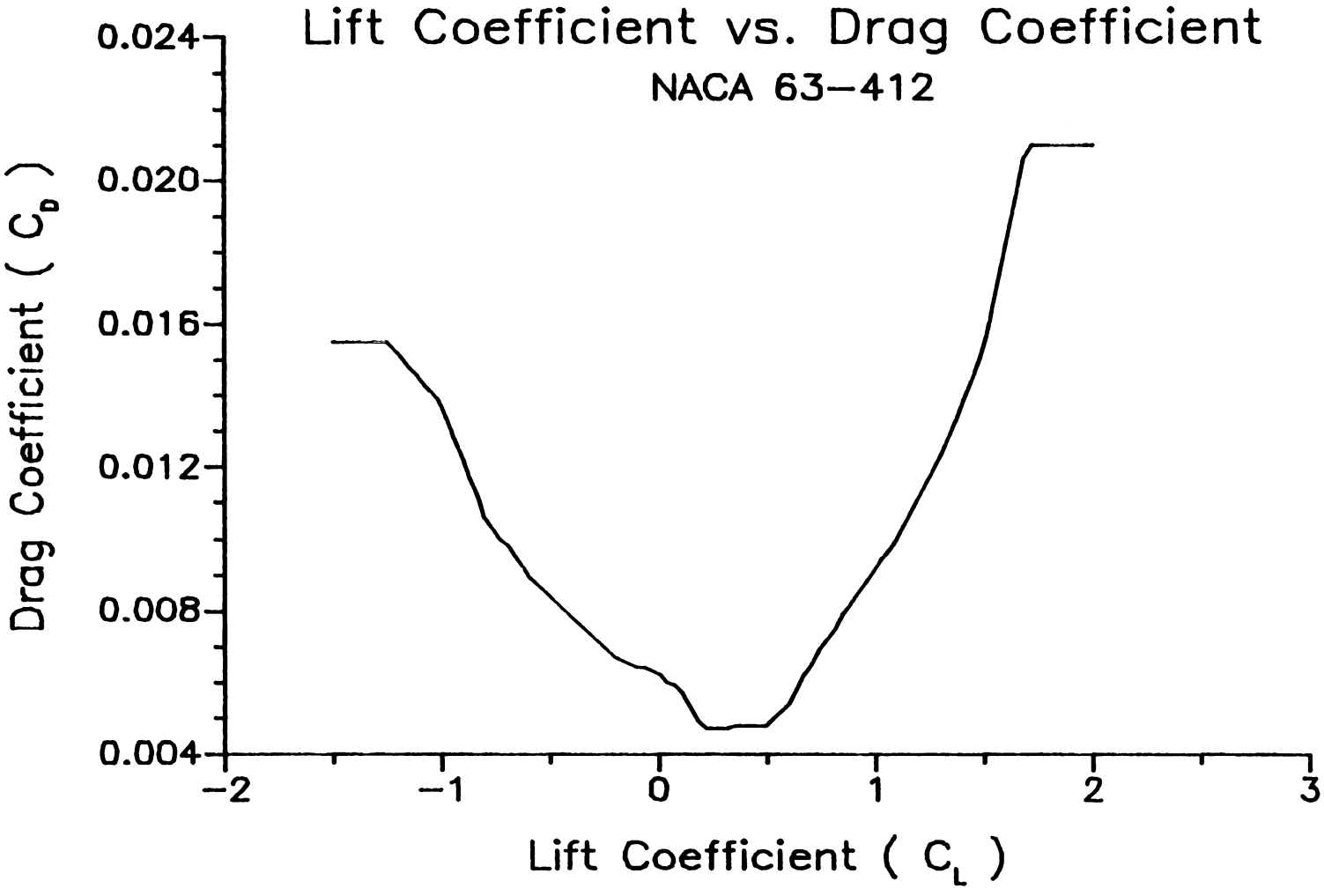


Figure B2. Lift Coefficient vs. Drag Coefficient Curve

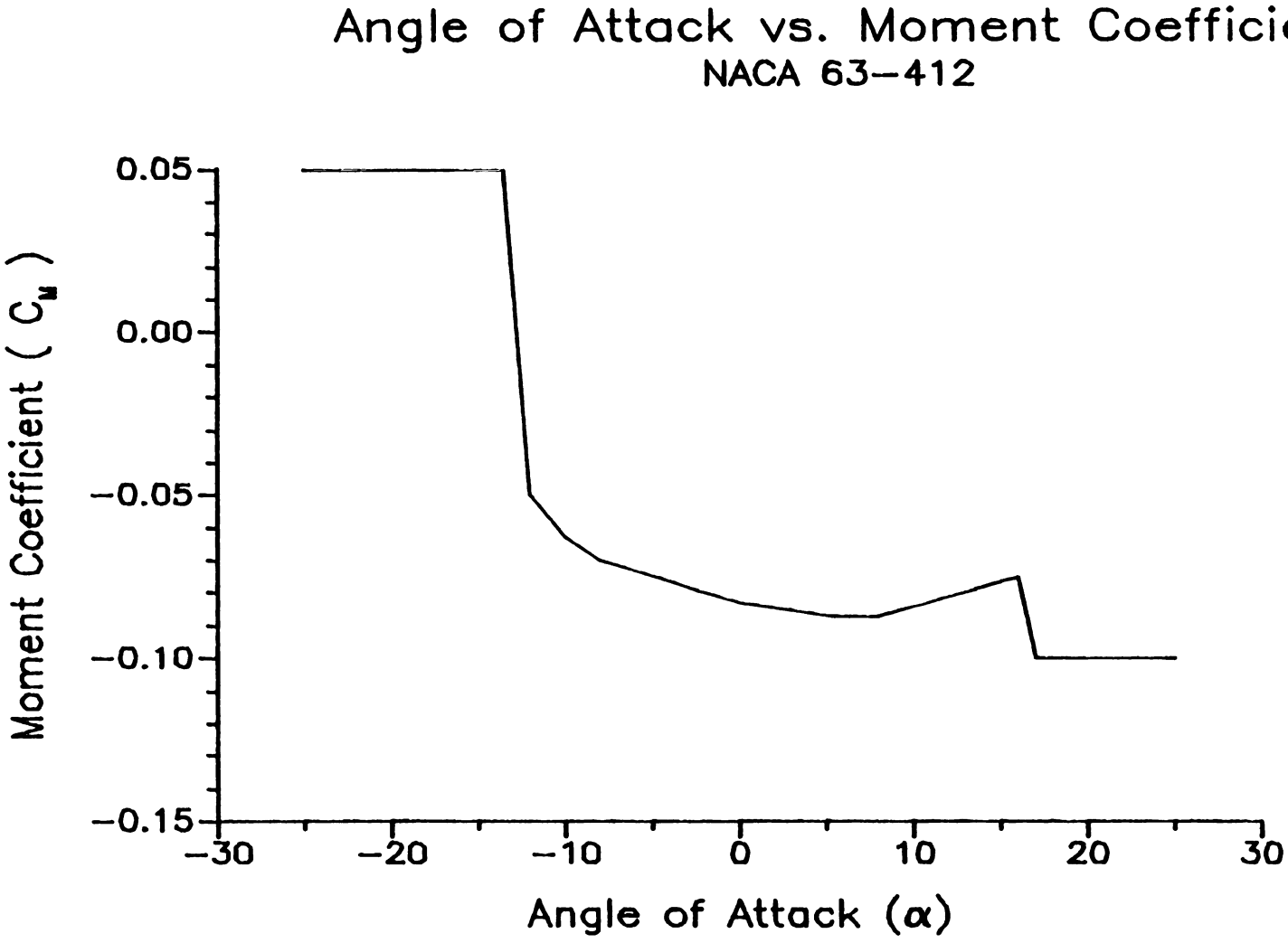


Figure B3. Angle of Attack vs. Moment Coefficient Curve

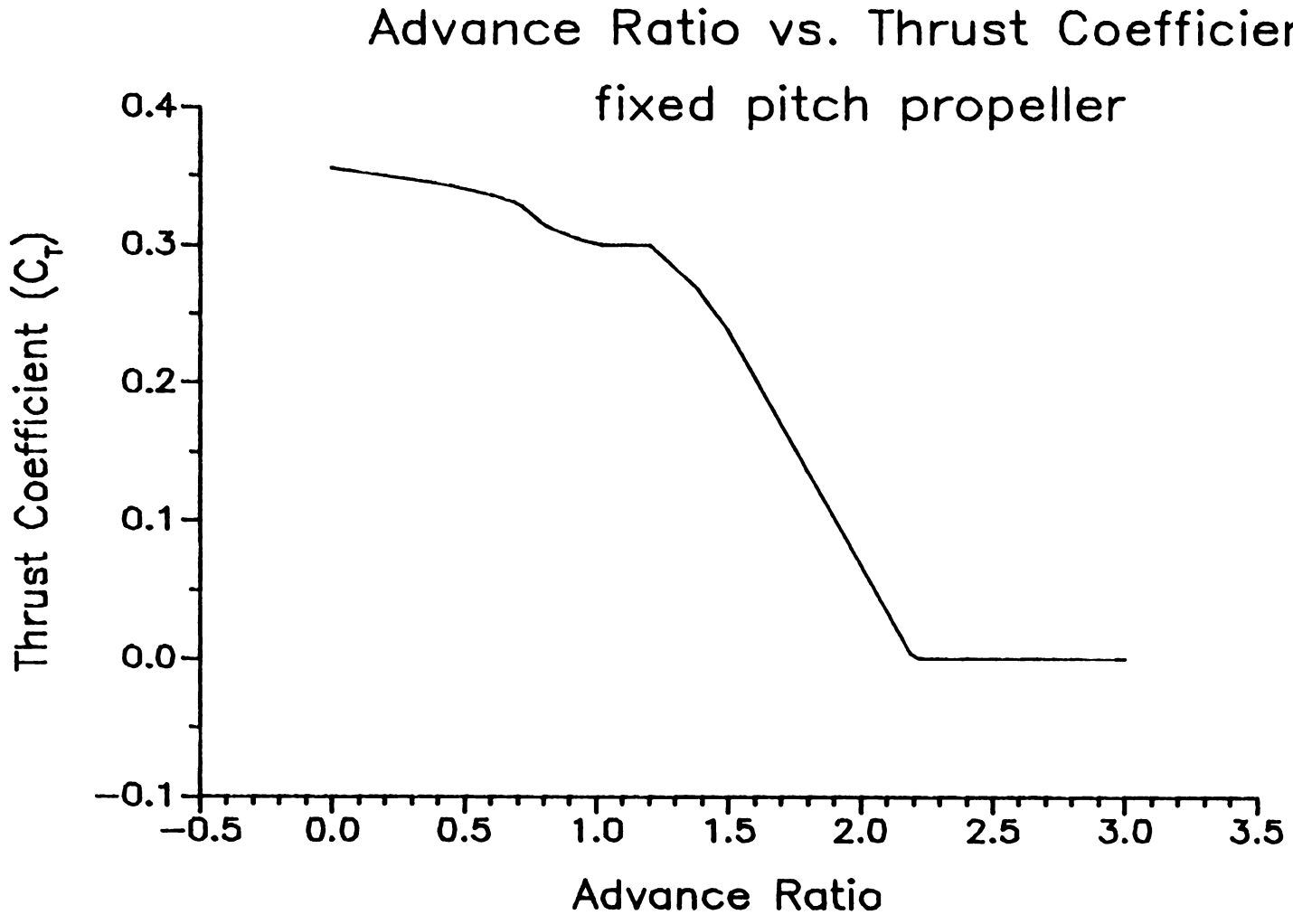


Figure B4. Advance Ratio vs. Thrust Coefficient Curve

APPENDIX C

Aircraft Layout

A complete data set of the necessary information of a specific light aircraft could not be found. Therefore, the parameters which define the simulation aircraft come from a variety of sources [6][7][9][12]. All of the data collected were typical of a light commerical plane. The following table presents the important parameters of the plane. Figure C1 gives the dimensions to the line of action for the various force generating elements. The line of action for the powerplant lies along the longitudinal axis (x axis) of the plane.

Inertial Parameters

Weight	2645	lbs.
I_{xx}	948	slugs·ft^2
I_{yy}	1346	slugs·ft^2
I_{zz}	1967	slugs·ft^2
I_{xz}	0	slugs·ft^2

Airfoil Section Parameters

Main Wing (inner)		Elevator	
span	12.0 ft	span	6.0 ft
chord	5.30 ft	chord	3.50 ft
dihedral	3.00°	dihedral	0.00°
washout	1.25°	washout	0.00°
sweep	0.00°	sweep	0.00°

Main Wing (outer)		Rudder	
span	5.90 ft	span	4.00 ft
chord	4.20 ft	chord	3.00 ft
dihedral	3.00°	dihedral	0.00°
washout	0.00°	washout	0.00°
sweep	0.00°	sweep	0.00°

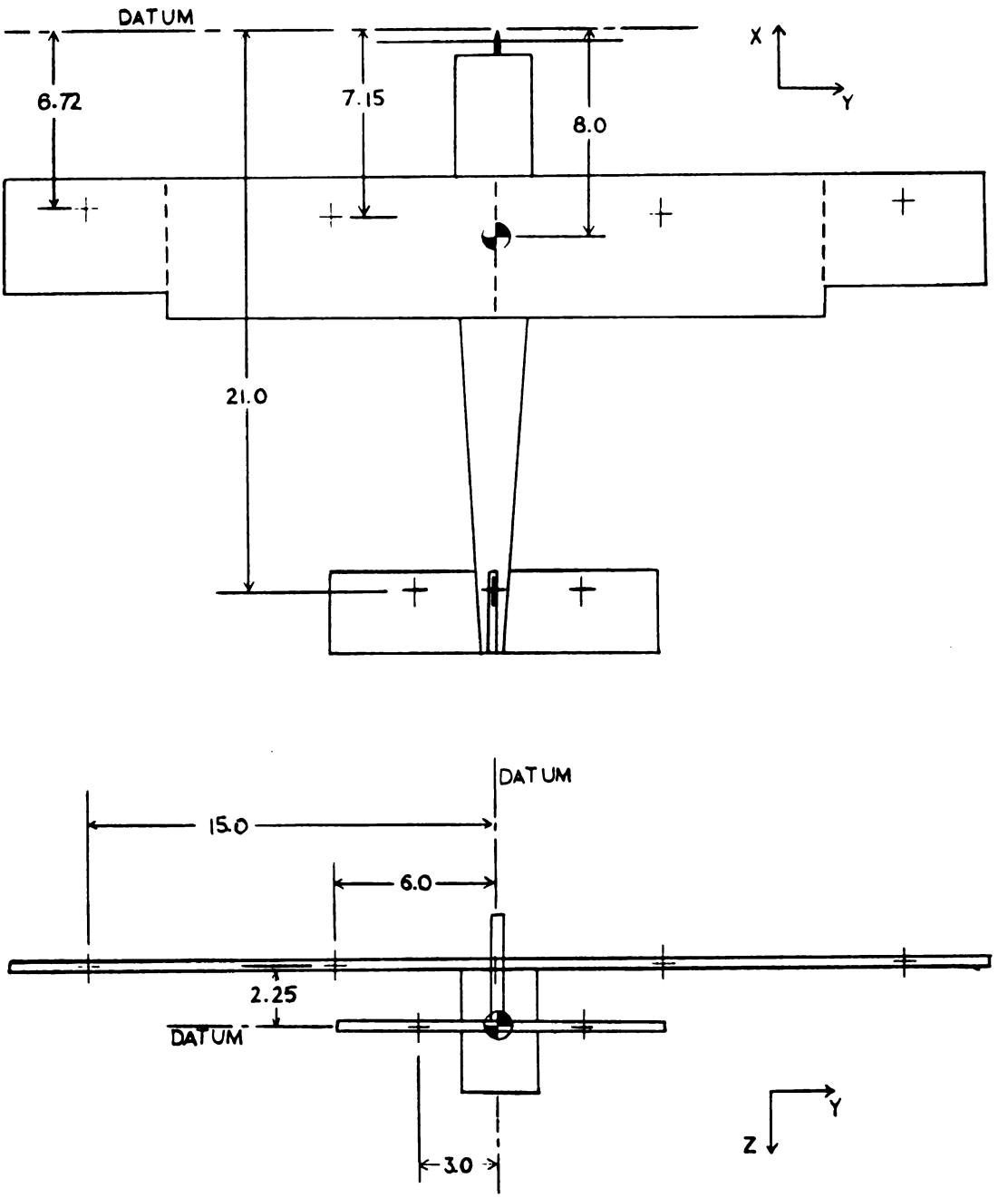


Figure C1. Dimensions to the line of actions for a light aircraft

APPENDIX D

DIFFEQ Verification

The purpose of using DIFFEQ is to show that the equations of motion derived do, in fact, characterize the dynamics of an aircraft. Using a simulation package will remove the possibility of numerical instabilities that can occur in a real-time simulation. To check the equations of motion, four different flight conditions were used -- steady-state flight, elevator-controlled flight, aileron-controlled flight, and rudder-controlled flight. These four conditions give an insight into the dynamic stability of the aircraft as well as the performance of the controls. Because the controls are implemented numerically without any feedback, it is difficult to perform complex controlled maneuvers with the DIFFEQ simulation. The graphic results

and a brief description of each flight condition follows.

Steady-State Flight

In this flight condition the plane was started with a longitudinal velocity slightly higher than steady-state, and the controls were trimmed for steady-state flight. The plane pitched upwards and gained altitude to lose speed. It passed through equilibrium, showed a slight stall, and pitched downward to gain back speed. The plane repeated this motion until the pertubations were damped out. The plane continued at its new equilibrium position in steady flight. As expected, there is no coupling of the longitudinal or vertical motion into the lateral motion.

Elevator-Controlled Flight

In this flight situation the aircraft was trimmed for steady-state flight, then an asymptotic upward elevator deflection was input. The plane started pitching upward and climbing. A transient oscillation was observed due to the control input. This died out and the plane continued its climb at a new angle of attack. The plane was observed to perform a gravity stall when the elevator input was maintained and the angle of attack of the plane increased past a critical point. No coupling of longitudinal and lateral motion was observed during this symmetric maneuver.

Aileron-Controlled Flight

The plane was initialized to steady-state flight. The ailerons were applied in an asymptotic manner. The plane began to roll, and lose altitude due to resultant vertical lift losses. A resultant lateral lift gain accompanied the bank, and the plane began to yaw. After a period of time, the aircraft had turned through 180 degrees. In this flight condition a coupling between the lateral and longitudinal motions was seen. The roll initiated a yaw motion as well as a pitch motion and a loss of altitude.

Rudder-Controlled Flight

The plane was started at steady-state flight. The rudder was applied in an asymptotic manner. This resulted in a yaw velocity, and the plane began to turn. The plane experienced a roll and pitch due the cross coupling of the lateral motion into the longitudinal motion. The plane continued to turn through 180 degrees. From this examination, it is seen that the motion produced by the rudder is very similar to that produced by the aileron.

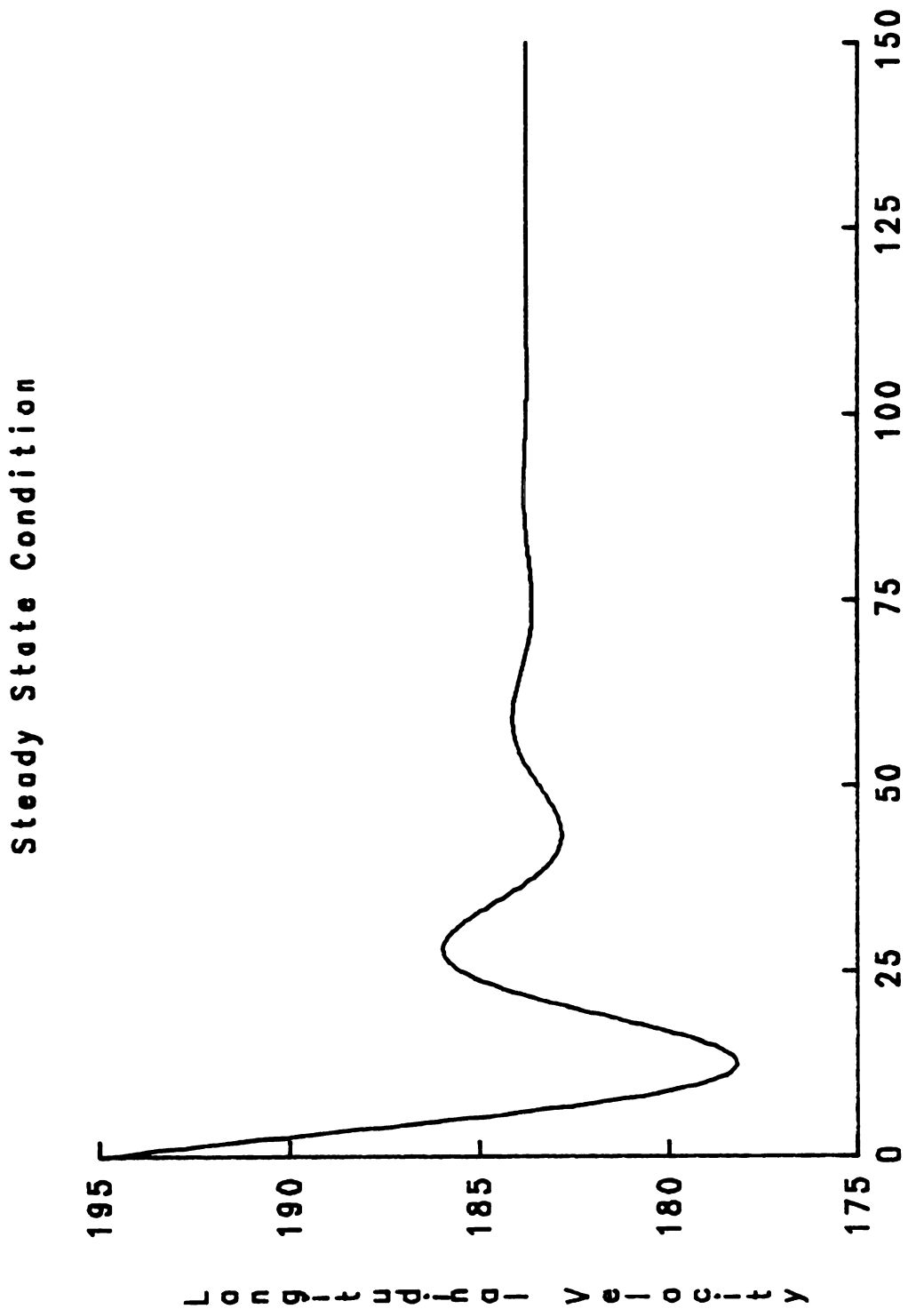


Figure D1a. DIFFEQ time response verification -- steady-state check

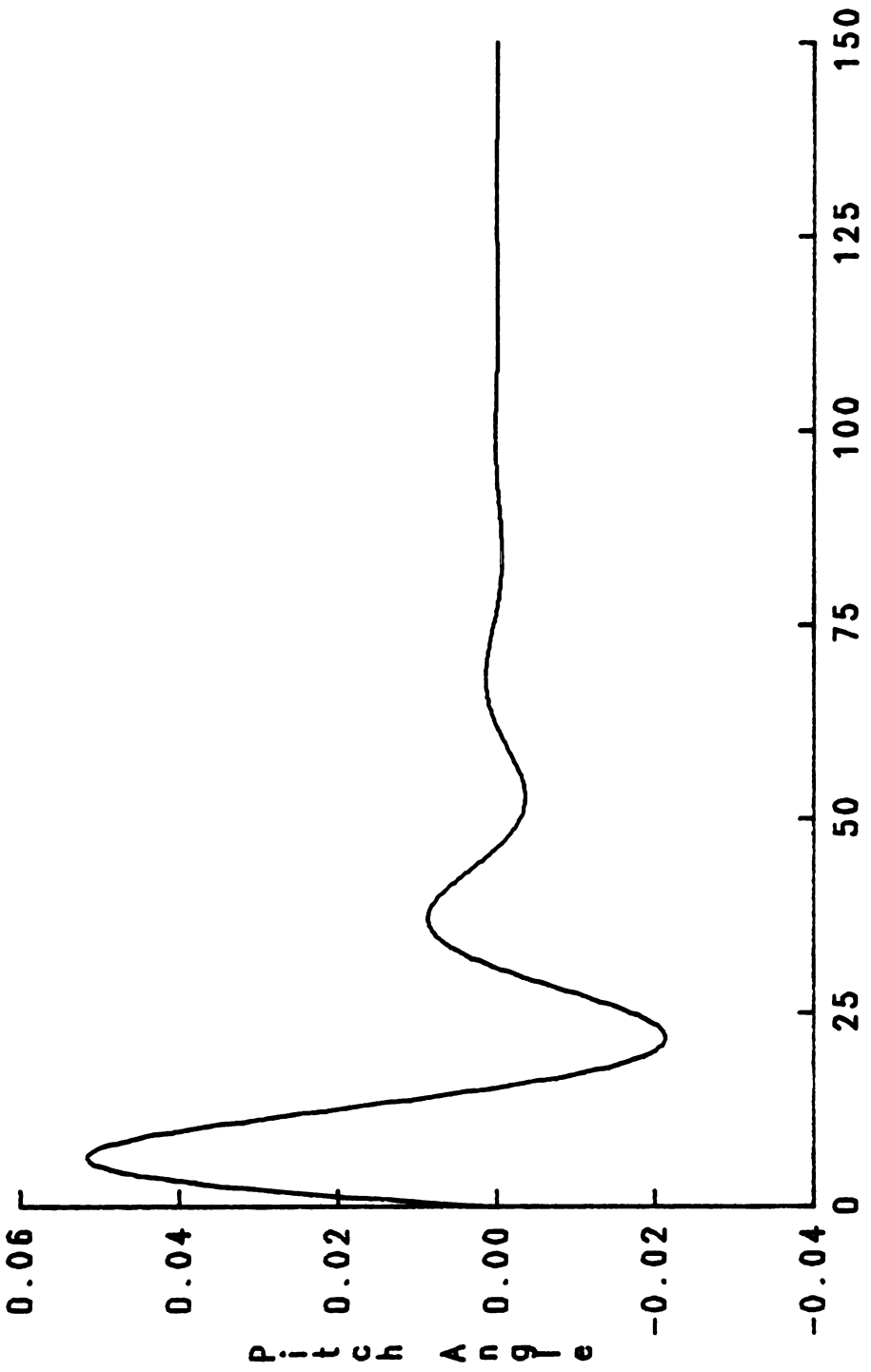


Figure D1b. DIFFEQ time response verification -- steady-state check

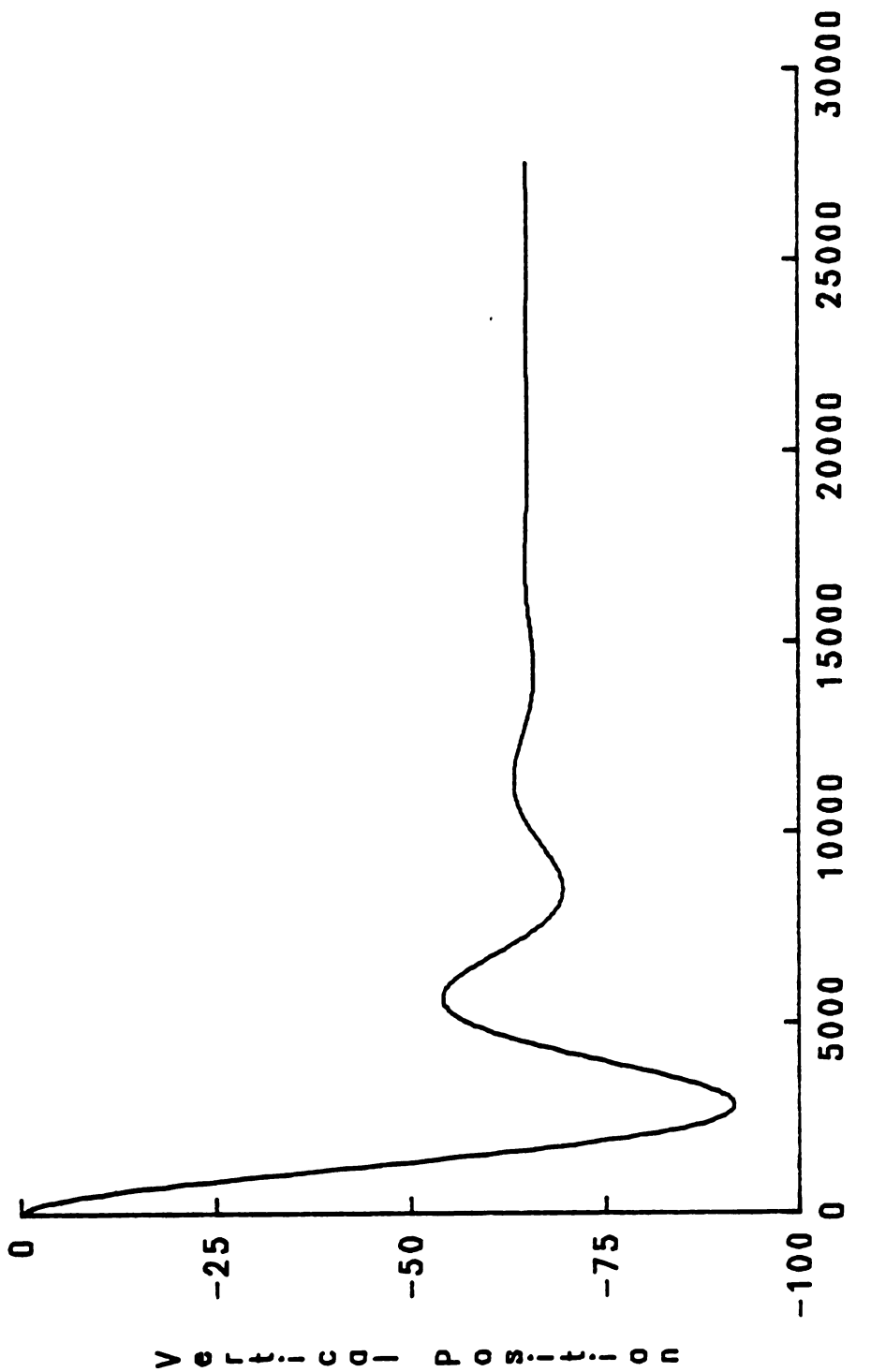


Figure D1c. DIFFEQ time response verification -- steady-state check

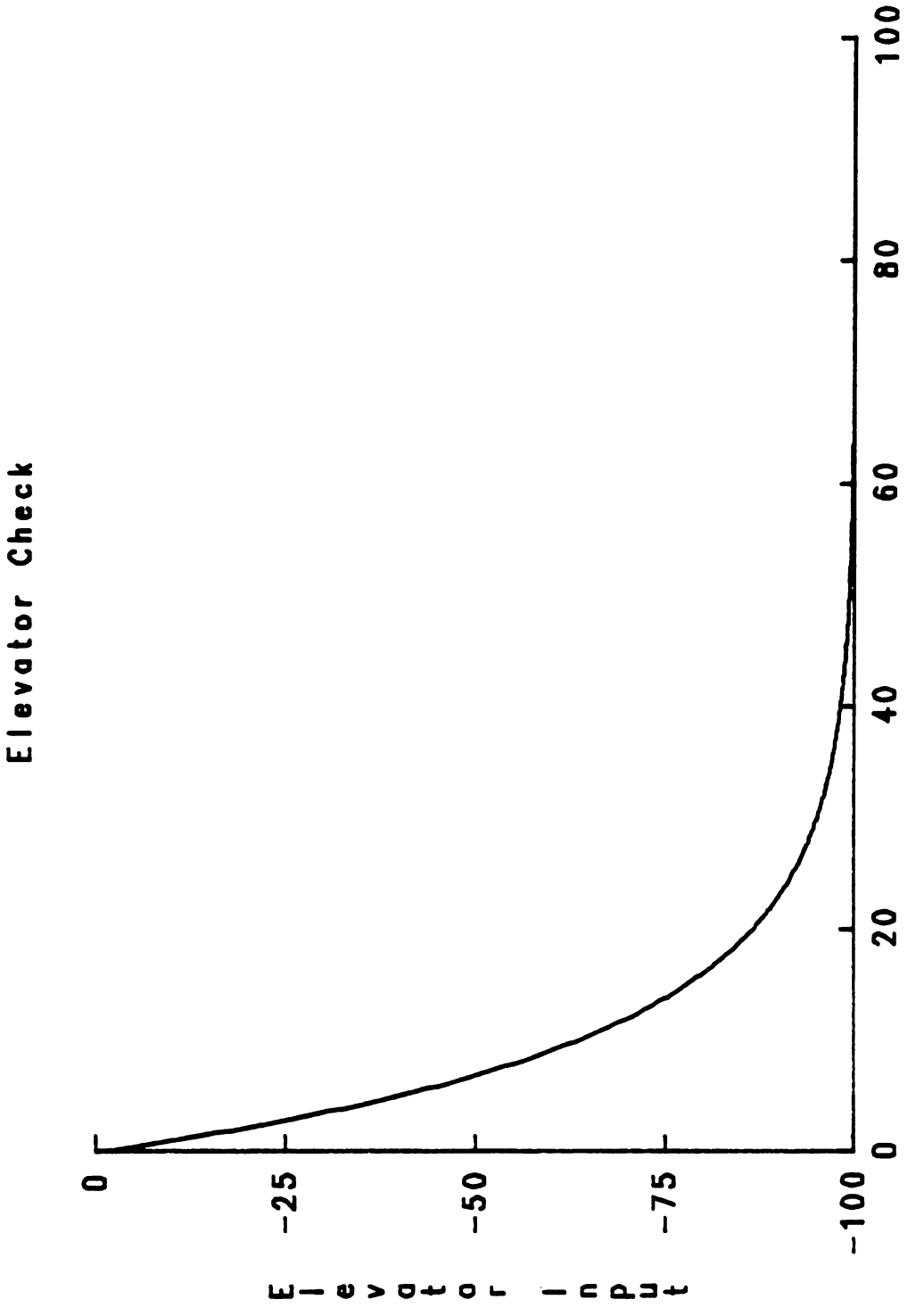


Figure D2a. DIFFEQ time response verification -- elevator check

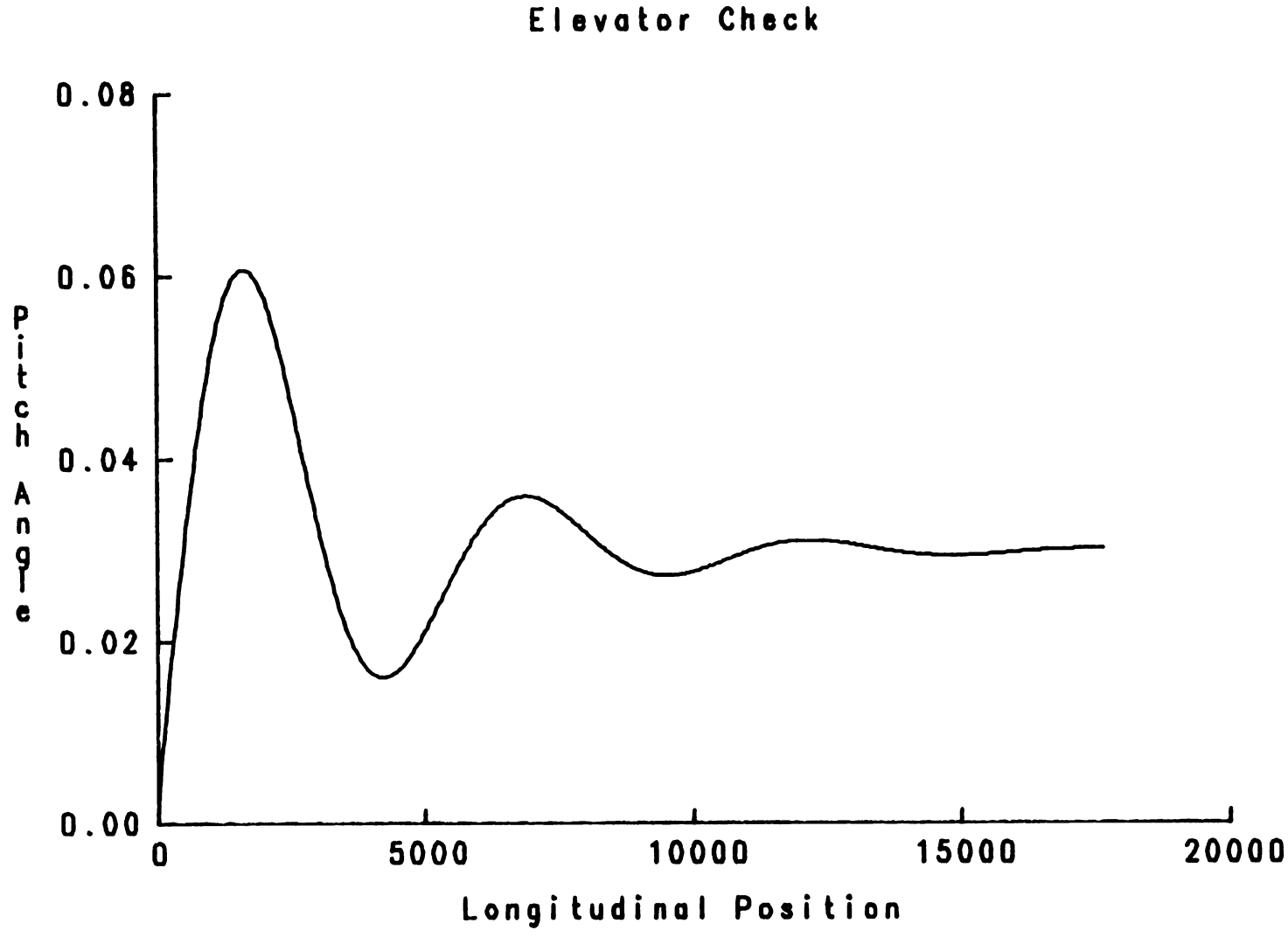


Figure D2b. DIFFEQ time response verification -- elevator check

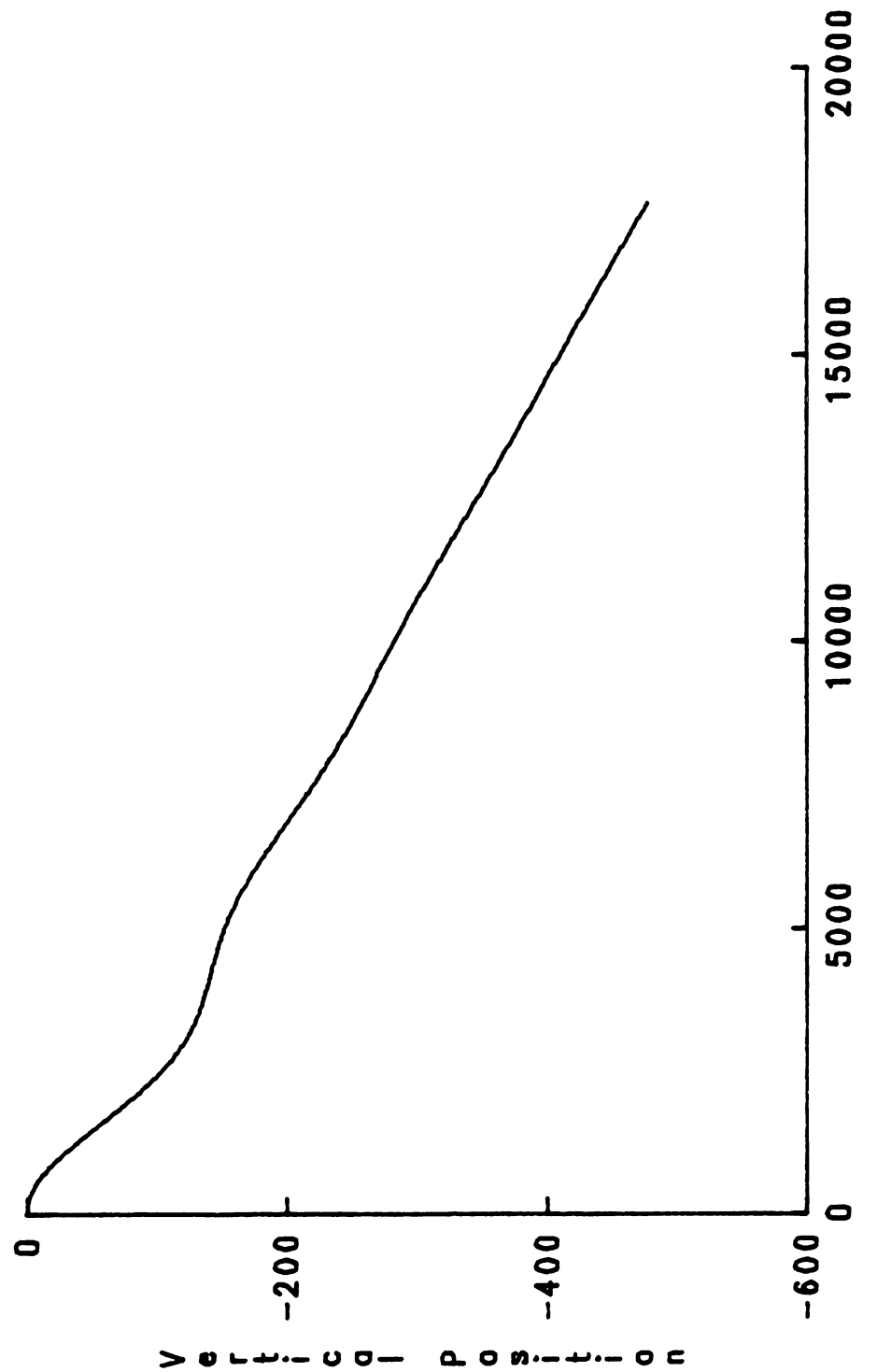


Figure D2c. DIFFEQ time response verification -- elevator check

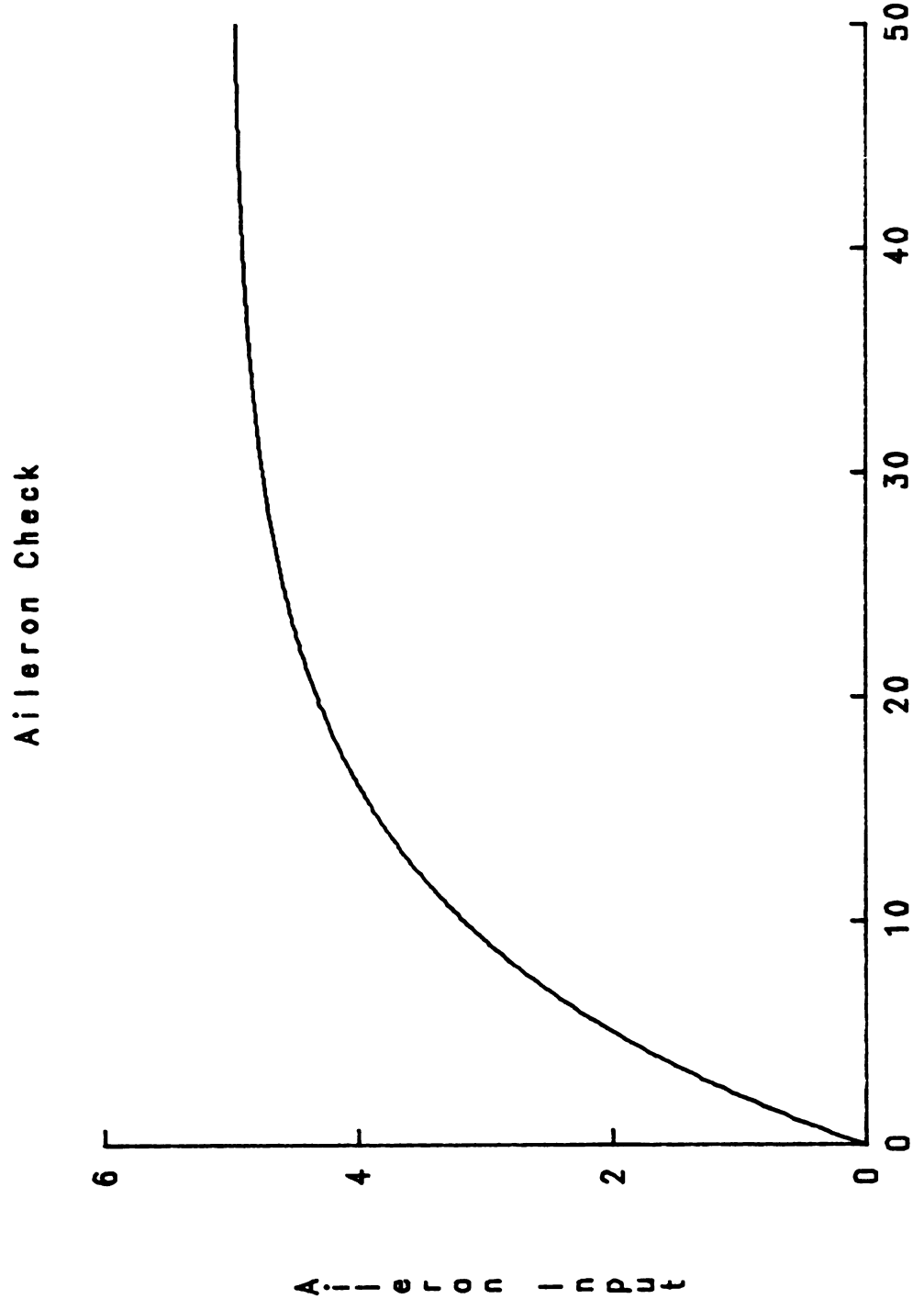


Figure D3a. DIFFEQ time response verification -- aileron check

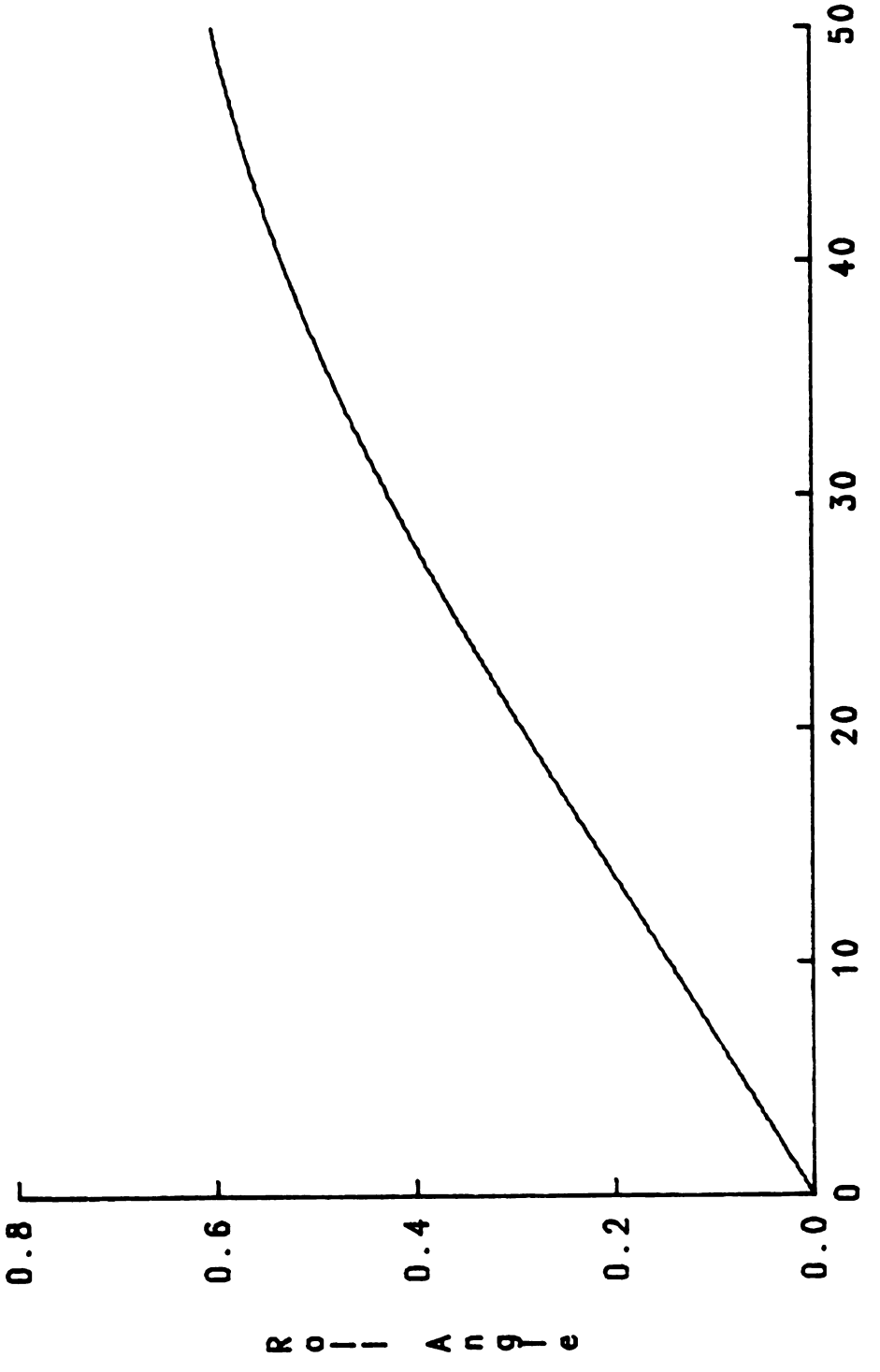


Figure D3b. DIFFEQ time response verification -- aileron check

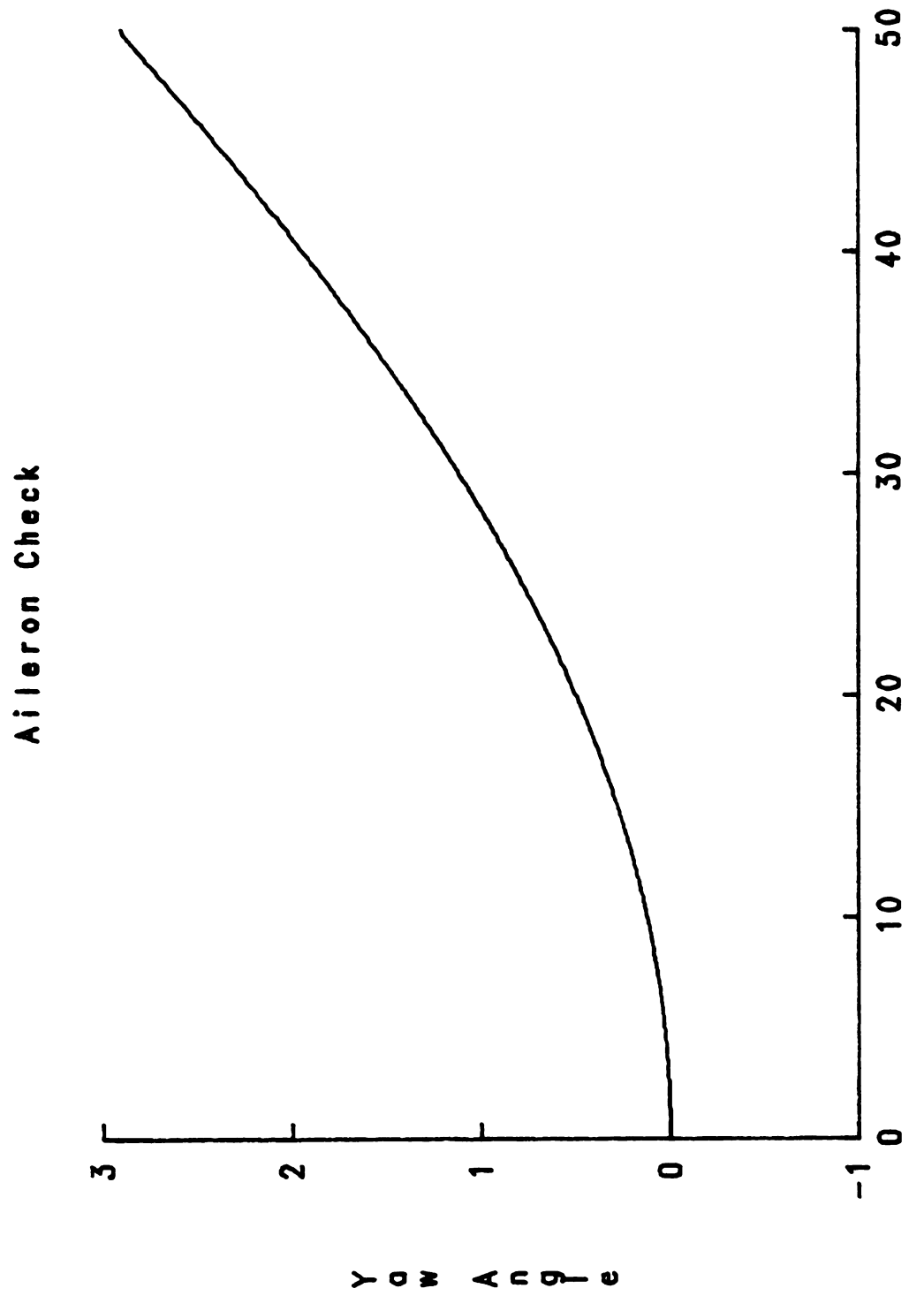


Figure D3c. DIFFEQ time response verification -- aileron check

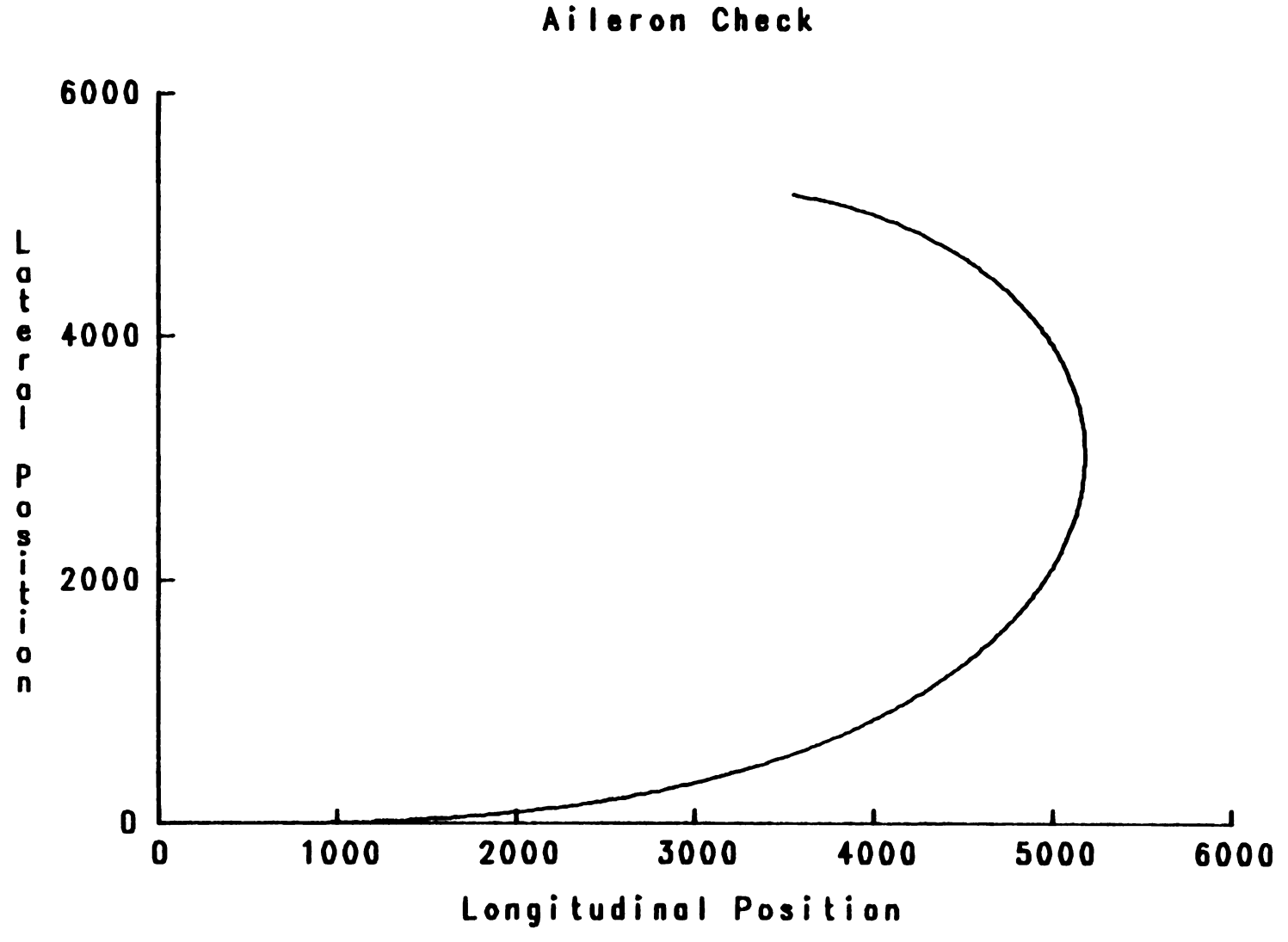


Figure D3d. DIFFEQ time response verification -- aileron check

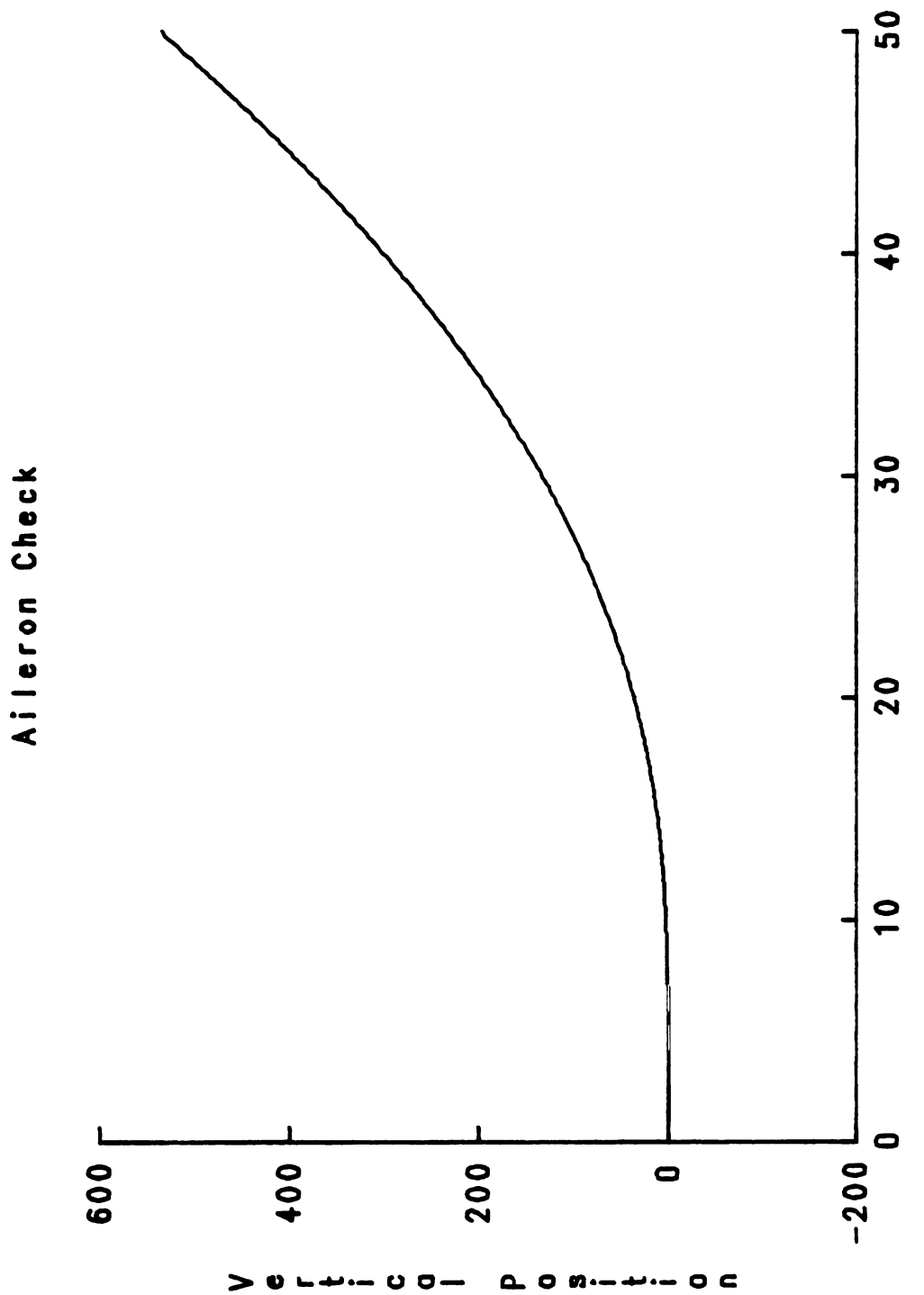


Figure D3e. DIFFEQ time response verification -- aileron check

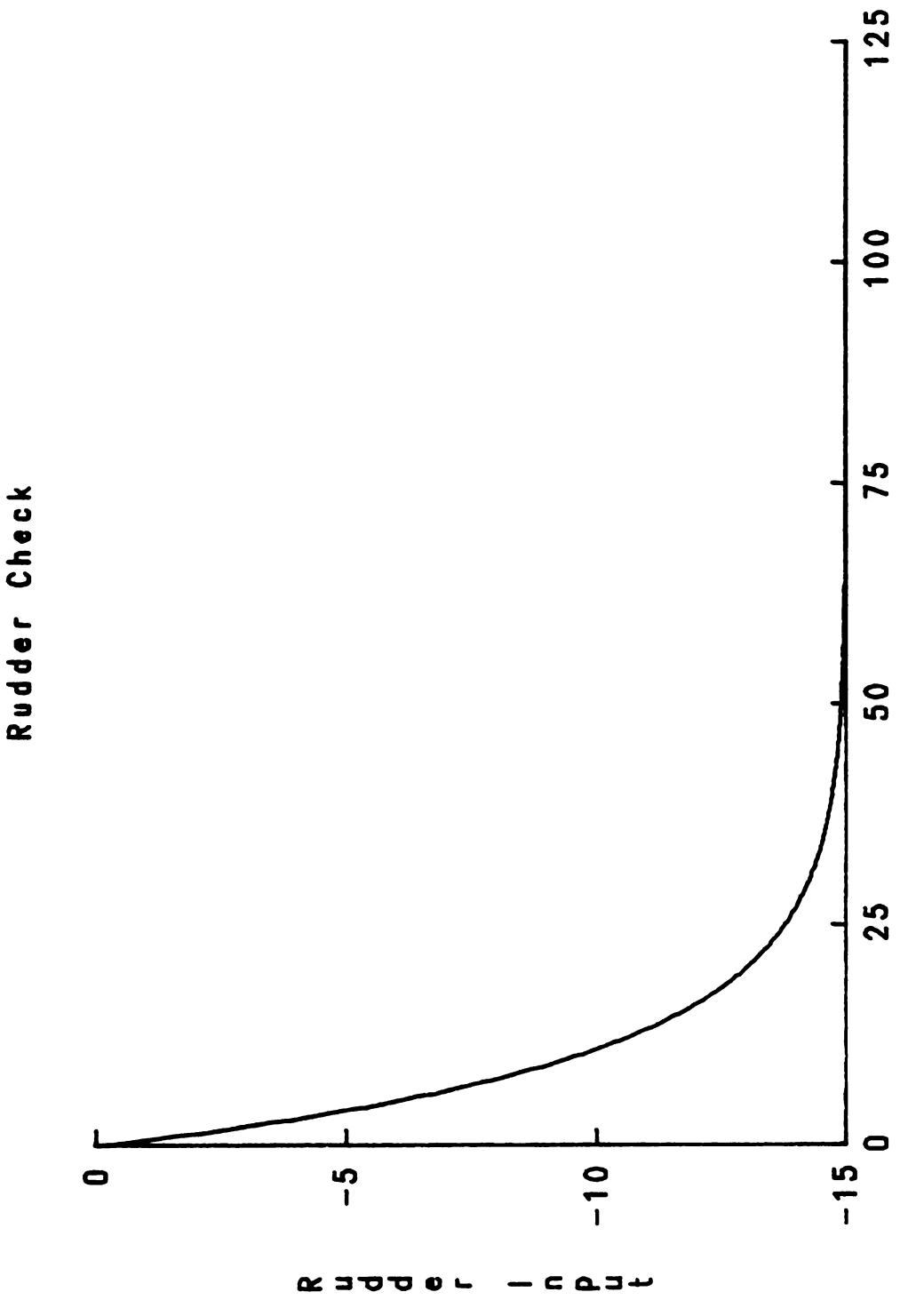


Figure D4a. DIFFEQ time response verification -- rudder check

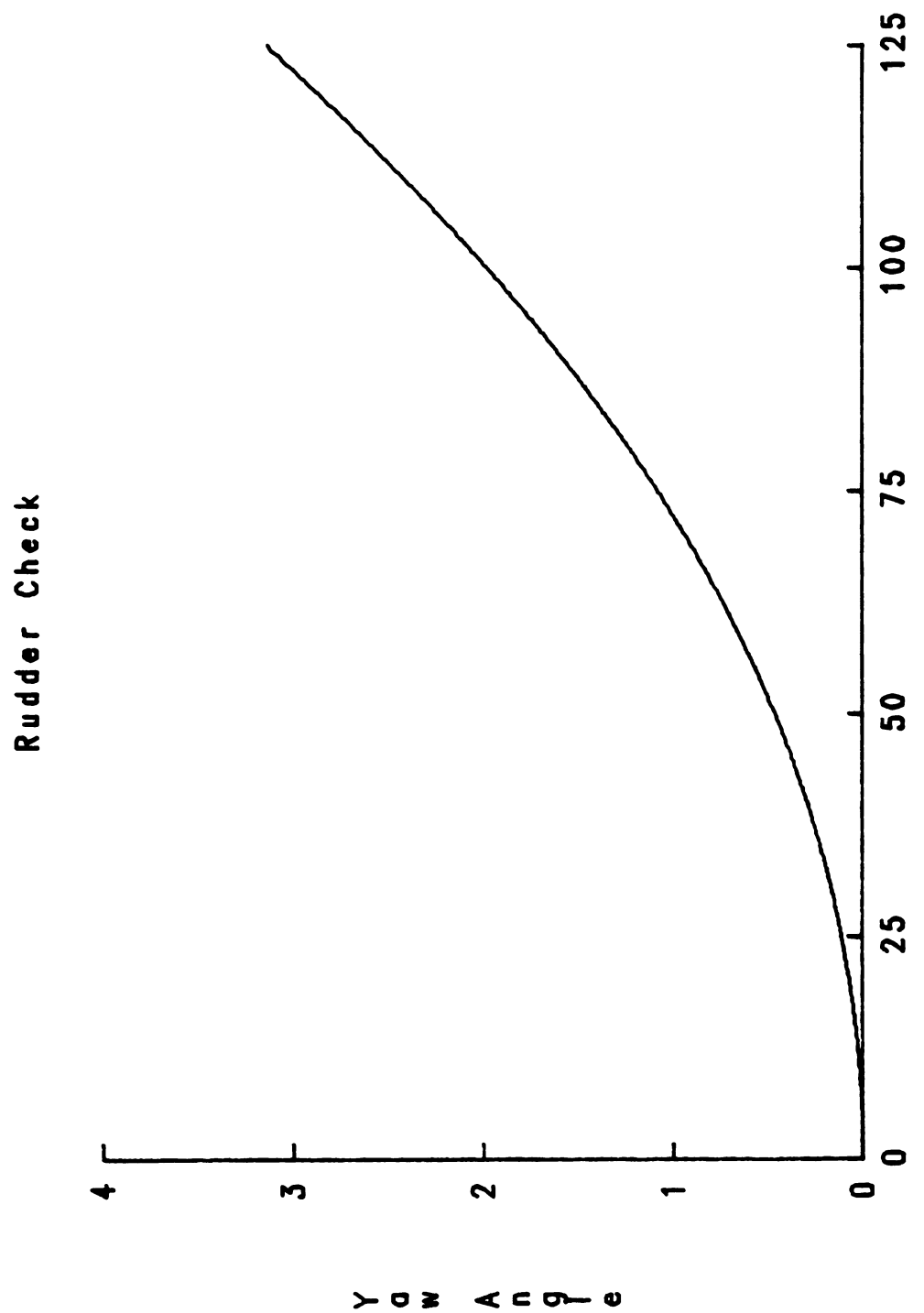


Figure D4b. DIFFEQ time response verification -- rudder check

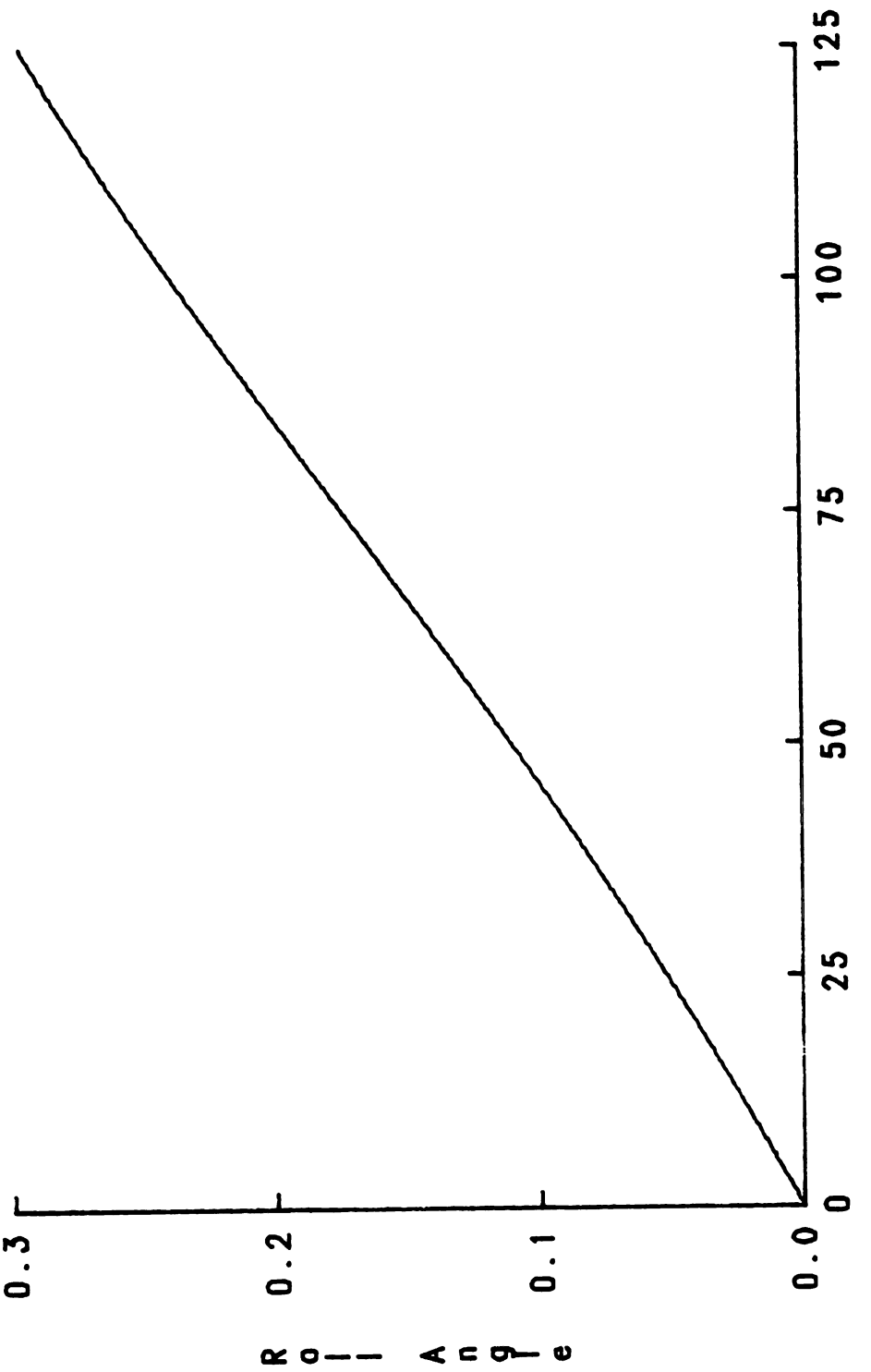


Figure D4c. DIFFEQ time response verification -- rudder check

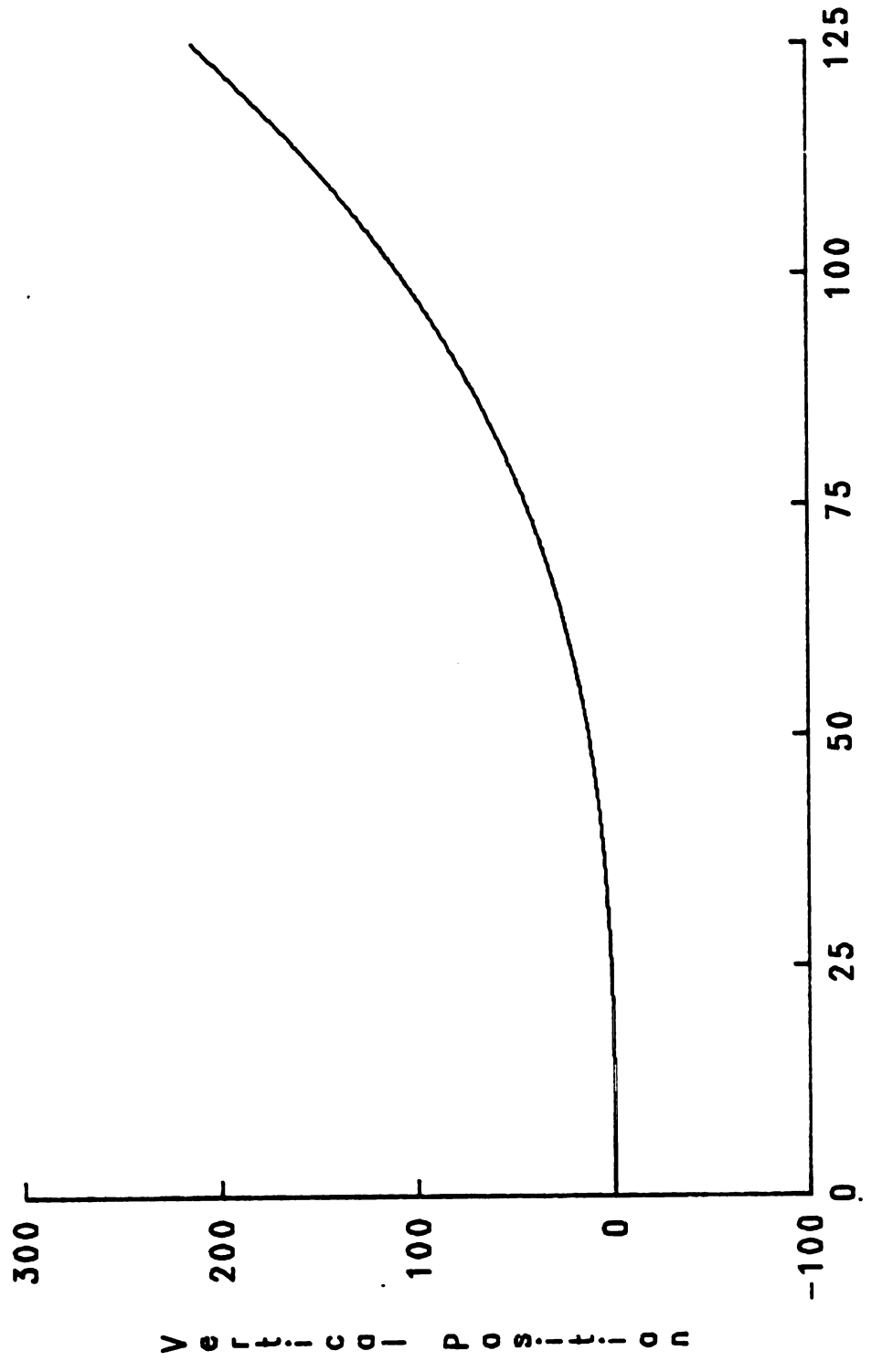


Figure D4d. DIFFEQ time response verification -- rudder check

APPENDIX E

Evans and Sutherland Display Function Network

Figure E1 outlines the functions performed by the E-S for the position integration and simulation display. Each of the blocks represents a firmware function or set of functions. The name(s) above each identify the variable used in the E-S programs. A brief description of the network follows.

The velocities are downloaded from the PRIME were they are stored in a variable, LINVEL or ROTVEL, on the E-S. A clock system triggers these velocities to be sent to the position integrators. The clock triggers a velocity transfer at a rate of ten transfers per second. Because an Euler integration method is used for position integration, an

accumulation function is used as the integrator. This accumulator scales the velocities by the appropriate time step and sums it with the last position information. The equation for the integration is as follows:

$$X_i = V_i \Delta t + X_{i-1} \qquad \text{(E1)}$$

The linear information is sent down three branches. The first branch updates the variables used for position feedback to the PRIME. The second branch is connected to the E-S display structure. The third branch is connected to a function that calculates the new viewpoint matrix, looking at the position where the plane has been translated to in branch two. This viewing matrix updates the E-S display structure.

The angular information is split into components -- roll, pitch and yaw. This information is sent down two branches. The first branch updates the variables used for position feedback to the PRIME. The second branch calculates the rotation matrices from the given angles. These rotation matrices are used to update the plane's angular position in the E-S display structure.

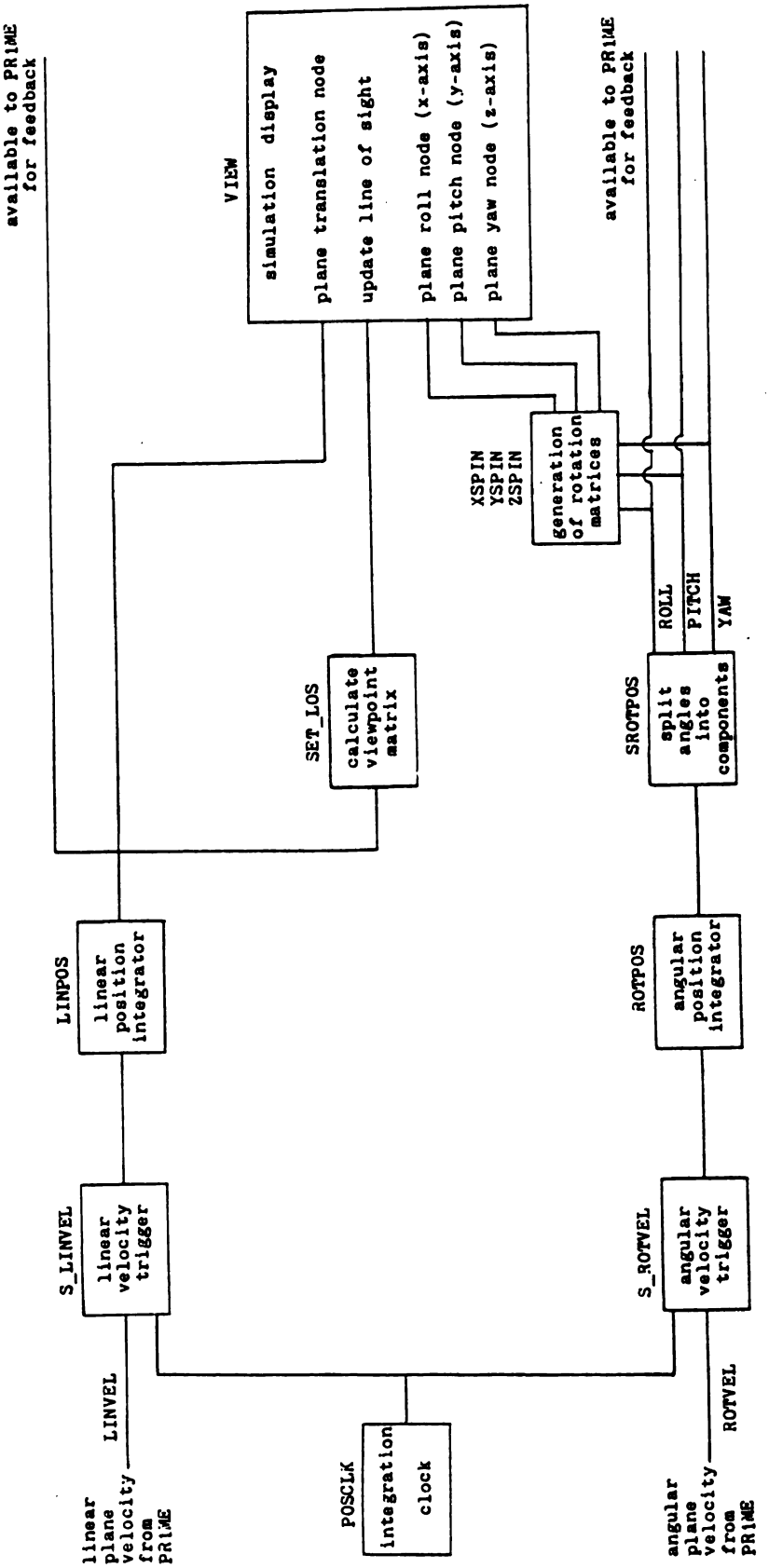


Figure E1. Evans and Sutherland Display Function Network

APPENDIX F

Simulation Computer Code

Common Variable Identification

Insert File Name	Common Block	Variables
ACELCM	ACCEL	AX, AY, AZ, AL, AM, AN
ANGPCM	POSITION	PX, PY, PZ, PL, PM, PN, ESPX, ESPY, ESPZ, ESPL, ESPM, ESPN
CTRLCM	CONTROL	PSPED, OWCSA, ECSA, RCSA
FRCCM	FORCE	X, Y, Z, L, M, N
INTECM	INTEGR	VEL(1:6), POS(1:6), DT, T1, T2, IRATE, NUMINT, IRESET
PARMSCM	PARAM	PI, RHOF2, T, RHOPDIA4, CWDI, CEDI
	RMATRIX	TROW, TLOW, TRIW, TLIW, TREL, TLEL, TRU
	TRACKX	LXROW, LXLOW, LXRIW, LXLIW, LXREL, LXLEL, LXRU, LXTH
PLANECM	INERTIA	IXX, IYY, IZZ, MASS, WGT
	OUTWING (OW)	B, C, X, Y, Z, DHA, WHA, SWA
	INNWING (W)	B, C, X, Y, Z, DHA, WHA, SWA
	ELEVATO (E)	B, C, X, Y, Z, DHA, WHA, SWA
	RUDDER (R)	B, C, X, Y, Z, DHA, WHA, SWA
	ENGPROP	PDIA, ENGDIA, XAREA
RVELCM	RELVEL	UR, VR, WR
TABLCM	TABLE	XAAVL, YAAVL, XLVD, YLVD, XAAVM, YAAVM, XJVT, YJVT
VELCM	VELOCITY	U, V, W, P, Q, R

```
      PROGRAM MSUSIM
CDCCCCCCCC              PROGRAM DESCRIPTION                      CCCCC
C                                                                    C
C     This is the driver program for the remote-controlled
c     flight simulation
C                                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                    C
CVCCCCCCCC           VARIABLE IDENTIFICATION                CCCCCCCC
C                                                                    C
C     IDBG - creates DEBUG output files
c          0- no DEBUGGING output
c          1- force DEBUGGING output
c          2- acceleration and velocity DEBUGGING output
c     IRATE - the slow motion factor
c     NUMINT - the number of velocity integrations on the PRIME
c              per one simulation loop
c      CRTMODE - current mode of operation for simulation
c      NEWMODE - new mode of operation for simulation
c      T1 - beginning time value of time step
c      T2 - ending time value of time step
C
C----- COMMON BLOCKS -----
C
C     INTECM
C                                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                    C
CSCCCCCCC            ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                    C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>INTECM
C
      CHARACTER NEWMODE*1,CRTMODE*1
      INTEGER IDBG,I
C
      EXTERNAL PSFILE,PSGRAF,PSTERM,TOPRIM,INPSET,INTSET,POSSET
      EXTERNAL PARMSET,RESET,TIME,REPEAT,INTE
C                                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                    C
CICCCCCCCC           INITIALIZATION BLOCK                 BLOCK 0100
C                                                                    C
      IDBG = 0
      IRATE = 12
      NUMINT = 3
      CRTMODE = 'E'
C
C<<<<< INITIALIZE THE PARAMETERS >>>>>
C
      CALL PARMSET
C
C<<<<< WRITE HEADER >>>>>
C
      DO 10 I = 1,14
         PRINT *,'  '
10    CONTINUE
      PRINT *,'          ****************************************'
      PRINT *,'          ****************************************'
      PRINT *,'          **                                    **'
      PRINT *,'          **                MSU                 **'
      PRINT *,'          **          REMOTE-CONTROLLED         **'
      PRINT *,'          **          FLIGHT SIMULATION         **'
      PRINT *,'          **                                    **'
      PRINT *,'          ****************************************'
      PRINT *,'          ****************************************'
C
C<<<<< PROMPT FOR SIMULATION MODE >>>>>
C
1000  PRINT *,'  '
      PRINT *,'Which flight simulation options would you like?'
      PRINT *,'  '
```

```
      PRINT *,'(D)emonstration mode or (F)light mode or (E)xit'
      READ(*,'(A)') NEWMODE
C
C<<<<< PAGE THE SCREEN >>>>>
C
      DO 20 I = 1,23
           PRINT *,'  '
20    CONTINUE
C
C<<<<< CHECK FOR VALID RESPONSE >>>>>
C
      IF((NEWMODE.NE.'D').AND.(NEWMODE.NE.'F')) THEN
           IF(NEWMODE.EQ.'E') GOTO 3000
           GOTO 1000
      ENDIF
C
C<<<<< CHECK TO SEE IF CURRENT MODE AND NEW MODE MATCH >>>>>
C
      IF(NEWMODE.EQ.'D') THEN
           IF(CRTMODE.EQ.'D') THEN
              PRINT *,'You are already in DEMO mode'
           ELSE
              PRINT *,'The EβS is initializing its function networks'
              PRINT *,'                    Please wait!'
              CALL PSFILE('ID58>SU14>SIMULATE.DIR>FLIGHT_SIM')
              CRTMODE = NEWMODE
           ENDIF
           GOTO 1000
      ENDIF
C
C<<<<< RESET EβS ROUTINES FOR NEW MODE >>>>>
C
      IF(NEWMODE.NE.CRTMODE) THEN
         PRINT *,'The EβS is initializing its function networks'
         PRINT *,'                    Please wait!'
         CALL PSGRAF
         IF(CRTMODE.EQ.'D') THEN
            PRINT *,'SEND FALSE TO <6>ROTCLK;'
         ENDIF
         PRINT *,'INIT;'
         PRINT *,'SEND FALSE TO <1>WARNING;'
         CALL PSTERM
         CALL TOPRIM
         CALL INPSET
         CALL INTSET
         CALL POSSET
         CALL PSFILE('ID58>SU14>SIMULATE.DIR>ENVIRON_DEF')
         CALL PSFILE('ID58>SU14>SIMULATE.DIR>SIM_REMOTE')
         CALL PSTERM
         CRTMODE = NEWMODE
      ENDIF
C
C<<<<< RESET THE PLANE'S POSITION >>>>>
C
      CALL RESET
C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CPCCCCCCCC            PROCESS BLOCK                        BLOCK 0200
C                                                                      C
C<<<<< START FORCE GENERATION ROUTINE >>>>>
C
      CALL TIME(T1)
C
C<<<<< PERFORM 4th ORDER RUNGE-KUTTA >>>>>
C
2000  CALL REPEAT(IDBG)
C
C<<<<< READ THE CURRENT TIME VALUE >>>>>
C
```

```
      CALL TIME(T2)
C
C<<<<< INTEGRATE TO VELOCITY AND START POSITION INTEGRATOR >>>>>
C<<<<<    T1 is set to T2 at the end of this subroutine    >>>>>
C
      CALL INTE
C
C<<<<< CHECK FOR RESET FLAG >>>>>
C
C --- false is reset --- true is continue ---
C
      IF(.NOT.IRESET) GOTO 1000
C
C<<<<< REPEAT FORCE GENERATION ROUTINE >>>>>
C
      GOTO 2000
C
C<<<<< END OF SIMULATION DRIVER PROGRAM
C
3000  CALL PSGRAF
      IF(CRTMODE.EQ.'D') THEN
           PRINT *,'SEND FALSE TO <6>ROTCLK;'
      ELSE
           PRINT *,'SEND FALSE TO <6>POSCLK;'
      ENDIF
      PRINT *,'INIT;'
      CALL PSTERM
      END
```

```
CDCCCCCCCC                SUBROUTINE DESCRIPTION                      CCCCC
C                                                                       C
c     This subroutine is called at the startup of the program
c     and calculates all of the necessary parameters that
c     are fixed once the plane geometry is defined in
c     the plane common block (PLANECM)
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC             VARIABLE IDENTIFICATION                 CCCCCCCC
C                                                                       C
c     name    definition      where used
c
c     PI    - 3.14159
c     RHO   - .00238
c     RHOF2 - rho/2.              HFORCE,VFORCE
c     CWDI  - pi*2.*(OWB+WB)      DRAG
c     CEDI  - pi*2.*EB            DRAG
c TROW(3,3)-  rotation            RELWND
c TLOW(3,3)-                      RELWND
c TRIW(3,3)-  matrices            RELWND
c TLIW(3,3)-                      RELWND
c TREL(3,3)-  for                 RELWND
c TLEL(3,3)-                      RELWND
c TRU(3,3) -  relative wind       RELWND
c
C----- these values are read from the plane data block (PLANEDAT)
c
C     OWDHA - outer wing dihedral angle
C     OWWHA - outer wing washout angle
C     OWSWA - outer wing sweep angle
C      WDHA - inner wing dihedral angle
C      WWHA - inner wing washout angle
C      WSWA - inner wing sweep angle
C      EDHA - elevator dihedral angle
C      EWHA - elevator washout angle
C      ESWA - elevator sweep angle
C      RDHA - rudder dihedral angle
C      RWHA - rudder washout angle
C      RSWA - rudder sweep angle
C
C----- COMMON BLOCKS -----
C
C      PLANECM,PARMSCM
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC              ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                       C
      SUBROUTINE PARMSET
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PLANECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PARMSCM
C
      INTRINSIC ATAN2,SIN,COS
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CICCCCCCCC             INITIALIZATION BLOCK                   BLOCK 0100
C                                                                       C
      RHO = .00238
      PI = ATAN2(1.,0.) * 2.
      RHOF2 = RHO/2.
      RHOPDIA4 = RHO * (PDIA**4.)
      CWDI = PI * (2. * (OWB + WB))**2.
      CEDI = PI * (2. * EB)**2.
c
c----- right outer wing rotation matrix -----
c
      TROW(1,1) = COS(OWWHA)*COS(OWSWA)
      TROW(2,1) = COS(OWWHA)*SIN(OWSWA)
```

```
      TROW(3,1) = -SIN(OWWHA)
C
      TROW(1,2) = SIN(-OWDHA)*SIN(OWWHA)*COS(OWSWA) -
     %            COS(-OWDHA)*SIN(OWSWA)
      TROW(2,2) = SIN(-OWDHA)*SIN(OWWHA)*SIN(OWSWA) +
     %            COS(-OWDHA)*COS(OWSWA)
      TROW(3,2) = SIN(-OWDHA)*COS(OWWHA)
C
      TROW(1,3) = COS(-OWDHA)*SIN(OWWHA)*COS(OWSWA) +
     %            SIN(-OWDHA)*SIN(OWSWA)
      TROW(2,3) = COS(-OWDHA)*SIN(OWWHA)*SIN(OWSWA) -
     %            SIN(-OWDHA)*COS(OWSWA)
      TROW(3,3) = COS(-OWDHA)*COS(OWWHA)
c
c----- left outer wing rotation matrix -----
c
      TLOW(1,1) = COS(OWWHA)*COS(-OWSWA)
      TLOW(2,1) = COS(OWWHA)*SIN(-OWSWA)
      TLOW(3,1) = -SIN(OWWHA)
C
      TLOW(1,2) = SIN(OWDHA)*SIN(OWWHA)*COS(-OWSWA) -
     %            COS(OWDHA)*SIN(-OWSWA)
      TLOW(2,2) = SIN(OWDHA)*SIN(OWWHA)*SIN(-OWSWA) +
     %            COS(OWDHA)*COS(-OWSWA)
      TLOW(3,2) = SIN(OWDHA)*COS(OWWHA)
C
      TLOW(1,3) = COS(OWDHA)*SIN(OWWHA)*COS(-OWSWA) +
     %            SIN(OWDHA)*SIN(-OWSWA)
      TLOW(2,3) = COS(OWDHA)*SIN(OWWHA)*SIN(-OWSWA) -
     %            SIN(OWDHA)*COS(-OWSWA)
      TLOW(3,3) = COS(OWDHA)*COS(OWWHA)
c
c----- right inner wing rotation matrix -----
c
      TRIW(1,1) = COS(WWHA)*COS(WSWA)
      TRIW(2,1) = COS(WWHA)*SIN(WSWA)
      TRIW(3,1) = -SIN(WWHA)
C
      TRIW(1,2) = SIN(-WDHA)*SIN(WWHA)*COS(WSWA) -
     %            COS(-WDHA)*SIN(WSWA)
      TRIW(2,2) = SIN(-WDHA)*SIN(WWHA)*SIN(WSWA) +
     %            COS(-WDHA)*COS(WSWA)
      TRIW(3,2) = SIN(-WDHA)*COS(WWHA)
CC
      TRIW(1,3) = COS(-WDHA)*SIN(WWHA)*COS(WSWA) +
     %            SIN(-WDHA)*SIN(WSWA)
      TRIW(2,3) = COS(-WDHA)*SIN(WWHA)*SIN(WSWA) -
     %            SIN(-WDHA)*COS(WSWA)
      TRIW(3,3) = COS(-WDHA)*COS(WWHA)
c
c----- left inner wing rotation matrix -----
c
      TLIW(1,1) = COS(WWHA)*COS(-WSWA)
      TLIW(2,1) = COS(WWHA)*SIN(-WSWA)
      TLIW(3,1) = -SIN(WWHA)
C
      TLIW(1,2) = SIN(WDHA)*SIN(WWHA)*COS(-WSWA) -
     %            COS(WDHA)*SIN(-WSWA)
      TLIW(2,2) = SIN(WDHA)*SIN(WWHA)*SIN(-WSWA) +
     %            COS(WDHA)*COS(-WSWA)
      TLIW(3,2) = SIN(WDHA)*COS(WWHA)
C
      TLIW(1,3) = COS(WDHA)*SIN(WWHA)*COS(-WSWA) +
     %            SIN(WDHA)*SIN(-WSWA)
      TLIW(2,3) = COS(WDHA)*SIN(WWHA)*SIN(-WSWA) -
     %            SIN(WDHA)*COS(-WSWA)
      TLIW(3,3) = COS(WDHA)*COS(WWHA)
c
c----- right elevator rotation matrix -----
c
      TREL(1,1) = COS(EWHA)*COS(ESWA)
```

```
      TREL(2,1) = COS(EWHA)*SIN(ESWA)
      TREL(3,1) = -SIN(EWHA)
C
      TREL(1,2) = SIN(-EDHA)*SIN(EWHA)*COS(ESWA) -
     %            COS(-EDHA)*SIN(ESWA)
      TREL(2,2) = SIN(-EDHA)*SIN(EWHA)*SIN(ESWA) +
     %            COS(-EDHA)*COS(ESWA)
      TREL(3,2) = SIN(-EDHA)*COS(EWHA)
C
      TREL(1,3) = COS(-EDHA)*SIN(EWHA)*COS(ESWA) +
     %            SIN(-EDHA)*SIN(ESWA)
      TREL(2,3) = COS(-EDHA)*SIN(EWHA)*SIN(ESWA) -
     %            SIN(-EDHA)*COS(ESWA)
      TREL(3,3) = COS(-EDHA)*COS(EWHA)
c
c----- left elevator rotation matrix -----
c
      TLEL(1,1) = COS(EWHA)*COS(-ESWA)
      TLEL(2,1) = COS(EWHA)*SIN(-ESWA)
      TLEL(3,1) = -SIN(EWHA)
C
      TLEL(1,2) = SIN(EDHA)*SIN(EWHA)*COS(-ESWA) -
     %            COS(EDHA)*SIN(-ESWA)
      TLEL(2,2) = SIN(EDHA)*SIN(EWHA)*SIN(-ESWA) +
     %            COS(EDHA)*COS(-ESWA)
      TLEL(3,2) = SIN(EDHA)*COS(EWHA)
C
      TLEL(1,3) = COS(EDHA)*SIN(EWHA)*COS(-ESWA) +
     %            SIN(EDHA)*SIN(-ESWA)
      TLEL(2,3) = COS(EDHA)*SIN(EWHA)*SIN(-ESWA) -
     %            SIN(EDHA)*COS(-ESWA)
      TLEL(3,3) = COS(EDHA)*COS(EWHA)
c
c----- rudder rotation matrix -----
c
      TRU(1,1) = COS(RWHA)*COS(RSWA)
      TRU(2,1) = COS(RWHA)*SIN(RSWA)
      TRU(3,1) = -SIN(RWHA)
C
      TRU(1,2) = SIN(RDHA)*SIN(RWHA)*COS(RSWA) -
     %           COS(RDHA)*SIN(RSWA)
      TRU(2,2) = SIN(RDHA)*SIN(RWHA)*SIN(RSWA) +
     %           COS(RDHA)*COS(RSWA)
      TRU(3,2) = SIN(RDHA)*COS(RWHA)
C
      TRU(1,3) = COS(RDHA)*SIN(RWHA)*COS(RSWA) +
     %           SIN(RDHA)*SIN(RSWA)
      TRU(2,3) = COS(RDHA)*SIN(RWHA)*SIN(RSWA) -
     %           SIN(RDHA)*COS(RSWA)
      TRU(3,3) = COS(RDHA)*COS(RWHA)
C
C<<<<< SET THE DISTANCE FROM FORCE ELEMENT TO C.G. >>>>>
C----- OUTER WING -----
C
      OWX = 1.28
      OWY = 15.0
      OWZ = -2.25
C
C----- INNER WING -----
C
      WX = .85
      WY = 6.0
      WZ = -2.25
C
C----- ELEVATOR -----
C
      EX = -13.0
      EY = 3.0
      EZ = 0.0
C
C----- RUDDER -----
```

```
C
      RUX = -13.0
      RUY = 0.0
      RUZ = -0.00
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
c
ccccccccccc            Subroutine Description                cccccc
c                                                                    c
c     This subroutine use HFORCE and TRNFRC to find  the
c     sum of the forces on the airplane.
c                                                                    c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
cVcccccccc              Variable Identification               cccc
c                                                                    c
c     B      - wing span  (ft)
c     C      - chord  (ft)
c  EX,EY,EZ - elevator moment arms  (ft)
c     L,M,N - moments about the planes c.g.  (ft*lbf)
c  RX,RY,RZ - rudder moment arms  (ft)
c     U,V,W - velocity components  (ft/sec)
c  WX,WY,WZ - main wing moment arms  (ft)
c     X,Y,Z - forces at the planes c.g.  (lbf)
c     WLIFT - lift due to wing  (lbf)
c     ELIFT -lift due to elevator  (lbf)
c     IFLG  - debug flag
c     OWCSA - outer wing control surface angle
c      ECSA - elevator control surface angle
c      RCSA - rudder control surface angle
c  OX,OY,OZ - linear forces used for DEBUG routine
c  OL,OM,ON - rotational forces used for DEBUG routine
C
C----- COMMON BLOCKS -----
C
C     FRCCM,VELCM,RVELCM,CTRLCM,PLANECM,ANGPCM,PARMSCM
c                                                                    c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
ccccccccccc            Entry and Storage Block          BLOCK 0000
c                                                                    c
      SUBROUTINE FORCES(IFLG)
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>FRCCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>RVELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>CTRLCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PLANECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ANGPCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PARMSCM
C
      REAL LWX,LWZ,LWM,LOWX,LOWZ
      REAL LOWM,LEX,LEZ,LEM
      REAL RWX,RWZ,RWM,ROWX,ROWZ,ROWM
      REAL REX,REZ,REM,RX,RY,RN
      REAL WLIFT,ELIFT
      REAL OX,OY,OZ,OL,OM,ON,ZERO
      INTEGER IFLG
C
```

```
      INTRINSIC SIN,COS,ATAN2
      EXTERNAL RELWND,HFORCE,TRNFRC,TRACK,VFORCE,WEIGHT
      EXTERNAL THRUST,DRAG
C
c                                                                       c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
cIcccccccc            Initialization Block            BLOCK 0100
C
C<<<<< RESET RELATIVE AND TOTAL FORCES >>>>>
C
      X = 0.
      Y = 0.
      Z = 0.
      L = 0.
      M = 0.
      N = 0.
      OX = 0.
      OY = 0.
      OZ = 0.
      OL = 0.
      OM = 0.
      ON = 0.
      ZERO = 0.
c
cPcccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
cPcccccccc            Process Block                   BLOCK 0200
c                                                                       c
c<<<<< inner main wing force calculation >>>>>
C--- right ---
c
      CALL RELWND(TRIW,WX,WY,WZ)
      CALL HFORCE(ZERO,RWZ,RWX,RWM,WC,WB,'IRW',LXRIW)
      CALL TRNFRC(WX,WY,WZ,RWX,ZERO,RWZ,ZERO,RWM,ZERO)
C
c--- left ---
c
      CALL RELWND(TLIW,WX,-WY,WZ)
      CALL HFORCE(ZERO,LWZ,LWX,LWM,WC,WB,'ILW',LXLIW)
      CALL TRNFRC(WX,-WY,WZ,LWX,ZERO,LWZ,ZERO,LWM,ZERO)
c                                                                       c
c<<<<< outer main wing force calculation >>>>>
c--- right ---
c
      CALL RELWND(TROW,OWX,OWY,OWZ)
      CALL HFORCE(OWCSA,ROWZ,ROWX,ROWM,OWC,OWB,'ORW',LXROW)
      CALL TRNFRC(OWX,OWY,OWZ,ROWX,ZERO,ROWZ,ZERO,ROWM,ZERO)
C
c--- left ---
c
      CALL RELWND(TLOW,OWX,-OWY,OWZ)
      CALL HFORCE(-OWCSA,LOWZ,LOWX,LOWM,OWC,OWB,'OLW',LXLOW)
      CALL TRNFRC(OWX,-OWY,OWZ,LOWX,ZERO,LOWZ,ZERO,LOWM,ZERO)
      CALL TRACK(OX,OY,OZ,OL,OM,ON,'WING      ',IFLG)
C
      WLIFT = Z
C
c<<<<< elevator force calculation >>>>>
C--- right ---
c
      CALL RELWND(TREL,EX,EY,EZ)
      CALL HFORCE(ECSA,REZ,REX,REM,EC,EB,'REL',LXREL)
      CALL TRNFRC(EX,EY,EZ,REX,ZERO,REZ,ZERO,REM,ZERO)
C
c--- left ---
c
      CALL RELWND(TLEL,EX,-EY,EZ)
      CALL HFORCE(ECSA,LEZ,LEX,LEM,EC,EB,'LEL',LXLEL)
      CALL TRNFRC(EX,-EY,EZ,LEX,ZERO,LEZ,ZERO,LEM,ZERO)
      CALL TRACK(OX,OY,OZ,OL,OM,ON,'ELEVATOR',IFLG)
```

```
C
      ELIFT = Z - WLIFT
C
c<<<<< rudder force calculation >>>>>
c
      CALL RELWND(TRU,RUX,RUY,RUZ)
      CALL VFORCE(RCSA,RY,RX,RN,RC,RB,'RUD',LXRU)
      CALL TRNFRC(RUX,RUY,RUZ,RX,RY,ZERO,ZERO,ZERO,RN)
      CALL TRACK(OX,OY,OZ,OL,OM,ON,'RUDDER  ',IFLG)
c
c<<<<< add in weight and thrust effects >>>>>
c
      CALL WEIGHT
      CALL TRACK(OX,OY,OZ,OL,OM,ON,'WEIGHT  ',IFLG)
      CALL THRUST ('THR',LXTH)
      CALL TRACK(OX,OY,OZ,OL,OM,ON,'THRUST  ',IFLG)
      CALL DRAG (WLIFT,ELIFT)
      CALL TRACK (OX,OY,OZ,OL,OM,ON,'DRAG    ',IFLG)
      RETURN
      END
```

```
c
ccccccccccc            Subroutine Description              cccccc
c                                                                 c
c     This subroutine will take the appropriate velocities
c      from the summation of forces to compute the local
c      forces at each horizontal flying surface.  It uses
c      a table lookup scheme to find the necessary wing
c      characteristics (CL,CD,CM).  It also assumes a
c      constant density of air.
c                                                                 c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
cVccccccccc             Variable Identification               cccc
c                                                                 c
c     B      - wing span (ft)
c     C      - chord of wing (ft)
c     CD     - drag coefficient
c     CL     - lift coefficient
c     CM     - moment coefficient
c     CSA    - control surface angle (radians)
c     DRAG   - drag force (lbf)
c     LIFT   - lift force (lbf)
c     MOM    - moment produced by forces above (ft*lbf)
c     DYNP   - dynamic pressure (lbf/ft**2)
c     RHO    - density of air (slugs/ft**3)
C     RHOF2  - RHO/2.
c     UR      - relative velocity in x direction (ft/sec)
c     WR      - relative velocity in z direction (ft/sec)
c     LASTX  - values of ALFA or CL used in previous TLOOK
C
C----- COMMON BLOCKS -----
C
C     RVELCM,PARMSCM
c                                                                 c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
ccccccccccc            Entry and Storage Block          BLOCK 0000
c                                                                 c
      SUBROUTINE HFORCE(CSA,LIFT,DRAG,MOM,C,B,IDENT,LASTX)
c                                                                 c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
c                                                                       c
cIcccccccc              Initialization Block            BLOCK 0100
c                                                                       c
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>RVELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PARMSCM
C
      REAL ALFA,ALFA0,B,C,CD,CL,CM,CSA,DRAG,LIFT,MOM,DYNP,S
      INTEGER LASTX(3)
      CHARACTER IDENT*3
C
      INTRINSIC ATAN2,SIN,COS
      EXTERNAL TLOOK
c                                                                       c
cPccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
cPcccccccc              Process Block                   BLOCK 0200
c                                                                       c
c     calculate the dynamic pressure and effective wing area
c
c<<<<< SET MAXIMUM VELOCITY >>>>>
c
      IF(UR.GT.100000.) THEN
           UR = 100000.
      ENDIF
      IF(WR.GT.100000.) THEN
           WR = 100000.
      ENDIF
c
c     calculate the dynamic pressure and effective wing area
c
      DYNP = RHOF2*UR*UR + RHOF2*WR*WR
      S = B*C
c
c     calculate the true angle of incidence
c
      IF((WR.EQ.0.).AND.(UR.EQ.0.)) THEN
           ALFA0 = 0.
           ALFA = (ALFA0 + CSA)*57.296
      ELSE
           ALFA0 = ATAN2(WR,UR)
           ALFA = (ALFA0 + CSA)*57.296
      ENDIF
c
c     use the lookup routine to find CD,CL,CM
c
      CL = TLOOK('LIFT',ALFA,IDENT,LASTX(1))
      CM = TLOOK('MOME',ALFA,IDENT,LASTX(2))
      CD = TLOOK('DRAG',CL,IDENT,LASTX(3))
c
c     calculate the lift, drag, and moments in body coords
c     force = coefficient * dynamic pressure * effective area
c
      DRAG = -(-CL*SIN(ALFA0) + CD*COS(ALFA0))*DYNP*S
      LIFT = -(CL*COS(ALFA0) + CD*SIN(ALFA0))*DYNP*S
      MOM = -CM*DYNP*S*C
c                                                                       c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
      RETURN
      END
```

c

```
ccccccccccc           Subroutine Description              cccccc
c                                                                 c
c     This subroutine will take the appropriate velocities
c      from the summation of forces to compute the local
c      forces at each vertical flying surface.  It uses
c      a table lookup scheme to find the necessary wing
c      characteristics (CL,CD,CM).  It also assumes a
c      constant density of air.
c                                                                 c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
cVccccccccc             Variable Identification               cccc
c                                                                 c
c     B      - wing span (ft)
c     C      - chord of wing (ft)
c     CD     - drag coefficient
c     CL     - lift coefficient
c     CM     - moment coefficient
c     CSA    - control surface angle (radians)
c     D      - drag force (lbf)
c     L      - lift force (lbf)
c     M      - moment produced by forces above (ft*lbf)
c     DYNP   - dynamic pressure (lbf/ft**2)
c     RHO    - density of air (slugs/ft**3)
C     RHOF2  - RHO/2.
c     UR      - relative velocity in x direction (ft/sec)
c     VR      - relative velocity in y direction (ft/sec)
c     LASTX  - values of ALFA or CL used in previuos TLOOK
C
C----- COMMON BLOCKS -----
C
C     RVELCM,PARMSCM
c                                                                 c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
ccccccccccc           Entry and Storage Block          BLOCK 0000
c                                                                 c
      SUBROUTINE VFORCE(CSA,LIFT,DRAG,MOM,C,B,IDENT,LASTX)
c                                                                 c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
cIccccccccc             Initialization Block           BLOCK 0100
c                                                                 c
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>RVELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PARMSCM
C
      REAL ALFA,ALFA0,B,C,CD,CL,CM,CSA,DYNP,S,SIGN
      REAL LIFT,DRAG,MOM
      INTEGER LASTX(3)
      CHARACTER IDENT*3
C
      INTRINSIC ATAN2,ABS,SIN,COS
      EXTERNAL TLOOK
c                                                                 c
cPccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
cPccccccccc             Process Block                  BLOCK 0200
c                                                                 c
c<<<<< SET MAXIMUM VELOCITY >>>>>
c
      IF(UR.GT.100000.) THEN
          UR = 100000.
      ENDIF
      IF(VR.GT.100000.) THEN
          VR = 100000.
      ENDIF
c
c     calculate the dynamic pressure and effective wing area
c
      DYNP = RHOF2*UR*UR + RHOF2*VR*VR
      S = B*C
```

```
c
c     calculate the true angle of incidence
c
      IF((VR.EQ.0.).AND.(UR.EQ.0.)) THEN
           ALFA0 = 0.0
           ALFA = (ALFA0 + CSA)*57.296
      ELSE
           ALFA0 = ATAN2(VR,UR)
           ALFA = (ALFA0 + CSA)*57.296
      ENDIF
C
C<<<<< CORRECT ANGLE OF ATTACK TO ALL POSITIVE >>>>>
C
      IF(ALFA.NE.0.0) THEN
           SIGN = ALFA/ABS(ALFA)
           ALFA = SIGN*ALFA
      ELSE
           SIGN = 1.0
      ENDIF
c
c     use the lookup routine to find CD,CL,CM
c     and adjust for symmetric airfoil
c
      CL = TLOOK('LIFT',ALFA,IDENT,LASTX(1)) - .31
      CM = TLOOK('MOME',ALFA,IDENT,LASTX(2)) +.0833
      CD = TLOOK('DRAG',CL,IDENT,LASTX(3))
c
c     calculate the lift, drag, and moments in body coords
c      force = coefficient * dynamic pressure * effective area
c
      DRAG = -(-CL*SIN(ALFA0) + CD*COS(ALFA0))*DYNP*S
      LIFT = -SIGN*(CL*COS(ALFA0) + CD*SIN(ALFA0))*DYNP*S
      MOM = -SIGN*CM*DYNP*S*C
c                                                                     c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                     c
      RETURN
      END
```

```
c
ccccccccccc           Subroutine Description              cccccc
c                                                                   c
c     This subroutine takes the forces found in
c     subroutine HFORCE and transforms them to the c.g.
c     of the airplane.
c                                                                   c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                   c
cVccccccccc             Variable Identification               cccc
c                                                                   c
c          X,Y,Z - force components at the plane c.g. (lbf)
c          L,M,N - moment components at the plane c.g. (ft*lbf)
c       XA,YA,ZA - line of action to c.g. of plane (ft)
c XLOC,YLOC,ZLOC - localized forces in wing coordinates (lbf)
c LLOC,MLOC,NLOC - localized moments in wing coordinates (ft*lbf)
C
C----- COMMON BLOCKS -----
C
C     FRCCM
c                                                                   c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                   c
```

```
ccccccccccc            Entry and Storage Block          BLOCK 0000
c                                                                  c
      SUBROUTINE TRNFRC(XA,YA,ZA,XLOC,YLOC,ZLOC,LLOC,MLOC,NLOC)
c                                                                  c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                  c
cIccccccccc            Initialization Block             BLOCK 0100
c                                                                  c
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>FRCCM
C
      REAL LLOC,MLOC,NLOC,XA,XLOC,YA,YLOC,ZA,ZLOC
c                                                                  c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                  c
cPccccccccc            Process Block                    BLOCK 0200
c
c <<<<< add the components from other 'wings' >>>>>
c
      X = X + XLOC
      Y = Y + YLOC
      Z = Z + ZLOC
      L = L + LLOC - ZA*YLOC + YA*ZLOC
      M = M + MLOC + ZA*XLOC - XA*ZLOC
      N = N + NLOC - YA*XLOC + XA*YLOC
c                                                                  c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                  c
      RETURN
      END
```

```
c
ccccccccccc            Subroutine Description             cccccc
c                                                                  c
c    This subroutine transforms the absolute wind into
c    relative wind coordinates for the plane surface to use.
c                                                                  c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                  c
cVccccccccc            Variable Identification              cccc
c                                                                  c
c     DHA   - dihedral angle  (radians)
c     WHA   - washout angle  (radians)
c     SWA   - sweep angle  (radians)
c  XA,YA,ZA - moment arms from plane c.g to surface c.g.  (ft)
c     T     - rotation matrices to find rel wind
c     P,Q,R - absolute rotational wind velocities  (rad/sec)
c     U,V,W - absolute linear wind components  (ft/sec)
c  UR,VR,WR - relative wind components  (ft/sec)
cUREL,VREL - wind components with P,Q,R effects  (ft/sec)
c    WREL
C
C----- COMMON BLOCKS -----
C
C     VELCM,RVELCM
c                                                                  c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                  c
ccccccccccc            Entry and Storage Block          BLOCK 0000
c                                                                  c
      SUBROUTINE RELWND(T,XA,YA,ZA)
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
```

```
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>RVELCM
C
      REAL T(3,3),XA,YA,ZA,UREL,VREL,WREL
c                                                                       c
cPccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
cPccccccccc             Process Block                    BLOCK 0200
c                                                                       c
C<<<<< ADJUST FOR ROTATING COORDINATES >>>>>
C
      UREL = U + Q*ZA - R*YA
      VREL = V + R*XA - P*ZA
      WREL = W + P*YA - Q*XA
C
C<<<<< ALIGN THE VELOCITIES WITH THE SURFACE AXIS >>>>>
C<<<<<           USING A 3X3 MATRIX               >>>>>
C
C----- below is the general form of the matrix -----
C----- refer to SUBROUTINE PARMSET where these -----
C-----         matrices are calculated           -----
C
C     T(1,1) = COS(Y)*COS(Z)
C     T(2,1) = COS(Y)*SIN(Z)
C     T(3,1) = -SIN(Z)
C
C     T(1,2) = SIN(X)*SIN(Y)*COS(Z) - COS(X)*SIN(Z)
C     T(2,2) = SIN(X)*SIN(Y)*SIN(Z) + COS(X)*COS(Z)
C     T(3,2) = SIN(X)*COS(Y)
C
C     T(1,3) = COS(X)*SIN(Y)*COS(Z) + SIN(X)*SIN(Z)
C     T(2,3) = COS(X)*SIN(Y)*SIN(Z) - SIN(X)*COS(Z)
C     T(3,3) = COS(X)*COS(Y)
C
      UR = UREL*T(1,1) + VREL*T(1,2) + WREL*T(1,3)
      VR = UREL*T(2,1) + VREL*T(2,2) + WREL*T(2,3)
      WR = UREL*T(3,1) + VREL*T(3,2) + WREL*T(3,3)
C
C
c                                                                       c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
      RETURN
      END
```

```
C
CDCCCCCCCC              SUBROUTINE DESCRIPTION                 CCCCC
C                                                                     C
C    This subroutine calculates the thrust generated
C    by the airplane's engine.
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
CVCCCCCCCC           VARIABLE IDENTIFICATION              CCCCCCCC
C                                                                     C
c    J      - advance ratio
c    T      - thrust component  (lbf)
c    RHO    - air density  (slugs/ft**3)
c   PSPED   - propeller speed  (rev/sec)
c    CT     - thrust coefficient
c    X      - total force in the x direction  (lbf)
c    U      - absolute velocity in the x direction  (ft/sec)
c    PDIA   - propeller diameter  (ft)
```

```
C RHOPDIA4 - RHO * PDIA ** 4.
c     LASTX - value of J used in previuos TLOOK
c     IDENT - character flag used by TLOOK
C
C----- COMMON BLOCKS -----
C
C     FRCCM,VELCM,PLANECM,CTRLCM,PARMSCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC            ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                       C
      SUBROUTINE THRUST(IDENT,LASTX)
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>FRCCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PLANECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>CTRLCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PARMSCM
C
      REAL J,T,CHECK,CT
      INTEGER LASTX(3)
      CHARACTER IDENT*3
C
      EXTERNAL TLOOK
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                          BLOCK 0200
C                                                                       C
      IF(PSPED.EQ.0.) THEN
           T = 0.
           GOTO 1000
      ENDIF
C
      CHECK = U/PSPED
      IF(CHECK.GT.15.4) THEN
           T = 0.
           GOTO 1000
      ENDIF
C
      J = CHECK/PDIA
C
C<<<<< USE LOOK UP FUNCTION TO FIND COEFFICIENTS >>>>>
C
      CT = TLOOK('SPED',J,IDENT,LASTX(1))
C
C<<<<< CALCULATE THRUST >>>>>
C<<<<< force = coefficient * advance speed * effective area >>>>>
C
      T = CT * PSPED * PSPED * RHOPDIA4
C
C<<<<< SUMMATION OF X DIRECTION FORCES >>>>>
C
1000  X = X + T
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
C
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                  CCCCC
C                                                                    C
C    This subroutine calculates the drag effect of the non-lift
C    generating surfaces.  The coefficients of darg assume the
C    angle of attack of the fuselage to be small.
C                                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                    C
CVCCCCCCCC            VARIABLE IDENTIFICATION               CCCCCCCC
C                                                                    C
c     RHO   - air density  (slug/ft**3)
c     TWB   - total wing span  (ft)
c     TEB   - total elevator span  (ft)
c     DYNP  - dynamic pressure  (lbf/ft**2)
c     DF    - drag due to fuselage effects  (lbf)
c     DE    - drag due to engine and nacelle effects  (lbf)
c     DL    - drag due to landing gear effects  (lbf)
c     WDI   - induced drag effects of main wing  (lbf)
c     EDI   - induced drag effects of elevator wing  (lbf)
c     X     - total force in the x direction  (lbf)
c     M     - total moment about the x axis  (ft*lbf)
c     U     - forward velocity of the plane (ft/sec)
C
C----- COMMON BLOCKS -----
C
C     FRCCM,VELCM,PLANECM,PARMSCM
C                                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                    C
CSCCCCCCC             ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                    C
      SUBROUTINE DRAG (WLIFT,ELIFT)
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>FRCCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PLANECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PARMSCM
C
      REAL DE,DF,DL,DYNP,WLIFT,ELIFT,WDI,EDI,UT
C                                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                    C
CPCCCCCCCC            PROCESS BLOCK                        BLOCK 0200
C
C<<<<< SET MAXIMUM VELOCITY >>>>>
C
      IF(U.GT.100000.) THEN
           UT = U
           U = 100000.
      ENDIF
C   dynamic pressure
c
      DYNP = .5*RHO*U*U
c
C   fuselage drag contributions
C
      DF = .000557*XAREA*U*U
C
C   engine and nacelle drag contribution
C
      DE = .000335*ENGDIA*ENGDIA*U*U
C
C   landing gear drag contribution
C
      DL = .001395*U*U
C
C   induced drag produced by wings and elevator
C
      IF(DYNP.EQ.0.) THEN
           WDI = 0.
           EDI = 0.
```

```
      ELSE
           WDI = WLIFT*WLIFT/(CWDI*DYNP)
           EDI = ELIFT*ELIFT/(CEDI*DYNP)
      ENDIF
C
C    summation of drag effects
C
      X = X - DF - DE - DL - WDI - EDI
      M = M - WDI*WX - EDI*EX
C
C     reset the maximum velocity
C
      IF(U.EQ.100000.) THEN
           U = UT
      ENDIF
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
C
ccccccccccc           Subroutine Description              cccccc
c                                                                 c
c     This subroutine will incorporate the weight of the
c     aircraft into the summation of the forces at the c.g.
c                                                                 c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
cVccccccccc             Variable Identification               cccc
c                                                                 c
c     WGT   - aircraft weight  (slugs)
c     PM    - pitch angle of the aircraft  (radians)
c     PL    - roll angle of the aircraft (radians)
c     X,Y,Z - force components acting on the aircraft c.g. (lbf)
C
C----- COMMON BLOCKS -----
C
C     FRCCM,PLANECM,ANGPCM
c                                                                 c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
cIccccccccc             Initialization Block           BLOCK 0100
c                                                                 c
      SUBROUTINE WEIGHT
c
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>FRCCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PLANECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ANGPCM
C
      INTRINSIC SIN,COS
c                                                                 c
cPccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                 c
cPccccccccc             Process Block                  BLOCK 0200
C
C<<<<< CALCULATE WEIGHT COMPONENTS >>>>>
C
      X = X - WGT*SIN(PM)
      Y = Y + WGT*(COS(PM)*SIN(PL))
      Z = Z + WGT*(COS(PM)*COS(PL))
c                                                                 c
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
      RETURN
      END




c
CDCCCCCCCC              FUNCTION DESCRIPTION                      CCCCC
C                                                                       C
C     This function will perform a 1-D table look up.  The
C     function requires the following arguments: TABLENAME,
C     NUM OF PTS IN TABLE, and XVAL.  From this info
C     the TLOOK command will find the corresponding YVAL
C     using a linear interpolation between data points.
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC            VARIABLE IDENTIFICATION                 CCCCCCCC
C                                                                       C
C       IFLAG - checks for illegal table name
C       NX    - number of data points in data table
C       TABLE - name of data table
C       TLOOK - value returned from function
C XAAVL,YAAVL - data for alfa vs. Lift Coefficient
C XAAVM,YAAVL - data for alfa vs. Moment Coefficient
C XLVD,YLVD   - data for Lift Coefficient vs. Drag Coefficient
C
C----- COMMON BLOCKS -----
C
C     TABLCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC             ENTRY AND STORAGE BLOCK               BLOCK 0000
C                                                                       C
      FUNCTION TLOOK(TABLE,XVAL,IDENT,LASTX)
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>TABLCM
C
      REAL XVAL,TLOOK
      INTEGER LASTX,IFLAG,NX
      CHARACTER TABLE*4,IDENT*3
C
      EXTERNAL FIND
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CICCCCCCCC            INITIALIZATION BLOCK                  BLOCK 0100
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                         BLOCK 0200
C                                                                       C
      IFLAG = 0
      IF(TABLE.EQ.'LIFT') THEN
           NX = 12
           CALL FIND (XAAVL,YAAVL,XVAL,TLOOK,NX,TABLE,IDENT,LASTX)
           IFLAG = 1
      ENDIF
C
      IF(TABLE.EQ.'DRAG') THEN
```

```
            NX = 17
            CALL FIND (XLVD,YLVD,XVAL,TLOOK,NX,TABLE,IDENT,LASTX)
            IFLAG = 1
      ENDIF
C
      IF(TABLE.EQ.'MOME') THEN
            NX = 11
            CALL FIND (XAAVM,YAAVM,XVAL,TLOOK,NX,TABLE,IDENT,LASTX)
            IFLAG = 1
      ENDIF
C
      IF(TABLE.EQ.'SPED') THEN
            NX = 14
            CALL FIND (XJVCT,YJVCT,XVAL,TLOOK,NX,TABLE,IDENT,LASTX)
            IFLAG = 1
      ENDIF
C
      IF(IFLAG.NE.1) GOTO 2000
      GOTO 1000
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CECCCCCCCC            ERROR HANDLING                        BLOCK 0900
C                                                                       C
2000  PRINT *,'An invalid table name has been given'
      STOP
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
1000  RETURN
      END
```

```
C
CDCCCCCCCC              SUBROUTINE DESCRIPTION                  CCCCC
C                                                                       C
C     This subroutine is used in conjunction with
c     the TLOOK function to find the value off of a table.
C     This subroutine uses linear interpolation between two
c     data points to find YVAL.
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC            VARIABLE IDENTIFICATION                 CCCCCCCC
C                                                                       C
C     NX    - number of data points
C     X,Y   - array of data points
C     XVAL  - value input to look up function
C     YVAL  - value output from look up function
C
C----- COMMON BLOCKS -----
C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC             ENTRY AND STORAGE BLOCK                 BLOCK 0000
C                                                                       C
      SUBROUTINE FIND(X,Y,XVAL,YVAL,NX,TABLE,IDENT,LASTX)
C
      REAL X(NX),Y(NX),XVAL,YVAL
      INTEGER LASTX,I,NX
      CHARACTER TABLE*4,IDENT*3
C                                                                       C
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
CPCCCCCCCC            PROCESS BLOCK                        BLOCK 0200
C                                                                     C
C<<<<< SET LOWER END OF TABLE >>>>>
C
      IF(XVAL.LE.X(1)) THEN
           YVAL = Y(1)
           LASTX = 1
C           PRINT 200, TABLE,IDENT
           GOTO 1000
      ENDIF
C
C<<<<< SET UPPER END OF TABLE >>>>>
C
      IF(XVAL.GE.X(NX)) THEN
           YVAL = Y(NX)
           LASTX = NX
C           PRINT 300, TABLE,IDENT
           GOTO 1000
      ENDIF
C
C<<<<< INTERPOLATE USING DATA FROM TABLE >>>>>
C
           I = LASTX
           IF(XVAL.GE.X(I)) THEN
2000          I = I + 1
              IF(XVAL.GT.X(I)) GOTO 2000
      YVAL = ((Y(I)-Y(I-1))/(X(I)-X(I-1)))*(XVAL-X(I-1)) + Y(I-1)
           ELSE
4000          I = I - 1
              IF(XVAL.LT.X(I)) GOTO 4000
      YVAL = ((Y(I+1)-Y(I))/(X(I+1)-X(I)))*(XVAL-X(I)) + Y(I)
      ENDIF
      LASTX = I
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
CECCCCCCCC            FORMAT STATEMENTS                    BLOCK 0800
C                                                                     C
C200    FORMAT('The lower limit of ',A4,' has been reached by ',A3)
C300    FORMAT('The upper limit of ',A4,' has been reached by ',A3)
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
1000  RETURN
      END
```

```
c
CDCCCCCCCC              SUBROUTINE DESCRIPTION               CCCCC
C                                                                     C
C    This subroutine calculates the acceleration of the plane's
c    center of mass using the forces calculated in FORCES.  It
c    takes into account that the accelerations are defined in
c    a rotating coordinate system.  Therefore the calculation
c    contains cross product terms.
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
CVCCCCCCCC            VARIABLE IDENTIFICATION              CCCCCCCC
C                                                                     C
c AX,AY,AZ - linear accelerations
```

```
c AL,AM,AN - rotational accelerations
c  U,V,W   - Linear velocities
c  P,Q,R   - Rotational velocities
c   MASS   - aircraft mass
c    IXX   - Aircraft moment of inertia (x axis)
c    IYY   - Aircraft moment of inertia (y axis)
c    IZZ   - Aircraft moment of inertia (z axis)
C
C----- COMMON BLOCKS -----
C
C     FRCCM,VELCM,PLANECM,ACELCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC           ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                       C
      SUBROUTINE CALACE
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CICCCCCCCC           INITIALIZATION BLOCK                 BLOCK 0100
C                                                                       C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>FRCCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>PLANECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ACELCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC           PROCESS BLOCK                        BLOCK 0200
C                                                                       C
      AX = X/MASS + R*V - Q*W
      AY = Y/MASS + P*W - R*U
      AZ = Z/MASS + Q*U - P*V
      AL = (L - Q*R*(IZZ - IYY))/IXX
      AM = (M - P*R*(IXX - IZZ))/IYY
      AN = (N - P*Q*(IYY - IXX))/IZZ
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
CDCCCCCCCC             SUBROUTINE DESCRIPTION              CCCCC
C                                                                       C
C     This subroutine provides the output for the DEBUG if option
c     1 is set.  The forces calculated for each of the force
c     generation subroutines is output.  A runnong total of the
c     forces is also output.
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC           VARIABLE IDENTIFICATION              CCCCCCCC
C                                                                       C
c     RX - differential force in x direction
c     RY - differential force in y direction
c     RZ - differential force in z direction
c     RL - differential moment about x axis
c     RM - differential moment about y axis
c     RN - differential moment about z axis
c     IFLG - flag for DEBUG option
c     ID - identifies the component with an eight character string
```

```
C
C----- COMMON BLOCKS -----
C
C     FRCCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC           ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                       C
      SUBROUTINE TRACK(RX,RY,RZ,RL,RM,RN,ID,IFLG)
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CICCCCCCCC           INITIALIZATION BLOCK                 BLOCK 0100
C                                                                       C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>FRCCM
C
      REAL RX,RY,RZ,RL,RM,RN
      INTEGER IFLG
      CHARACTER ID*8
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC           PROCESS BLOCK                        BLOCK 0200
C
C<<<<< CHECK FOR DEBUG OPTIONS >>>>>
C
      IF(IFLG.EQ.1) THEN
C
C<<<<< CALCULATE THE RELATIVE FORCES >>>>>
C
           RX = X - RX
           RY = Y - RY
           RZ = Z - RZ
           RL = L - RL
           RM = M - RM
           RN = N - RN
C
C<<<<< OUTPUT INFO TO FILE >>>>>
C
           WRITE(54,50) ID
50         FORMAT(/'The forces generated by the ',A8,' ARE:'/)
           WRITE(54,100)
100        FORMAT(/8X,'Relative',18X,'Total')
           WRITE (54,200) RX,X
200        FORMAT('X:    ',F15.5,10X,F15.5)
           WRITE (54,300) RY,Y
300        FORMAT('Y:    ',F15.5,10X,F15.5)
           WRITE (54,400) RZ,Z
400        FORMAT('Z:    ',F15.5,10X,F15.5)
           WRITE (54,500) RL,L
500        FORMAT('L:    ',F15.5,10X,F15.5)
           WRITE (54,600) RM,M
600        FORMAT('M:    ',F15.5,10X,F15.5)
           WRITE (54,700) RN,N
700        FORMAT('N:    ',F15.5,10X,F15.5//)
C
C<<<<< RESET RELATIVE FORCE VALUES >>>>>
C
      RX = X
      RY = Y
      RZ = Z
      RL = L
      RM = M
      RN = N
      ENDIF
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                      C
C    This subroutine performs a fourth order Runge-Kutta
c    integration to calculate the velocities at the
c    next time step.  This routine uses another
c    subroutine DERIV to calculate the derivatives.
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CVCCCCCCCC           VARIABLE IDENTIFICATION                 CCCCCCCC
C                                                                      C
c    DPSI  - the derivative of the function (acceleration)
c    DT    - the time step (delta time)
c    VEL   - the function integral (velocity)
c    TEMP  - the intermediate values of velocity
c  TEMPVEL - the temporary evaluation of the velocity
C    IRATE - slow motion factor
c     IFLG - DEBUG flag to write output to a file
C
C----- COMMON BLOCKS -----
C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CSCCCCCCC            ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                      C
      SUBROUTINE RUNGE(VEL,DT,IRATE,IFLG)
C
      REAL DPSI(6),TEMP1(6),TEMP2(6),TEMP3(6),TEMP4(6),TEMPVEL(6)
      REAL VEL(6),DT
      INTEGER IRATE,IFLG,I
C
      INTRINSIC FLOAT
      EXTERNAL DEBUG,DERIV
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CPCCCCCCCC           PROCESS BLOCK                        BLOCK 0200
C
C
C<<<<< TIME STEP AT SLOW MOTION RATE >>>>>
C
      DT = DT/FLOAT(IRATE)
C
C<<<<< WRITE TO FILE IF DEBUG IS ON >>>>>
C
      CALL DEBUG(DPSI,VEL,IFLG,DT,0)
C                                                                      C
C<<<<< EVALUATE THE FUNCTION FOR THE FIRST TIME >>>>>
C
      CALL DERIV(DPSI,VEL,IFLG)
      DO 10 I = 1,6
           TEMP1(I) = DPSI(I) * DT
           TEMPVEL(I) = VEL(I) + TEMP1(I)/2.
10    CONTINUE
      CALL DEBUG(DPSI,TEMPVEL,IFLG,DT,1)
C
C<<<<< SECOND EVALUATION OF VELOCITY >>>>>
C
      CALL DERIV(DPSI,TEMPVEL,IFLG)
      DO 20 I = 1,6
           TEMP2(I) = DPSI(I) * DT
           TEMPVEL(I) = VEL(I) + TEMP2(I)/2.
20    CONTINUE
      CALL DEBUG(DPSI,TEMPVEL,IFLG,DT,2)
C
C<<<<< THIRD EVALUATION OF VELOCITY >>>>>
C
      CALL DERIV(DPSI,TEMPVEL,IFLG)
      DO 30 I = 1,6
           TEMP3(I) = DPSI(I) * DT
```

```
            TEMPVEL(I)= VEL(I) + TEMP3(I)
30      CONTINUE
        CALL DEBUG(DPSI,TEMPVEL,IFLG,DT,3)
C
C<<<<< FOURTH EVALUATION OF VELOCITY >>>>>
C
        CALL DERIV(DPSI,TEMPVEL,IFLG)
        DO 40 I = 1,6
            TEMP4(I) = DPSI(I) * DT
            TEMPVEL(I) = VEL(I) + TEMP4(I)
40      CONTINUE
        CALL DEBUG(DPSI,TEMPVEL,IFLG,DT,4)
C
C<<<<< EVALUATE THE VELOCITY USING 4th ORDER RUNGE-KUTTA >>>>>
C
        DO 50 I = 1,6
            VEL(I) = (TEMP1(I)+2.*TEMP2(I)+2.*TEMP3(I)+TEMP4(I))/6.
     %               + VEL(I)
50      CONTINUE
C
C<<<<< WRITE TO FILE IF DEBUG IS ON >>>>>
C
        CALL DEBUG(DPSI,VEL,IFLG,DT,5)
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C<<<<< RESET TIME STEP BACK TO REAL TIME >>>>>
C
        DT = DT*FLOAT(IRATE)
C
        RETURN
        END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                       C
C     This subroutine is used in conjuction with RUNGE and
c     FORCES to calculate the derivative of the velocity.
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC           VARIABLE IDENTIFICATION                CCCCCCCC
C                                                                       C
c     DPSI  - the derivative fnction used in RUNGE
c     PSI   - the function used in RUNGE
c     U,V,W - the linear velocities used in FORCES
c     P,Q,R - the rotational velocities used in FORCES
c  AX,AY,AZ - the linear accelerations used in FORCES
c  AL,AM,AN - the rotational ac celerations used in FORCES
c     IFLG  - debug flag to have force data written to an output file
C
C----- COMMON BLOCKS -----
C
C       VELCM,ACELCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC            ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                       C
        SUBROUTINE DERIV(DPSI,PSI,IFLG)
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ACELCM
```

```
C
      REAL DPSI(6),PSI(6)
      INTEGER IFLG
C
      EXTERNAL FORCES,CALACE
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                          BLOCK 0200
C                                                                       C
C<<<<< SET RUNGE VARIABLES TO FORCES VARIABLE >>>>>
C
      U = PSI(1)
      V = PSI(2)
      W = PSI(3)
      P = PSI(4)
      Q = PSI(5)
      R = PSI(6)
C
C<<<<< CALL FORCES TO EVALUATE ACCELERATIONS >>>>>
C
      CALL FORCES(IFLG)
      CALL CALACE
C
C<<<<< SET FORCES VARIABLES TO RUNGE VARIABLES >>>>>
C
      DPSI(1) = AX
      DPSI(2) = AY
      DPSI(3) = AZ
      DPSI(4) = AL
      DPSI(5) = AM
      DPSI(6) = AN
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                   CCCCC
C                                                                       C
C    This subroutine will output the calcuated accelerations
C    and velocities used in the RUNGE subroutine.
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC            VARIABLE IDENTIFICATION                CCCCCCCC
C                                                                       C
C    VEL - CURRENT VELOCITY EVALUATION
C    ACC - CURRENT ACCELERATION EVALUATION
C    IFLG - DEBUG FLAG  ( 2 - ON )
C    DT - CURRENT TIME STEP
C    I - RUNGE-KUTTA STEP COUNTER
C
C----- COMMON BLOCKS -----
C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC             ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                       C
      SUBROUTINE DEBUG(ACC,VEL,IFLG,DT,I)
C
```

```
      REAL VEL(6),ACC(6),DT
      INTEGER IFLG,I
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                          BLOCK 0200
C                                                                       C
C
C<<<<< WRITE DEBUG DATA TO FILE >>>>>
C
      IF(IFLG.NE.2) GOTO 1000
        IF(I.EQ.0) THEN
           WRITE(53,*) ' '
           WRITE(53,*) ' '
           WRITE(53,*)'Initiate new Runge-Kutta calculation,DT=',DT
        ENDIF
        WRITE(53,*)  ' '
        WRITE(53,*) 'Acceleration and Velocity data for step',I
        WRITE(53,*) 'X:',ACC(1),VEL(1)
        WRITE(53,*) 'Y:',ACC(2),VEL(2)
        WRITE(53,*) 'Z:',ACC(3),VEL(3)
        WRITE(53,*) 'L:',ACC(4),VEL(4)
        WRITE(53,*) 'M:',ACC(5),VEL(5)
        WRITE(53,*) 'N:',ACC(6),VEL(6)
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
1000  RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                   CCCCC
C                                                                       C
C    This subroutine enable multiple Runge-Kutta evaluation
c    for every time step on the PRIME.  It utilizes subroutines
c    RUNGE, DERIV, and DEBUG.
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC            VARIABLE IDENTIFICATION                 CCCCCCCC
C                                                                       C
c      VEL - plane's velocity vector
c TEMPVEL - intermediate velocity vector
c      DTN - modified time step
c       DT - time step from real-time clock
c  NUMINT - number of integration steps per time loop
C
C----- COMMON BLOCKS -----
C
C      INTECM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC             ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                       C
      SUBROUTINE REPEAT(IFLG)
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>INTECM
C
      REAL TEMPVEL(6),DTN
      INTEGER I,IFLG,J
C
      INTRINSIC FLOAT
```

```
      EXTERNAL RUNGE,PSGRAF,PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CICCCCCCCC            INITIALIZATION BLOCK                 BLOCK 0100
C                                                                       C
C
      DO 10 I = 1,6
           TEMPVEL(I) = VEL(I)
10    CONTINUE
C
      DTN = DT/FLOAT(NUMINT)
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                        BLOCK 0200
C                                                                       C
      DO 20 J = 1,NUMINT
           CALL RUNGE(TEMPVEL,DTN,IRATE,IFLG)
C
C<<<<< PROMPT THE E-S TO SEND CONTROL AND POSITION INFO >>>>>
C<<<<<      AFTER THE FIRST EVALUATION OF VELOCITY       >>>>>
C
           IF(J.EQ.1) THEN
                CALL PSGRAF
                   PRINT *,'SEND TRUE TO <1>S_PSPED;'
                CALL PSTERM
           ENDIF
20    CONTINUE
C
C<<<<< CALCULATE THE AVERAGE VELOCITY OVER TIME STEP >>>>>
C
      DO 40 I = 1,6
           VEL(I) = TEMPVEL(I)
40    CONTINUE
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
c
CDCCCCCCCC              SUBROUTINE DESCRIPTION                   CCCCC
C                                                                       C
C    This subroutine sets up the E-S to receive the plane
C    velocities and integrate them into position data.
C    The integration step uses the E-S CLFRAMES with a
C    time step of 1/30 of a second (the refresh rate).
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC            VARIABLE IDENTIFICATION                CCCCCCCC
C                                                                       C
C    POSCLK - timing function for the position integration
c    LINVEL - variable that contains the current linear velocity
c    ROTVEL - variable that contains the current rotational velocity
c    LINPOS - position integrator for linear position
c    ROTPOS - position integrator for rotational position
C
C----- COMMON BLOCKS -----
C
```

```
C     INTECM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC           ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                       C
      SUBROUTINE INTSET
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>INTECM
C
      REAL POSCAL
C
      INTRINSIC FLOAT
      EXTERNAL PSGRAF,PSTERM
C
C<<<<< SET THE SLOW MOTION INDEX FOR E-S CLOCK >>>>>
C
      POSCAL = 0.1/FLOAT(IRATE)
C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC           PROCESS BLOCK                          BLOCK 0200
C                                                                       C
C<<<<< SET UP CLOCK OUTPUT EVERY 1/10 SEC >>>>>
C
      CALL PSGRAF
      PRINT *,'POSCLK:=F:CLCSECONDS;'
      PRINT *,'SEND FIX(20) TO <1>POSCLK;'
      PRINT *,'SEND FIX(1) TO <2>POSCLK;'
      PRINT *,'SEND TRUE TO <3>POSCLK;'
      PRINT *,'SEND FIX(1) TO <4>POSCLK;'
      PRINT *,'SEND FIX(0) TO <5>POSCLK;'
      PRINT *,'SEND FALSE TO <6>POSCLK;'
C
C<<<<< VELOCITY VECTORS (linear and rotational) >>>>>
C
      PRINT *,'VARIABLE LINVEL;'
      PRINT *,'VARIABLE ROTVEL;'
C
      PRINT *,'SEND V3D(0.,0.,0.) TO <1>LINVEL;'
      PRINT *,'SEND V3D(0.,0.,0.) TO <1>ROTVEL;'
C
C<<<<< TRIGGER TO SEND VELOCITY TO POSITION INTE >>>>
C
      PRINT *,'S_LINVEL:=F:FETCH;'
      PRINT *,'S_ROTVEL:=F:FETCH;'
      PRINT *,'SEND ''LINVEL'' TO <2>S_LINVEL;'
      PRINT *,'SEND ''ROTVEL'' TO <2>S_ROTVEL;'
      PRINT *,'CONN POSCLK<2>:<1>S_LINVEL;'
      PRINT *,'CONN POSCLK<2>:<1>S_ROTVEL;'
C
C<<<<< SET UP POSITION INTEGRATORS >>>>>
C
      PRINT *,'LINPOS:=F:ACCUMULATE;'
      PRINT *,'SEND V(-700.,0.,100.) TO <2>LINPOS;'
      PRINT *,'SEND 0.001 TO <3>LINPOS;'
      PRINT *,'SEND ',POSCAL,' TO <4>LINPOS;'
      PRINT *,'SEND 10000. TO <5>LINPOS;'
      PRINT *,'SEND -10000. TO <6>LINPOS;'
C
      PRINT *,'ROTPOS:=F:ACCUMULATE;'
      PRINT *,'SEND V(0.,0.,0.) TO <2>ROTPOS;'
      PRINT *,'SEND 0.001 TO <3>ROTPOS;'
      PRINT *,'SEND ',POSCAL,' TO <4>ROTPOS;'
      PRINT *,'SEND 10000. TO <5>ROTPOS;'
      PRINT *,'SEND -10000. TO <6>ROTPOS;'
C
C<<<<< CONNECT VELOCITIES INTO ACCUMULATORS >>>>>
C
      PRINT *,'CONN S_LINVEL<1>:<1>LINPOS;'
```

```
      PRINT *,'CONN S_ROTVEL<1>:<1>ROTPOS;'
c
      CALL PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
      RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                       C
C     This subroutine performs the position integration using
c     the velocity calculation from REPEAT.  It also calculates
c     the velocity to be sent to the E-S for display.  A position
c     feedback loop to keep the E-S 'on track' with the simulation.
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC            VARIABLE IDENTIFICATION               CCCCCCCC
C                                                                       C
c        T1 - beginning value of current time step
c        T2 - ending value of current time step
c  U1,V1,W1 - linear velocity in fixed (earth) coordinates
c  P1,Q1,R1 - angular velocity in fixed (earth) coordinates
c  U2,V2,W2 - linear velocity sent to the E-S for integration
c  P2,Q2,R2 - angular velocity sent to the E-S for integration
c   LINGAIN - gain used for linear position feedback
c   ROTGAIN - gain used for angular position feedback
C
C----- COMMON BLOCKS -----
C
C     INTECM,ANGPCM,VELCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC             ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                       C
      SUBROUTINE INTE
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>INTECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ANGPCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
c
      REAL LINGAIN,ROTGAIN,U1,V1,W1,P1,Q1,R1,U2,V2,W2,P2,Q2,R2
      REAL TIME,COSPL,SINPL,COSPM,SINPM,TANPM,COSPN,SINPN
C
      INTRINSIC SIN,COS,TAN,FLOAT
      EXTERNAL READES,PSGRAF,PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                        BLOCK 0200
C                                                                       C
C<<<<< CALCULATE THE CURRENT TIME STEP >>>>>
C
      IF(T2.LT.T1) THEN
           TIME = T2 + 86400 - T1
      ELSE
           TIME = T2 - T1
      ENDIF
      PRINT *,'The time step for the step',TIME
C
```

```
C<<<<< READ BUFFER WITH CONTROL AND POSITION INFO FROM E-S >>>>>
C
      CALL READES
C
C<<<<< CALCULATE THE PLANE VELOCITIES IN WORLD COORDINATES >>>>>
C
C-----define SIN and COS of euler angles ----
C
      COSPL = COS(ESPL)
      SINPL = SIN(ESPL)
      COSPM = COS(ESPM)
      SINPM = SIN(ESPM)
      TANPM = TAN(ESPM)
      COSPN = COS(ESPN)
      SINPN = SIN(ESPN)
C
C----- velocities in right handed world coordinates -----
C
      U1 = VEL(1)*COSPM*COSPN + VEL(2)*(SINPL*SINPM*COSPN-COSPL*SINPN)
     % +  VEL(3)*(COSPL*SINPM*COSPN+SINPL*SINPN)
      V1 = VEL(1)*COSPM*SINPN + VEL(2)*(SINPL*SINPM*SINPN+COSPL*COSPN)
     % + VEL(3)*(COSPL*SINPM*SINPN-SINPL*COSPN)
      W1 = -VEL(1)*SINPM + VEL(2)*SINPL*COSPM + VEL(3)*COSPL*COSPM
      P1 = VEL(4) + VEL(5)*SINPL*TANPM + VEL(6)*COSPL*TANPM
      Q1 = VEL(5)*COSPL - VEL(6)*SINPL
      R1 = (VEL(5)*SINPL + VEL(6)*COSPL)/COSPM
C
C<<<<< CALCULATE THE VELOCITY FOR THE E-S >>>>>
C<<<<< CONVERTED TO LEFT HAND COORDINATES >>>>>
C<<<<<   AND POSITION FEEDBACK INCLUDED   >>>>>
C
      LINGAIN = 0.0
      ROTGAIN = 1.0
      U2 = U1 + LINGAIN*(PX-ESPX)/TIME
      V2 = V1 + LINGAIN*(PY-ESPY)/TIME
      W2 = -1.*(W1 + LINGAIN*(PZ-ESPZ)/TIME)
      P2 = -57.296*(P1 + ROTGAIN*(PL-ESPL)/TIME)
      Q2 = -57.296*(Q1 + ROTGAIN*(PM-ESPM)/TIME)
      R2 = 57.296*(R1 + ROTGAIN*(PN-ESPN)/TIME)
C
C<<<<<SEND VELOCITIES DOWN TO THE E-S >>>>>
C
      CALL PSGRAF
      PRINT *,'SEND V3D(',U2,',',V2,',',W2,') TO <1>LINVEL;'
      PRINT *,'SEND V3D(',P2,',',Q2,',',R2,') TO <1>ROTVEL;'
      CALL PSTERM
C
C<<<<< DISPLAY THE CURRENT VALUES OF ROLL PITCH AND YAW >>>>>
C
C      D1 = PL*57.296
C      D2 = PM*57.296
C      D3 = PN*57.296
C      D4 = ESPL*57.296
C      D5 = ESPM*57.296
C      D6 = ESPN*57.296
C      WRITE(1,999) D1,D4,D2,D5,D3,D6
C999   FORMAT(/'roll:',F7.3,1X,F7.3,5X,'pitch:',F7.3,1X,F7.3,
C     %10X,'yaw:',F7.3,1X,F7.3)
C
C<<<<< INTEGRATE THE PRIME VELOCITY TO POSITION >>>>>
C
      PX = U1*DT/FLOAT(IRATE) + PX
      PY = V1*DT/FLOAT(IRATE) + PY
      PZ = W1*DT/FLOAT(IRATE) + PZ
      PL = P1*DT/FLOAT(IRATE) + PL
      PM = Q1*DT/FLOAT(IRATE) + PM
      PN = R1*DT/FLOAT(IRATE) + PN
C
C<<<<< UPDATE THE TIME INFORMATION FOR INTEGRATIONS >>>>>
C
      DT = TIME
      T1 = T2
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                      C
c     This subroutine will input the flight configuration (i.e.
c     throttle and ctrl surface settings) for the force generation
c     routines to use.
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CVCCCCCCCC           VARIABLE IDENTIFICATION                 CCCCCCCC
C                                                                      C
c ELEVATOR - elevator accumulator
c   ECSA   - elevator angle (real)
c   EOUT   - elevator angle (string)
c AILERON  - aileron accumulator
c   OWCSA  - aileron angle (real)
c    AOUT  - aileron angle (string)
c  RUDDER  - rudder accumulator
c    RCSA  - rudder angle (real)
c    ROUT  - rudder angle (string)
c THROTTLE - throttle accumulator
c    PSPED - throttle setting (real)
c    TOUT  - throttle setting (string)
C
C----- COMMON BLOCKS -----
C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CSCCCCCCC            ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                      C
      SUBROUTINE INPSET
C
      EXTERNAL PSGRAF,PSTERM
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CICCCCCCCC           INITIALIZATION BLOCK                 BLOCK 0100
C                                                                      C
      CALL PSGRAF
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CPCCCCCCCC           PROCESS BLOCK                        BLOCK 0200
C                                                                      C
c
c<<<<< SETTING UP ELEV AND AILERON >>>>>
c
      PRINT *,'ELEVATOR:=F:ACCUMULATE;'
      PRINT *,'SEND 0.0 TO <2>ELEVATOR;'
      PRINT *,'SEND 1. TO <3>ELEVATOR;'
      PRINT *,'SEND -135. TO <4>ELEVATOR;'
      PRINT *,'SEND 100.0 TO <5>ELEVATOR;'
      PRINT *,'SEND -100.0 TO <6>ELEVATOR;'
      PRINT *,'CONN DIALS<4>:<1>ELEVATOR;'
C
      PRINT *,'AILERON:=F:ACCUMULATE;'
      PRINT *,'SEND 0.0 TO <2>AILERON;'
      PRINT *,'SEND 1. TO <3>AILERON;'
      PRINT *,'SEND 100. TO <4>AILERON;'
      PRINT *,'SEND 100.0 TO <5>AILERON;'
      PRINT *,'SEND -100.0 TO <6>AILERON;'
      PRINT *,'CONN DIALS<6>:<1>AILERON;'
c
C<<<< LABEL AND VALUE TO AILERON OUTPUT >>>>>
c
      PRINT *,'EOUT:=F:PRINT;'
      PRINT *,'VARIABLE ECSA;'
      PRINT *,'CONN EOUT<1>:<1>ECSA;'
      PRINT *,'CONN ELEVATOR<1>:<1>EOUT;'
      PRINT *,'CONN EOUT<1>:<1>DLABEL8;'
      PRINT *,'SEND ''ELEVATOR'' TO <1>DLABEL4;'
```

```
C
C<<<<< ELEVATOR CONNECTION TO PRIME >>>>>
C
      PRINT *,'S_ECSA:=F:FETCH;'
      PRINT *,'SEND ''ECSA'' TO <2>S_ECSA;'
      PRINT *,'CONN S_ECSA<1>:<1>HOST;'
C
C<<<<< LABEL AND VALUE TO AILERON OUTPUT >>>>>
C
      PRINT *,'AOUT:=F:PRINT;'
      PRINT *,'VARIABLE OWCSA;'
      PRINT *,'CONN AOUT<1>:<1>OWCSA;'
      PRINT *,'CONN AILERON<1>:<1>AOUT;'
      PRINT *,'CONN AOUT<1>:<1>DLABEL2;'
      PRINT *,'SEND '' AILERON '' TO <1>DLABEL6;'
C
C<<<<< AILERON CONNECTION TO PRIME >>>>>
C
      PRINT *,'S_OWCSA:=F:FETCH;'
      PRINT *,'SEND ''OWCSA'' TO <2>S_OWCSA;'
      PRINT *,'CONN S_OWCSA<1>:<1>HOST;'
c
C<<<<< SETTING UP THROTTLE CONTROL >>>>>
c
      PRINT *,'THROTTLE:=F:ACCUMULATE;'
      PRINT *,'SEND 0. TO <2>THROTTLE;'
      PRINT *,'SEND 0. TO <3>THROTTLE;'
      PRINT *,'SEND 133.33333 TO <4>THROTTLE;'
      PRINT *,'SEND 100. TO <5>THROTTLE;'
      PRINT *,'SEND 0. TO <6>THROTTLE;'
      PRINT *,'CONN DIALS<1>:<1>THROTTLE;'
C
C<<<<< LABEL AND VALUE TO THROTTLE DIAL >>>>>
C
      PRINT *,'TOUT:=F:PRINT;'
      PRINT *,'VARIABLE PSPED;'
      PRINT *,'CONN TOUT<1>:<1>PSPED;'
      PRINT *,'CONN THROTTLE<1>:<1>TOUT;'
      PRINT *,'CONN TOUT<1>:<1>DLABEL5;'
      PRINT *,'SEND ''THROTTLE'' TO <1>DLABEL1;'
C
C<<<<< THROTTLE CONNECTION TO PRIME >>>>>
C
      PRINT *,'S_PSPED:=F:FETCH;'
      PRINT *,'SEND ''PSPED'' TO <2>S_PSPED;'
      PRINT *,'CONN S_PSPED<1>:<1>HOST;'
c
C<<<<< SETTING UP RUDDER CONTROL DIAL >>>>>
C
      PRINT *,'RUDDER:=F:ACCUMULATE;'
      PRINT *,'SEND 0. TO <2>RUDDER;'
      PRINT *,'SEND 1. TO <3>RUDDER;'
      PRINT *,'SEND 100. TO <4>RUDDER;'
      PRINT *,'SEND 100. TO <5>RUDDER;'
      PRINT *,'SEND -100. TO <6>RUDDER;'
      PRINT *,'CONN DIALS<7>:<1>RUDDER;'
C
C<<<<< LABEL AND VALUE TO RUDDER DIAL >>>>>
C
      PRINT *,'ROUT:=F:PRINT;'
      PRINT *,'VARIABLE RCSA;'
      PRINT *,'CONN ROUT<1>:<1>RCSA;'
      PRINT *,'CONN RUDDER<1>:<1>ROUT;'
      PRINT *,'CONN ROUT<1>:<1>DLABEL3;'
      PRINT *,'SEND ''RUDDER '' TO <1>DLABEL7;'
C
C<<<<< RUDDER CONNECTION TO PRIME >>>>>
C
      PRINT *,'S_RCSA:=F:FETCH;'
      PRINT *,'SEND ''RCSA'' TO <2>S_RCSA;'
      PRINT *,'CONN S_RCSA<1>:<1>HOST;'
```

```
C
C<<<<< TRANSMISSION LINK TO PRIME >>>>>
C
      PRINT *,'CONN S_PSPED<1>:<1>S_OWCSA;'
      PRINT *,'CONN S_OWCSA<1>:<1>S_ECSA;'
      PRINT *,'CONN S_ECSA<1>:<1>S_RCSA;'
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      CALL PSTERM
      RETURN
      END
C
C
C
C
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                       C
C     This subroutine sets up the function network to send
c     character information from the E-S to the terminal
c     screen.  It will pack the information from ten
c     different sources (throttle, aileron, elevator, rudder,
c     and six plane positions) separated by commas and ending
c     with a carriage return. It is triggered by sending
c     a message to <1>S_PSPED which is initialized in POSSET.
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC            VARIABLE IDENTIFICATION               CCCCCCCC
C                                                                       C
C     HOST  - adds a comma to a string variable
C LAST_VAL - adds a <cr> to the end of a string variable
C     PACK  - concatenates the variables into one string
C  TRIGGER - sends the string variable to the PRIME
C     INFO  - variable containing information sent from E-S to PRIME
C    S_INFO - function used to send INFO to PRIME
C     CLEAR - clears the string variable after it has been sent
C
C----- COMMON BLOCKS -----
C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CSCCCCCCC             ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                       C
      SUBROUTINE TOPRIM
C
      EXTERNAL PSGRAF,PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                        BLOCK 0200
C                                                                       C
      CALL PSGRAF
C
C<<<<< ALLOW FOR CHARACTER PACKING SEPARATED BY COMMAS >>>>>
C<<<<<              AND ENDING WITH A <cr>              >>>>>
C
C----- add a comma to the string -----
C
      PRINT *,'HOST:=F:CONCATENATEC;'
```

```
      PRINT *,'SEND '','' TO <2>HOST;'
C
C----- add a <cr> to the last string -----
C
      PRINT *,'LAST_VAL:=F:CONCATENATEC;'
      PRINT *,'SEND CHAR(13) TO <2>LAST_VAL;'
C
C----- pack the string variables -----
C
      PRINT *,'PACK:=F:CCONCATENATE;'
      PRINT *,'SEND '' '' TO <1>PACK;'
      PRINT *,'CONN PACK<1>:<1>PACK;'
      PRINT *,'CONN HOST<1>:<2>PACK;'
      PRINT *,'CONN LAST_VAL<1>:<2>PACK;'
C
C----- create trigger to send info -----
C
      PRINT *,'TRIGGER:=F:NOP;'
      PRINT *,'CONN LAST_VAL<1>:<1>TRIGGER;'
C
C----- write the string into a variable -----
C----- and send the info when requested -----
C
      PRINT *,'VARIABLE INFO;'
      PRINT *,'CONN PACK<1>:<1>INFO;'
      PRINT *,'S_INFO:=F:FETCH;'
      PRINT *,'CONN TRIGGER<1>:<1>S_INFO;'
      PRINT *,'SEND ''INFO'' TO <2>S_INFO;'
      PRINT *,'CONN S_INFO<1>:<1>HOSTOUT;'
C
C----- clear the packing order after variable is sent -----
C
      PRINT *,'CLEAR:=F:CONSTANT;'
      PRINT *,'CONN S_INFO<1>:<1>CLEAR;'
      PRINT *,'SEND '' '' TO <2>CLEAR;'
      PRINT *,'CONN CLEAR<1>:<1>PACK;'
C
      CALL PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                  C
C    This subroutine sets up the routine to send the plane
c    position to the PRIME.  The current plane position is
c    updated everytime a position integration takes place.
C                                                                  C
C                                                                  C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                  C
CVCCCCCCCC           VARIABLE IDENTIFICATION              CCCCCCCC
C                                                                  C
c   ANGLES - splits the angular position vector into components
c   LOCALE - splits the linear position vector into components
c    ROLL  - roll angle in world coordinates (string variable)
c    ROLL  - roll angle in world coordinates (string variable)
c    PITCH - pitch angle in world coordinates (string variable)
c    YAW   - yaw angle in world coordinates (string variable)
c    XOUT  - x position in world coordinates (string variable)
```

```
c     YOUT  - y position in world coordinates (string variable)
c     ZOUT  - z position in world coordinates (string variable)
c    ROLOUT - real to string conversion of roll angle
c    PITOUT - real to string conversion of pitch angle
c    YAWOUT - real to string conversion of yaw angle
c     XOUT - real to string conversionof x position
c     YOUT - real to string conversion of y position
c     ZOUT - real to string conversion of z position
c  S_ROLL - sends the current roll angle variable
c S_PITCH - sends the current pitch angle variable
c   S_YAW - sends the current yaw angle variable
c  S_XOUT - send the current x position variable
c  S_YOUT - send the current y position variable
c  S_ZOUT - send the current z position variable
C
C----- COMMON BLOCKS -----
C
C     ANGPCM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CSCCCCCCC           ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                       C
      SUBROUTINE POSSET
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ANGPCM
C
      EXTERNAL PSGRAF,PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC           PROCESS BLOCK                          BLOCK 0200
C
      CALL PSGRAF
C<<<<< SET UP OUTPUT OF PLANE POSITION >>>>>
C
      PRINT *,'ANGLES:=F:PARTS;'
      PRINT *,'CONN ROTPOS<1>:<1>ANGLES;'
C
C<<<<< INITIALIZE OUTPUT PORTS >>>>>
C
      PRINT *,'ROLOUT:=F:PRINT;'
      PRINT *,'CONN ANGLES<1>:<1>ROLOUT;'
      PRINT *,'PITOUT:=F:PRINT;'
      PRINT *,'CONN ANGLES<2>:<1>PITOUT;'
      PRINT *,'YAWOUT:=F:PRINT;'
      PRINT *,'CONN ANGLES<3>:<1>YAWOUT;'
C
C<<<<< DEFINE AND WRITE TO VARIABLE >>>>>
C
      PRINT *,'VARIABLE ROLL,PITCH,YAW;'
      PRINT *,'CONN ROLOUT<1>:<1>ROLL;'
      PRINT *,'CONN PITOUT<1>:<1>PITCH;'
      PRINT *,'CONN YAWOUT<1>:<1>YAW;'
C
C<<<<< SET UP TRIGGER TO SEND DATA >>>>>
C
      PRINT *,'S_ROLL:=F:FETCH;'
      PRINT *,'SEND ''ROLL'' TO <2>S_ROLL;'
      PRINT *,'S_PITCH:=F:FETCH;'
      PRINT *,'SEND ''PITCH'' TO <2>S_PITCH;'
      PRINT *,'S_YAW:=F:FETCH;'
      PRINT *,'SEND ''YAW'' TO <2>S_YAW;'
      PRINT *,'CONN S_RCSA<1>:<1>S_ROLL;'
      PRINT *,'CONN S_ROLL<1>:<1>S_PITCH;'
      PRINT *,'CONN S_PITCH<1>:<1>S_YAW;'
      PRINT *,'CONN S_ROLL<1>:<1>HOST;'
      PRINT *,'CONN S_PITCH<1>:<1>HOST;'
      PRINT *,'CONN S_YAW<1>:<1>HOST;'
C
C<<<<< SET UP OUTPUT OF PLANE POSITION >>>>>
```

```
C
      PRINT *,'LOCALE:=F:PARTS;'
      PRINT *,'CONN LINPOS<1>:<1>LOCALE;'
C
C<<<<< INITIALIZE OUTPUT PORTS >>>>>
C
      PRINT *,'XOUT:=F:PRINT;'
      PRINT *,'CONN LOCALE<1>:<1>XOUT;'
      PRINT *,'YOUT:=F:PRINT;'
      PRINT *,'CONN LOCALE<2>:<1>YOUT;'
      PRINT *,'ZOUT:=F:PRINT;'
      PRINT *,'CONN LOCALE<3>:<1>ZOUT;'
C
C<<<<< DEFINE AND WRITE TO VARIABLE >>>>>
C
      PRINT *,'VARIABLE XPOS,YPOS,ZPOS;'
      PRINT *,'CONN XOUT<1>:<1>XPOS;'
      PRINT *,'CONN YOUT<1>:<1>YPOS;'
      PRINT *,'CONN ZOUT<1>:<1>ZPOS;'
C
C<<<<< SET UP TRIGGER TO SEND DATA >>>>>
C
      PRINT *,'S_XPOS:=F:FETCH;'
      PRINT *,'SEND ''XPOS'' TO <2>S_XPOS;'
      PRINT *,'S_YPOS:=F:FETCH;'
      PRINT *,'SEND ''YPOS'' TO <2>S_YPOS;'
      PRINT *,'S_ZPOS:=F:FETCH;'
      PRINT *,'SEND ''ZPOS'' TO <2>S_ZPOS;'
      PRINT *,'CONN S_YAW<1>:<1>S_XPOS;'
      PRINT *,'CONN S_XPOS<1>:<1>S_YPOS;'
      PRINT *,'CONN S_YPOS<1>:<1>S_ZPOS;'
      PRINT *,'CONN S_XPOS<1>:<1>HOST;'
      PRINT *,'CONN S_YPOS<1>:<1>HOST;'
      PRINT *,'CONN S_ZPOS<1>:<1>LAST_VAL;'
C
      CALL PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                       C
C    This routine uses the PRIMOS subroutine TIMDAT to find the
c    current real time since midnight using the system clock.  It
c    converts the minutes to seconds and the ticks to fractions of
c    a second.
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CVCCCCCCCC           VARIABLE IDENTIFICATION              CCCCCCCC
C                                                                       C
c    SEC   - time in seconds since midnight returned
c    INFO  - arrary that holds the info returned from TIMDAT
C
C----- COMMON BLOCKS -----
C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
```

```
CSCCCCCCC            ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                      C
      SUBROUTINE TIME(SEC)
C
      REAL SEC
      INTEGER*2 INFO(28)
C
      INTRINSIC INTS
      EXTERNAL TIMDAT
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CPCCCCCCCC            PROCESS BLOCK                          BLOCK 0200
C                                                                      C
C<<<<< GET THE TIME FROM THE SYSTEM >>>>>
C
      CALL TIMDAT(INFO,INTS(28))
C
C<<<<< CONVERT TO SECONDS >>>>>
C
      SEC = INFO(4)*60. + INFO(5) + INFO(6)/330.
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                   CCCCC
C                                                                      C
c     This subroutine will read the information that has
c     been sent to the terminal via a function network.  The
c     routine will then convert the data to be consistant
c     with the data needed by the PRIME.
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CVCCCCCCCC            VARIABLE IDENTIFICATION                CCCCCCCC
C                                                                      C
c     PSPED - Throttle setting in %throttle
c     OWCSA - Aileron setting
c     ECSA  - Elevator setting
c     RCSA  - Rudder setting
c      ESPL - Plane position (roll) [in radians]
c      ESPM - Plane position (pitch) [in radians]
c      ESPN - Plane position (yaw) [in radians]
c      ESPX - Plane position (forward)
c      ESPY - Plane position (horizontal)
c      ESPZ - Plane position (vertical)
C    IRESET - reset flag to return to start of simulation
c     TRIM  - elevator offset added to stabilize for steady flight
C
C----- COMMON BLOCKS -----
C
C     CTRLCM,ANGPCM,INTECM
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CSCCCCCCC            ENTRY AND STORAGE BLOCK                BLOCK 0000
C                                                                      C
      SUBROUTINE READES
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>CTRLCM
```

```
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ANGPCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>INTECM
C
      REAL TRIM
C
      EXTERNAL PSGRAF,PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                          BLOCK 0200
C                                                                       C
C<<<<< READ THE INFO FROM THE BUFFER >>>>>
c
c--- if an error is detected the old control and position info
c--- is used and the reset flag is turned on
C
      READ (*,*,ERR=1000) PSPED,OWCSA,ECSA,RCSA,ESPL,ESPM,ESPN
     %,ESPX,ESPY,ESPZ
C
C<<<<< CONVERT DATA TO BE COMPATIABLE WITH PRIME >>>>>
C
      ESPL = -ESPL/57.296
      ESPM = -ESPM/57.296
      ESPN = ESPN/57.296
C     ESPX = ESPX
C     ESPY = ESPY
      ESPZ = -ESPZ
C
C<<<<< CONVERT INPUT DISPLAY VALUES TO TRUE CONTROL VALUES >>>>>
C
      TRIM = .01255661
C
      PSPED = PSPED*.4166667
      OWCSA = 2.*(OWCSA * .00034907)
      ECSA = 15.*(ECSA * .000029088) - TRIM
      RCSA = -15.*(RCSA * .00014544)
C
C<<<<< CHECK IF PLANE HAS CRASHED >>>>>
C
      IF(ESPZ.GE.1.1) THEN
           CALL PSGRAF
            PRINT *,'SEND FALSE TO <6>POSCLK;'
            PRINT *,'DISPLAY DEATH;'
            PRINT *,'REMOVE VIEW;'
           CALL PSTERM
           GOTO 1000
      ENDIF
      GOTO 2000
C
C<<<<< SET THE RESET FLAG AND STOP CLOCK >>>>>
C
1000  IRESET = .FALSE.
      CALL PSGRAF
         PRINT *,'SEND FALSE TO <6>POSCLK;'
      CALL PSTERM
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
2000  RETURN
      END
```

c

```
CDCCCCCCCC              SUBROUTINE DESCRIPTION                    CCCCC
C                                                                      C
C    This subroutine is used to set or reset the E-S initial
c    conditions to start a simuation over.  The initial
c    values are enterinteractively by the user.
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CVCCCCCCCC           VARIABLE IDENTIFICATION                 CCCCCCCC
C                                                                      C
c    refer to E-S subroutines POSSET,INPSET,INTSET AND SIM.WORLD
c    for definitions of the E-S variable names.
c    VEL   - velocity used in PRIME (plane coordinates)
c U1,V1,W1 - intermediate velocity in left hand coordinate system
c P1,Q1,R1 -      (plane coordinates)
c U2,V2,R2 - velocity sent to the E-S for integration
c P2,Q2,R2 -       (world coordinates)
c    PSPED - throttle setting
c    OWCSA - aileron setting
c    ECSA  - elevator setting
c    RCSA  - rudder setting
C
C----- COMMON BLOCKS -----
C
C     VELCM,INTECM,CTRLCM,ANGPCM
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CSCCCCCCC            ENTRY AND STORAGE BLOCK              BLOCK 0000
C                                                                      C
      SUBROUTINE RESET
C
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>VELCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>INTECM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>CTRLCM
$INSERT ID58>SU14>SIMULATE.DIR>INSERT.DIR>ANGPCM
C
      REAL TL(3,3),TR(3,3),U1,V1,W1,P1,Q1,R1
      REAL U2,V2,W2,P2,Q2,R2,POSCAL
      CHARACTER HOLD*1
C
      INTRINSIC SIN,COS,FLOAT,TAN
      EXTERNAL PSGRAF,PSTERM
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
CICCCCCCCC           INITIALIZATION BLOCK                 BLOCK 0100
C                                                                      C
C
C<<<<< INITIALIZE THE STARTING LOCATION OF THE PLANE
C
C      PRINT *,'  '
C      PRINT *,'The velocities and controls have been set to steady',
C     %' state values.'
C      PRINT *,'USE DECIMALS FOR INPUTTING POSITIONS!'
C      PRINT *,'  '
C1000  PRINT *,'What is the linear position in world coordinates?'
C      PRINT *,'        A good location is -1000.,-150.,100'
C      READ(*,*,ERR=1000) PX,PY,PZ
C2000  PRINT *,'What is the angular position in world coordinates?'
C      PRINT *,'          A good orientation is 0.,0.,90.'
C      READ(*,*,ERR=2000) PL,PM,PN
      PX  = -1000.
      PY = 0.
      PZ = 100.
      PL = 0.
      PM = 0.
      PN = 0.
C
C<<<<< INITIALIZE THE STEADY-STATE VELOCITY AND CONTROLS >>>>>
C
```

```
      VEL(1) = 214.177
      VEL(2) = 0.
      VEL(3) = 0.
      VEL(4) = 0.
      VEL(5) = 0.
      VEL(6) = 0.
      PSPED = 56.3262
      OWCSA = 0.0
      ECSA = 0.0
      RCSA = 0.0
      POSCAL =0.1/FLOAT(IRATE)
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
CPCCCCCCCC            PROCESS BLOCK                          BLOCK 0200
C                                                                       C
C
C<<<<< SEND THE RESET VALUES TO THE APPROPRIATE E-S VARIABLES >>>>>
C
      CALL PSGRAF
c
c----- reset E-S screen -----
c
      PRINT *,'REMOVE VIEW;'
      PRINT *,'REMOVE DEATH;'
      PRINT *,'DISPLAY VIEW;'
c
c----- reset slow motion factor on E-S -----
c
      PRINT *,'SEND ',POSCAL,' TO <4>LINPOS;'
      PRINT *,'SEND ',POSCAL,' TO <4>ROTPOS;'
c
c--- reset plane position
c
      PRINT *,'SEND ''',PX,''' TO <1>XPOS;'
      PRINT *,'SEND ''',PY,''' TO <1>YPOS;'
      PRINT *,'SEND ''',PZ,''' TO <1>ZPOS;'
      PRINT *,'SEND ''',PL,''' TO <1>ROLL;'
      PRINT *,'SEND ''',PM,''' TO <1>PITCH;'
      PRINT *,'SEND ''',PN,''' TO <1>YAW;'
      PRINT *,'SEND V3D(',PX,',',PY,',',PZ,') TO <2>LINPOS;'
      PRINT *,'SEND V3D(',PL,',',PM,',',PN,') TO <2>ROTPOS;'
      PRINT *,'SEND V3D(',PL,',',PM,',',PN,') TO <2>DELTA_ROT;'
      PRINT *,'SEND V3D(',PX,',',PY,',',PZ,') TO <1>TRANSCAL;'
      PRINT *,'SEND ',PL,' TO <1>LSCAL;'
      PRINT *,'SEND ',PM,' TO <1>MSCAL;'
      PRINT *,'SEND ',PN,' TO <1>NSCAL;'
      PRINT *,'SEND ',PN,' TO <1>PLANE_RESET;'
c
c--- reset plane controls
c
      PRINT *,'SEND ''0.0'' TO <1>RCSA;'
      PRINT *,'SEND ''0.0'' TO <1>ECSA;'
      PRINT *,'SEND ''0.0'' TO <1>OWCSA;'
      PRINT *,'SEND ''56.3262'' TO <1>PSPED;'
      PRINT *,'SEND ''0.0'' TO <1>ROUT;'
      PRINT *,'SEND ''0.0'' TO <1>EOUT;'
      PRINT *,'SEND ''0.0'' TO <1>AOUT;'
      PRINT *,'SEND ''56.3262'' TO <1>TOUT;'
      PRINT *,'SEND ',RCSA,' TO <2>RUDDER;'
      PRINT *,'SEND ',ECSA,' TO <2>ELEVATOR;'
      PRINT *,'SEND ',OWCSA,' TO <2>AILERON;'
      PRINT *,'SEND ',PSPED,' TO <2>THROTTLE;'
      CALL PSTERM
C                                                                       C
C<<<<< CONVERT DEGREES TO RADIANS >>>>>
C
      PL = PL/57.296
      PM = PM/57.296
      PN = PN/57.296
C
```

```
C<<<<< CONVERT RIGHT HAND COORDINATE TO LEFT HAND COORDINATES >>>>>
C
      U1 = VEL(1)
      V1 = VEL(2)
      W1 = -VEL(3)
      P1 = -VEL(4)
      Q1 = -VEL(5)
      R1 = VEL(6)
C
C<<<<< CALCULATE THE VELOCITIES IN FIXED COORDINATES >>>>>
C<<<<<       FROM PLANE COORDINATES        >>>>>
C
C--- LINEAR ---
C
      TL(1,1) = COS(PM)*COS(PN)
      TL(2,1) = COS(PM)*SIN(PN)
      TL(3,1) = -SIN(PM)
C
      TL(1,2) = SIN(PL)*SIN(PM)*COS(PN) - COS(PL)*SIN(PN)
      TL(2,2) = SIN(PL)*SIN(PM)*COS(PN) + COS(PL)*COS(PN)
      TL(3,2) = SIN(PL)*COS(PM)
C
      TL(1,3) = COS(PL)*SIN(PM)*COS(PN) + SIN(PL)*SIN(PN)
      TL(2,3) = COS(PL)*SIN(PM)*SIN(PN) - SIN(PL)*COS(PN)
      TL(3,3) = COS(PL)*COS(PM)
      U2 = U1*TL(1,1) + V1*TL(1,2) + W1*TL(1,3)
      V2 = U1*TL(2,1) + V1*TL(2,2) + W1*TL(2,3)
      W2 = U1*TL(3,1) + V1*TL(3,2) + W1*TL(3,3)
C
C--- ROTATIONAL ---
C
      TR(1,1) = 1.
      TR(1,2) = SIN(PL)*TAN(PM)
      TR(1,3) = COS(PL)*TAN(PM)
      TR(2,1) = 0.
      TR(2,2) = COS(PL)
      TR(2,3) = -SIN(PL)
      TR(3,1) = 0.
      TR(3,2) = SIN(PL)/COS(PM)
      TR(3,3) = COS(PL)/COS(PM)
      P2 = P1*TR(1,1) + Q1*TR(1,2) + R1*TR(1,3)
      Q2 = P1*TR(2,1) + Q1*TR(2,2) + R1*TR(2,3)
      R2 = P1*TR(3,1) + Q1*TR(3,2) + R1*TR(3,3)
C
C<<<<< CONVERT POSITIONS FROM LHCS TO RHCS >>>>>
C
      PX = PX
      PY = PY
      PZ = -PZ
      PL = -PL
      PM = -PM
      PN = PN
      ESPX = PX
      ESPY = PY
      ESPZ = -PZ
      ESPL = -PL
      ESPM = -PM
      ESPN = PN
      POS(1) = PX
      POS(2) = PY
      POS(3) = PZ
      POS(4) = PL
      POS(5) = PM
      POS(6) = PN
C
C<<<<< CONVERT CONTROLS TO FORCE CALCULATION UNITS >>>>>
C
      PSPED = 20.0926
      ECSA = -0.007994611
C
C<<<<< SET THE RESET FLAG >>>>>
```

```
C
      IRESET = .TRUE.
C
C<<<<< WAITING FOR PROMPT -- DEMO STALL >>>>>
C
      PRINT *,'Grab the control box and let''s storm the barn!'
      PRINT *,'                Hit <cr> to GO'
      READ(*,'(A)') HOLD
c
C---reset plane velocity and start position update on the E-S -----
c
      CALL PSGRAF
      PRINT *,'SEND V3D(',U2,',',V2,',',W2,') TO <1>LINVEL;'
      PRINT *,'SEND V3D(',P2,',',Q2,',',R2,') TO <1>ROTVEL;'
      PRINT *,'SEND TRUE TO <6>POSCLK;'
      CALL PSTERM
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```