OMV--A SIMPLIFIED MATHEMATICAL MODEL OF THE ORBITAL MANEUVERING VEHICLE

NASA-CR-171123
19840021833

Kenneth E. Johnson
Environmental and Energy Center
The University Of Alabama
In Huntsville

August 1984
OMV — A Simplified Mathematical Model
Of The
Orbital Maneuvering Vehicle
Interim Report

Prepared for
George C. Marshall Space Flight Center
Marshall Space Flight Center
Huntsville, AL 35812

by
Dr. William Teoh
Kenneth Johnson Energy & Environment Center
University of Alabama in Huntsville
Huntsville, AL 35899

Date prepared: August 9, 1984

Contract # NAS8-35670

Abstract

A model of the Orbital Maneuvering Vehicle is presented. In this model, several simplifications have been made. A set of hand controller signals may be used to control the motion of the OMV.

Model verification is carried out using a sequence of tests. The dynamic variables generated by the model is compared, whenever possible, with the corresponding analytical ones. The result of the tests show conclusively that the present model is behaving correctly. Further, this model interfaces properly with the State Vector Transformation module (SVX) developed previously. Correct command sequences are generated by the OMV and SVX system, and these command sequences can be used to drive the flat floor simulation system here at MSFC.
INTRODUCTION

This report discusses the design and implementation of OMV -- a mathematical model of the Orbital Maneuvering Vehicle. To avoid confusion, the term OMV shall be used to mean the mathematical model as well as the software that performs the modelling, while the full term "Orbital Maneuvering Vehicle" shall be used to mean the actual flight hardware.

The Orbital Maneuvering Vehicle can be maneuvered by remote operator control. Its motion is completely specified by its equations of motion. The solution of the equations of motion yields its position $[X,Y,Z]^T$, velocity $[\dot{X},\dot{Y},\dot{Z}]^T$, orientation $[r,p,y]^T$ and their rates $[\ddot{r},\ddot{p},\ddot{y}]^T$ where $r$, $p$ and $y$ stand for roll, pitch and yaw respectively. From these dynamic quantities, a 14-component state vector can be generated. This state vector contains all the necessary information to completely specify the state of the vehicle in space at any time.

The OMV simulates the motion of the Orbital Maneuvering Vehicle in space. OMV is a software subsystem that is an integral part of the software system used to drive the MSFC flat floor simulation system. In this installation, a set of hand controllers is used to maneuver the OMV (Mathematical model) and the state vector obtained is used as input to a second software module called SVX (the State Vector Transformation module) which transforms it to a suitable set of commands to be transmitted to, and thereby controlling the mobile base on the flat floor. The overall relation is as shown in Figure 1. As can be seen in this
figure, the OMV module encompasses the vehicle response module as well as the orbital mechanics module. In order to optimize execution speed, these two modules are not implemented as separate entities.

The State Vector Transformation Module has been discussed elsewhere (see Reference 1) and will not be elaborated here. Throughout this report, it is important to bear in mind that the OMV simulates the motion of the Orbital Maneuvering vehicle but otherwise has no physical relationship with the Orbital Maneuvering Vehicle. The mobile base on the flat floor will attempt to move in such a manner that a mock-up module mounted on it will replicate the motion of the Orbital Maneuvering Vehicle, using a set of commands derived from the state vectors generated by OMV. Otherwise the mobile base is not related to the OMV. The mock-up module is not the Orbital Maneuvering Vehicle. One of the objectives of the flat floor system is to simulate docking of the OMV with a target vehicle.

THE OMV MODEL

This report describes a simplified mathematical model of the Orbital Maneuvering Vehicle. A more detailed model is being developed elsewhere at MSFC. In the present model, several simplifications and assumptions have been made. The objective is to develop quickly (and hence the simplification) a model that can be used to drive the flat floor system.

Before discussing the model in any detail, it is necessary
to define the various coordinate systems used in this work.

A. The Local Vertical Frame (LVF)

Imagine a spacecraft in an orbit around the earth. It is immaterial whether this is the Orbital Maneuvering Vehicle or the target vehicle. LVF is a coordinate system with its origin at the center of mass of this spacecraft such that Z-axis lies in the plane of the orbit and is directed away from the center of the earth. The Y-axis is chosen to be parallel to the orbital angular momentum vector and X-axis is tangential to the orbit as shown in Figure 2. The position, velocity as well as orientation of the second vehicle are described in LVF and is therefore relative to the orbiting vehicle. Throughout this work, we shall assume that the target vehicle is the orbiting vehicle.

B. OMV Body Frame

This is a body fixed reference frame with its origin fixed at the center of mass of the OMV, and its axes will be denoted by 1, 2 and 3 respectively. Initially, at the start of the simulation, 1, 2 and 3 axes line up with X, Y and Z axes respectively. As can be seen from Figure 3, the axis of symmetry is the 1-axis.

In order to construct the model of the Orbital Maneuvering Vehicle, the following assumptions are made:

1. The OMV is assumed to be a circular disk of constant mass and having a uniform mass distribution. This assumption may seem unreasonable at first glance, but one quickly realizes
that the detail shape of the OMV is unimportant as long as one knows the mass and propulsion characteristics of the Orbital Maneuvering Vehicle. In the present model, the mass characteristics are summarized in Table 1. These figures are taken from the MSFC Preliminary Definition Studies (see Reference 2).

2. The OMV is manipulated using signals from a set of hand controllers. These signal can be classified into two groups. The first group is used to simulate a force acting through the center of mass of the OMV. In other words, one can, from this group of signals, generate an acceleration vector \( \mathbf{a} = [a_1, a_2, a_3]^T \) in the body frame. The other group of signals simulates rotations about 1, 2 and 3 axes, namely, a vector \( \mathbf{w} = [w_1, w_2, w_3]^T \). Assumptions 1 and 2 mean that detailed knowledge of the shape, thrust level and placement of the thruster and so forth are not really needed. The present control mode is the only mode implemented.

3. Circular orbits are assumed. The altitude of the orbit can be anything from 150 to 1500 nautical miles which is the designed operating range of the Orbital Maneuvering Vehicle.

4. Orbital mechanics is an important part in describing the motion of the OMV and is therefore implemented. Other secondary perturbation effects are totally ignored.

5. The state of the OMV is computed and updated 10 times per second. The period of 0.1 second will be referred to as a major cycle throughout this report.
The equations of motion of the OMV can be discussed in terms of the rotational part and translational part.

**Rotational Equations of Motion**

The rotational equation of motion can be written as :

$$\tau = \dot{L}$$

where $L = Iw$ is the angular momentum vector and $\tau$ is the applied torque. $I$ is the moment of inertia tensor and $w$ is the body rate. The solution can be drastically simplified by choosing the body axes 1, 2 and 3 such that $I$ is diagonal (Please see References 3 and 4), that is:

$$I = \begin{bmatrix}
I_{11} & 0 & 0 \\
0 & I_{22} & 0 \\
0 & 0 & I_{33}
\end{bmatrix}$$

Remember that $w = [w_1, w_2, w_3]^T$ is obtained from the hand controller signals. The solution of the rotational equations of motion yields $\phi$, $\theta$ and $\psi$ the three Euler angles. The order and sense of rotation is chosen in the conventional manner (Please see Reference 5), that is:

$$[\phi]_1[\theta]_2[\psi]_3$$

To reduce computational overhead, quaternions are used to specify the attitude of the OMV rather than the Euler angles themselves. It has been proven that the two representations are exactly equivalent (Reference 6). A quaternion $q$ may be written as:
An object whose attitude is described by the three Euler angles relative to some reference frame can be treated as a single rotation by $\alpha$ about an Euler axis $E = [E_1, E_2, E_3]^T$. Theory has shown that this is the shortest angular path (Reference 7) in the sense that $\alpha$ is less than the algebraic sum of $\phi$, $\theta$ and $\psi$. The angle $\alpha$ and the Euler axis can be expressed in terms of the quaternion $q$ as:

$$
\cos \frac{\alpha}{2} = q_4
E = \frac{(iq_1 + jq_2 + kq_3)}{(q_1 + q_2 + q_3)}^{1/2}
$$

Since the attitude control system of the OMV can control the roll, pitch and yaw axis independently, we expect the roll, pitch and yaw $[r, p, y]^T$ to be proportional to the respective components of $E$ (Reference 7). In fact, the following relation holds:

$$
[r, p, y]^T = [\alpha E_x, \alpha E_y, \alpha E_z]^T
$$

Quaternion algebra leads to further computational economy when successive rotations need to be calculated. Let say, at any instant, the attitude of the OMV is specified by the quaternion $q_1$ relative to some non-rotating frame. Suppose further that an instant later, the vehicle's attitude has changed, having rotated by $\phi$, $\theta$ and $\psi$. These angular displacements are measured relative to the rotated body frame. If the new attitude is described
by a second quaternion $q_2$, the attitude of the vehicle, relative to the non-rotating frame (References 8, 9) is then given by

$$q = q_1 q_2$$

This is an important advantage because if at the beginning of the simulation, the body frame is aligned with the LVF (as specified by the quaternion $q_0 = [0,0,0,1]^T$), then the attitude of the OMV relative to the LVF, after $n$ successive rotations is simply:

$$q = q_0 q_1 q_2 \ldots q_n$$

Of course, the attitude of the vehicle after the $n+1$-th rotation is $q = q_n q_{n+1}$. Thus, the attitude of the vehicle can be computed from the previous quaternions. This recursive property gives rise to quite a computational advantage, especially since there are only four elements in a given quaternion versus the nine elements of a direction cosine matrix.

**TRANSLATIONAL EQUATION OF MOTION**

The translational equations of motion has been derived in detail in Appendix I, and will not be repeated here. In essence, we seek solutions to a set of three simultaneous, coupled second order differential equations of the form:

$$\ddot{X} = A_x - 2\omega \dot{Z}$$

$$\ddot{Y} = A_y - \omega^2 Y$$

$$\ddot{Z} = A_z + 2\omega \dot{X} + 3\omega^2 Z$$

Here, the position and velocity vectors $[X,Y,Z]^T$ and $[\dot{X},\dot{Y},\dot{Z}]^T$
refer to the position and velocity of the OMV relative to the target vehicle, as expressed in Local Vertical Frame. \( \omega \) is the orbital velocity, and \( \mathbf{A} = [A_x, A_y, A_z]^T \) is the linear acceleration vector in LVF. Remember that the hand controller signals give rise to an acceleration vector \( \mathbf{a} = [a_1, a_2, a_3]^T \) in OMV body frame. Thus, one can obtain \( \mathbf{A} \) from \( \mathbf{a} \) using the transformation:

\[
\mathbf{A} = \mathbf{C}^{-1} \mathbf{a}
\]

where \( \mathbf{C}^{-1} \) is the inverse of the direction cosine matrix which can be derived from the quaternion \( \mathbf{q} = [q_1, q_2, q_3, q_4]^T \) as:

\[
\mathbf{C}^{-1} = \begin{bmatrix}
q_4 + q_1 - q_2 - q_3 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\
2(q_1q_2 + q_3q_4) & q_4 - q_1 + q_2 - q_3 & 2(q_2q_3 - q_1q_4) \\
2(q_1q_3 - q_2q_4) & 2(q_2q_3 + q_1q_4) & q_4 - q_1 - q_2 + q_3
\end{bmatrix}
\]

It is obviously impractical to seek an analytical solution to the translational equations of motion. Numerical methods must be used. In the present work, the Adam-Bashforth method is used. For this purpose, each major cycle is subdivided into \( N \) (normally 10, but see later section) sub-intervals, each of which will be referred to as a minor cycle. It is necessary that the acceleration vector \( \mathbf{A} \) be computed for each minor cycle, and stored in an acceleration matrix. At the end of \( N \) minor cycles, this acceleration matrix is used to obtain the numerical solution for the entire major cycle. A 14-component state vector is then assembled, and their components are listed below:

\[
\begin{align*}
S(1) - S(3) & \quad \text{relative position vector in LVF} \\
S(4) - S(6) & \quad \text{relative velocity vector in LVF} \\
S(7) - S(9) & \quad \text{angular momentum vector in LVF}
\end{align*}
\]
The angular momentum vector in LVF can be deduced as follows. Since the body rate \( \mathbf{w} = [w_1, w_2, w_3]^T \) is known, one can calculate \( \mathbf{L}_B \) in body frame using the relation (Reference 10):

\[
\mathbf{L}_B = \mathbf{I} \mathbf{w} \\
\mathbf{L} = \mathbf{C}^{-1} \mathbf{L}_B
\]

where \( \mathbf{C}^{-1} \) is the inverse of the direction cosine matrix.

The state vector serves as input to the State Vector Transformation module (SVX). This module has been designed and implemented (Reference 1) and will not be repeated here. For completeness, a copy of the updated report is included in Appendix 2.

System Design and Implementation

The design and implementation of the present system is best discussed in the following sub-sections:

A) Hand Controllers

The hand controllers allow the operator to manipulate the Orbital Maneuvering Vehicle in terms of translation and attitude. In the present system, hand controller signals are used to maneuver the OMV. The hardware is configured to provide 12 bits of information. The first 6 bits pertain to translation, while the remaining 6 bits pertain to attitude control. During development, the 12 bits are simulated by reading them from a disk file (HNDSGL.DAT) as 12 single digit integers. This process is carried
out in a subprogram called HNDCTL. In actual implementation, this subprogram must be replaced by a suitable device driver.

The bit assignment is shown in Table 2. It will be noted that 1 will be used to denote the "on" state while 0 will be used to denote the "off" state. The subroutine HNDCTL contains sufficient logic to ensure that when both bits assigned to a given axis are on, they will be treated as both off (that is, no acceleration along, or rotation about, that axis) to conserve fuel usage. The main purpose of this subroutine is to examine the 12 bits from the hand controllers and return two vectors $a$ and $w$ where

$$a = [a_1, a_2, a_3]^T \quad \text{and} \quad w = [w_1, w_2, w_3]^T$$

whose meaning have been explained in the previous section. It is important to remember that both $a$ and $w$ are expressed in the OMV body frame.

Ideally, the hand controllers signals should be sensed and updated every minor cycle. But because of timing considerations they will be sensed once every major cycle, and it is explicitly assumed that the bit states do not change during the entire major cycle. This is not an unreasonable assumption, since one major cycle is 0.1 second, which is in the neighbourhood of the average reaction time of the human operator. Besides, the OMV does not have a fast response because of its large mass and low thrust levels.

The acceleration vector $a$ must be expressed in LVF before it
can be used in solving the equations of motion. In the OMV software, this is carried out as mentioned previously by:

a) calculating the inverse of the direction cosine matrix \( C^{-1} \),

b) transforming the vector \( \mathbf{a} \) to \( \mathbf{A} \) in LVF, and
c) placing \( \mathbf{A} \) in an acceleration matrix \( \mathbf{AA} \).

Step a) is carried out by a subroutine called DCSINV while steps b) and c) are carried out by subroutines DMUL and STORE in subroutine MOTION. At the end of the \( N \) minor cycles, the subroutine SOLVE is invoked to obtain solutions to the equations of motion numerically.

B) Numerical Solutions:

A three step Adam-Bashforth method (References 11-14) is used to obtain solutions to the equations of motion. This method is well known, and will not be elaborated here. Essentially, the set of three coupled second differential equations are re-written as a set of six simultaneous first order differential equations, and the solution computed. The six initial conditions needed for the computation are provided by the six components of the relative position and velocity vectors. Subroutine SOLVE takes the relative displacement and velocity vectors as initial conditions of the previous major cycle, and returns the new position and velocity vectors. A subroutine called STATE is then invoked to assemble the state vector.

C) Output Section:

A subroutine called OUTPUT is responsible for conveying information to the outside world. In normal operations, no output
is generally expected, but during testing, it is necessary to be able to monitor the progress of the simulation. At present, one can, via the use of flags, control the form and type of output. By way of example, one can request OMV to print a time sequence of state vectors at 1 second intervals on the printer, or display the position and orientation of the mobile base (on the flat floor) graphically, or disable all outputs altogether.

A fairly simple graphics package called PLOT is implemented to provide graphics output. This package is developed for the initial software checking only; namely to provide the operator with some form of visual output. It must be emphasized that this package is hardware dependent, and is not compatible with the PDP 11/34 mini-computer. The present graphics package runs on an IBM Personal Computer fitted with a TECMAR GRAPHICS MASTER board and an IBM monochrome monitor. A resolution of 640 by 352 is used for the package, although the system has a potential resolution of 720 by 700 pixels (Reference 15). PLOT uses escape codes to generate the top or side view of the mobile base (including the mock up module). A listing of this package, written in FORTRAN 77, is included in Appendix 3. It is anticipated that this package can be modified to run on the Evans and Sutherland color graphics terminal driven by a VAX 780.

The entire OMV module is written in FORTRAN 77, and all floating point computations are carried out in double precision. The usual structured programming technique is used. Modular design is faithfully adhered to, so that subroutines can be easily updated or replaced. At times, efficiency may be sacrificed for
code clarity, thereby making the code much easier to maintain and modify. During the design phase, flexibility is emphasized. Model parameters are inputted from disk files. Thus, modifications on the flat floor system will not involve any changes to the OMV source code. Appendix 4 shows the various data files used. Explanations for the various quantities are included as part of the record so that one can easily modify the configuration, initial conditions and so forth without having to refer to the source listing. A complete listing of OMV is included in Appendix 5, and a hierarchial chart is shown in Figure 4.

V) Testing and Results

Initial testing of the OMV software is conducted using an IBM Personal Computer with 8087 arithmetic co-processor. The same source code without the graphics option has been uploaded to the PDP 11/34 at MSFC and executed successfully.

The nature of the model is such that the major source of error would arise from the numerical solutions of the equations of motion. Thus, much effort has been spent to ensure that the Adam-Bashforth method yields accurate results. An error analysis of this method shows that the error is of the order of $h^5$ where $h$ is the step size. In the present work, the step size is typically 0.01. This, coupled with the fact that all computations are carried out in double precision, means that the expected truncation error is of the order of $10^{-10}$ -- a figure that is too good to be true.

The following tests were conducted to verify that this
method does indeed give accurate solutions. The homogeneous case is first considered. Physically, this corresponds to the situation where the operator leaves all the controls in neutral so that

\[ a = [0,0,0]^T \quad \text{and} \quad w = [0,0,0]^T \]

Thus, the equations of motion reduce to:

\[
\begin{align*}
\ddot{x} &= -2 \omega \dot{z} \\
\ddot{y} &= -\omega^2 \dot{y} \\
\ddot{z} &= 2 \omega \dot{x} + 3 \omega^2 z
\end{align*}
\]

This set of equations can be solved numerically using the Adam-Bashforth method. Further, if \( X_1, X_2, X_3 \) and \( V_1, V_2, V_3 \) are the initial conditions, it can be shown that the analytical solutions are:

\[
\begin{align*}
X(t) &= X_1 - \frac{(3 \Omega t - 4 \sin \Omega t)}{\Omega} V_1 - 6(\Omega t - \sin \Omega t) X_3 - \frac{(1 - \cos \Omega t)}{\Omega} V_3 \\
Y(t) &= (\cos \Omega t) X_2 + \frac{\sin \Omega t}{\Omega} V_2 \\
Z(t) &= -\Omega (\sin \Omega t) X_2 + (\cos \Omega t) V_2 \\
\end{align*}
\]

\[
\begin{align*}
Y(t) &= \frac{2(1-\cos \Omega t)}{\Omega} V_1 + (4-3\cos \Omega t) X_3 + \frac{\sin \Omega t}{\Omega} V_3
\end{align*}
\]
\[ \dot{Z}(t) = 2(\sin nt) V_1 + 3n(\sin nt) X_3 + (\cos nt) V_3 \]

Thus, the numerical solutions can be compared directly with the analytical ones. Here, \( \Omega \) is the orbital velocity, and for a circular orbit, \( \Omega \) can be calculated:

\[ \Omega = \frac{G M_e}{(R_o + H)^3} \]

where \( G \) is the universal gravitation constant, \( M_e \) is the mass of the earth, \( R_o \) is the mean earth radius and \( H \) is the altitude. Note that at higher orbits, \( \Omega \) approaches 0 and the equations of motion approach

\[
\begin{align*}
\ddot{X} & \rightarrow 0 \\
\ddot{Y} & \rightarrow 0 \\
\ddot{Z} & \rightarrow 0
\end{align*}
\]

and better agreement between numerical and analytical results are expected for high altitudes than lower orbits. A computer program called ADAM has been developed that would, given a set of initial conditions, calculate both the numerical and analytical solutions to the equations of motion. The source listing of ADAM is shown in Appendix 5. In the present set of tests, an altitude of 200 kilometers (\( \Omega = 0.00118 \text{ rad/sec} \)) is used throughout. This altitude represents the lowest design orbit of the Orbital Maneuvering Vehicle. Table 3 shows a comparison between the analytical and numerical solutions at this altitude, using the initial conditions:
\( X_1 = 0, \quad X_2 = X_3 = 0 \)
\( V_1 = 0.05, \quad V_2 = V_3 = 0 \)

The result shows that the two solutions agree to better than \( 3 \times 10^{-8} \) in 60 minutes, or about 0.03 millimeters. This figure is well below the expected accuracy of the flat floor simulation system. This surprisingly small error comes from the fact that the angular velocity \( \Omega \) is quite small. When \( \Omega = 1.0 \) is used, (this angular frequency does not make sense physically, as it represents an orbit well below the earth's surface, but constitutes a valid situation mathematically), the errors propagate quite fast as to render the comparison meaningless after 10 minutes.

A second test was carried out at the same altitude, using null initial conditions:
\( X_1 = X_2 = X_3 = 0 \)
\( V_1 = V_2 = V_3 = 0 \)

The hand controller signals were chosen to yield a constant acceleration along the X-axis in the LVF, that is \( a = [0.025, 0, 0]^T \), and the orientation of the OMV is chosen to be aligned to the LVF at \( t = 0 \). The result after 4 seconds of simulation is shown in Table 4. A plot of the relevant dynamic variables as a function of time is shown in Figure 5. The result shows that the model behaves exactly as expected; namely that an acceleration along the X-axis gives rise to a Z component, as dictated by orbital mechanics. If we ignore the Z contribution for the time being, one can estimate the value of \( X \) and \( \dot{X} \) using Newton's laws (this is not an invalid estimate as the time inter-
val is quite short compared with the period of rotation to be \( X = 0.2 \) meters, and \( X = 0.1 \) meter/sec respectively. These figures compare very favourably with the numerical results at \( t = 4 \) seconds.

A very interesting test was conducted in which the OMV is made to execute a pure pitch motion. In this test, it is assumed that the OMV is originally at rest, the initial conditions being:

\[
\begin{align*}
X_1 &= X_2 = X_3 = 0 \\
V_1 &= V_2 = V_3 = 0 \\
r &= p = y = 0
\end{align*}
\]

where \( r, p, y \) represent the roll, pitch and yaw respectively. A pure pitch motion would correspond to a rotation about the 2-axis. Mathematically,

\[
\begin{align*}
r &= y = 0, \text{ and } p &= w_2 = 0
\end{align*}
\]

When the OMV is executed in this mode, the state vectors are fed into the SVX module, with the result that the state vector is translated into a sequence of commands CMD. This sequence of commands is to be transmitted to the flat floor. Table 5 shows the relevant commands for the mobile base. As verified by the graphics display, the mock up module mounted on the mobile base executes a pure pitch at the same rate as the OMV, while the mobile base has to translate along the +X direction. In addition, the pivot point is progressively lowered as expected. This test shows that the modules OMV and SVX are properly interfaced, and that correct results are produced. The command strings as out-

17
putted by the system to the flat floor is shown in Figure 6.

To further ascertain that the system is functioning properly, the hand controller signals corresponding to a translation along 1-axis and a yaw is generated. The relevant commands to the flat floor system is shown in Table 6. A pictorial representation of the mobile base and mock up is as shown in Figure 7. Note that the path of the center of mass of the mock up exactly duplicates that of the OMV.

In summary, various tests conducted have shown that the OMV-SVX system functions properly. By way of example, a pure yaw motion of the OMV demands that the mobile base describes a circular path as shown in Figure 8. There is just one area that needs further investigation, namely timing considerations. This system must be able to complete all the computation within 0.1 second — a major cycle. When the system is uploaded to the PDP 11/34, it was discovered that the computer took more than 0.1 seconds to complete one major cycle of computation. At this juncture, one can take one of the following three corrective actions:

a) Use a faster host computer (VAX 780)
b) Use single precision computation, or
c) Increase the step size in the numerical methods.

Of the three choices, the first method is clearly desirable, but until the VAX is installed, one must explore the remaining alternatives. Table 7 shows a time comparison between single and double precision arithmetic when the OMV is run until identical
parameters on the PDP 11/34 computer. The result shows little improvement in execution time. This is not surprising since the computer is equipped with hardware floating point capability. The only remaining recourse is to increase the step size, thereby reducing the number of steps (and hence the number of iterations). It is discovered that the numerical solution to the equations of motion took most of the computation time. Table 8 shows a similar time test for various steps N and retaining double precision arithmetic after the code has been suitably optimized. The data show that a step size of \( h = 0.025 \) seconds (\( N = 4 \)) satisfies the time requirement. The price to be paid is that the error associated with the numerical process may increase. Table 9 shows a comparison test for \( N = 10 \) and \( N = 4 \) using the program ADAM. The result suggests that there is an optimum \( N \) somewhere between 4 and 10 in which the error is a minimum, but this question is not pursued any further. The result also shows that the error does not increase substantially over the same period of 60 minutes whether we use \( N = 10 \) or \( N = 4 \). Using \( N = 4 \), the deviation from the analytical solution is still much less than the accuracy of the flat floor system.

Conclusion

The series of tests conducted, some of which are not reported here, shows that the simplified mathematical of the Orbital Maneuvering Vehicle is functioning properly, and that it interfaces properly with the State Vector Transformation module SVX to produce correct sequences of commands to the flat floor.
By choosing a coarser step in the numerical integration process, ONV is able to complete all the necessary computation within a major cycle, without compromising on the accuracy. The final acid test cannot be conducted until the flat floor hardware is operational.
List of Tables
<table>
<thead>
<tr>
<th>Dynamic Variable</th>
<th>Value</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass M</td>
<td>3282.75</td>
<td>kg</td>
</tr>
<tr>
<td>$I_{11}$</td>
<td>7048.37</td>
<td>kg m$^2$</td>
</tr>
<tr>
<td>$I_{22}$</td>
<td>3713.95</td>
<td>kg m$^2$</td>
</tr>
<tr>
<td>$I_{33}$</td>
<td>3713.95</td>
<td>kg m$^2$</td>
</tr>
<tr>
<td>bit</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Acceleration along +1 direction</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Acceleration along -1 direction</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Acceleration along +2 direction</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Acceleration along -2 direction</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Acceleration along +3 direction</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Acceleration along -3 direction</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>+ roll; CCW rotation about 1-axis</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>- roll; CW rotation about 1-axis</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>+ pitch; CCW rotation about 2-axis</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>- pitch; CW rotation about 2-axis</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>+ yaw; CCW rotation about 3-axis</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>- yaw; CW rotation about 3-axis</td>
<td></td>
</tr>
</tbody>
</table>
Table 3
Comparison Between Analytical and Numerical Solutions

| Time in Minutes | X (meters) |   | Z (meters) |   |
|-----------------|------------|-----------------|------------|
|                 | Numerical  | Analytical      | Numerical  | Analytical |
| 0               | 0.000000   | 0.000000        | 0.000000   | 0.000000   |
| 5               | 13.746736  | 13.746736       | 5.271240   | 5.271240   |
| 10              | 20.161917  | 20.161917       | 20.427114  | 20.427114  |
| 15              | 12.828962  | 12.828962       | 43.576117  | 43.576178  |
| 20              | -12.952950 | -12.952952      | 71.829442  | 71.829444  |
| 25              | -59.582227 | -59.582233      | 101.660919 | 101.660923 |
| 30              | -126.855533| -126.855544     | 129.347660 | 129.347664 |
| 35              | -211.993176| -211.993191     | 151.434377 | 151.434380 |
| 40              | -309.986003| -309.986022     | 165.164663 | 165.164664 |
| 45              | -414.220544| -414.220565     | 168.824984 | 168.824985 |
| 50              | -517.304365| -517.304388     | 161.958539 | 161.958536 |
| 55              | -611.988664| -611.988666     | 145.422253 | 145.422248 |
| 60              | -692.072815| -692.072834     | 121.279843 | 121.279843 |

Note: X and Z are expressed in Local Vertical Frame.
Table 4

OMV Acceleration Along +X Direction

<table>
<thead>
<tr>
<th>Time (Seconds)</th>
<th>X (meters)</th>
<th>X (meters)</th>
<th>Z (meters)</th>
<th>Z (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.5</td>
<td>0.002940</td>
<td>0.012125</td>
<td>0.000001</td>
<td>0.000007</td>
</tr>
<tr>
<td>1.0</td>
<td>0.012128</td>
<td>0.024625</td>
<td>0.000009</td>
<td>0.000029</td>
</tr>
<tr>
<td>1.5</td>
<td>0.027565</td>
<td>0.037125</td>
<td>0.000032</td>
<td>0.000065</td>
</tr>
<tr>
<td>2.0</td>
<td>0.049253</td>
<td>0.049625</td>
<td>0.000077</td>
<td>0.000117</td>
</tr>
<tr>
<td>2.5</td>
<td>0.077190</td>
<td>0.062125</td>
<td>0.000152</td>
<td>0.000183</td>
</tr>
<tr>
<td>3.0</td>
<td>0.111377</td>
<td>0.074624</td>
<td>0.000263</td>
<td>0.000264</td>
</tr>
<tr>
<td>3.5</td>
<td>0.151814</td>
<td>0.087124</td>
<td>0.000418</td>
<td>0.000360</td>
</tr>
<tr>
<td>4.0</td>
<td>0.198501</td>
<td>0.099624</td>
<td>0.000625</td>
<td>0.000471</td>
</tr>
</tbody>
</table>

Initial conditions:

\[ X_1 = X_2 = X_3 = 0 \] and

\[ V_1 = V_2 = V_3 = 0 \]

Note: All quantities are expressed in Local Vertical Frame.
Table 5

OMV -- Pure pitch motion at 0.017453 rad/sec

<table>
<thead>
<tr>
<th>Time (Sec)</th>
<th>Pitch (Rad)</th>
<th>X (meters)</th>
<th>Z (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>5.0000</td>
<td>2.4384</td>
</tr>
<tr>
<td>4</td>
<td>0.0698</td>
<td>5.0010</td>
<td>2.3852</td>
</tr>
<tr>
<td>8</td>
<td>0.1396</td>
<td>5.0074</td>
<td>2.3324</td>
</tr>
<tr>
<td>12</td>
<td>0.2094</td>
<td>5.0167</td>
<td>2.2800</td>
</tr>
<tr>
<td>16</td>
<td>0.2793</td>
<td>5.0295</td>
<td>2.2284</td>
</tr>
<tr>
<td>20</td>
<td>0.3491</td>
<td>5.0460</td>
<td>2.1778</td>
</tr>
<tr>
<td>24</td>
<td>0.4189</td>
<td>5.0659</td>
<td>2.1285</td>
</tr>
</tbody>
</table>

Note: All measurements are in flat floor coordinates. Please see Appendix 1.
Table 6

Motion of the Mobile Base under constant acceleration of [0.025,0,0]T and constant yaw at 0.08675 rad/sec

<table>
<thead>
<tr>
<th>Time (Sec)</th>
<th>X (meters)</th>
<th>Y (meters)</th>
<th>Z (meters)</th>
<th>Yaw (rad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>11.6680</td>
<td>2.4384</td>
<td>0.0000</td>
</tr>
<tr>
<td>4</td>
<td>0.2752</td>
<td>11.2418</td>
<td>2.4390</td>
<td>0.3470</td>
</tr>
<tr>
<td>8</td>
<td>1.0709</td>
<td>11.0039</td>
<td>2.4433</td>
<td>0.6940</td>
</tr>
<tr>
<td>12</td>
<td>2.2919</td>
<td>11.1199</td>
<td>2.4545</td>
<td>1.0410</td>
</tr>
<tr>
<td>16</td>
<td>3.7925</td>
<td>11.7135</td>
<td>2.4750</td>
<td>1.3880</td>
</tr>
<tr>
<td>20</td>
<td>5.3934</td>
<td>12.8512</td>
<td>2.5062</td>
<td>1.7350</td>
</tr>
<tr>
<td>24</td>
<td>6.9035</td>
<td>14.5350</td>
<td>2.5480</td>
<td>2.0820</td>
</tr>
</tbody>
</table>

Note: X, Y and Z are expressed in flat floor coordinates. Please see Appendix 1.
Table 7

OMV Time Test

<table>
<thead>
<tr>
<th>No of Steps</th>
<th>Average execution time per major cycle</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single Precision</td>
<td>Double Precision</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.077</td>
<td>0.084</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.090</td>
<td>0.099</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.103</td>
<td>0.113</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.117</td>
<td>0.128</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.130</td>
<td>0.143</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.144</td>
<td>0.158</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.157</td>
<td>0.173</td>
<td></td>
</tr>
</tbody>
</table>
Table 8

Optimized OMV Execution Times Per Major Cycle
As A Function Of Number Of Steps N

<table>
<thead>
<tr>
<th>N</th>
<th>Execution time (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.068</td>
</tr>
<tr>
<td>5</td>
<td>0.079</td>
</tr>
<tr>
<td>6</td>
<td>0.090</td>
</tr>
<tr>
<td>7</td>
<td>0.100</td>
</tr>
<tr>
<td>8</td>
<td>0.111</td>
</tr>
<tr>
<td>9</td>
<td>0.122</td>
</tr>
<tr>
<td>10</td>
<td>0.132</td>
</tr>
</tbody>
</table>
Table 9

Comparison Test Between $N = 4$ and $N = 10$ Steps

<table>
<thead>
<tr>
<th>Time in Minutes</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analytic</td>
</tr>
<tr>
<td></td>
<td>$N = 10$</td>
</tr>
<tr>
<td>0</td>
<td>0.000000</td>
</tr>
<tr>
<td>5</td>
<td>13.746736</td>
</tr>
<tr>
<td>10</td>
<td>20.161917</td>
</tr>
<tr>
<td>15</td>
<td>12.828962</td>
</tr>
<tr>
<td>20</td>
<td>-12.952950</td>
</tr>
<tr>
<td>25</td>
<td>-59.582233</td>
</tr>
<tr>
<td>30</td>
<td>-126.855544</td>
</tr>
<tr>
<td>35</td>
<td>-211.993191</td>
</tr>
<tr>
<td>40</td>
<td>-309.986022</td>
</tr>
<tr>
<td>45</td>
<td>-414.220565</td>
</tr>
<tr>
<td>50</td>
<td>-517.304388</td>
</tr>
<tr>
<td>55</td>
<td>-611.988666</td>
</tr>
<tr>
<td>60</td>
<td>-692.072834</td>
</tr>
</tbody>
</table>
List of Figures
Figure 1. MSFC Flatfloor Simulation System
Fig. 2 Local Vertical Frame (L)
OMV Body Frame

Fig 3
Figure 4

OMV Heirarchial Chart

OMV

- OUTPUT
  - INITPL
  - QUITGM
  - HNDCTL
    - FUDGE
      - DOTPRD
      - PUT
      - DCSINV
      - DMUL
      - MATCH
      - UPDQ
    - DETQ
      - SINCOS
    - STORE
    - SOLVE
      - INNIT
    - ANGFRE
      - F
    - VECTOR
  - OMVMDL
  - SVX

(Note 1)
Note 1: Hardware incompatible graphics package.

(Note 2)
Figure 5. Translation Along X-Axis
Figure 6. Pure Pitch Motion at 0.017453 rad/sec.
Figure 7. Trajectory of Mobile Base When OMV is Executing a Translation & Yaw
Figure 8. OMV Pure Yaw Motion
List of References
List of References


Appendix 1

OMV Translational Equations of Motion
Consider a target vehicle orbiting the earth with an angular velocity $\omega$ and an orbit radius of $R_o$. We can define a local vertical frame (LVF) at the center of gravity of this vehicle as shown in the figure below:

Here, $X_L$, $Y_L$ and $Z_L$ are the three orthogonal axes of the LVF. We can imagine that the center of the earth may be considered as the origin of the inertial coordinate frame. We can choose the axes of this coordinate system as shown. In particular, $Y_E$ is parallel to $Y_L$. We shall use the subscript $L$ to denote those quantities that are expressed in the LVF, while the subscript $E$ shall be used for those quantities expressed in the inertial frame. The point $C$ in the above figure represents the center of mass of the chase vehicle (OMV).
The equation of motion of the chase vehicle is easily deduced from Newton's second law, namely,

\[ M_c \ddot{r} = F_g + F_c \]  

This equation is written in the inertial frame. Here, \( M_c \) is the mass of the chase vehicle, \( F_g \) is the gravitational force exerted on the vehicle by the earth, and \( F_c \) is the control force exerted on the vehicle from the on-board thrusters and jets. The objective of this exercise is to derive the equation of motion in terms of \( r \) and its time derivatives. Namely, we wish to express the motion of the chase vehicle (OMV) in local vertical frame. This choice turns out to be very convenient for docking maneuvers.

From the above figure, it is obvious that

\[ R = R_o + r_E \]  

it follows that

\[ \dddot{R} = \dddot{R}_o + \dddot{r}_E \]  

Since the LVF is a rotating frame, we can use the operator:

\[ \{ d/dt \}_E = \{ d/dt + \omega \times \}_L \]

Applying this operator to \( r \) twice, we have

\[ \dot{r}_E = \dot{r}_L + \omega \times r_L \]

and

\[ \ddot{r}_E = \frac{d}{dt} (\dot{r}_L + \omega \times r_L) + \omega \times (\dot{r}_L + \omega \times r_L) = \dddot{r}_L + \omega \times \dddot{r}_L + 2\omega \times (\omega \times r_L) \]

\[ \dddot{r}_E = \dddot{r}_L + 2\omega \times \dddot{r}_L + \omega \times (\omega \times r_L) \]  

(4)
From equations (3) and (4), we have:

\[
\ddot{R} = \ddot{R}_o + \dot{r}_L \\
= \ddot{R}_o + 2w \times \dot{r}_L + \dot{w} \times (w \times r_L)
\]

Furthermore, for a circular orbit,

\[
\ddot{R}_o + w^2 R_o = 0
\]

therefore,

\[
\ddot{R} = -w^2 R_o + 2w \times \dot{r}_L + \dot{w} \times (w \times r_L) \quad (5)
\]

It is clear at this point that the equations of motion (1) can be rewritten in terms of \( r_L \) and \( R_o \) and their time derivatives. Thus the subscript will be dropped from here on. Recall that

\[
R = R_o + r \\
R^2 = (R_o + r) \cdot (R_o + r) \\
= R_o^2 + r^2 + 2R_o \cdot r \\
= R_o^2 + 2R_o \cdot r \\
= R_o^2 \{ 1 + 2(R_o \cdot r) / R_o^2 \}
\]

so that

\[
R^{-3} = R_o^{-3} \{ 1 - 3(R_o \cdot r) / R_o^2 \}^{-3/2} \\
= R_o^{-3} \{ 1 - 3(R_o \cdot r) / R_o^2 \}
\]

Thus, \( \mathbf{F}_g = -\frac{GM_e M_c}{R^3} R \)

\[
= -\frac{GM_e M_c}{R_o^3} (R_o + r) \left( 1 - 3(R_o \cdot r) / R_o^2 \right) \\
= -w^2 M_c (R_o + r) \left( 1 - 3(R_o \cdot r) / R_o^2 \right) \\
eq -w^2 M_c (R_o + r - 3(R_o \cdot r / R_o^2) R_o \quad (6)
\]

since for a circular orbit, \( w^2 = \frac{GM_e}{R_o^3} \). Substituting equations (5) and (6)
into (1), we have:

\[ M_c(-w^2 R_o + \mathbf{r} + 2w \times \mathbf{r} + \mathbf{w} \times (\mathbf{w} \times \mathbf{r})) = F - M_c w^2 (R_o + \mathbf{r} - 3(R_o \cdot \mathbf{r})/R_o^2) \]

If we define \( A = F_c / M_c \), then we have:

\[ -w^2 R_o + \mathbf{r} + 2w \times \mathbf{r} + \mathbf{w} \times (\mathbf{w} \times \mathbf{r}) = A - w^2 R_o - w^2 \mathbf{r} + 3w^2 (R_o \cdot \mathbf{r}/R_o^2) R_o \]

which, after re-arranging, gives:

\[ \ddot{\mathbf{r}} = A - 2w \times \dot{\mathbf{r}} - w^2 \mathbf{r} - \mathbf{w} \times (\mathbf{w} \times \mathbf{r}) + 3w^2 (R_o \cdot \mathbf{r}/R_o^2) R_o \]

(7)

Now, we shall state \( \mathbf{r}, R_o \) and \( \mathbf{w} \) in cartesian coordinates. It is explicitly assumed that the unit vectors \( \mathbf{i}, \mathbf{j} \) and \( \mathbf{k} \) are directed along \( X_L, Y_L \) and \( Z_L \) axes respectively. Thus,

\[
\begin{align*}
\mathbf{r} &= [X, Y, Z]^T \\
R_o &= [0, 0, R_o]^T \\
\mathbf{w} &= [0, w, 0]^T \\
A &= [A_x, A_y, A_z]^T
\end{align*}
\]

and it can easily be shown that:

\[
\begin{align*}
2w \times \dot{\mathbf{r}} &= [2wZ, 0, -2wX]^T \\
\mathbf{w} \times (\mathbf{w} \times \mathbf{r}) &= [-w^2X, 0, -w^2Z]^T \\
3w(R_o \cdot \mathbf{r}/R_o^2) R_o &= [0, 0, 3w^2Z]^T \\
w^2 \mathbf{r} &= [w^2X, w^2Y, w^2Z]^T
\end{align*}
\]

and substituting into equation (7) yields

\[
\begin{align*}
[X, Y, Z]^T &= [-2wZ, 0, 2wX]^T + [w^2X, 0, w^2Z]^T \\
&+ [-w^2X, -w^2Y, -w^2Z]^T + [0, 0, 3w^2Z]^T \\
&+ [A_x, A_y, A_z]^T
\end{align*}
\]
or

\[
\begin{align*}
\dddot{X} &= A_x - 2w\dot{Z} \\
\dddot{Y} &= A_y - \omega^2 Y \\
\dddot{Z} &= A_z + 2w\dot{X} + 3\omega^2 Z
\end{align*}
\]  

(8)

Equation (8) is the equation of motion of the chase vehicle relative to the target vehicle in local vertical frame.
Appendix 2

State Vector Transformation Module  SVX
Technical Report
State Vector Transformation Module

Technical Report

Prepared for
George C. Marshall Space Flight Center
Marshall Space Flight Center
Huntsville, AL 35812

by

Dr. William Teoh
Kenneth Johnson Energy & Environment Center
University of Alabama in Huntsville
Huntsville, AL 35899

Date prepared: May 9, 1984

Contract # NAS8-35670
INTRODUCTION

The State Vector Transformation Module (SVX) is an interface between the OMV simulation model and the mobile base (TOM_B) of the flat floor simulation system. We can imagine the OMV simulation to be a free flying vehicle in space under human operator control, and at any particular instant, its state can be summarized as a fourteen-component vector called the state vector $S$. SVX takes this state vector as an input and generates an appropriate string of commands that is transmitted to TOM_B with the stipulation that if TOM_B executes this command string exactly, then the mock-up module mounted on TOM_B will exactly replicate the motion of the OMV as perceived by the operator.

References 1), 2) and 3) are reports that pertain to the various aspects of the OMV. From these reports, the various components that make up the state vector can be deduced and are presented below:

<table>
<thead>
<tr>
<th>Component</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X$</td>
<td>position of the target vehicle relative to the OMV in local vertical frame LVF.</td>
</tr>
<tr>
<td>2</td>
<td>$Y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$Z$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$V_x$</td>
<td>relative velocity of the chase vehicle in LVF</td>
</tr>
<tr>
<td>5</td>
<td>$V_y$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$V_z$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$L_x$</td>
<td>angular momentum vector in LVF</td>
</tr>
<tr>
<td>8</td>
<td>$L_y$</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$L_z$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$q_1$</td>
<td>attitude quaternions in body frame</td>
</tr>
<tr>
<td>11</td>
<td>$q_2$</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>$q_3$</td>
<td></td>
</tr>
</tbody>
</table>
It is often more convenient to consider the state vector to be made up of the following four vectors: \( X = [X, Y, Z]^T \), \( V = [V_x, V_y, V_z]^T \), \( L = [L_x, L_y, L_z] \) and the unit quaternion \( q = [q_1, q_2, q_3, q_4]^T \).

As mentioned earlier, the required command string must be derived from this state vector, and is transmitted to TOM_B as seven 16-bit words. The last word can either be a zero or a one, which is interpreted by the TOM_B Executive as rate or position control respectively. A brief explanation of the command string is shown below:

<table>
<thead>
<tr>
<th>Component</th>
<th>Position control</th>
<th>rate control</th>
<th>coord. system</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol</td>
<td>meaning</td>
<td>symbol</td>
<td>meaning</td>
</tr>
<tr>
<td>1</td>
<td>( y )</td>
<td>( y )</td>
<td>( )</td>
</tr>
<tr>
<td>2</td>
<td>( X )</td>
<td>( V_x )</td>
<td>( )</td>
</tr>
<tr>
<td>3</td>
<td>( Y )</td>
<td>( V_y )</td>
<td>( )</td>
</tr>
<tr>
<td>4</td>
<td>( Z )</td>
<td>( V_z )</td>
<td>( )</td>
</tr>
<tr>
<td>5</td>
<td>( p )</td>
<td>( p )</td>
<td>( )</td>
</tr>
<tr>
<td>6</td>
<td>( r )</td>
<td>( r )</td>
<td>( )</td>
</tr>
<tr>
<td>7</td>
<td>( l )</td>
<td>( 0 )</td>
<td>( )</td>
</tr>
</tbody>
</table>

Before the detailed analysis is presented, it is necessary to define the various coordinate systems used.

**COORDINATE SYSTEMS**

Several coordinate systems are used in this software module. Specifically, motion of the OMV is described in Local Vertical Frame (LVF) while the orienta-
tion of the OMV is described in body frame. Similarly, the position and velocity of the mobile base TOM_B is described in floor coordinates while the orientation of the mock-up module and TOM_B are described by their respective body frames.

A) the Local Vertical Frame (LVF)

Imagine a circular orbit that is inclined at an angle \(i\) with respect to the equatorial plane. A Local Vertical Frame is a non-stationary frame that has its origin at a point on this orbit such that

(i) its \(Z_L\) axis is directed away from the earth's center,
(ii) its \(X_L\) axis is directed tangential to the orbit and is perpendicular to its \(Z_L\) axis, and
(iii) the \(Y_L\) axis is directed parallel to the angular momentum vector, as shown in Figure 1.

A subscript \(L\) will be used to indicated quantities defined in this coordinate system.

B) the Floor Coordinates (F)

The floor coordinates has its origin at one corner of the flat floor as shown in Figure 2. Its \(X_F\) axis is directed along the width of the floor, while the \(Y_F\) axis is directed along the length of the floor. Naturally, \(Z_F\) axis is directed vertically up.

C) the TOM_B body Frame (B)

This coordinate system is fixed with respect to the mobile base, and has its origin at the center of mass of the mobile base. Its \(X_B\) axis is directed towards the front of TOM_B, while its \(Z_B\) axis is parallel to the \(Z_F\) axis of the flat floor. A third axis \(Y_B\) is chosen so as to form an orthogonal right-handed
coordinate system, a top view of which is shown in Figure 3.

D) the Mock-Up Module Body Frame (M)

We shall assume that the mock-up module resembles the OMV in shape (that is, not unlike a pancake). The origin of its body frame coincides with its center of mass, and the \( X_M \) axis is directed towards the front of the module. Initially, at the start of the simulation, the \( Z_M \) axis is chosen to be parallel to \( Z_F \), and the appropriate orthogonal axis is chosen as its \( Y_M \) axis, as indicated in Figure 4.

**ANALYSIS**

From the references mentioned above, it is obvious that the relative position and attitude from the state vector are relative quantities. Thus, initial conditions at the start of the simulation must be known. Figures 5 a) and b) shows the initial state of the mobile base and mock-up module at the start of the simulation. The quantities \( a, c, l, h \) and \( o \) can be obtained from measurement.

A necessary initial condition is that the operator must leave the hand controllers in the neutral position for at least one second so that the initial position of the OMV \( [ X_o, Y_o, Z_o ]^T \) can be obtained. It is also assumed that the initial orientations of both the OMV and mock-up module are set in their home position. If the notation \( r, p, \) and \( y \) is used to indicate the roll, pitch and yaw of both the OMV and the mock-up, then,

\[
[ r_{OMV}, p_{OMV}, y_{OMV} ]^T = [ r_M, p_M, y_M ]^T = [ 0, 0, 0 ]^T
\]

It is obvious that the corresponding axes of the coordinate frames \( M, B \) and \( F \) are all parallel at this point in time. At any later time, the position of the OMV can be calculated from the state vector:
Here, $S_1$, $S_2$, and $S_3$ are the first three components of the state vector. This position is measured relative to the starting point in the beginning of the simulation, and can be transformed to the position of the mock-up module in floor coordinates using the equation:

$$
\begin{bmatrix}
X_M \\
Y_M \\
Z_M
\end{bmatrix} =
\begin{bmatrix}
X_L \\
Y_L \\
Z_L
\end{bmatrix} +
\begin{bmatrix}
c + 1 - X_0 \\
a - Y_0 \\
h - Z_0
\end{bmatrix}
$$

Equation [I] governs the transformation of the position vector of the OMV in LVF to a position vector for the mock-up module in floor coordinates, based on the initial conditions and the first three components of the state vector. Given that the instantaneous orientation of the module is $[r_M, p_M, r_M]^T$ as shown in Figure 6 a) and b), the position of TOM_B $[X_F, Y_F, Z_F]^T$ in floor coordinates is given by:

$$
\begin{bmatrix}
X_F \\
Y_F \\
Z_F
\end{bmatrix} =
\begin{bmatrix}
X_M - (c + 1\cos(p))\cos(y) \\
Y_M - (c + 1\cos(p))\sin(y) \\
\delta
\end{bmatrix}
$$

Equation [II] governs the transformation of the position vector of the OMV in LVF to a position vector for the mock-up module in floor coordinates, based on the initial conditions and the first three components of the state vector. Given that the instantaneous orientation of the module is $[r_M, p_M, r_M]^T$ as shown in Figure 6 a) and b), the position of TOM_B $[X_F, Y_F, Z_F]^T$ in floor coordinates is given by:

$$
Z = Z_M - 1\sin(p)
$$

Note that $Z_F$ is the height of the center of mass of TOM_B from the floor (a constant quantity), and is not of interest here. Instead, the quantity of interest is $Z$, which is the height of the pivot point from the floor as shown in Figure 6, and

$$
Z = Z_M - 1\sin(p)
$$

It follows that the velocity of TOM_B and the pivot point is given by...
The above transformations take care of the position and velocity quantities.

The quaternions \( q_1, q_2, q_3, q_4 \) from the state vector specifies the OMV's attitude in body frame, as discussed in References 4) and 5). At any instant, its orientation is given by (see Ref 4):

\[
[r, \ p, \ y]^T = \alpha [0_x, 0_y, 0_z]^T
\]

where

\[
\alpha = 2 \cos^{-1}(q_4)
\]

\[
[0_x, 0_y, 0_z]^T = \frac{[iq_1 + jq_2 + kq_3]}{(q_1 + q_2 + q_3)^{0.5}} \quad [VI]
\]

while their rates are \( \omega_B = [w_1, w_2, w_3]^T \) which can be calculated in the following manner:

Since the angular momentum vector \( L = [L_x, L_y, L_z]^T \) from the state vector is expressed in LVF, it is necessary to transform it to body frame using the equation:

\[
L_B = A L \quad [VII]
\]

here \( A \) is the direction cosine matrix which can be constructed from the attitude quaternions \( q_1, q_2, q_3, \) and \( q_4 \)

\[
A = \begin{bmatrix}
q_4 + q_1 - q_2 - q_3 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\
2(q_1q_2 - q_3q_4) & q_4 - q_1 + q_2 - q_3 & 2(q_2q_3 + q_3q_4) \\
2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & q_4 - q_1 - q_2 + q_3
\end{bmatrix} \quad [VIII]
\]
Knowing the moment of inertia tensor $\mathbf{I}$, one can calculate the angular rates

$$\mathbf{w}_B = [w_1, w_2, w_3]^T = \mathbf{I}^{-1} \mathbf{L}_B = \mathbf{I}^{-1} (\mathbf{A} \mathbf{L})$$

Thus, one has all the needed information from the state vector to yield the necessary position or rate control commands.

**ALGORITHM**

The algorithm for SVX makes use of all the transformations described in the above section. Essentially, the algorithm uses the state vector and depending on the value of \textit{MODE}, generates the appropriate command string \textit{CMDRAW}.

**Case 1 \textit{MODE} $\leftrightarrow$ 0 (position control)**

In this case, both orientation and position of the OMV are updated. A transformation is made to yield the position of the center of mass of TOM\textsubscript{B} using equation [I] through [III]. The orientation of the mock-up module is obtained using equation [VI]. Using the previous notation, a seven element vector

$$[y, x_B, y_B, z, p, r, 1]^T$$

is generated. Each element of this vector is suitably scaled and round off to the nearest integer (16-bit word) and is the sole output of the SVX module. Rate information is not of interest when the system is in position control, and is therefore not transmitted. Throughout this module, the scale factors for all angular and displacement quantities are $10^4$ and $10^3$ respectively.
Case 2      \( \text{MODE} = 0 \)  (rate control)

In this rate control mode, it is still necessary to update the orientation (equation [VI]) although it is no longer necessary to update the position of the OMV. The velocity of TOM_B in floor coordinates is determined from equation [IV] while the rates for roll, pitch and yaw are determined using equations [VII] through [X]. The seven 16-bit word command string is

\[
[ y, X_B, Y_B, Z, p, r, 0 ]^T
\]

As before, each component of this vector is similarly scaled and rounded before returning.

Case 3 \( \text{MODE} \neq 0 \) AND \( \text{MODE} \neq 1 \)

In this case, \( \text{MODE} \) is set to 1, and position control is assumed.

IMPLEMENTATION

This algorithm is implemented as a subroutine named \( \text{SVX} \) (\( S, \text{CMDRAW}, \text{MODE} \)) where the three items on the parameter list are the state vector output command string and control mode respectively.

The subroutine is implemented in FORTRAN 77, and the usual programming practices are adhered to. Most of the major steps are either properly documented in the form of COMMENT statements or implemented as subprograms, following a modular design approach. Whenever possible, structured codes are used unless severe degradation of execution speed may result.

\( \text{SVX} \) is compiled and tested using an IBM Personal Computer, and the source code, on completion of the testing, is uploaded to the PDP 11/34 computer at MSFC. Appendix I shows a complete listing of this module. A more detailed description of the testing procedure will be presented later in this section.
A local counter (COUNT) is initialized at load time, and updated during execution to enable SVX to determine the initial state on start up. During this period, other tasks are carried out as an integral part of the initialization process. This include reading a file (SVXINT.DAT) for the values of \( c, l, a, h \) and \( o \), as well as the inverse of the moment of inertia tensor \( I^{-1} \).

This module assumes that the operator will, at start up, leave the hand controller at a neutral position for at least a second. During this interval, the initial state of the OMV is recorded, and the vector \( E \) where

\[
E = [ E_1, E_2, E_3 ]^T = [ c + l - x_0, a - y_0, h - z_0 ]^T
\]

is calculated. The roll, pitch and yaw of both the OMV and the mock-up module are initialized to zero during this process by invoking subroutine ZERO.

Subsequent calls to SVX causes a seven 16-bit command string in an INTEGER array called CMDRAW to be produced. Computation here depends on the value of MODE.

When MODE is non-zero, position control is assumed. SVX invokes subroutines QTRPY and UPDPOS to calculate the desired orientation and position of the OMV. A transformation is then made to determine the required position (of the mobile base TOM_B in floor coordinates) and orientation (of the mock-up module in body frame). Since the value of MODE cannot be changed in the course of a simulation, no rate information is calculated or retained.

When MODE is zero, rate control is used. First, QTRPY is called to calculate the orientation of the OMV; its position is not computed because it is not of interest while in rate control mode. The direction cosine matrix \( A \) is formed by invoking subroutine DIRCOS, and a simple matrix multiplication transforms
the angular momentum to body frame. Finally, the velocity of the OMV (from the state vector) is suitably transformed to yield the velocity of TOM_B in floor coordinates, and the appropriate command string assembled.

When MODE is neither zero nor one, it is set to one and defaults to position control. One frequently used subroutine in both modes is DECOMP which takes the state vector S and decomposes it to form the vectors X, V, L and q which correspond to the displacement, velocity, angular momentum and the unit quaternion vectors respectively. Throughout this module, no attempt is ever made to ensure that the magnitude of q is unity.

To ensure that SVX generates the correct command string, a series of tests were conducted using the IBM PC. First, a simple State Vector Editor is written. This editor allows one to create and edit, interactively, state vectors which are placed in sequence in a disk file. Next, a simple main program is written and linked to the SVX module. The main program consists of a driver loop that reads each state vector from the disk file and invokes SVX. The command string outputted by SVX is sent to a printer and the process is repeated until the file of state vectors is exhausted. This simple arrangement allows one to verify the correctness of SVX without disturbing it.

Since it is difficult, if not impossible, to represent the results graphically in three dimensions, state vectors are chosen such that one can easily displays the results in two dimensions. By way of example, a sequence of 60 state vectors of the form:

\[
[0,0,0,0,0,0,0,0,0,0,sin(7.5),cos(7.5),1500]^T
\]

is generated. This set of state vectors simulates 50 seconds of run time in which position control is used. The meaning of this state vector is that the
OMV is to remain stationary, but executes a yaw at a rate of 15° per major cycle (0.1 second). Here, we have assumed that the OMV is a disk shaped object having a uniform mass distribution and a constant mass of 1500 pounds. Note that in case of position control, the angular momentum vector is inconsequential, so a null vector is used. These figures may not be very realistic, but they are adequate for testing the SVX module. Figure 7 shows the result of a portion of the output command string. In this and subsequent figures, a circle or dot indicates the position of the center of mass of TOM_B in floor coordinates, while an attached arrow shows its yaw. This figure depicts that TOM_B moves in a circular path and its yaw is changing at a rate of 15° per major cycle. It is noted that the radius of the circular path is equal to the distance between the centers of mass of TOM_B and the mock-up module. Thus, the mock-up module would be spinning about its ZM axis at the same rate, exactly as expected.

When the state vectors are changed to

\[
[ 0.5,0,0, 0,0,0, 0,0,0, 0,0,\sin(7.5),\cos(7.5), 1500 ]^T
\]

in position control, the path of TOM_B is shown in Figure 8. In this figure, TOM_B attempts to move in a circular path with a net displacement of 0.5 feet per major cycle. It is easy to conclude that the mock-up module would be rotating about its ZM axis and translate along the XM axis simultaneously, as demanded by this state vector.

CONCLUSION

Other similar tests have been conducted. For example, the state vector in the beginning of this section has been used as input for rate control, and the result is plotted in Figure 9. This and similar results have demonstrated that
the module SVX is functioning properly and that correct command strings are obtained. One must remember that the outputs of this module are commands to TOM_B, indicating the desired position, (or velocity) and attitude (or angular rates). The proper interpretation, and subsequent execution, of these commands are performed by the TOM_B Executive, and is outside the scope of the SVX module.
Fig. 1  Local Vertical Frame (L)
Fig 2. Floor coordinates (F)
Fig 3. TOM_B Body Frame (B)

Fig 4. MOCK-UP MODULE BODY FRAME (B)
Fig 5 a) Initial position (top view)
Fig 5 b) Initial position (side view)
Figure 6 a) Position and yaw of TOM_B
Figure 6 b) Pitch and roll of Mock-up Module
Figure 7 Position of TOM B in floor coordinates
Figure 9  Velocity components of TOM_B
List of References


STATE VECTOR TRANSFORMATION MODULE (SVX)

by

Dr. W. Teoh

U A H 1984

SUBROUTINE SVX (S, CMDRAW, MODE)

This is the state vector transformation module which accepts a
14 element state vector S of the OMV as input and generates a
6-element command string CMDRAW as output. The argument MODE
conveys the following meaning:

<table>
<thead>
<tr>
<th>MODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rate control</td>
</tr>
<tr>
<td>1</td>
<td>position control</td>
</tr>
<tr>
<td>anything else</td>
<td>defaults to 1</td>
</tr>
</tbody>
</table>

Summary of the state vector components are as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X position of target vehicle from the chase vehicle in LVF</td>
</tr>
<tr>
<td>2</td>
<td>Y relative velocity of the two vehicles in LVF</td>
</tr>
<tr>
<td>3</td>
<td>Z angular momentum vector in LVF</td>
</tr>
<tr>
<td>4</td>
<td>VX attitude quaternions in body frame</td>
</tr>
<tr>
<td>5</td>
<td>VY instantaneous mass in kg.</td>
</tr>
</tbody>
</table>
Summary of command string components:

<table>
<thead>
<tr>
<th>component</th>
<th>meaning</th>
<th>coord system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YAW</td>
<td>body frame</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>floor coordinate</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>floor coordinate</td>
</tr>
<tr>
<td>4</td>
<td>Z</td>
<td>floor coordinate</td>
</tr>
<tr>
<td>5</td>
<td>PITCH</td>
<td>body frame</td>
</tr>
<tr>
<td>6</td>
<td>ROLL</td>
<td>body frame</td>
</tr>
<tr>
<td>7</td>
<td>MODE</td>
<td>integer</td>
</tr>
</tbody>
</table>

This module maintains a local counter to process initial conditions at the start of the simulation.

REAL * 8  S(14)
REAL * 8  X(3), V(3), L(3), Q(4)
REAL * 8  XO(3), XM(3), E(3), XHOLD(3)
REAL * 8  IINV(3), LB(3), W(4)
REAL * 8  RPY(3), QDOT(4), QW(4,4), A(3,3)
REAL * 8  ROLL, PITCH, YAW, ROLDOT, PITDOT, YAWDOT
REAL * 8  Q1, Q2, SY, CY, VX, VY, VZ

INTEGER CMDRAW(7), COUNT, MODE

*** load-time initialization

DATA COUNT /0/

*** decompose state vector and process it

CALL DECOMP (S, X, V, L, Q)
IF (COUNT .NE. 0) GOTO 300

*** initialization before start

CALL ZERO (XO, 3)

*** read parameters

OPEN (1, FILE = 'SVXINT.DAT', STATUS = 'OLD')
READ (1, 20) CC, LL, AA, HH
READ (1, 20) IINV
*** calculate inverse of moment of inertia tensor

DO 50 K = 1, 3
    IINV(K) = 1.0 / IINV(K)
CONTINUE

*** set conversion factors
UL = 10000.0
UA = UL
COUNT = COUNT + 1

*** set transformation matrix elements to floor coord.
E(1) = CC + LL - XO(1)
E(2) = AA - XO(2)
E(3) = HH - XO(3)

*** initialize to home orientation
CALL ZERO (RPY, 3)
COUNT = COUNT + 1

IF (MODE .NE. 1) GO TO 400

*** position commands
*** update orientation and position
CALL QTRPY (Q, ROLL, PITCH, YAW)
CALL UPDPOS (XM, X, XHOLD, E, 3)

*** set orientation part of the command string
CMDRAW(7) = 1
CMDRAW(6) = JFIX(ROLL * UA)
CMDRAW(5) = JFIX(PITCH * UA)
CMDRAW(1) = JFIX(YAW * UA)

*** transform to TOM_B position in floor coordinates
QQ = CC + LL * DCOS(PITCH)
TX = XM(1) - QQ * DCOS(YAW)
CMDRAW(2) = JFIX (TX * UL)
TY = XM(2) - QQ * DSIN(YAW)

*** Z-component

Z = XM(3) - LL * DSIN(PITCH)

*** This is a good place to call the I/O driver to
*** transmit to TOM_B, but we won't for now

IF (MODE .NE. 0) GO TO 900

*** rate control

CALL QTRPY (Q, ROLL, PITCH, YAW)

*** form direction cosine matrix and calculate angular
*** momentum in body frame

CALL D IRCOS (A, Q)
CALL MMUL (A, L, LB, 3)

*** compute body rate

ROLDOT = IINV(1) * LB(1)
PITDOT = IINV(2) * LB(2)
YAWDOT = IINV(3) * LB(3)

*** construct orientation part of command string

CMDRAW(7) = 0
CMDRAW(6) = JFIX (ROLDOT * UA)
CMDRAW(5) = JFIX (PITDOT * UA)
CMDRAW(1) = JFIX (YAWDOT * UA)

*** compute velocity of TOM_B in floor coordinates

Q1 = LL * DSIN(PITCH) * PITDOT
Q2 = (CC + LL * DCOS(PITCH)) * YAWDOT
SY = DSIN(YAW)
CY = DCOS(YAW)

*** X-component of velocity in floor coordinate
**Official FORTRAN Code**

```fortran
VX = V(1) + Q1 * CY + Q2 * SY
CMDRAW(2) = JFIX (VX * UL)

*** Y-component of velocity in floor coordinate

VY = V(2) + Q1 * SY - Q2 * CY
CMDRAW(3) = JFIX (VY * UL)

*** Z-component

VZ = V(3) - LL * DCOS(PITCH) * PITDOT
CMDRAW(4) = JFIX (VZ * UL)
RETURN

CONTINUE

*** We have an un-recognizable code, default to 1 for
*** position control

MODE = 1
GO TO 300

FORMAT (4F10.2)
FORMAT (F15.8)
END
```

**Variables and Offset Table**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>REAL*8</td>
<td>466</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>REAL*8</td>
<td>558</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JC</td>
<td>REAL*8</td>
<td>542</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMDRAW</td>
<td>INTEGER*4</td>
<td>4</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>COUNT</td>
<td>INTEGER*4</td>
<td>538</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CY</td>
<td>REAL*8</td>
<td>698</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCOS</td>
<td></td>
<td></td>
<td></td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>DSIN</td>
<td></td>
<td></td>
<td></td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>E</td>
<td>REAL*8</td>
<td>418</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IH</td>
<td>REAL*8</td>
<td>566</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INV</td>
<td>REAL*8</td>
<td>442</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>574</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>REAL*8</td>
<td>370</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td>REAL*8</td>
<td>394</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LL</td>
<td>REAL*8</td>
<td>550</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td>INTEGER*4</td>
<td>8</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>PITCH</td>
<td>REAL*8</td>
<td>602</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PITDOT</td>
<td>REAL*8</td>
<td>658</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td>REAL*8</td>
<td>674</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Miscellaneous**

- The code includes several mathematical operations and intrinsic functions.
- The offset table provides the location of each variable within memory.
- The code is modular, with separate sections for different calculations.
- The use of intrinsic functions like DCOS and DSIN indicates the code's precision and accuracy.
- The CONTINUE statement is used to continue execution after a specific condition is met.

---

**Note:**
- The use of comments (***, ...*) helps explain the purpose of each section of code.
- The code is well-structured, making it easy to read and understand.
SUBROUTINE DECOMP (S, X, V, L, Q)

This procedure decomposes the State vector S into its components which are also vectors. They have the following meaning:

<table>
<thead>
<tr>
<th>Vector</th>
<th>Dimension</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>3</td>
<td>Position vector in LVF</td>
</tr>
<tr>
<td>V</td>
<td>3</td>
<td>Velocity vector in LVF</td>
</tr>
<tr>
<td>L</td>
<td>3</td>
<td>Angular momentum in LVF</td>
</tr>
<tr>
<td>Q</td>
<td>4</td>
<td>Unit quaternion in body frame</td>
</tr>
</tbody>
</table>

REAL * 8 S(14), X(3), V(3), L(3), Q(4)

CALL LD (S, X, 1, 3)
CALL LD (S, V, 4, 3)
CALL LD (S, L, 7, 3)
CALL LD (S, Q, 10, 4)

RETURN
END
SUBROUTINE LD (A, B, M, N)

This procedure copies N elements of vector A to vector B, starting at the M-th element

REAL * 8 A(14), B(N)
DO 100 K = 1, N
   B(K) = A(M + K - 1)
100 CONTINUE
RETURN
END
SUBROUTINE MMUL (A, B, C, N)

This procedure performs a matrix multiplication of an N x N matrix A to an N-element column matrix B to yield an N-element column matrix C.

REAL * 8 A(N,N), B(N), C(N), S

DO 100 I = 1, N
   S = 0.0
   DO 200 J = 1, N
      S = S + A(I,J) * B(J)
   CONTINUE
   C(I) = S
1 CONTINUE
100 CONTINUE
RETURN
END
SUBROUTINE ZERO (A, N)

This procedure initializes an N-element array A to zero at run time

REAL * 8 A(N)
DO 100 K = 1, N
A(K) = 0.0
100 CONTINUE
RETURN
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>782</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>4</td>
<td>*</td>
</tr>
</tbody>
</table>
SUBROUTINE UPDPOS (XM, X, XHOLD, E, N)

This procedure updates the position of the OMV in local vertical frame (XHOLD).

The new position of the module in floor coordinates is then computed (XM)

REAL * 8 XM(N), X(N), XHOLD(N), E(N)

DO 100 K = 1, N
   XM(K) = X(K) + E(K)
100 CONTINUE

END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>XM</td>
<td>REAL*8</td>
<td>12</td>
<td>*</td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>790</td>
<td>*</td>
</tr>
<tr>
<td>N</td>
<td>INTEGER*4</td>
<td>16</td>
<td>*</td>
</tr>
<tr>
<td>E(N)</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>XHOLD</td>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
</tr>
<tr>
<td>XM</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
</tr>
</tbody>
</table>
This procedure properly rounds a real number \( R \) to the nearest integer.

```
 INTEGER FUNCTION JFIX (RR)

 REAL * 8 RR
 REAL R

 IF (R .GE. 0) THEN
   JFIX = IFIX (R + 0.5)
 ELSE
   JFIX = IFIX (R - 0.5)
 END IF
 RETURN
END
```

Type Offset P Class

FIX REAL 798 INTRINSIC
: REAL*8 0 *

356 $PAGE
SUBROUTINE SETQ (QW, Q)

This procedure constructs a 4x4 transformation matrix QW from the attitude quaternions Q.

For reference, please see "Software Specifications For Docking Simulation Of The OMV" by J. Micheals, January, 1984.

REAL * 8 QW(4,4), Q(4)

DO 100 I = 1, 3
   DO 110 J = I+1, 4
      KK = I + J
      K = KK - (KK/4) * 4
      IF (K .EQ. 0) K = 2
      ISGNN = 1
      IF (J .EQ. I+1) .AND. (J.NE. 4)) ISGNN = -1
      QW(I,J) = ISGNN * Q(K)
   CONTINUE
1  QW(I,I) = Q(4)
100 CONTINUE

QW(4,4) = Q(4)

DO 200 I = 2, 4
   KK = I - 1
   DO 200 J = 1, KK
      QW(I,J) = -QW(J,I)
   CONTINUE
200 RETURN

END

Name | Type     | Offset P Class
--- | -------- | -------------
ISGN | INTEGER*4 | 802           
J    | INTEGER*4 | 818           
\`   | INTEGER*4 | 806           
\:K  | INTEGER*4 | 814           
Q    | REAL*8    | 4 *           
QW   | REAL*8    | 0 *           

393 $PAGE
SUBROUTINE DIRCOS (A, Q)

This procedure takes the quaternion vector and generates a 3 X 3 direction cosine matrix A.

REAL * 8 Q(4), A(3,3), QKS, QRS, SI

DO 100 K = 1, 3

*** initialize diagonal elements
A(K,K) = Q(4) ** 2
DO 100 J = 1, 3

*** fix up the diagonal elements
A(K,K) = A(K,K) + DLTKR(K,J) * Q(J) ** 2

*** now do the off-diagonal elements
IF ( J .GT. K ) THEN

*** calculate index I <> J & K
I = 6 / (J * K)

*** calculate the proper sign
SI = QSIGN (K,J)
QKJ = Q(K) * Q(J)
QRS = Q(I) * Q(4) * SI
A(K,J) = 2.0 * (QKJ + QRS)
A(J,K) = 2.0 * (QKJ - QRS)

END IF
CONTINUE
RETURN
END
<table>
<thead>
<tr>
<th>D Line# 1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>INTEGER*4</td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
</tr>
<tr>
<td>QKJ</td>
<td>REAL</td>
</tr>
<tr>
<td>QKS</td>
<td>REAL*8</td>
</tr>
<tr>
<td>QRS</td>
<td>REAL*8</td>
</tr>
<tr>
<td>S1</td>
<td>REAL*8</td>
</tr>
</tbody>
</table>

438 $PAGE
REAL FUNCTION DLTKRK (K,J)

INTEGER K, J

S = 1.0

IF (K .NE. J) S = -1.0

DLTKRK = S

RETURN

END
REAL FUNCTION QSIGN(K,J)

463 S = 1.0
464 L = J + K
465 IF (MOD(L,2) .EQ. 0) S = -1.0
466 QSIGN = S
467 RETURN
468 END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>INTEGER*4</td>
<td>874</td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>INTRINSIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>REAL</td>
<td>870</td>
<td></td>
</tr>
</tbody>
</table>

469 $PAGE
SUBROUTINE QTRPY (Q, R, P, Y)

This subroutine calculates a reasonable set of roll, pitch and yaw from the quaternion Q

REAL * 8 Q(4), R, P, Y, M, THETA, CA, CB, CG

M = DSQRT (Q(1)**2 + Q(2)**2 + Q(3)**2)

calculate direction cosines CA, CB, CG

IF (DABS(M) .LE. 1.0D-20) THEN
  CA = 0.0
  CB = 0.0
  CG = 0.0
ELSE
  CA = Q(1) / M
  CB = Q(2) / M
  CG = Q(3) / M
END IF

calculate angle of rotation about Euler axis

THETA = 2.0 * DACOS(Q(4))

now determine the roll, pitch and yaw

R = CA * THETA
P = CB * THETA
Y = CG * THETA

RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>REAL*8</td>
<td>886</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>REAL*8</td>
<td>894</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>REAL*8</td>
<td>902</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABS</td>
<td>REAL*8</td>
<td>878</td>
<td></td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>ACOS</td>
<td>REAL*8</td>
<td>8 *</td>
<td></td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>DSQRT</td>
<td>REAL*8</td>
<td>0 *</td>
<td></td>
<td>INTRINSIC</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>4 *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Size</td>
<td>Class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECOMP</td>
<td></td>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRCOS</td>
<td></td>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KRR REAL</td>
<td></td>
<td>FUNCTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIX</td>
<td>INTEGER*4</td>
<td>FUNCTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.IGN</td>
<td>REAL</td>
<td>FUNCTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRPY</td>
<td></td>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TQ</td>
<td></td>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDPOS</td>
<td></td>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERO</td>
<td></td>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pass One  
No Errors Detected  
506 Source Lines
Appendix 3

OMV PLOT — Source Listing
Program OMVPLOT

by

Dr. W. Teoh

U A H 1984

This is a graphical package that accepts a command string and uses this information to generate and display the position and orientation of TOM_B and the attached mock-up module in two dimensions. One can choose to display either the top or side view of the system.

This package is developed in FORTRAN 77 to run on an IBM PC with at least 128K of RAM, and fitted with a TECMAR GRAPHICS MASTER board. An IBM Monochrome monitor is used for the actual display. The resolution in this work is chosen to be 640 x 350.

SUBROUTINE SIDEVIEW (H, X, P)
REAL * 8 H, X, P, C, S
REAL XFORM(3,3), SDFORM(3,3), VO(3,10), V(3,10)
REAL ROT(3,3), FLOOR(3,3), V1(3,10)
REAL CC, DD, LL, RR, WW, TT
INTEGER FLAG, N, CLR, EF, EEF, PRTFG

COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
COMMON /MF/ XFORM, SDFORM, VO, V1
COMMON /ME/ EF, EEF, PRTFG

N = 10
AA = 1.0
*** define mock-up module shape at origin
DO 100 K = 1, N
  V(3, K) = 1.0
CONTINUE
V(1,1) = TT
V(2,1) = -DD
V(1,2) = -TT
V(2,2) = -DD
V(1,3) = -TT
V(2,3) = DD
V(1,4) = TT
V(2,4) = DD
*** rotate it by P radians
CALL SINCOS (P, S, C)
CALL NOTHNG (ROT, 3)
ROT(1,1) = C
ROT(1,2) = -S
ROT(2,1) = S
ROT(2,2) = C
CALL XMUL (ROT, V, 4)
*** calculate translation
Line# 1  7

99 C
100  PX = CC + LL * C  
101  PY = H +    LL * S  
102 C
103 C  *** move the rotated module out there  
104 C
105  CALL NOTHING (ROT, 3)  
106  ROT(1,3) = PX  
107  ROT(2,3) = PY  
108  CALL XMUL (ROT, V, 4)  
109 C
110 C  *** now calculate the shape of the base  
111 C
112 C  XX = X + CC  
113 C  << point E >>  
114  V(1,5) = CC  
115  V(2,5) = H  
116 C  << point F >>  
117  V(1,6) = CC  
118  V(2,6) = AA  
119 C  << point G >>  
120  V(1,7) = CC  
121  V(2,7) = 0.  
122 C  << point H >>  
123  V(1,8) = -RR  
124  V(2,8) = 0.  
125 C  << point I >>  
126  V(1,9) = -RR  
127  V(2,9) = AA  
128 C
129  V(1,10) = PX  
130  V(2,10) = PY  
131 C
132 C  *** Transform to floor coordinates  
133 C
134  CALL NOTHING (FLOOR, 3)  
135  FLOOR(1,3) = X  
136  CALL XMUL (FLOOR, V, N)  
137 C
138 C  *** transform to screen coordinates  
139 C
140  CALL XMUL (SDFORM, V, N)  
141 C
142 C  *** erase old picture  
143 C
144  CALL DRWFLR (VO)  
145 IF ((EF.EQ. 0).AND. (EEF .NE. 0)) THEN  
146  CLR = 0  
147  CALL SDRAW (V1, N, CLR)
<< O M V P L O T >>>

D Line# 1  7
148      END IF
149      CLR = 1
150      CALL SDRAW (V, N, CLR)
151      CALL MOVE (V, V1, N)
152      EEF = 1
153 C
154      RETURN
155      END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>REAL</td>
<td>198</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>REAL*8</td>
<td>218</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>REAL</td>
<td>4</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>INTEGER*4</td>
<td>234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>REAL</td>
<td>8</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>EEF</td>
<td>INTEGER*4</td>
<td>4</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>EF</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>FLAG</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>FLOOR</td>
<td>REAL</td>
<td>158</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>202</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LL</td>
<td>REAL</td>
<td>12</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>INTEGER*4</td>
<td>194</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>PRTFG</td>
<td>INTEGER*4</td>
<td>8</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>PX</td>
<td>REAL</td>
<td>226</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PY</td>
<td>REAL</td>
<td>230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROT</td>
<td>REAL</td>
<td>122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>REAL</td>
<td>16</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>REAL*8</td>
<td>210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDFORM</td>
<td>REAL</td>
<td>36</td>
<td>/MF</td>
<td></td>
</tr>
<tr>
<td>IT</td>
<td>REAL</td>
<td>24</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>REAL</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>REAL</td>
<td>72</td>
<td>/MF</td>
<td></td>
</tr>
<tr>
<td>V1</td>
<td>REAL</td>
<td>192</td>
<td>/MF</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>REAL</td>
<td>20</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>XFORM</td>
<td>REAL</td>
<td>0</td>
<td>/MF</td>
<td></td>
</tr>
</tbody>
</table>

156 $PAGE
SUBROUTINE SDRAW (V, N, CLR)

This procedure draws the side view of TOM_B

REAL V(3,10)
INTEGER N, CLR, X1, X2, Y1, Y2

*** draw mobile base
CALL RCT (V, 5, CLR)

*** draw linkage
X1 = V(1,6)
Y1 = V(2,6)
X2 = V(1,5)
Y2 = V(2,5)
CALL LINE (X1, Y1, X2, Y2, CLR)
X1 = V(1,10)
Y1 = V(2,10)
CALL LINE (X2, Y2, X1, Y1, CLR)

*** draw mock-up module
CALL RCT (V, 0, CLR)
CALL PURGE
CALL GRFRDY
CALL HOME
RETURN
END

%me  Type  Offset  P Class

INTEGER*4  8  *
INTEGER*4  4  *
REAL      0  *
INTEGER*4 238
<table>
<thead>
<tr>
<th>Line#</th>
<th>X2</th>
<th>INTEGER*4</th>
<th>246</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y1</td>
<td>INTEGER*4</td>
<td>242</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>INTEGER*4</td>
<td>250</td>
</tr>
</tbody>
</table>

199 $PAGE$
SUBROUTINE RCT (V, OFF, CLR)

This procedure draws a rectangle

REAL V(3,10)
INTEGER OFF, CLR, X(10), Y(10)

DO 100 K = 1, 4
    J = K + OFF
    X(K) = V(1,J)
    Y(K) = V(2,J)
100 CONTINUE
CALL POLYGN(4, X, Y, CLR)
RETURN
END
SUBROUTINE PLOT (CMD)

This is the plot part of the graphical package, and can be directly callable from OMV or SVX. The value of FLAG obtained from the disk file named SIZE.DAT dictates one of top or side view to be displayed.

INTEGER CMD(7), FLAG
REAL * 8  X, Y, T, UL, UA, H
REAL XFORM(3,3), SDFORM(3,3), CC, LL, DD, RR, WW, TT
REAL VO(3,10), VL(3,10)

COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
COMMON /MF/ XFORM, SDFORM, VO, VL

UL = 10000.0
UA = UL

IF (FLAG .EQ. 0) THEN
   T = CMD(1) / UA
   X = CMD(2) / UL
   Y = CMD(3) / UL
   CALL TOPVIEW (X, Y, T)
ELSE
   H = CMD(4) / UL
   X = CMD(2) / UL
   T = CMD(5) / UA
   CALL SIDEVIEW (H, X, T)
END IF

RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>REAL</td>
<td>4</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>CMD</td>
<td>INTEGER*4</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>REAL</td>
<td>8</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>FLAG</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>REAL*8</td>
<td>382</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>REAL</td>
<td>12</td>
<td>/MG</td>
<td></td>
</tr>
</tbody>
</table>
### Neo# 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>16</td>
<td>/MG</td>
</tr>
<tr>
<td>REAL</td>
<td>36</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL*8</td>
<td>358</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>24</td>
<td>/MG</td>
</tr>
<tr>
<td>REAL*8</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>342</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>72</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL</td>
<td>192</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL</td>
<td>20</td>
<td>/MG</td>
</tr>
<tr>
<td>REAL*8</td>
<td>366</td>
<td></td>
</tr>
<tr>
<td>FORM REAL</td>
<td>0</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL*8</td>
<td>374</td>
<td></td>
</tr>
</tbody>
</table>

265 $PAGE
SUBROUTINE TOPVIEW (PX, PY, THETA)

This procedure constructs the top view of TOM_B. No correction to perspective distortion is implemented.

REAL * 8 PX, PY, THETA, S, C
REAL V(3,10), VO(3,10), SDFORM(3,3)
REAL ROT(3,3), FLOOR(3,3), XFORM(3,3)
REAL CC, DD, LL, RR, WW, TT, VI(3,10)
INTEGER FLAG, N, CLR, EF, EEF, PRTFG
COMMON /MG/ FLAG, CC, DD, LL, RR, TT
COMMON /MF/ XFORM, SDFORM, VO, VI
COMMON /HE/ EF, EEF, PRTFG

N = 10

*** get TOM_B shape at the origin
CALL ORGPOS (V, N)

*** rotate by THETA if needed
IF (THETA .NE. 0.0) THEN
    *** construct rotation matrix
    CALL NOTHNG (ROT, 3)
    CALL SINCOS (THETA, S, C)
    ROT(1,1) = C
    ROT(1,2) = -S
    ROT(2,1) = S
    ROT(2,2) = C
    *** rotate it
    CALL XMUL (ROT, V, N)
END IF

*** transform to floor coordinates
CALL NOTHNG (FLOOR, 3)
FLOOR(1,3) = PX
FLOOR(2,3) = PY
CALL XMUL (FLOOR, V, N)
*** transform to screen coordinates
CALL XMUL (XFORM, V, N)

*** get ready to draw, but first erase old picture
CALL DRWFLR (V1)
IF ((EF .EQ. 0) .AND. (EEF .NE. 0)) THEN
CLR = 0
CALL DRAW (VO, N, CLR)
END IF

CLR = 1
CALL DRAW (V, N, CLR)
CALL MOVE (V, VO, N)
EEF = 1
RETURN
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>594</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>4</td>
<td>/MG</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>602</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>8</td>
<td>/MG</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>4</td>
<td>/MG</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>/MG</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>/MG</td>
</tr>
<tr>
<td>REAL*8</td>
<td>546</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>12</td>
<td>/MG</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>582</td>
<td></td>
</tr>
<tr>
<td>TF</td>
<td>8</td>
<td>/ME</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>/ME</td>
</tr>
<tr>
<td>REAL*8</td>
<td>510</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>16</td>
<td>/ME</td>
</tr>
<tr>
<td>REAL*8</td>
<td>586</td>
<td></td>
</tr>
<tr>
<td>FORM</td>
<td>36</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL*8</td>
<td>8</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL</td>
<td>24</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL</td>
<td>390</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL</td>
<td>72</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL</td>
<td>192</td>
<td>/MF</td>
</tr>
<tr>
<td>REAL</td>
<td>20</td>
<td>/MF</td>
</tr>
<tr>
<td>ORM*</td>
<td>0</td>
<td>/MF</td>
</tr>
</tbody>
</table>
SUBROUTINE MOVE (V, VO, N)

This procedure saves the shape vector V.

REAL V(3,10), VO(3,10)

DO 100 K = 1, N
   DO 100 J = 1, 3
      VO(J,K) = V(J,K)
   CONTINUE

100 CONTINUE

RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>614</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>606</td>
<td>*</td>
</tr>
<tr>
<td>N</td>
<td>INTEGER*4</td>
<td>8</td>
<td>*</td>
</tr>
<tr>
<td>V</td>
<td>REAL</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>VO</td>
<td>REAL</td>
<td>4</td>
<td>*</td>
</tr>
</tbody>
</table>

359 $PAGE
SUBROUTINE NOTHING (A, N)

This procedure initializes an N x N matrix A to a unit matrix.

REAL A(N,N)

DO 100 K = 1, N
   DO 200 J = 1, N
      A(K,J) = 0.0
   CONTINUE
   A(K,K) = 1.0
100 CONTINUE

RETURN

END
SUBROUTINE XMUL (R, V, N)

This procedure uses a transformation matrix R and transforms the shape vector V having N columns.

Type | Offset | P Class
--- | --- | ---
INTEGER*4 | 646 | |
INTEGER*4 | 662 | |
INTEGER*4 | 666 | |
INTEGER*4 | 8 | *
REAL | 0 | *
INTEGER*4 | 654 | |
REAL | 658 | |
REAL | 634 | |
REAL | 4 | *
SUBROUTINE ORGPOS (V, N)

This procedure calculates the shape vector V of TOM_B at the origin. Only the top view is considered here.

DO 100 K = 1, N
   V(3, K) = 1.0
Continuing

*** set up shape matrix V

CL = CC + LL

Corner << A >>
V(1, 1) = CC
V(2, 1) = 0

Corner << B >>
V(1, 2) = CC
V(2, 2) = -WW

Corner << C >>
V(1, 3) = -RR
V(2, 3) = -WW

Corner << D >>
V(1, 4) = -RR
V(2, 4) = WW

Corner << E >>
V(1, 5) = CC
V(2, 5) = WW

Corner << MM >>
V(1, 6) = CL
```
 Line#  7  V(2, 6) = 0
 470  C
 471  V(1, 7) = CL + TT
 472  V(2, 7) = -DD
 473  C
 474  V(1, 8) = CL - TT
 475  V(2, 8) = -DD
 476  C
 477  V(1, 9) = CL - TT
 478  V(2, 9) = DD
 479  C
 480  V(1,10) = CL + TT
 481  V(2,10) = DD
 482  C
 483  RETURN
 484  END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CL</td>
<td>REAL</td>
<td>702</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORN</td>
<td>INTEGER*4</td>
<td>678</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>REAL</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEF</td>
<td>INTEGER*4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EF</td>
<td>INTEGER*4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLAG</td>
<td>INTEGER*4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>694</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LL</td>
<td>REAL</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>REAL</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDFORM</td>
<td>REAL</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IT</td>
<td>REAL</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>REAL</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>REAL</td>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'I</td>
<td>REAL</td>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'J</td>
<td>REAL</td>
<td>670</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WW</td>
<td>REAL</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFORM</td>
<td>REAL</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

486 PAGE
```
SUBROUTINE INITPL

This procedure initializes the system and calculates all the necessary transformation matrices based on the data obtained from SIZE.DAT

REAL VO(3,10), XFORM(3,3), SDFORM(3,3), W(2)
REAL CC, DD, LL, RR, WW, TT, V1(3,10)
REAL CORNR(2,2), W(2)
INTEGER CORNR(2,2), W(2)
REAL VO(3,10), XFORM(3,3), SDFORM(3,3), W(2)
REAL CC, DD, LL, RR, WW, TT, V1(3,10)
REAL CORNR(2,2), W(2)
INTEGER CORNR(2,2), W(2)
COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
COMMON /MF/ XFORM, SDFORM, VO, V1
COMMON /ME/ EF, EEF, PRTFG

EEF = 0
OPEN (7, FILE = 'SIZE.DAT')
READ (7, 10) CC, DD, LL, RR, WW, TT
DO 200 K = 1, 2
   READ (7, 20) (CORNR(K,J), J=1, 2)
200 CONTINUE
W(1) = 12.2
W(2) = 24.4
CALL CORDX (CORNR, XFORM, W)
DO 300 K = 1, 2
   READ (7, 20) (CORNS(K,J), J=1, 2)
300 CONTINUE
W(1) = 12.2
W(2) = 6.096
CALL CORDX (CORNS, SDFORM, W)
READ (7, 20) EF
READ (7, 20) FLAG
READ (7, 20) PRTFG
CLOSE (7)
FLG = 1
*** calculate floor shape
IF (FLAG .EQ. 0) THEN
   J1 = CORNR(1,1)
   L1 = CORNR(1,2)
   J2 = CORNR(2,1)
   L2 = CORNR(2,2)
   JJ = (L2 - L1 + 1) / 2
   V1(1,1) = J1
   V1(2,1) = L1
   V1(1,2) = J2
   V1(2,2) = L1
   V1(1,3) = J2
   V1(2,3) = L2
   V1(1,4) = J1
   V1(2,4) = L2
   V1(1,5) = J1
   V1(2,5) = L2 + JW - JJ
   V1(1,6) = J1 - JL
   V1(2,6) = L2 + JW - JJ
   V1(1,7) = J1 - JL
   V1(2,7) = L2 - JL - JJ
   V1(1,8) = J1
   V1(2,8) = L2 - JL - JJ
   V1(1,9) = -1000.0
   V1(2,9) = -1000.0
ELSE
   J1 = CORNS(1,1)
   L1 = CORNS(1,2)
   J2 = CORNS(2,1)
   L2 = CORNS(2,2)
   VO(1,1) = J1 - JL
   VO(2,1) = L2 + 1
   VO(1,2) = J2 + JL
   VO(2,2) = L2 + 1
   VO(1,3) = -1000.0
   VO(2,3) = -1000.0
END IF

CALL GRAFICS
RETURN

OFFSET P CLASS
Line# 1  7

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>REAL</td>
<td>4</td>
</tr>
<tr>
<td>ORNR</td>
<td>INTEGER*4</td>
<td>714</td>
</tr>
<tr>
<td>ORNS</td>
<td>INTEGER*4</td>
<td>730</td>
</tr>
<tr>
<td>DD</td>
<td>REAL</td>
<td>8</td>
</tr>
<tr>
<td>SEEF</td>
<td>INTEGER*4</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>INTEGER*4</td>
<td>0</td>
</tr>
<tr>
<td>FLAG</td>
<td>INTEGER*4</td>
<td>0</td>
</tr>
<tr>
<td>ELG</td>
<td>REAL</td>
<td>758</td>
</tr>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>770</td>
</tr>
<tr>
<td>J2</td>
<td>INTEGER*4</td>
<td>778</td>
</tr>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>786</td>
</tr>
<tr>
<td>JW</td>
<td>INTEGER*4</td>
<td>766</td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>746</td>
</tr>
<tr>
<td>L</td>
<td>INTEGER*4</td>
<td>774</td>
</tr>
<tr>
<td>L2</td>
<td>INTEGER*4</td>
<td>782</td>
</tr>
<tr>
<td>LL</td>
<td>REAL</td>
<td>12</td>
</tr>
<tr>
<td>RTFG</td>
<td>INTEGER*4</td>
<td>8</td>
</tr>
<tr>
<td>R</td>
<td>REAL</td>
<td>16</td>
</tr>
<tr>
<td>SDFORM</td>
<td>REAL</td>
<td>36</td>
</tr>
<tr>
<td>T</td>
<td>REAL</td>
<td>24</td>
</tr>
<tr>
<td>J</td>
<td>REAL</td>
<td>72</td>
</tr>
<tr>
<td>V1</td>
<td>REAL</td>
<td>192</td>
</tr>
<tr>
<td>W</td>
<td>REAL</td>
<td>706</td>
</tr>
<tr>
<td>J</td>
<td>REAL</td>
<td>20</td>
</tr>
<tr>
<td>AFORM</td>
<td>REAL</td>
<td>0</td>
</tr>
</tbody>
</table>

582 $PAGE
SUBROUTINE DRWFLR (V)

This subroutine draws the floor portion of graphics

REAL V(3,10)
INTEGER CT, X(10), Y(10)

CT = 1

REPEAT

K = CT

X(K) = V(1,K)
Y(K) = V(2,K)

CT = CT + 1

IF (V(1,CT) .GE. -100.0) GO TO 100

UNTIL V(1,CT) < -100.0

CALL POLYGN (K, X, Y, 1)

RETURN

END
SUBROUTINE DRAW (V, N, CLR)

This procedure actually draws the top view of TOM_B.

This procedure must be modified if different hardware
is used for the graphics display

REAL V(3, 10)
INTEGER X1, X2, Y1, Y2
INTEGER CLR

*** draw mobile base
CALL RCT (V, 1, CLR)

*** draw connecting line
X1 = V(1,1)
Y1 = V(2,1)
X2 = V(1,6)
Y2 = V(2,6)
CALL LINE (X1, Y1, X2, Y2, CLR)

*** draw mocked-up
CALL RCT (V, 6, CLR)

CALL PURGE
CALL GRFRDY
CALL HOME

RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLR</td>
<td>INTEGER*4</td>
<td>8</td>
<td>*</td>
</tr>
<tr>
<td>I</td>
<td>INTEGER*4</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td>0</td>
<td>*</td>
</tr>
</tbody>
</table>
```
Integer*4  892
Integer*4  900
Integer*4  896
Integer*4  904
```
SUBROUTINE CORDX (C, T, W)

This procedure computes the necessary transformation matrices from floor to screen coordinates

** set up transformation matrix T

```
677    T(1,3) = C(1,1)
678    T(2,3) = C(2,2)
679    T(3,3) = 1.0
680    T(1,1) = (C(2,1) - T(1,3)) / W(1)
681    T(2,1) = (C(2,2) - T(2,3)) / W(1)
682    T(3,1) = 0.0
683    T(1,2) = (C(1,1) - T(1,3)) / W(2)
684    T(2,2) = (C(1,2) - T(2,3)) / W(2)
685    T(3,2) = 0.0
```

RETURN

END
This is a graphics package for the TECMAR GRAPHICS MASTER board written under Microsoft's FORTRAN 77. To use this package, one must include this package in the source file. A graphics master must already be installed, or the software will hang.

SUBROUTINE PURGE

This procedure purges the graphics buffer and forces the board to complete the drawing by closing the graphics channel.

INTEGER GRF
CHARACTER ESC
COMMON /GMBD/ GRF, ESC
CLOSE (GRF)
RETURN
END

```c
<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR*1</td>
<td>4</td>
<td>/GMBD</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>/GMBD</td>
</tr>
</tbody>
</table>
```

710 $PAGE
**SUBROUTINE GRFRDY**

This procedure opens the graphics channel and sets it ready for communication.

```fortran
INTEGER GRF
CHARACTER*1 ESC
COMMON /GMBD/ GRF, ESC
OPEN (GRF, FILE = 'gm')
RETURN
END
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC</td>
<td>CHAR*1</td>
<td>4</td>
<td>/GMBD</td>
<td>/</td>
</tr>
<tr>
<td>GRF</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/GMBD</td>
<td>/</td>
</tr>
</tbody>
</table>

$PAGE$
SUBROUTINE SETFB (FG, BG)

This procedure sets the foreground color to FG and the background color to BG. Both arguments must be of INTEGER type.

INTEGER GRF, FG, BG
CHARACTER ESC
COMMON /GMBD/ GRF, ESC
WRITE (GRF, 10) ESC, FG, BG
RETURN

FORMAT (' ', A1, '!', I2, ';', I2, 'c')

END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>CHAR*1</td>
<td>4</td>
<td>/GMBD</td>
</tr>
<tr>
<td>G</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/GMBD</td>
</tr>
<tr>
<td>F</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/GMBD</td>
</tr>
</tbody>
</table>

742 $PAGE
SUBROUTINE GRAFICS

This procedure enters the GM graphics mode with a four-line text window at the bottom.

INTEGER GRF
CHARACTER ESC
COMMON /GMBD/ GRF, ESC

GRF = 9
ESC = CHAR(27)
CALL GRFRDY
WRITE (GRF, 10) ESC
WRITE (GRF, 20) ESC
WRITE (GRF, 30) ESC
CALL SETFB (1, 0)
CALL HOME
RETURN

FORMAT (' ', A1, '[!Om']
FORMAT (' ', A1, '[!640;352;2g']
FORMAT (' ', A1, '[21;24r']
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>INTRINSIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESC</td>
<td>CHAR*1</td>
<td>4</td>
<td>/GMBD</td>
</tr>
<tr>
<td>GRF</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/GMBD</td>
</tr>
</tbody>
</table>

770 $PAGE
SUBROUTINE QUITGM

This procedure gets one out of graphics mode and returns to text mode

CHARACTER CH, ESC
INTEGER GRF
COMMON /GMBD/ GRF, ESC

CALL HOME
WRITE (GRF, 30)
CALL PURGE
READ (*, 10) CH
CALL GRFRDY
CALL TEXT
RETURN

FORMAT (AI)
FORMAT ('Press <CR> to continue ... ')
END
SUBROUTINE TEXT

This procedure returns the system to text mode

INTEGER GRF
CHARACTER ESC
COMMON /GMBD/ GRF, ESC
WRITE (GRF, 10) ESC
RETURN

NAME   TYPE    OFFSET P CLASS

ESC    CHAR*1   4   /GMBD /
GRF    INTEGER*4 0   /GMBD /

$PAGE
SUBROUTINE LINE (X1, Y1, X2, Y2, COLOR)

This procedure draws a line from (X1,Y1) to (X2,Y2) in COLOR

INTEGER GRF, X1, Y1, X2, Y2, COLOR
CHARACTER ESC

COMMON /GMBD/ GRF, ESC

WRITE (GRF, 10) ESC, X1, Y1, X2, Y2, COLOR

10 FORMAT (' ', A1, ['!', 4(I3,';'), I3, '1'])

END

MOR INTEGER*4 16 *
CHAR*1 4 /GMBD /
INTEGER*4 0 /GMBD /
INTEGER*4 0 *
INTEGER*4 8 *
INTEGER*4 4 *
INTEGER*4 12 *

826 $PAGE
SUBROUTINE HIDE LN (X1, Y1, X2, Y2, COLOR)

This procedure draws the line (X1,Y1) - (X2,Y2) but aborts drawing before reaching target if a dot in a color other than BG is encountered.

```
INTEGER GRF, X1, Y1, X2, Y2, COLOR
CHARACTER ESC
COMMON /GMBD/ GRF, ESC
WRITE (GRF, 10) ESC, X1, Y1, X2, Y2, COLOR
RETURN
10 FORMAT (' ', A1, '[!]', 4(I3, ';'), I3, 'S')
```

Offset P Class

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>16</td>
<td>*</td>
</tr>
<tr>
<td>3C</td>
<td>4</td>
<td>/GMBD</td>
</tr>
<tr>
<td>GRF</td>
<td>0</td>
<td>/GMBD</td>
</tr>
<tr>
<td>X1</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>X2</td>
<td>8</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>Y2</td>
<td>12</td>
<td>*</td>
</tr>
</tbody>
</table>

$PAGE

SUBROUTINE POLYGN(N, X, Y, COLOR)

This procedure draws a closed polygon whose N vertices are stored in the arrays X and Y. The color to be used is COLOR.

INTEGER GRF, X(N), Y(N), COLOR
CHARACTER ESC

COMMON /GHBD/ GRF, ESC

WRITE (GRF, 10) ESC
DO 100 K = 1, N
    WRITE (GRF, 20) X(K), Y(K)
CONTINUE
WRITE (GRF, 30) COLOR
RETURN

FORMAT (' ', A1, '[]')
FORMAT ( 2(I3, '[:])
FORMAT (   I3, 'p')
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOR</td>
<td>INTEGER*4</td>
<td>12     *</td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>INTEGER*4</td>
<td>4 GMBD</td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>INTEGER*4</td>
<td>0 GMBD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>1160</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>0 *</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>4 *</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>8 *</td>
<td></td>
</tr>
</tbody>
</table>

PAGE
SUBROUTINE HOME

THIS SUBROUTINE HOMES THE CURSOR

INTEGER GRF

COMMON /GMBD/ GRF, ESC

WRITE (GRF, 10) ESC

RETURN

FORMAT (',', A1, '[ 1;1 f']

END

Name  Offset P Class
---  ----- ----
ESC   CHAR*1 4 /GMBD /
GRF   INTEGER*4 0 /GMBD /

885 $PAGE
<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDX</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TLR</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>BLD</td>
<td>5</td>
<td>COMMON</td>
</tr>
<tr>
<td>AFIC</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>RDY</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>ELN</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>ME</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TPL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td>312</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>COMMON</td>
</tr>
<tr>
<td>RE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>THING</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>POS</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>JT</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>DLYGN</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>GE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TGM</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>CT</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>DRAW</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>IFB</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>DEVE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>INCOS</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>XT</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>PVIEW</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
</tbody>
</table>

Pass One  No Errors Detected
-  885 Source Lines
Appendix 4

OMV -- Data files
File: INITCON.DAT

This file contains all the needed initial conditions

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>POS(1) -- initial condition</td>
</tr>
<tr>
<td>0.0</td>
<td>POS(2) -- initial condition</td>
</tr>
<tr>
<td>0.0</td>
<td>POS(3) -- initial condition</td>
</tr>
<tr>
<td>0.00</td>
<td>VEL(1) -- initial condition</td>
</tr>
<tr>
<td>0.0</td>
<td>VEL(2) -- initial condition</td>
</tr>
<tr>
<td>0.0</td>
<td>VEL(3) -- initial condition</td>
</tr>
<tr>
<td>0.0</td>
<td>EUL(1) -- initial condition .. ROLL</td>
</tr>
<tr>
<td>0.0</td>
<td>EUL(2) -- initial condition .. PITCH</td>
</tr>
<tr>
<td>0.0</td>
<td>EUL(3) -- initial condition .. YAW</td>
</tr>
</tbody>
</table>
File: MDLPRM.DAT

This file contains all the model parameters needed by OMV

```
00.075  ACC(1) : Acc along X-axis (body)
00.075  ACC(2) : Acc along Y-axis (body)
00.075  ACC(3) : Acc along Z-axis (body)
000.52359878 WWB(1) : body rate about X axis
000.52359878 WWB(2) : body rate about Y axis
000.52359878 WWB(3) : body rate about Z axis
7048.37  III(1) principal moment of inertia along 1 axis
3713.95  III(2) principal moment of inertia along 2 axis
3713.95  III(3) principal moment of inertia along 3 axis
3282.75  Mass in kilograms
0.1     major cycle period in seconds
1      MODE : 1 for position control
10     No. of steps per major cycle
200.0   altitude of orbit in kilo-meters
```
File: SVXINT.DAT

This file contains all the system initialization data needed by the SVX module

<table>
<thead>
<tr>
<th>Value</th>
<th>Variable</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5588</td>
<td>CC</td>
<td>IN METERS</td>
</tr>
<tr>
<td>0.762</td>
<td>LL</td>
<td>IN METERS</td>
</tr>
<tr>
<td>11.668</td>
<td>AA</td>
<td>IN METERS</td>
</tr>
<tr>
<td>2.4384</td>
<td>HH</td>
<td>IN METERS</td>
</tr>
<tr>
<td>7048.37</td>
<td>IINV(1)</td>
<td></td>
</tr>
<tr>
<td>3713.95</td>
<td>IINV(2)</td>
<td></td>
</tr>
<tr>
<td>3713.95</td>
<td>IINV(3)</td>
<td></td>
</tr>
</tbody>
</table>
File: HNDGSL.DAT

This file contains the simulated hand controller signals  
(Partial list)

100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
File: SIZE.DAT

This file contains all the plot parameters for the graphics package PLOT

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5588</td>
<td>CC: 22 inches</td>
</tr>
<tr>
<td>2.1336</td>
<td>DD: 84 inches</td>
</tr>
<tr>
<td>0.762</td>
<td>LL: 30 inches</td>
</tr>
<tr>
<td>1.016</td>
<td>RR: 40 inches</td>
</tr>
<tr>
<td>0.6096</td>
<td>WW: 24 inches</td>
</tr>
<tr>
<td>0.3048</td>
<td>TT: 12 inches</td>
</tr>
<tr>
<td>409</td>
<td>CORNR(1,1)</td>
</tr>
<tr>
<td>001</td>
<td>CORNR(1,2)</td>
</tr>
<tr>
<td>630</td>
<td>CORNR(2,1)</td>
</tr>
<tr>
<td>350</td>
<td>CORNR(2,2)</td>
</tr>
<tr>
<td>100</td>
<td>CORNR(1,1) SIDE VIEW</td>
</tr>
<tr>
<td>152</td>
<td>CORNR(1,2) SIDE VIEW</td>
</tr>
<tr>
<td>500</td>
<td>CORNR(2,1) SIDE VIEW</td>
</tr>
<tr>
<td>300</td>
<td>CORNR(2,2) SIDE VIEW</td>
</tr>
<tr>
<td>000</td>
<td>PLOT MODE: &lt;&gt; 0 MEANS NO CLEAR</td>
</tr>
<tr>
<td>000</td>
<td>VIEW: 0 = TOP VIEW, &lt;&gt; 0 = SIDE VIEW</td>
</tr>
<tr>
<td>001</td>
<td>PRTFG: 1-PLOT 2-PRINT 3-PLOT &amp; PRINT</td>
</tr>
</tbody>
</table>
Appendix 5

ONV — Source Listing
This is a simplified version of a mathematical simulation model of the OMV. In this model, the following simplifications and assumptions are made:

1. The hand controllers provide signals that are interpreted as a force at the center of mass and/or a torque about the center of mass to provide a rotation of constant angular velocity.

2. The target vehicle is in a circular orbit; the altitude of this orbit is inputted from the MDLPRM.DAT file.

3. Orbital mechanics is implemented, but smaller perturbation effects are totally ignored.

4. Detailed placement of thrusters is not considered (Please see assumption 1. above)

5. Roll, pitch and yaw denote the instantaneous orientation of the OMV.

A 14 component state vector is generated by this model, and this state vector serves as input to the SVX module.

```fortran
REAL * 8 X(3), V(3), E(3), A(3), W(3), Q(4)
REAL * 8 POS(3), VEL(3), EUL(3), OMEGA
REAL * 8 III(3), S(14), MASS, CYCLE
INTEGER CMD(7), IN, FLAG, MODE, STEP
INTEGER * 4 TIME
COMMON /MC/ III, MASS, CYCLE, MODE, STEP
COMMON /PC/ POS, VEL, EUL, OMEGA
```
*** system initialization

IN = 2
TIME = -1
CALL OMVMDL (IN)
OPEN (IN, FILE = 'HNDSL.GAT')

*** Note: this invokes graphics routines, and can be
eliminated if no graphics output.

CALL INITPL

*** calculate the initial quaternions at the start of the
*** simulation and read hand controller

CALL DETQ (EUL, Q)
CALL HNDCTRL (IN, FLAG, A, W)
CALL MATCH (EUL, POS, VEL, E, X, V, 3)
CALL STATE (Q, S, W)
CALL SVX (S, CMD, MODE)
CALL OUTPUT (A, W, X, V, E, Q, S, CMD, TIME)
TIME = 0

*** main processing loop

WHILE (FLAG = 0) DO
  IF (FLAG .NE. 0) GOTO 900

  *** copy initial state into work vectors and use these
  *** work vectors for solving the equations of motion
  CALL MOTION (X, V, E, A, W, Q)

  *** update dynamic state
  CALL MATCH (E, X, V, EUL, POS, VEL, 3)

  *** calculate state vector and pass it on to the State
  *** Vector Transformation module
  CALL STATE (Q, S, W)
  CALL SVX (S, CMD, MODE)
  CALL OUTPUT (A, W, X, V, E, Q, S, CMD, TIME)

  *** poll hand controller and get the next set of signals
  CALL HNDCTRL (IN, FLAG, A, W)
  GOTO 100
END WHILE
CONTINUE
CLOSE (IN)
*** *** This is also a call to the graphics package
CALL QUITGM
*** Grand exit, stage left
STOP
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>242</td>
<td>/MC</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>266</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>32</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>74</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>48</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>302</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>294</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>24</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>40</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>72</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>24</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>50</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>2</td>
<td>/MC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE OMVMDL (IN)

This procedure obtains the necessary parameters of the OMV by reading them from a disk file called MDLPRM.DAT after getting the initial state of the OMV (from a file called INITCON.DAT)

REAL * 8  POS(3), VEL(3), EUL(3), OMEGA
REAL * 8  ACC(3), III(3), WWB(3), INV(3)
REAL * 8  MASS, CYCLE, ORBIT
INTEGER   IN, MODE, STEP

COMMON /DC/   ACC, WWB
COMMON /MC/   III, MASS, CYCLE, MODE, STEP
COMMON /PC/   POS, VEL, EUL, OMEGA

*** get initial conditions of the OMV
OPEN  (IN, FILE = 'INITCON.DAT')
CALL  VECTOR (IN, POS, 3)
CALL  VECTOR (IN, VEL, 3)
CALL  VECTOR (IN, EUL, 3)
CLOSE (IN)

*** read acceleration, angular rates and
*** principal moments of inertia in body frame
OPEN  (IN, FILE = 'MDLPRM.DAT')
CALL  VECTOR (IN, ACC, 3)
CALL  VECTOR (IN, WWB, 3)
CALL  VECTOR (IN, III, 3)

*** read mass characteristics & other parameters
READ  (IN, 10)  MASS
READ  (IN, 10)  CYCLE
READ  (IN, 20)  MODE
READ  (IN, 30)  STEP
READ  (IN, 10)  ORBIT
CLOSE (IN)

*** calculate orbital frequency
Line# 7
160 CALL ANGFRE (ORBIT, OMEGA)
161 C
162 C
163 RETURN
164 10 FORMAT (F15.8)
165 20 FORMAT (I1)
166 30 FORMAT (I2)
167 END

<table>
<thead>
<tr>
<th>name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>REAL*8</td>
<td>0</td>
<td>/DC</td>
</tr>
<tr>
<td>JLE</td>
<td>REAL*8</td>
<td>32</td>
<td>/MC</td>
</tr>
<tr>
<td>UL</td>
<td>REAL*8</td>
<td>48</td>
<td>/PC</td>
</tr>
<tr>
<td>t</td>
<td>REAL*8</td>
<td>0</td>
<td>/MC</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>REAL*8</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>SSS</td>
<td>REAL*8</td>
<td>24</td>
<td>/MC</td>
</tr>
<tr>
<td>UE</td>
<td>INTEGER*4</td>
<td>40</td>
<td>/MC</td>
</tr>
<tr>
<td>WEGA</td>
<td>REAL*8</td>
<td>72</td>
<td>/PC</td>
</tr>
<tr>
<td>B</td>
<td>REAL*8</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>REAL*8</td>
<td>0</td>
<td>/PC</td>
</tr>
<tr>
<td>EP</td>
<td>INTEGER*4</td>
<td>44</td>
<td>/MC</td>
</tr>
<tr>
<td>EL</td>
<td>REAL*8</td>
<td>24</td>
<td>/PC</td>
</tr>
<tr>
<td>B</td>
<td>REAL*8</td>
<td>24</td>
<td>/DC</td>
</tr>
</tbody>
</table>
SUBROUTINE ANGFRE(ORB, W)

This procedure calculates the orbital angular frequency at a given altitude. In this calculation, the altitude must be given in kilo-meters. This is necessary in order for the calculations to be carried out without losing precision. The angular frequency W is in rad/second.

REAL * 8 ORB
REAL * 8 ALT, R3, W

ALT = ORB * 0.001
R3 = (6.370 + ALT) ** 3
W = DSQRT (398.86 / R3) * 0.001
RETURN
END

Name          Type    Offset  P Class
ALT           REAL*8   358          INTRINSIC
DSQRT         REAL*8   0 *
ORB           REAL*8   0 *
R3            REAL*8   366          *
W             REAL*8   4 *
SUBROUTINE VECTOR (M, A, N)

This procedure reads a vector A of N elements from input unit M

INTEGER M, N
REAL * 8 A(N)

DO 100 K = 1, N
READ (M, 10) A(K)
CONTINUE
RETURN

FORMAT (F15.8)
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>374</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

$PAGE
SUBROUTINE HNDCTL (IN, FLAG, A, W)

Simulates hand controllers input by reading from a file (called HNDSGL.DAT 12) integers to simulate a 12 bit output of the hand controllers. Bit assignments are as follows:

<table>
<thead>
<tr>
<th>bit</th>
<th>meaning (direction in body frame)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accelerate along +1 axis</td>
</tr>
<tr>
<td>2</td>
<td>Accelerate along -1 axis</td>
</tr>
<tr>
<td>3</td>
<td>Accelerate along +2 axis</td>
</tr>
<tr>
<td>4</td>
<td>Accelerate along -2 axis</td>
</tr>
<tr>
<td>5</td>
<td>Accelerate along +3 axis</td>
</tr>
<tr>
<td>6</td>
<td>Accelerate along -3 axis</td>
</tr>
<tr>
<td>7</td>
<td>Rotate about +1 axis</td>
</tr>
<tr>
<td>8</td>
<td>Rotate about -1 axis</td>
</tr>
<tr>
<td>9</td>
<td>Rotate about +2 axis</td>
</tr>
<tr>
<td>10</td>
<td>Rotate about -2 axis</td>
</tr>
<tr>
<td>11</td>
<td>Rotate about +3 axis</td>
</tr>
<tr>
<td>12</td>
<td>Rotate about -3 axis</td>
</tr>
</tbody>
</table>

REAL * 8 ACC(3), WWB(3)
REAL * 8 A(3), W(3)
INTEGER SL(6), SA(6), FLAG
COMMON /DC/ ACC, WWB

FLAG = 0
READ (IN, 10, END = 90, ERR = 90) SL, SA

*** no error, generate matrices A and W

CALL FUDGE (A, ACC, SL)
CALL FUDGE (W, WWB, SA)
RETURN

90 CONTINUE

*** error condition

FLAG = 1
RETURN

10 FORMAT (20I1)

END
<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>/DC</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>414</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>390</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>12</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>24</td>
<td>/DC</td>
<td></td>
</tr>
</tbody>
</table>

264 $PAGE$
SUBROUTINE FUDGE (A, ACC, SL)

*** Sets appropriate components based on SL

DO 100 K = 1, 6, 2
   J = (K + 1) / 2
   X = 0.0
   T = SL(K) + SL(K+1)
   IF (T .EQ. 1) THEN
      X = ACC(J)
      IF (SL(K) .EQ. 0) X = -X
   END IF
   A(J) = X
CONTINUE
RETURN
END

Name   Type        Offset P Class

A      REAL*8      0 *
ACC    REAL*8      4 *
J      INTEGER*4   450
K      INTEGER*4   446
SL     INTEGER*4   8 *
T      INTEGER*4   462
X      REAL*8      454

$PAGE
SUBROUTINE STATE (Q, S, W)

This procedure uses the dynamic quantities of the OMV and constructs a State Vector of the OMV. The 14 components of this State Vector S are defined as follows:

<table>
<thead>
<tr>
<th>Components</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(1) .. S(3)</td>
<td>Relative displacement between OMV and target</td>
</tr>
<tr>
<td>S(4) .. S(6)</td>
<td>Relative velocity between OMV &amp; target</td>
</tr>
<tr>
<td>S(7) .. S(9)</td>
<td>Angular momentum vector of OMV in LVF</td>
</tr>
<tr>
<td>S(10) .. S(13)</td>
<td>Attitude quaternions expressed in body frame, and</td>
</tr>
<tr>
<td>S(14)</td>
<td>Instantaneous mass, assumed constant throughout the simulation.</td>
</tr>
</tbody>
</table>

REAL * 8 POS(3), VEL(3), EUL(3), OMEGA
REAL * 8 III(3), QQ(4), MASS, CYCLE
REAL * 8 LB(3), LL(3), B(3,3), A(3,3)
REAL * 8 Q(4), W(3), L(3), S(14)
INTEGER MODE, STEP

COMMON /MC/ III, MASS, CYCLE, MODE, STEP
COMMON /PC/ POS, VEL, EUL, OMEGA

*** calculate angular momentum in body frame
CALL DOTPRD (III, W, LB, 3)

*** transforms it to local vertical frame
CALL DCSINV (Q, B)
CALL DMUL (B, LB, LL, 3)

*** Build state vector
N = 0
CALL PUT (N, S, POS, 3)
CALL PUT (N, S, VEL, 3)
CALL PUT (N, S, LL, 3)
CALL PUT (N, S, Q, 4)
```
<< O N V >>

<table>
<thead>
<tr>
<th>Line#</th>
<th>Description</th>
<th>Offset</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>339</td>
<td>S(14) = MASS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>340 C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>341</td>
<td>RETURN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>342</td>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name  Type   Offset P Class
------ ----- ------ -------
    J    REAL*8   642          
    I    REAL*8   570          
CYCLE REAL*8   32 /MC /      
UL     REAL*8   48 /PC /      
II     REAL*8   0 /MC /      
L      REAL*8   546          
LAMBDA REAL*8  498          
L      REAL*8   522          
MASS   REAL*8   24 /MC /      
MODE   INTEGER*4 40 /MC /    
        INTEGER*4 714         
MEGA   REAL*8   72 /PC /      
POS    REAL*8   0 /PC /      
    J    REAL*8   0 *         
    Q    REAL*8   466         
    S    REAL*8   4 *         
        INTEGER*4 44 /MC /    
EL     REAL*8   24 /PC /      
    w    REAL*8   8 *         

343 $PAGE
```
SUBROUTINE PUT(N, S, A, M)

*** The procedure copies a vector A into a larger one S starting at the N-th element of S

100

DO 100 K = 1, M
    N = N + 1
    S(N) = A(K)

100 CONTINUE
RETURN
END

me Type Offset P Class

REAL*8  8 *
INTEGER*4  718
INTEGER*4  12 *
INTEGER*4  0 *
REAL*8  4 *
SUBROUTINE DOTPRD (A, B, C, N)

*** This procedure calculates a vector C from two other vectors A and B such that C(I) = A(I) * B(I) for all i = 1 to N

REAL * 8 A(N), B(N), C(N)

DO 100 K = 1, N
   C(K) = A(K) * B(K)
100 CONTINUE
RETURN
END

Name   Type    Offset P Class
A      REAL*8   0 *
-B     REAL*8   4 *
C      REAL*8   8 *
K      INTEGER*4 726
_N     INTEGER*4 12 *

384 $PAGE
SUBROUTINE DETQ (E, Q)

*** calculates quaternions from the Euler angles using expression given by Zack.

REAL * 8 E(3), Q(4)
REAL * 8 C1, S1, C2, S2, C3, S3, THETA

THETA = E(1) / 2.0
CALL SINCOS (THETA, S1, C1)
THETA = E(2) / 2.0
CALL SINCOS (THETA, S2, C2)
THETA = E(3) / 2.0
CALL SINCOS (THETA, S3, C3)

Q(1) = S1 * C3 * C2 + C1 * S3 * S2
Q(2) = S1 * S3 * C2 + C1 * C3 * S2
Q(3) = C1 * S3 * C2 - S1 * C3 * S2
Q(4) = C1 * C3 * C2 - S1 * S3 * S2
RETURN
END
SUBROUTINE SINCOS (THETA, S, C)

*** this procedure returns the sine and cosine of an angle THETA.

REAL * 8 THETA, S, C, A

C = DCOS(THETA)
S = DSIN(THETA)
RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>REAL*8</td>
<td>*****</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>REAL*8</td>
<td>8</td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>DCOS</td>
<td>REAL*8</td>
<td>4</td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>DSIN</td>
<td>REAL*8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>REAL*8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THETA</td>
<td>REAL*8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

432 $PAGE
SUBROUTINE MOTION (X, V, E, A, W, Q)

*** This procedure solves the equation of motion

REAL * 8 POS(3), VEL(3), EUL(3), OMEGA
REAL * 8 X(3), V(3), E(3), A(3), W(3), Q(4)
REAL * 8 CIN(3,3), C(3,3), AA(3,10), B(3), QQ(4)
REAL * 8 WW(3), PI, TWO
REAL * 8 III(3), MASS, CYCLE
INTEGER MODE, STEP

COMMON /MC/ III, MASS, CYCLE, MODE, STEP
COMMON /PC/ POS, VEL, EUL, OMEGA

H = CYCLE / FLOAT(STEP)
N = STEP
PI = 355.0 / 113.0
TWO= PI * 2.0

*** Divide 1 major cycle into N equal subintervals and
*** determine the OMV state for each interval

DO 100 KK = 1, N

*** Update orientation

DO 200 J = 1, 3
    WW(J) = W(J) * H
    E(J) = E(J) + WW(J)
    IF (E(J) .GT. TWO) E(J) = E(J) - TWO
200 CONTINUE

*** Calculate quaternion for this rotation, and transform
*** it to local vertical frame with respect to initial frame

CALL DCSINV(Q, CIN)
CALL DMUL (CIN, A, B, 3)
CALL STORE (B, AA, KK)
CONTINUE

*** Solve the equation of motion using the Adam-Brashford method
CALL SOLVE (X, V, AA, N, H, OMEGA)
RETURN
END

Name   Type  Offset P Class
\   REAL*8  12 *
AA    REAL*8  990
B     REAL*8  1230
CIN   REAL*8  918
CYCLE REAL*8  32 /MC /
\   REAL*8  8 *
\   REAL*8  48 /PC /
FLOAT INTRINSIC
H     REAL  1254
III   REAL*8  0 /MC /
J     INTEGER*4 1286
KK    INTEGER*4 1278
MASS  REAL*8  24 /MC /
MODE  INTEGER*4 40 /MC /
N     INTEGER*4 1258
-OMEGA REAL*8  72 /PC /
PI    REAL*8  1262
POS   REAL*8  0 /PC /
Q     REAL*8  20 *
\Q    REAL*8  814
\STEP INTEGER*4 44 /MC /
TWO   REAL*8  1270
\Y   REAL*8  4 *
\EL   REAL*8  24 /PC /
W     REAL*8  16 *
\W   REAL*8  790
\   REAL*8  0 *
SUBROUTINE MATCH (A, B, C, P, Q, R, N)

*** This procedure makes an exact duplicate B of a vector A of N elements

REAL * 8 A(N), B(N), C(N), P(N), Q(N), R(N)

DO 100 K = 1, 3
   P(K) = A(K)
   Q(K) = B(K)
   R(K) = C(K)
100 CONTINUE

RETURN
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>1290</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

514 $PAGE
SUBROUTINE STORE (AAA, AA, K)

This procedure takes an instantaneous acceleration vector AAA and stores it in the acceleration matrix AA which is needed by the numerical integration process.

REAL * 8 AA(3, 10)
REAL * 8 AAA(3)
DO 100 J = 1, 3
   AA(J,K) = AAA(J)
100 CONTINUE
RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>AAA</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>1294</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE SOLVE(X, V, A, N, H, W)

This subroutine produces the numerical solution to the system of equations of motion using a 3 step Adam-Brashford method.

LOGICAL FLAG
REAL*8 X(3), V(3), A(3,10), AA(3,13), U(6,13)
REAL*8 WX2, WXW, WXWX3, HD12, F, W
COMMON /BLOCK/ AA, U, WX2, WXW, WXWX3, HD12
DATA FLAG /.TRUE./

*** pack user supplied nonhomomogenous part of DE into the higher part of AA
DO 10 I = 1,10
   DO 10 K = 1,3
      AA(K,I+3) = A(K,I)
   CONTINUE
10 CONTINUE

*** if this is the first call to solve (FLAG = T), then it is necessary to initialize some parameters
IF (FLAG) THEN
   CALL INIT(X, V, W, H)
   FLAG = .FALSE.
END IF

*** use the Adams-Brashford 3-step method to advance the solution H time units. Place the solution back into X and V.
DO 100 I = 4,N+3
   DO 100 J = 1,6
      U(J,I) = U(J,I-1) +
      HD12*(23*F(J,I-1)-16*F(J,I-2)+S*F(J,I-3))
   CONTINUE
100 X(1) = U(1,N+3)
101 V(1) = U(2,N+3)
102 X(2) = U(3,N+3)
103 V(2) = U(4,N+3)
104 X(3) = U(5,N+3)
105 V(3) = U(6,N+3)
*** reset U and AA for the next call to SOLVE

DO 200 J = 1,6
   DO 200 I = 1,3
      U(J,I) = U(J,N+I)
   IF (J .LE. 3) AA(I,J) = AA(I,N+J)
CONTINUE
200
RETURN
END

Name  Type  Offset  P  Class
-A    REAL*8     8  *  
AA    REAL*8     0  /BLOCK /
F     REAL*8     0  /FUNCTION /
_FLAG LOGICAL*4  1298  
H     REAL      16  *  
HD12  REAL*8     960  /BLOCK /
I     INTEGER*4  1302  
-J    INTEGER*4  1314  
K     INTEGER*4  1306  
N     INTEGER*4  12  *  
-U    REAL*8     312  /BLOCK /
V     REAL*8     4  *  
W     REAL*8     20  *  
WX2   REAL*8     936  /BLOCK /
WXW   REAL*8     944  /BLOCK /
WXWX3 REAL*8     952  /BLOCK /
X     REAL*8     0  *  

595 $PAGE
SUBROUTINE INIT(X, V, W, H)

This procedure initializes all the necessary parameters before solving the system of ordinary differential equations. This procedure is invoked only once.

REAL * 8 X(3), V(3), AA(3,13), U(6,13), WX2, WXW, WXWX3
REAL * 8 CWT, SWT, T, W, HD12
COMMON /BLOCK/ AA, U, WX2, WXW, WXWX3, HD12
WXW = W*W
WXWX3 = 3*WXW
WX2 = 2*W
HD12 = DBLE(H)/12.0

DO 100 K = 1,3
    U(2*K-1,3) = X(K)
    U(2*K,3) = V(K)
    DO 100 J = 1,6
    AA(J,K) = 0.0
100 CONTINUE

CONTINUE
DO 300 I = 1,2
    T = H*(I-3)
    CWT = DCOS(W*T)
    SWT = DSIN(W*T)
    U(1,I) = X(1) + V(1)*(4*SWT-3*T)/W + 6*X(3)*(SWT-W*T) + 2*V(3)*(CWT-1.0)/W
    U(2,I) = V(1)*(4*CWT-3.0) + 6*W*X(3)*(CWT-1.0) - 2*V(3)*SWT

    U(3,I) = X(2)*CWT + V(2)*SWT/W
    U(4,I) = -X(2)*W*SWT + V(2)*CWT
    U(5,I) = 2*V(1)*(1.0-CWT)/W + X(3)*(4.0-3*CWT) + V(3)*SWT/W

    U(6,I) = 2*V(1)*SWT + 3*X(3)*W*SWT + V(3)*CWT
300 CONTINUE
RETURN
END
D Line# 1    7
CWT       REAL*8    1362
DOUBLE    INTRINSIC
DCOS      INTRINSIC
DSIN      INTRINSIC

I        REAL    12 *
D12      REAL*8   960 /BLOCK /
I        INTEGER*4 1350
J        INTEGER*4 1346
T        INTEGER*4 1342
SWT      REAL*8   1370
T        REAL*8   1354
J        REAL*8   312 /BLOCK /
/        REAL*8   4 *
W        REAL*8   8 *
\X2      REAL*8   936 /BLOCK /
\XW      REAL*8   944 /BLOCK /
\WXW     REAL*8   952 /BLOCK /
\X      REAL*8   0 *
FUNCTION F(J,I)

REAL*8 AA(3,13), U(6,13), WX2, WXW, WXWX3, HD12, F

COMMON /BLOCK/ AA, U, WX2, WXW, WXWX3, HD12

GO TO (10, 20, 30, 40, 50, 60), J
CONTINUE

F = U(2,I)
RETURN

CONTINUE

F = -WX2*U(6,I) + AA(1,I)
RETURN

CONTINUE

F = U(4,I)
RETURN

CONTINUE

F = -WXW*U(3,I) + AA(2,I)
RETURN

CONTINUE

F = U(6,I)
RETURN

CONTINUE

F = WX2*U(2,I) + WXWX3*U(5,I) + AA(3,I)
RETURN

END

Name  Type  Offset  P  Class

AA  REAL*8  0  /BLOCK /
HD12 REAL*8  960 /BLOCK /
I  INTEGER*4  4 *
-J INTEGER*4  0 *
U  REAL*8  312 /BLOCK /
WX2 REAL*8  936 /BLOCK /
WXW REAL*8  944 /BLOCK /
WXWX3 REAL*8  952 /BLOCK /

$PAGE
SUBROUTINE OUTPUT (A, W, X, V, E, Q, S, CMD, TIME)

This is the output section of the system. Any further modification of the output requirements of this model must be done in this procedure. In particular, if no output to the CRT or printer is needed, it is recommended that C's be inserted into column 1 of all the WRITE statements. The simulation clock is updated in this procedure.

REAL * 8 A(3), W(3), X(3), V(3), E(3), Q(4), S(14)
INTEGER CMD(7), EF, EEF, PRTFG
INTEGER * 4 TIME, T

COMMON /ME/ EF, EEF, PRTFG

TIME = TIME + 1
T = (TIME / 10) * 10 - TIME
IF ( (T .NE. 0) .OR. (PRTFG .EQ. 0)) RETURN
IF (PRTFG .EQ. 1) GO TO 100
OPEN (4, FILE = 'LPT1: ')
WRITE (4, 15) TIME / 10
WRITE (4, 20) A, W
WRITE (4, 30) X, V
WRITE (4, 40) E, W
WRITE (4, 50) S
WRITE (4, 90) CMD
CLOSE (4)
IF (PRTFG .NE. 2) CALL PLOT (CMD)
RETURN

FORMAT (' A, W = ', 3F10.6, 3X, 3F10.6)
FORMAT (' ', 7I10)
FORMAT (' TIME = ', I6, ' Seconds')
FORMAT (' X, V = ', 3F10.6, 3X, 3F10.6)
FORMAT (' E, W = ', 3F10.6, 3X, 3F10.6/
FORMAT (' S = ', 3F10.6, 3X, 3F10.6/
FORMAT ('', 3F10.3/
FORMAT ('', 4F10.6, 3X,F10.3/)
### Name | Type | Offset | P | Class
--- | --- | --- | --- | ---
_A | REAL*8 | 0 | * | 
_CMD | INTEGER*4 | 28 | * | 
_E | REAL*8 | 16 | * | 
_EEF | INTEGER*4 | 4 | /ME | 
_EF | INTEGER*4 | 0 | /ME | 
_PRTFG | INTEGER*4 | 8 | /ME | 
_Q | REAL*8 | 20 | * | 
_S | REAL*8 | 24 | * | 
_WARNING | INTEGER*4 | 1378 | | 
_TIME | INTEGER*4 | 32 | * | 
_V | REAL*8 | 12 | * | 
_W | REAL*8 | 4 | * | 
_X | REAL*8 | 8 | * | 

723 $PAGE
SUBROUTINE DMUL (A, B, C, N)

This procedure performs a matrix multiplication of an NxN matrix A to an N-element column matrix B to yield an N-element column matrix C.

REAL * 8 A(N,N), B(N), C(N), S

DO 100 I = 1, N
   S = 0.0
   DO 200 J = 1, N
      S = S + A(I,J) * B(J)
   CONTINUE
   C(I) = S
CONTINUE
RETURN
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>1714</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>1730</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>1722</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE UPDQ (Q, QQ)

This subroutine uses the previous quaternion and generates the present quaternions with respect to the local vertical frame LVF. Quaternion algebra is used to deduce the needed computation beforehand to simplify the algorithm.

REAL*8 Q(4), QQ(4), Q1, Q2, Q3, Q4

Q1 = Q(1)*QQ(4) + Q(4)*QQ(1) - Q(3)*QQ(2) + Q(2)*QQ(3)
Q2 = Q(2)*QQ(4) + Q(3)*QQ(1) + Q(4)*QQ(2) - Q(1)*QQ(3)
Q3 = Q(3)*QQ(4) - Q(2)*QQ(1) + Q(1)*QQ(2) + Q(4)*QQ(3)
Q4 = Q(4)*QQ(4) - Q(1)*QQ(1) - Q(2)*QQ(2) - Q(3)*QQ(3)

Q(1) = Q1
Q(2) = Q2
Q(3) = Q3
Q(4) = Q4
RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>REAL*8</td>
<td>1738</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>REAL*8</td>
<td>1746</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>REAL*8</td>
<td>1754</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>REAL*8</td>
<td>1762</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>1762</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

777 $PAGE
SUBROUTINE DCSINV (Q, C)

This subroutine takes the attitude quaternion Q and returns the transpose of the direction cosine matrix.

REAL * 8 Q(4), C(3,3)
REAL * 8 Q1, Q2, Q3, Q4
REAL * 8 Q11, Q22, Q33, Q44
REAL * 8 Q12, Q13, Q23
REAL * 8 Q14, Q24, Q34

Q1 = Q(1)
Q2 = Q(2)
Q3 = Q(3)
Q4 = Q(4)
Q11 = Q1 * Q1
Q22 = Q2 * Q2
Q33 = Q3 * Q3
Q44 = Q4 * Q4
Q12 = 2.0 * Q1 * Q2
Q13 = 2.0 * Q1 * Q3
Q23 = 2.0 * Q2 * Q3
Q14 = 2.0 * Q1 * Q4
Q24 = 2.0 * Q2 * Q4
Q34 = 2.0 * Q3 * Q4

C(1,1) = Q11 - Q22 - Q33 + Q44
C(2,2) = -Q11 + Q22 - Q33 + Q44
C(3,3) = -Q11 - Q22 + Q33 + Q44
C(1,2) = Q12 - Q34
C(2,1) = Q12 + Q34
C(1,3) = Q13 + Q24
C(3,1) = Q13 - Q24
C(2,3) = Q23 - Q14
C(3,2) = Q23 + Q14

RETURN
END
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>REAL*8</td>
<td>4 *</td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>0 *</td>
</tr>
<tr>
<td>Q1</td>
<td>REAL*8</td>
<td>1770</td>
</tr>
<tr>
<td>Q11</td>
<td>REAL*8</td>
<td>1802</td>
</tr>
<tr>
<td>Q12</td>
<td>REAL*8</td>
<td>1834</td>
</tr>
<tr>
<td>Q13</td>
<td>REAL*8</td>
<td>1842</td>
</tr>
<tr>
<td>Q14</td>
<td>REAL*8</td>
<td>1858</td>
</tr>
<tr>
<td>Q2</td>
<td>REAL*8</td>
<td>1778</td>
</tr>
<tr>
<td>Q22</td>
<td>REAL*8</td>
<td>1810</td>
</tr>
<tr>
<td>Q23</td>
<td>REAL*8</td>
<td>1850</td>
</tr>
<tr>
<td>Q24</td>
<td>REAL*8</td>
<td>1866</td>
</tr>
<tr>
<td>Q3</td>
<td>REAL*8</td>
<td>1786</td>
</tr>
<tr>
<td>Q33</td>
<td>REAL*8</td>
<td>1818</td>
</tr>
<tr>
<td>Q34</td>
<td>REAL*8</td>
<td>1874</td>
</tr>
<tr>
<td>Q4</td>
<td>REAL*8</td>
<td>1794</td>
</tr>
<tr>
<td>Q44</td>
<td>REAL*8</td>
<td>1826</td>
</tr>
</tbody>
</table>

826 $PAGE
<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRE</td>
<td>968</td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>JCK</td>
<td>48</td>
<td>COMMON</td>
</tr>
<tr>
<td>FINV</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>JL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>JTPRDR</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>REAL*8</td>
<td></td>
<td>FUNCTION</td>
</tr>
<tr>
<td>GE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>NCTL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TTPL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TTPD</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>AIN</td>
<td>48</td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>ATCH</td>
<td>12</td>
<td>COMMON</td>
</tr>
<tr>
<td>OPTION</td>
<td>80</td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>MDSL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>INPUT</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>COMMON</td>
</tr>
<tr>
<td>OT</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>ITGM</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>INCOS</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>LVE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TATE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TORE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>DQ</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>VECTOR</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
</tbody>
</table>

Pass One  No Errors Detected

826 Source Lines
Appendix 6

ADAM -- Source Listing
This program uses the Adam Brashforth method to solve the equation of motion (homogeneous case) numerically and compares the solution with the analytical results such that both outputs are printed.

REAL*8 XE(3), VE(3), X(3), V(3), A(3,10), W
REAL*8 XO(3), VO(3)
DATA A/30*0.0/
DATA N, H /10, 0.01/
WRITE (*, 30)
READ (*,32) W
get initial conditions
CALL GETINT (XO, VO, 3)
DO 100 K = 1, 3
   X(K) = XO(K)
   V(K) = VO(K)
CONTINUE
DO 10 I = 1, 36000
   T = 0.1*I
   *** calculate the analytical solution
   CALL EXACT(T, XE, VE, W, XO, VO)
Now get the numerical solution

CALL SOLVE(X,V,A,N,H,W)

** Output every 60 seconds

JJ = (I / 600) * 600

IF (JJ .EQ. I) THEN

WRITE(*,20) T,XE,VE

WRITE(*,20) T,X,V

WRITE(*,22)

END IF

CONTINUE

STOP

END

Name | Type | Offset  |
---- |------ |-------- |
-A   | REAL*8 | 146    |
H    | REAL  | 390    |
I    | INTEGER*4 | 406    |
-JJ  | INTEGER*4 | 414    |
K    | INTEGER*4 | 402    |
N    | INTEGER*4 | 386    |
-T   | REAL  | 410    |
-V   | REAL*8 | 98     |
-V0  | REAL*8 | 122    |
-VE  | REAL*8 | 26     |
-W   | REAL*8 | 394    |
-X   | REAL*8 | 50     |
-X0  | REAL*8 | 74     |
-XE  | REAL*8 | 2      |
SUBROUTINE EXACT(T,XE,VE,W,X,V)

** This subroutine calculates the exact solution
of the homogeneous ODEs

REAL*8 XE(3),VE(3),CWT,SWT,W, WT, X(3), V(3)

WT = W * T
SWT = DSIN(WT)
CWT = DCOS(WT)

XE(1) = X(1) + (4 * SWT - 3 * WT) * V(1) / W + 6 * (SWT - WT) * X(3)
XE(2) = CWT * X(2) + SWT * V(2) / W
XE(3) = 2 * (1 - CWT) * V(1) / W + (4 - 3 * CWT) * X(3)

VE(1) = (4 * CWT - 3) * V(1) + 6 * W * (CWT - 1) * X(3)
VE(2) = CWT * V(2) - W * SWT * X(2)
VE(3) = 2 * SWT * V(1) + 3 * W * SWT * X(3) + CWT * V(3)

RETURN
END
SUBROUTINE SOLVE(X,V,A,N,H,W)

** This subroutine produces the numerical solution to the system of equations of motion

LOGICAL FLAG
REAL*8 X(3), V(3), A(3,10), AA(3,13), U(6,13)
REAL*8 WX2, WXW, WXW3, HD12, F, W
COMMON /BLOCK/ AA, U, WX2, WXW, WXW3, HD12
DATA FLAG / .TRUE. /

pack user supplied nonhomogeneous part of DE into the higher part of AA

DO 10 I = 1,10
   DO 10 K = 1,3
      AA(K,I+3) = A(K,I)
   CONTINUE

if this is the first call to solve (FLAG = T), then initialize

IF (FLAG) THEN
   CALL ININIT(X,V,W,H)
   FLAG = .FALSE.
END IF

use the Adam-Brashford 3-step method to advance the solution h time units. Place the solution back into X and V.

DO 100 I = 4,N+3
   DO 100 J = 1,6
      U(J,I) = U(J,I-1) + HD12*(23*F(J,I-1)-16*F(J,I-2)+5*F(J,I-3))
   CONTINUE
X(1) = U(1,N+3)
RESET U AND AA FOR THE NEXT CALL TO SOLVE

DO 200 J = 1, 6
   DO 200 I = 1, 3
      U(J, I) = U(J, N+I)
   CONTINUE

DO 300 I = 1, 3
   DO 300 K = 1, 3
      AA(K, I) = AA(K, N+I)
   CONTINUE

RETURN
END
SUBROUTINE INNIT(X,V,W,H)

This is the initialization routine which is called only once

DO 100 I = 1,3
   DO 100 J = 1,6
      AA(J,I) = 0.0
   CONTINUE

DO 200 K = 1,3
   U(2*K-1,3) = X(K)
   U(2*K  ,3) = V(K)
CONTINUE

DO 300 I = 1,2
   T = H*(I-3)
   CWT = DCOS(W*T)
   SWT = DSIN(W*T)
   U(1,I) = X(1) + V(1)*(4*CWT-3*W*T)/W + 6*X(3)*(SWT-W*T) + 2*V(3)*(CWT-1.0)/W
   U(2,I) = V(1)*(4*CWT-3.0) + 6*W*X(3)*(CWT-1.0) - 2*V(3)*SWT
   U(3,I) = X(2)*CWT + V(2)*SWT/W
   U(4,I) = -X(2)*W*SWT + V(2)*CWT
   U(5,I) = 2*V(1)*(1.0-CWT)/W + X(3)*(4.0-3*CWT) + V(3)*SWT/W
   U(6,I) = 2*V(1)*SWT + 3*X(3)*W*SWT + V(3)*CWT
CONTINUE
RETURN
END
<table>
<thead>
<tr>
<th>e</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>REAL*8</td>
<td>0</td>
<td>/BLOCK /</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>560</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>INTRINSIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>U0S</td>
<td>INTRINSIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JIN</td>
<td>INTRINSIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL</td>
<td>12</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>960</td>
<td>/BLOCK /</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>540</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>544</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>548</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VT</td>
<td>REAL*8</td>
<td>568</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>552</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>312</td>
<td>/BLOCK /</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Z</td>
<td>REAL*8</td>
<td>936</td>
<td>/BLOCK /</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>REAL*8</td>
<td>944</td>
<td>/BLOCK /</td>
</tr>
<tr>
<td></td>
<td>XWX3</td>
<td>REAL*8</td>
<td>952</td>
<td>/BLOCK /</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>
FUNCTION F(J,I)

C----------------------------------------------------------------------
C
C

User supplied function

C----------------------------------------------------------------------

REAL*8 AA(3,13),U(6,13),WX2,WXW,WXWX3,HD12,F
COMMON /BLOCK/ AA,U,WX2,WXW,WXWX3,HD12

GO TO (10,20,30,40,50,60), J
CONTINUE

F = U(2,I)
RETURN

CONTINUE

F = -WX2*U(6,I) + AA(1,I)
RETURN

CONTINUE

F = U(4,I)
RETURN

CONTINUE

F = -WXW*U(3,I) + AA(2,I)
RETURN

CONTINUE

F = U(6,I)
RETURN

CONTINUE

F = WX2*U(2,I) + WXWX3*U(5,I) + AA(3,I)
RETURN

END

Name   Type    Offset P Class
AA     REAL*8   0    /BLOCK /
HD12   REAL*8   960   /BLOCK /
I      INTEGER*4 4    *
J      INTEGER*4 0    *
U      REAL*8   312   /BLOCK /
WX2    REAL*8   936   /BLOCK /
WXW    REAL*8   944   /BLOCK /
WXWX3  REAL*8   952   /BLOCK /
258    $PAGE
SUBROUTINE GETINT (X, V, N)

REAL * 8 X(N), V(N)

OPEN (1, FILE = 'INITCON.DAT')

DO 100 K = 1, N
  READ (1, 10) X(K)
  CONTINUE

DO 200 K = 1, N
  READ (1,10) V(K)
  CONTINUE

RETURN

FORMAT (F15.6)
END
Pass One
No Errors Detected
285 Source Lines
End of Document