NASA CR-171123

UAH RESEARCH REPORT NO. 407

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 $[\dot{r},\dot{p},\dot{y}]^T$ 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 over-all 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 space craft 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 space craft 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 **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 **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 :

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

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

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

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

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

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 $\mathbf{q}$ may be written as :

$$q = iq_1 + jq_2 + kq_3 + q_4 = [q_1,q_2,q_3,q_4]^T$$

and satisfies the relation

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

An object whose attitude is described by the three Euler angles relative to some reference frame can be treated as a single rotation by α about an Euler axis $\mathbf{E} = [E_1,E_2,E_3]^T$. Theory has shown that this is the shortest angular path (Reference 7) in the sense that α is less than the algebraic sum of ϕ , θ and ψ . The angle α and the Euler axis can be expressed in terms of the quaternion $\mathbf{q}$ as :

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

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 $\mathbf{E}$ (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 <u>relative to some non-rotating frame</u>. Suppose further that an instant later, the vehicle's attitude has changed, having rotated by ϕ , θ and ψ . 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, <u>relative to the non-rotating frame</u> (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:

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

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 :

$$\begin{aligned} \ddot{X} &= A_x - 2\omega\dot{Z} \\ \ddot{Y} &= A_y - \omega^2 Y \\ \ddot{Z} &= A_z + 2\omega\dot{X} + 3\omega^2 Z \end{aligned}$$

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. ω 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 **A** from a using the transformation :

$$\mathbf{A} \quad = \quad \mathbf{C}^{-1}\mathbf{a}$$

where $\mathbf{C}^{-1}$ is the inverse of the direction cosine matrix which can be derived from the quaternion $\mathbf{q} = [q_1, a_2, a_3, a_4]^T$ as:

$$\mathbf{C}^{-1} = \begin{bmatrix} q_4 + q_1 - q_2 - q_3 & 2(q_1 q_2 - q_3 q_4) & 2(q_1 q_3 + q_2 q_4) \\ 2(q_1 q_2 + q_3 q_4) & q_4 - q_1 + q_2 - q_3 & 2(q_2 q_3 - q_1 q_4) \\ 2(q_1 q_3 - q_2 q_4) & 2(q_2 q_3 + q_1 q_4) & q_4 - q_1 - q_2 + q_3 \end{bmatrix}$$

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 **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 :

S(1) - S(3) -- relative position vector in LVF

S(4) - S(6) -- relative velocity vector in LVF

S(7) - S(9) -- angular momentum vector in LVF

S(10) - S(13) -- attitude quaternion

S(14) -- mass in kilograms

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 LB in body frame using the relation (Reference 10):

$$\mathbf{L}_B = \mathbf{Iw}$$
$$\mathbf{L} = \mathbf{C}^{-1}\mathbf{L}_B$$

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

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

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 **a** to **A** in LVF, and

c) placing **A** in an acceleration matrix 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 surboutine 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 competable 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

$$\mathbf{a} = [0,0,0]^T \qquad \text{and} \qquad \mathbf{w} = [0,0,0]^T$$

Thus, the equations of motion reduce to :

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

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 :

$$X(t) = X_1 - \frac{(3\Omega t - 4\sin\Omega t)}{\Omega} V_1 - 6(\Omega t - \sin\Omega t)X_3 - \frac{(1-\cos\Omega t)}{\Omega} V_3$$

$$\dot{X}(t) = -(3 - 4\cos\Omega t)V_1 - 6\Omega(1-\cos\Omega t)X_3 - 2(\sin\Omega t)V_3$$

$$Y(t) = (\cos\Omega t)X_2 + \left(\frac{\sin\Omega t}{\Omega}\right)V_2$$

$$\dot{Y}(t) = -\Omega(\sin\Omega t)X_2 + (\cos\Omega t)V_2$$

$$Z(t) = \frac{2(1-\cos\Omega t)}{\Omega} V_1 + (4-3\cos\Omega t)X_3 + \frac{\sin\Omega t}{\Omega} V_3$$

$$\dot{Z}(t) = 2(\sin\Omega t)\, V_1 + 3\Omega(\sin\Omega t)\, X_3 + (\cos\Omega t)\, V_3$$

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

$$\Omega = G\, M_e \,/\, (R_o + H)^3$$

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, Ω approaches 0 and the equations of motion approach

$$\ddot{X} \rightarrow 0$$
$$\ddot{Y} \rightarrow 0$$
$$\ddot{Z} \rightarrow 0$$

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 (Ω = 0.00118 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 x 10^{-8} in 60 minutes, or about 0.03 milli-meters. This figure is well below the expected accuracy of the flat floor simulation system. This suprisingly small error comes from the fact that the angular velocity Ω 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 consititutes 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 revelant 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 interval 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 orginally at rest, the initial conditions being :

$$X_1 = X_2 = X_3 = 0$$

$$V_1 = V_2 = V_3 = 0$$

$$r = p = y = 0$$

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,

$$r = y = 0, \text{ and } \quad p = w_2 = 0$$

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 revelent 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 outputted 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 revelant 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 timeing 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, OMV 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 1

OMV Mass Characteristics

Dynamic Variable	Value	unit
Mass M	3282.75	kg
I_{11}	7048.37	$kg\ m^2$
I_{22}	3713.95	$kg\ m^2$
I_{33}	3713.95	$kg\ m^2$

Table 2

Hand Controller Bit Assignments

bit	Meaning
1	Acceleration along +1 direction
2	Acceleration along -1 direction
3	Acceleration along +2 direction
4	Acceleration along -2 direction
5	Acceleration along +3 direction
6	Acceleration along -3 direction
7	+ roll; CCW rotation about 1-axis
8	- roll; CW rotation about 1-axis
9	+ pitch; CCW rotation about 2-axis
10	- pitch; CW rotation about 2-axis
11	+ yaw; CCW rotation about 3-axis
12	- yaw; CW rotation about 3-axis

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	12.828963	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	-611.988644	-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

Time in Seconds	X in meters	$\dot{X}$ in meters	Z in meters	$\dot{Z}$ in meters
.0	0.000000	0.000000	0.000000	0.000000
.5	0.002940	0.012125	0.000001	0.000007
1.0	0.012128	0.024625	0.000009	0.000029
1.5	0.027565	0.037125	0.000032	0.000065
2.0	0.049253	0.049625	0.000077	0.000117
2.5	0.077190	0.062125	0.000152	0.000183
3.0	0.111377	0.074624	0.000263	0.000264
3.5	0.151814	0.087124	0.000418	0.000360
4.0	0.198501	0.099624	0.000625	0.000471

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

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

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

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

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

Table 7

OMV Time Test

No of Steps	Average execution time per major cycle	
	Single Precision	Double Precision
4	0.077	0.084
5	0.090	0.099
6	0.103	0.113
7	0.117	0.128
8	0.130	0.143
9	0.144	0.158
10	0.157	0.173

Table 8

Optimized OMV Execution Times Per Major Cycle

As A Function Of Number Of Steps N

N	Execution time (Sec)
4	0.068
5	0.079
6	0.090
7	0.100
8	0.111
9	0.122
10	0.132

Table 9

Comparison Test Between N = 4 and N = 10 Steps

Time in Minutes	Solution: Analytic	Solution: Numeric N = 10	Solution: Numeric N = 4
0	0.000000	0.000000	0.000000
5	13.746736	13.746736	13.746736
10	20.161917	20.161917	20.161917
15	12.828962	12.828963	12.828961
20	-12.952953	-12.952950	-12.952956
25	-59.582233	-59.582227	-59.582237
30	-126.855544	-126.855533	-126.855551
35	-211.993191	-211.993176	-211.993200
40	-309.986022	-309.986003	-309.986034
45	-414.220565	-414.220544	-414.220579
50	-517.304388	-517.304365	-517.304403
55	-611.988666	-611.988644	-611.988681
60	-692.072834	-692.072815	-692.072847

List of Figures

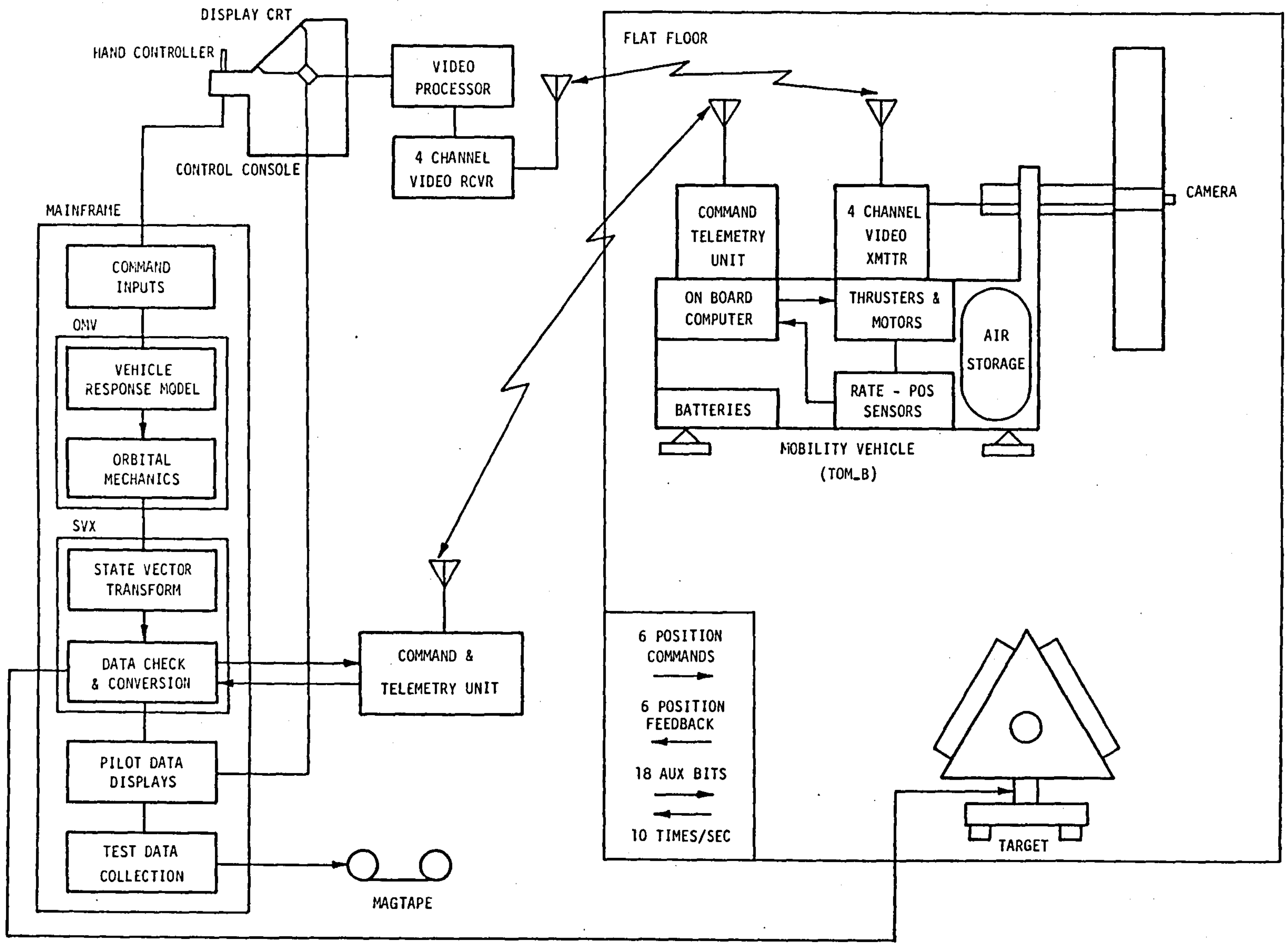


Figure 1. MSFC Flatfloor Simulation System

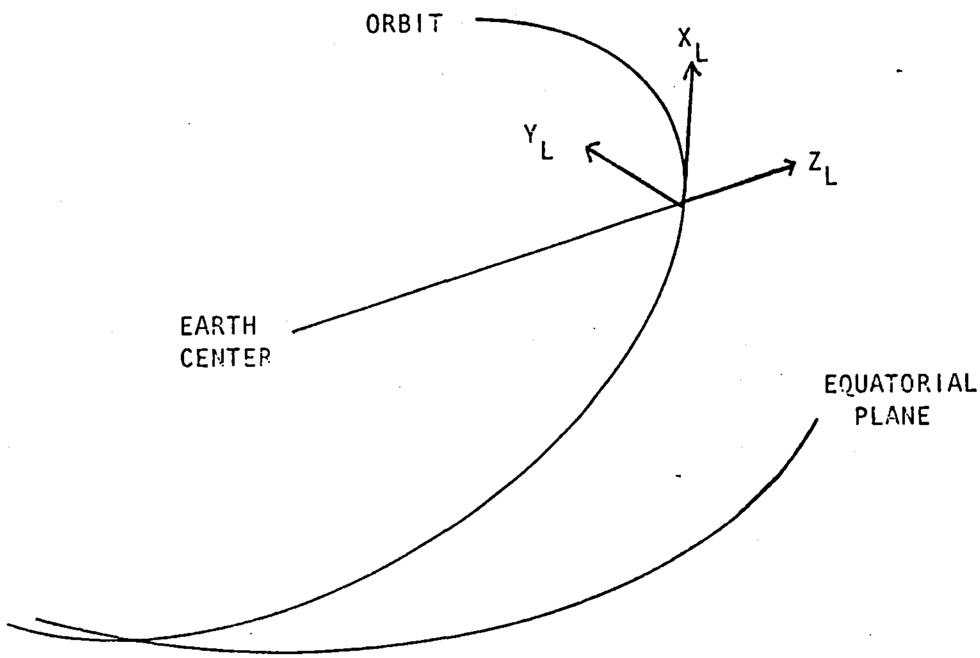
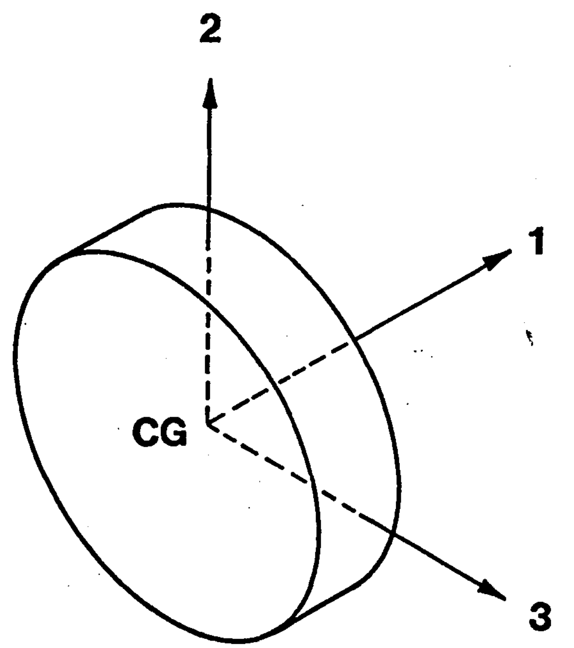


Fig. 2 Local Vertical Frame (L)

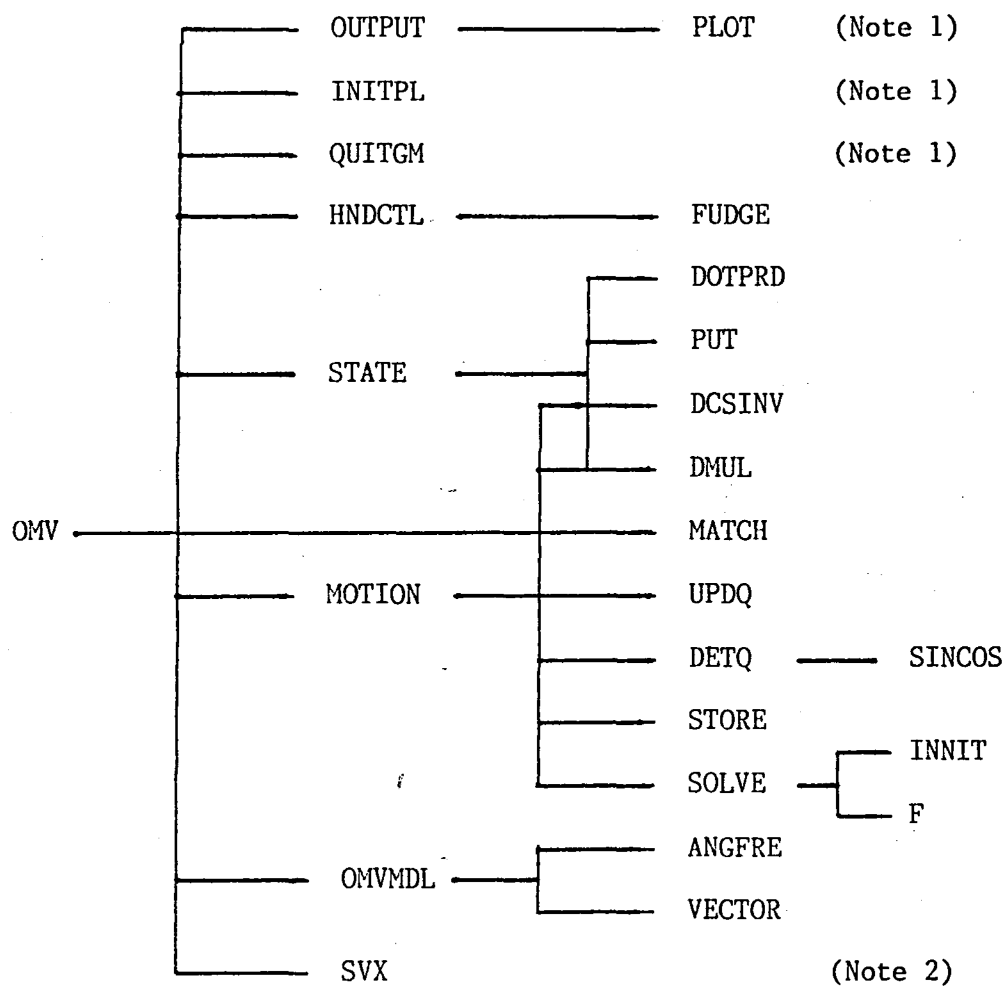


OMV Body Frame

Fig 3

Figure 4

OMV Heirarchial Chart



Note 1 : Hardware incompatible graphics package.

Note 2 : Vector Transformation Module. See Reference 1.

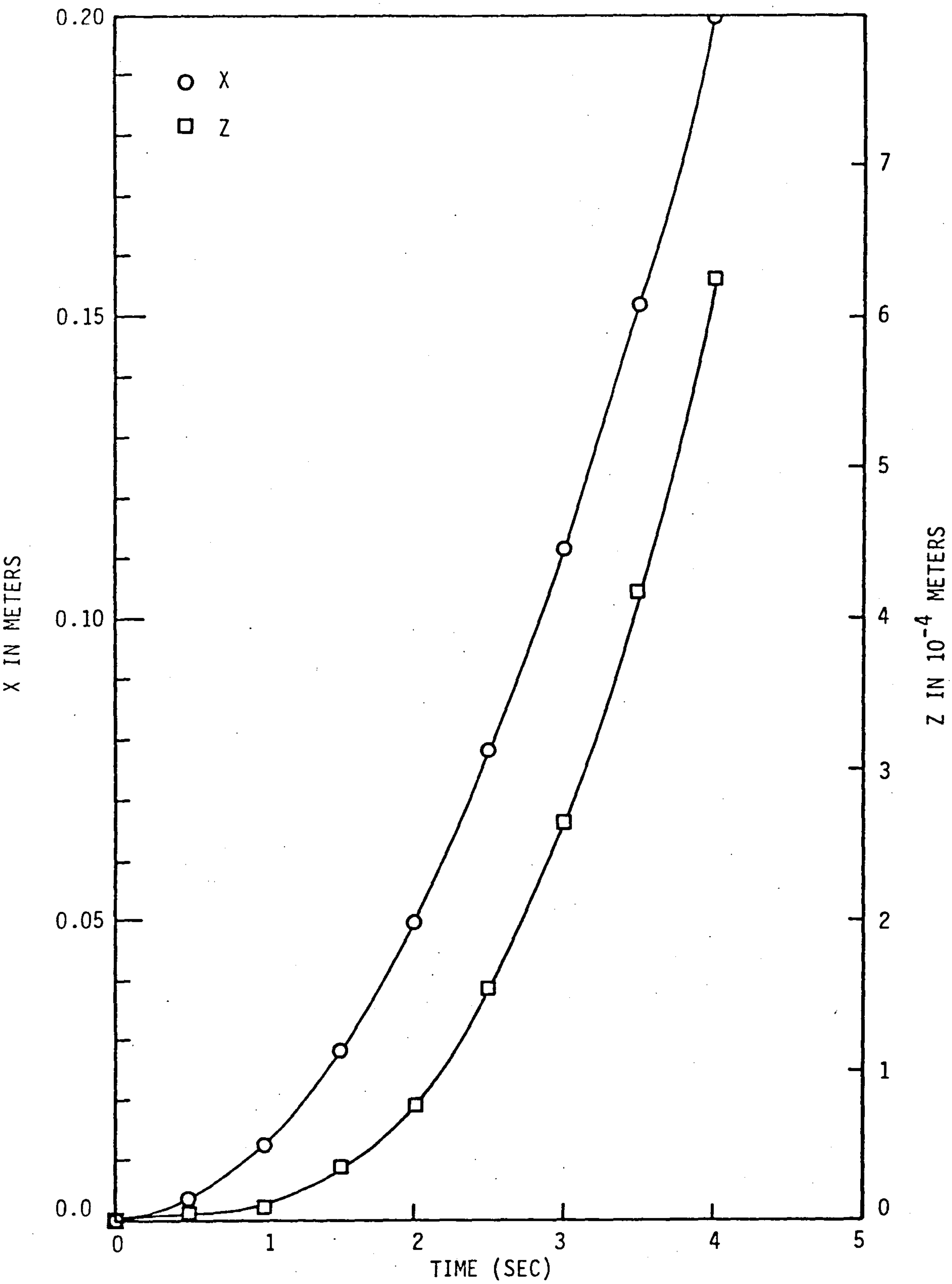


Figure 5. Translation Along X-Axis

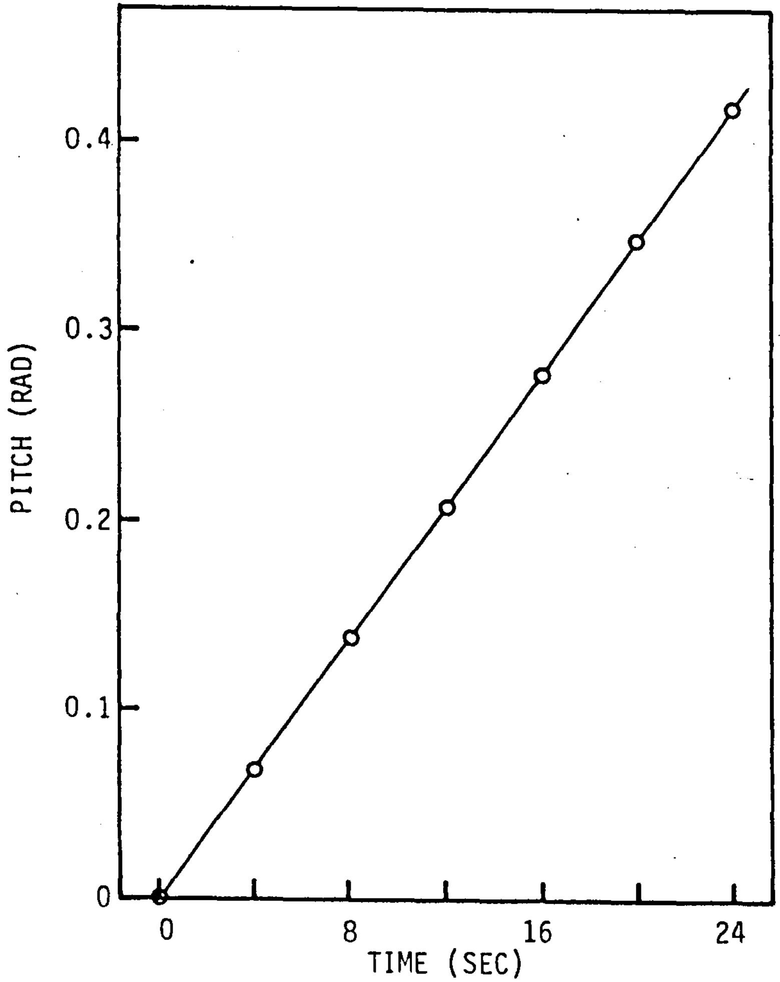
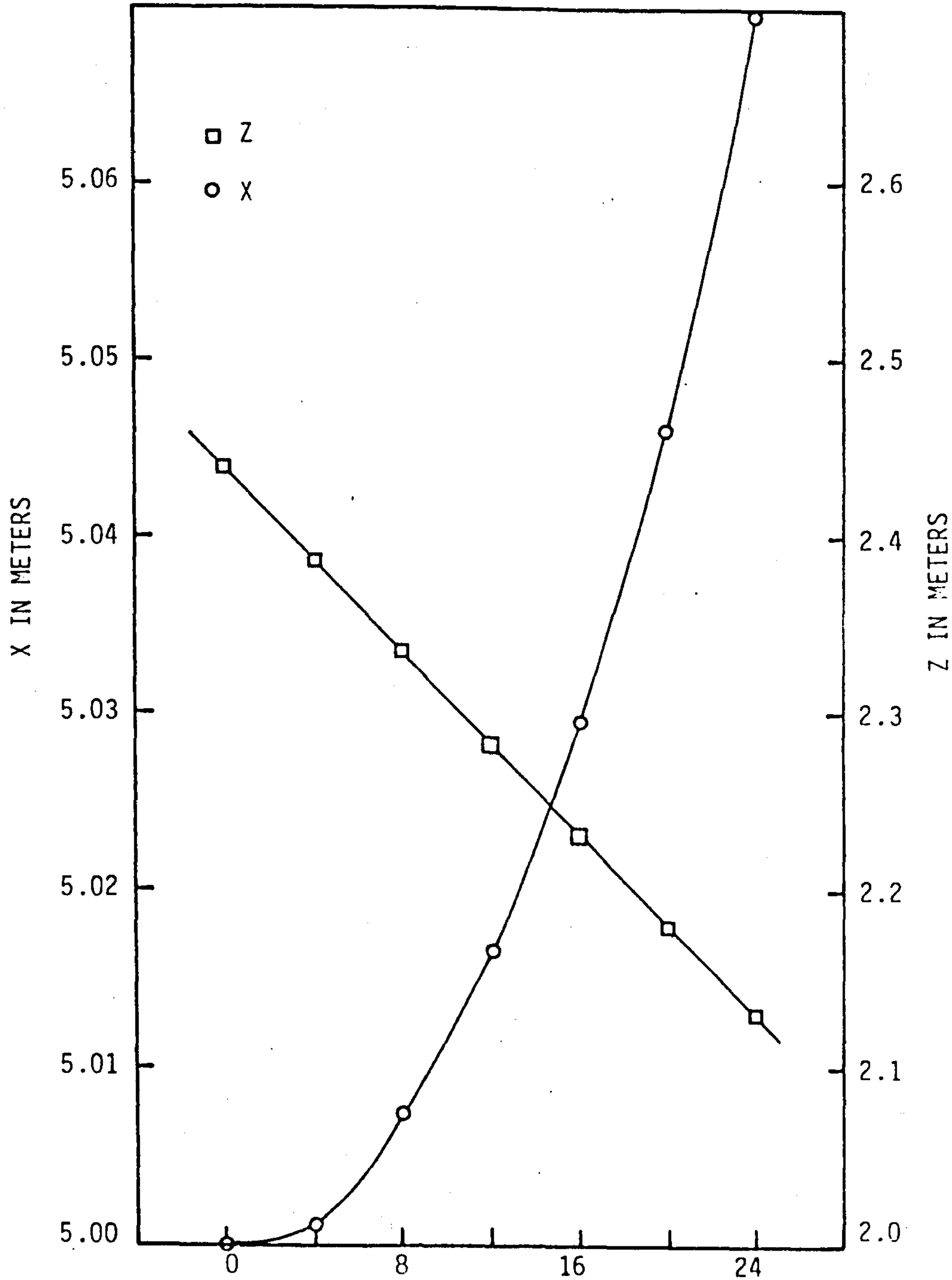


Figure 6. Pure Pitch Motion
at 0.017453 rad/sec.



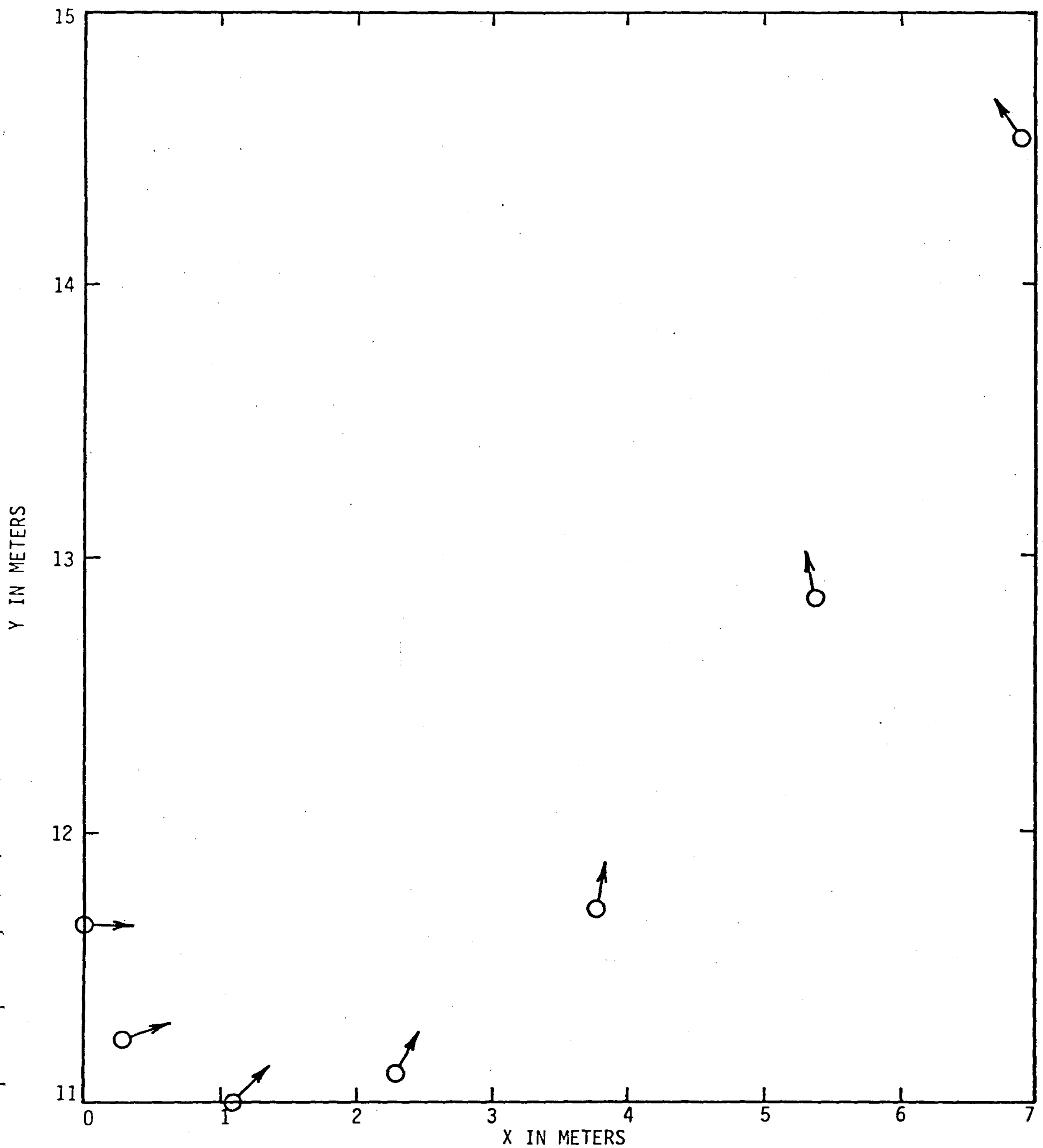


Figure 7. Trajectory of Mobile Base When OMV is Executing a Translation & Yaw

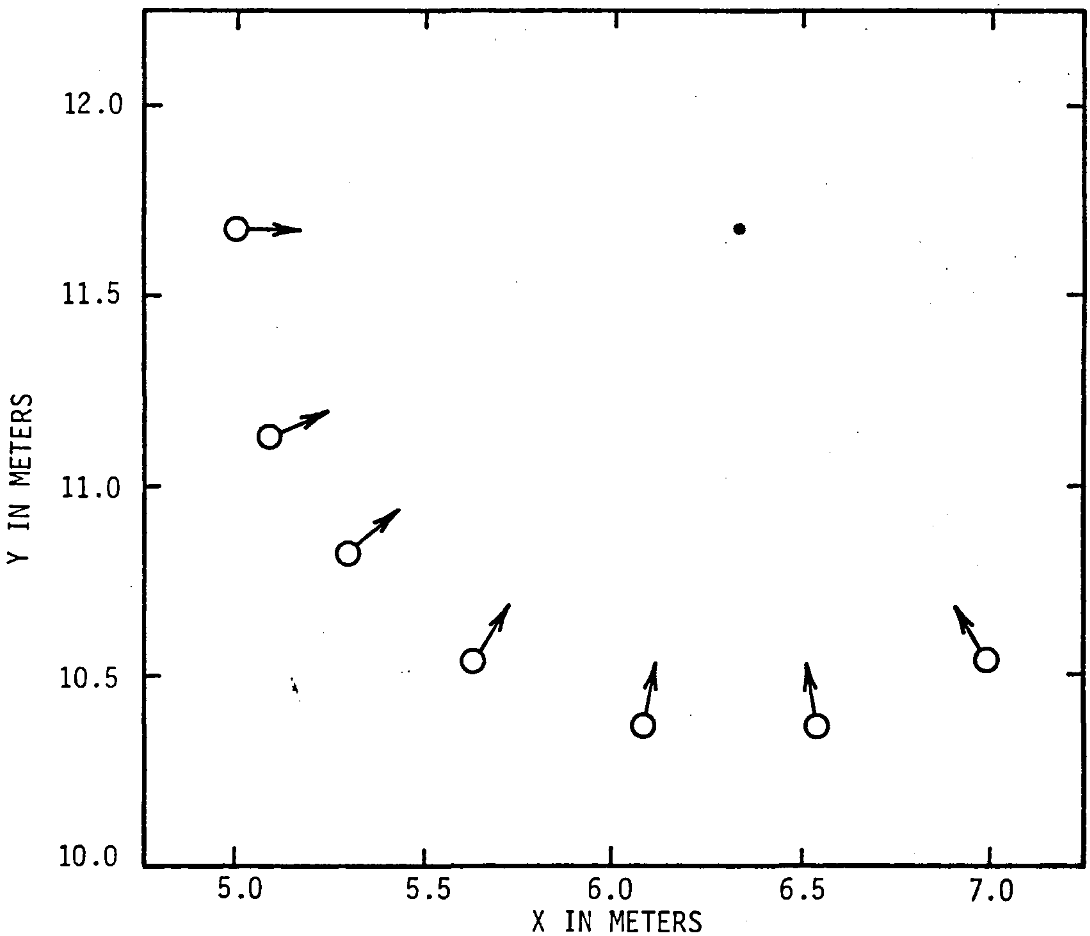


Figure 8. OMV Pure Yaw Motion

List of References

List of References

1. "State Vector Transformation Module", W. Teoh, NASA Technical Report, May 9, 1984. Contract # NAS8-35670.

2. "Teleoperator Maneuvering System Preliminary Definition Study", prepared by Program Development, MSFC, June 1983.

3. "Classical Mechanics", H. Goldstein, Addison-Wesley Inc, 1965.

4. "Mechanics" K. Symon, Addison-Westley Inc, 1964.

5. "Equations of Motion for Six Degrees Of Freedom TRS Simulation", J. Galaboff, NASA MSFC System Dynamics Laboratory Memo ED15-78-34, August, 1978.

6. "On The Use Of Quaternions In Simulation Of Rigid Body Motion", A. C. Robinson, Wright Air Development Center, 1958.

7. "A New Method For Performing Digital Control System Attitude Computation Using Quaternions", B. P. Ickes, AIAA Journal 8(1970)13.

8. "Design Of A Terminal Pointer Hand Controller For Teleoperator Applications:, E. L. Saenger and W. S. Woltosz, URS/Matrix Company Final Report, NASA Contract # NAS8-28760, 1973.

9. "Quaternions For Control Of Space Vehicles", A. C. Hendley, Sperry Rand Corp, NASA Contract # NAS8-20055, 1973.

10. "Software Specifications For Docking Simulations Of The Orbital Maneuvering Vehicle (OMV)", J. D. Micheal, NASAN. MSFC System Dynamics Lab, Memo ED15-83-64, 1984.

11. "Discrete Variable Methods In Ordinary Differential Equations", P. Henrici, John Wiley & Sons, 1962.

12. "Numerical Methods With FORTRAN IV", D. McCracken, John Wiley & Son, 1972.

13. "Mathematical Modelling And Digital Simulation For Engineerers And Scientists", J. Smith, John Wiley & Son, 1977.

14. "Applied Numerical Methods", B. Carnahan, H. Luther and J. Wilkes, John Wiley & Son, 1969.

15. "GRAPHICS MASTER Reference Mannual", Tecmar Corp, 1983.

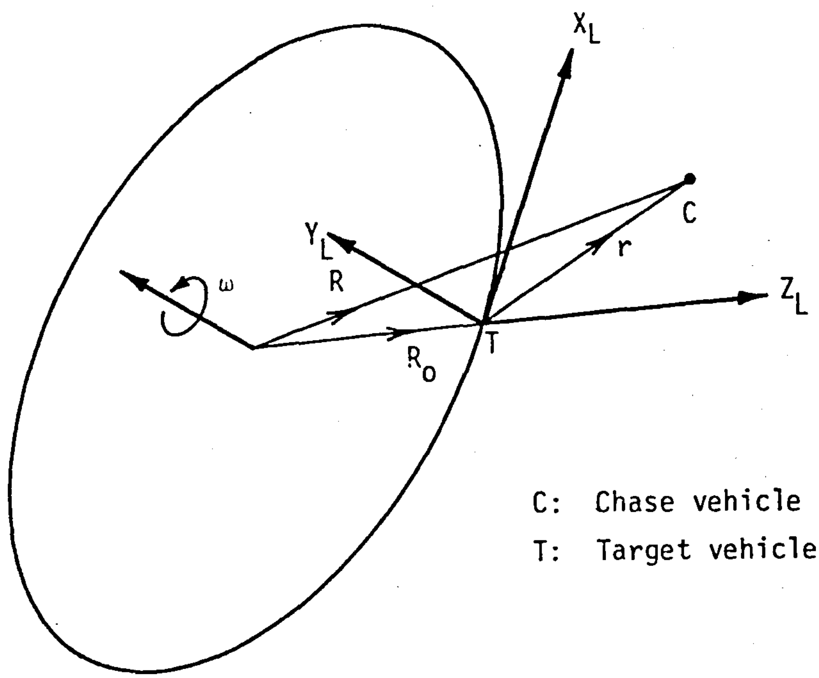
Appendix 1

OMV Translational Equations of Motion

Appendix 1.

OMV Translational Equations Of Motion

Consider a target vehicle orbiting the earth with an angular velocity **w** 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 chose 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,

$$M_c \ddot{\mathbf{R}} = \mathbf{F}_g + \mathbf{F}_c \tag{1}$$

This equation is written in the inertial frame. Here, M_c is the mass of the chase vehicle, $\mathbf{F}_g$ is the gravitational force exerted on the vehicle by the earth, and $\mathbf{F}_c$ 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 $\mathbf{r}$ 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

$$\mathbf{R} = \mathbf{R}_o + \mathbf{r}_E \tag{2}$$

it follows that

$$\ddot{\mathbf{R}} = \ddot{\mathbf{R}}_o + \ddot{\mathbf{r}}_E \tag{3}$$

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

$$\{ d/dt \}_E = \{ d/dt + \mathbf{w} \times \}_L$$

Applying this operator to $\mathbf{r}$ twice, we have

$$\dot{\mathbf{r}}_E = \dot{\mathbf{r}}_L + \mathbf{w} \times \mathbf{r}_L$$

and

$$\begin{aligned} \ddot{\mathbf{r}}_E &= d/dt\,(\dot{\mathbf{r}}_L + \mathbf{w} \times \mathbf{r}_L) + \mathbf{w} \times (\dot{\mathbf{r}}_L + \mathbf{w} \times \mathbf{r}_L) \\ &= \ddot{\mathbf{r}}_L + \mathbf{w} \times \dot{\mathbf{r}}_L + \mathbf{w} \times \dot{\mathbf{r}}_L + \mathbf{w} \times (\mathbf{w} \times \mathbf{r}_L) \\ &= \ddot{\mathbf{r}}_L + 2\mathbf{w} \times \dot{\mathbf{r}}_L + \mathbf{w} \times (\mathbf{w} \times \mathbf{r}_L) \end{aligned} \tag{4}$$

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

$$\ddot{\mathbf{R}} = \ddot{\mathbf{R}}_o + \ddot{\mathbf{r}}_E$$
$$= \ddot{\mathbf{R}}_o + \ddot{\mathbf{r}}_L + 2\mathbf{w} \times \dot{\mathbf{r}}_L + \mathbf{w} \times (\mathbf{w} \times \mathbf{r}_L)$$

Furthermore, for a circular orbit,

$$\ddot{\mathbf{R}}_o + w^2\mathbf{R}_o = 0$$

therefore,

$$\ddot{\mathbf{R}} = -w^2\mathbf{R}_o + \ddot{\mathbf{r}}_L + 2\mathbf{w} \times \dot{\mathbf{r}}_L + \mathbf{w} \times (\mathbf{w} \times \mathbf{r}_L) \qquad (5)$$

It is clear at this point that the equations of motion (1) can be rewritten in terms of $\mathbf{r}_L$ and $\mathbf{R}_o$ and their time derivatives. Thus the subscript will be dropped from here on. Recall that

$$\mathbf{R} = \mathbf{R}_o + \mathbf{r}$$
$$R^2 = (\mathbf{R}_o + \mathbf{r}) \cdot (\mathbf{R}_o + \mathbf{r})$$
$$= R_o^2 + r^2 + 2\mathbf{R}_o \cdot \mathbf{r}$$
$$= R_o^2 + 2\mathbf{R}_o \cdot \mathbf{r}$$
$$= R_o^2 \{1 + 2(\mathbf{R}_o \cdot \mathbf{r}) / R_o^2 \}$$

so that

$$R^{-3} = R_o^{-3} \{1 + 2(\mathbf{R}_o \cdot \mathbf{r}) / R_o^2 \}^{-3/2}$$
$$\cong R_o^{-3} \{1 - 3(\mathbf{R}_o \cdot \mathbf{r}) / R_o^2 \}$$

Thus,

$$\mathbf{F}_g = -(GM_eM_c/R^3)\,\mathbf{R}$$
$$= -(GM_eM_c/R_o^3)\,(\mathbf{R}_o + \mathbf{r})\,(1 - 3(\mathbf{R}_o \cdot \mathbf{r}) / R_o^2)$$
$$= -w^2M_c(\mathbf{R}_o + \mathbf{r})\,(1 - 3(\mathbf{R}_o \cdot \mathbf{r}) / R_o^2)$$
$$\cong -w^2M_c\,\{\mathbf{R}_o + \mathbf{r} - 3(\mathbf{R}_o \cdot \mathbf{r}/R_o^2)\mathbf{R}_o \qquad (6)$$

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

$$M_c\{-w^2\mathbf{R}_o + \ddot{\mathbf{r}} + 2\mathbf{w} \times \dot{\mathbf{r}} + \mathbf{w} \times (\mathbf{w} \times \mathbf{r})\} = \mathbf{F} - M_c w^2\{\mathbf{R}_o + \mathbf{r} - 3(\mathbf{R}_o.\mathbf{r})/R_o^2\}$$

If we define $\mathbf{A} = \mathbf{F}_c / M_c$, then we have :

$$-w^2\mathbf{R}_o + \ddot{\mathbf{r}} + 2\mathbf{w} \times \dot{\mathbf{r}} + \mathbf{w} \times (\mathbf{w} \times \mathbf{r}) = \mathbf{A} - w^2\mathbf{R}_o - w^2\mathbf{r} + 3w^2(\mathbf{R}_o.\mathbf{r}/R_o^2)\mathbf{R}_o$$

which, after re-arranging, gives :

$$\ddot{\mathbf{r}} = \mathbf{A} - 2\mathbf{w} \times \dot{\mathbf{r}} - w^2\mathbf{r} - \mathbf{w} \times (\mathbf{w} \times \mathbf{r}) + 3w^2(\mathbf{R}_o.\mathbf{r}/R_o^2)\mathbf{R}_o \qquad (7)$$

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

$$\mathbf{r} = [X, Y, Z]^T$$
$$\mathbf{R}_o = [0, 0, R_o]^T$$
$$\mathbf{w} = [0, w, 0]^T \qquad \text{and}$$
$$\mathbf{A} = [A_x, A_y, A_z]^T$$

and it can easily be shown that :

$$2\mathbf{w} \times \dot{\mathbf{r}} = [\,2w\dot{Z},\ 0,\ -2w\dot{X}]^T$$
$$\mathbf{w} \times (\mathbf{w} \times \mathbf{r}) = [-w^2X,\ 0,\ -w^2Z]^T$$
$$3\mathbf{w}(\mathbf{R}_o.\mathbf{r}/R_o^2)\mathbf{R}_o = [\,0,\ 0,\ 3w^2Z]^T \qquad \text{and}$$
$$w^2\mathbf{r} = [\,w^2X,\ w^2Y,\ w^2Z]^T$$

and substituting into equation (7) yields

$$[X, Y, Z]^T = [-2w\dot{Z},\ 0,\ 2w\dot{X}]^T + [\,w^2X,\ 0,\ w^2Z]^T$$
$$+ [-w^2X,\ -w^2Y,\ -w^2Z]^T + [\,0,\ 0,\ 3w^2Z]^T$$
$$+ [\,A_x,\ A_y,\ A_z]^T$$

or

$$
\begin{aligned}
\ddot{X} &= A_x - 2w\dot{Z} \\
\ddot{Y} &= A_y - w^2 Y \\
\ddot{Z} &= A_z + 2w\dot{X} + 3w^2 Z
\end{aligned} \tag{8}
$$

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 percieved 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 :

Component	Symbol	Meaning
1	X	position of the target vehicle relative
2	Y	to the OMV in local vertical frame LVF.
3	Z	
4	V_x	relative velocity of the chase vehicle
5	V_y	in LVF
6	V_z	
7	L_x	angular momentum vector in LVF
8	L_y	
9	L_z	
10	q_1	attitude quaternions in body frame
11	q_2	
12	q_3	

13	q_4	
14	m	mass of OMV

It is often more convenient to consider the state vector to be made up of the following four vectors : $\mathbf{X} = [\ X, Y, Z\ ]^T$, $\mathbf{V} = [\ V_x, V_y, V_z\ ]^T$, $\mathbf{L} = [\ L_x, L_y, L_z\ ]$ and the unit quaternion $\mathbf{q} = [\ q_1, q_2, q_3, q_4\ ]^T$.

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 :

Component	Position control symbol	menaing	rate control symbol	meaning	coord. system
1	y	yaw of TOM_B	y	yaw rate	body frame
2	X	position of	V_x	velocity of	LVF
3	Y	TOM_B	V_y	TOM_B	
4	Z	pos of pivot	V_z	vel of pivot	
5	p	pitch angle	p	pitch rate	body frame
6	r	roll angle	r	roll rate	
7	1	pos. control	0	rate control	

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 orientation 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 :

$$\begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} \qquad [0]$$

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 :

$$\begin{bmatrix} X_M \\ Y_M \\ Z_M \end{bmatrix} = \begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} + \begin{bmatrix} c + 1 - X_o \\ a - Y_o \\ h - Z_o \end{bmatrix} \qquad [I]$$

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 :

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

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. In stead, the quantity of interest is Z, which is the height of the pivot point from the floor as shown in Figure 6, and

$$Z = Z_M - 1\sin(p) \qquad [III]$$

It follows that the velocity of TOM_B and the pivot point is given by

$$\begin{bmatrix} X_F \\ Y_F \\ Z \end{bmatrix} = \begin{bmatrix} X_M + \{c + l\cos(p)\}\sin(p)y + l\sin(p)\cos(y)p \\ Y_M - \{c + l\cos(p)\}\cos(p)y + l\sin(p)\sin(y)p \\ Z_M - l\cos(p)p \end{bmatrix} \quad [IV]$$

The above transformations take care of the position and velocity quantities.

The quaternions q_1, q_2, $q3$, 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) :

$$[\ r,\ p,\ y\ ]^T = \alpha[O_x, O_y, O_z]^T \quad [V]$$

where

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

$$[\ O_x,\ O_y,\ O_z\ ]^T = (\mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3) / (q_1 + q_2 + q_3)^{0.5} \quad [VI]$$

while their rates are $\mathbf{w}_B = [\ w_1,\ w_2,\ w_3]^T$ which can be calculated in the following manner:

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

$$\mathbf{L}_B = \mathbf{A}\ \mathbf{L} \quad [VII]$$

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

$$\mathbf{A} = \begin{bmatrix} q_4 + q_1 - q_2 - q_3 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & q_4 - q_1 + q_2 - q_3 & 2(q_2q_3 + q_3q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & q_4 - q_1 - q_2 + q_3 \end{bmatrix} \quad [VIII]$$

Knowing the moment of inertia tensor **I**, one can calculate the angular rates

$$w_B = [\ w_1,\ w_2,\ w_3\ ]^T$$
$$= I^{-1}\ L_B \quad = I^{-1}\ (A\ L) \qquad [IX]$$

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 MODE, generates the appropriate command string CMDRAW.

Case 1 MODE <> 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_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 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 MODE <> 0 AND MODE <> 1

In this case, MODE is set to 1, and position control is assumed.

IMPLEMENTATION

This algorithm is implemented as a subroutine named SVX (S, CMDRAW, 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.

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 intial state on strat 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, 1, 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

$$E = [\ E_1,\ E_2,\ E_3\ ]^T$$

$$= [\ c + 1 - X_o,\ a - Y_o,\ h - Z_o\ ]^T$$

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 :

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

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^{o} 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 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^{o} 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 Z_M 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 Z_M axis and translate along the X_M axis simultaneously, as demanded by this state vector.

CONCLUSION

Other simular 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 simular 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.

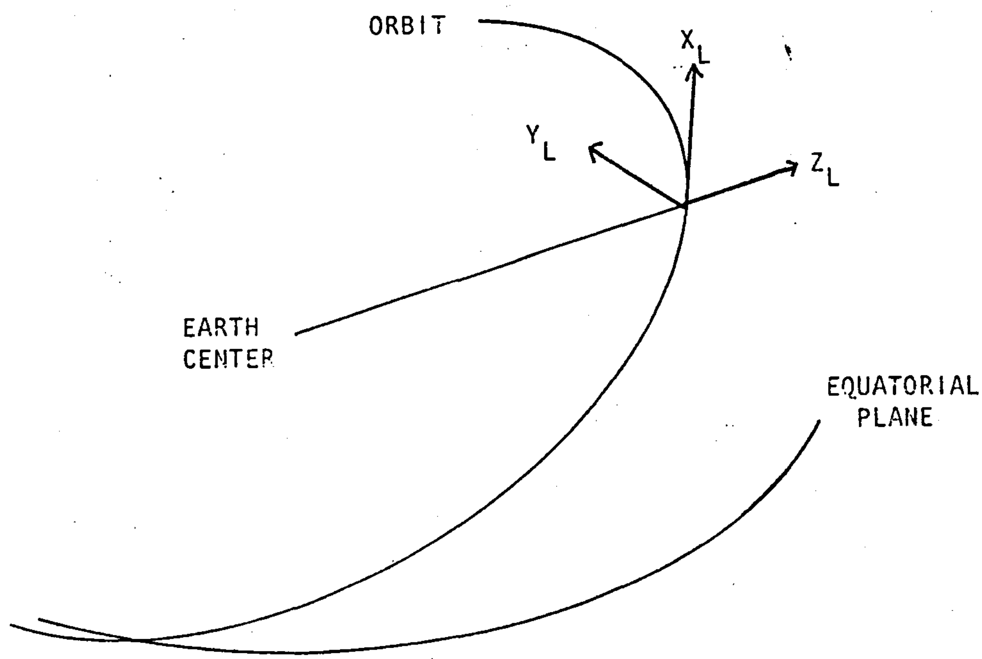


Fig. 1 Local Vertical Frame (L)

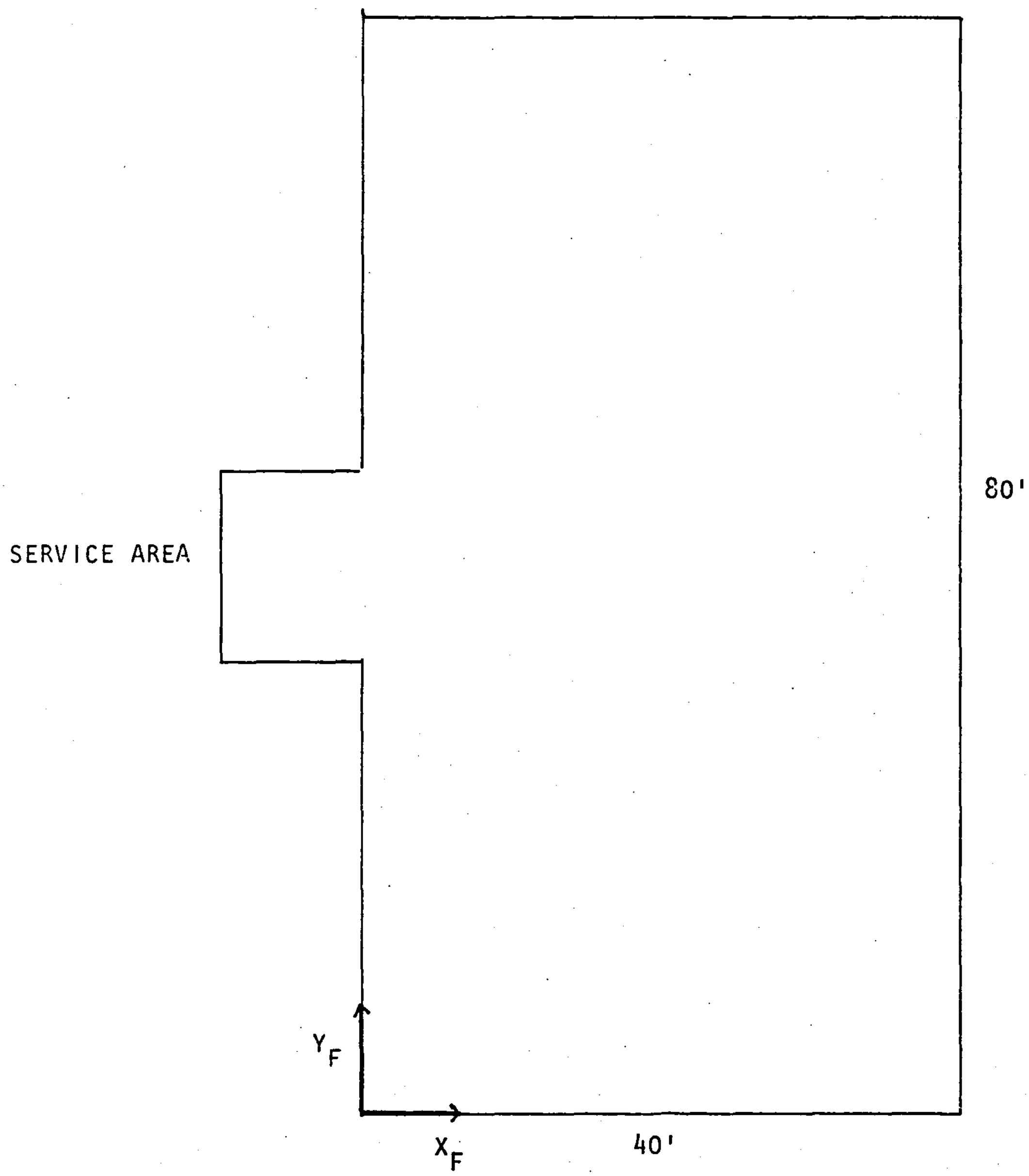


Fig 2. Floor coordinates (F)

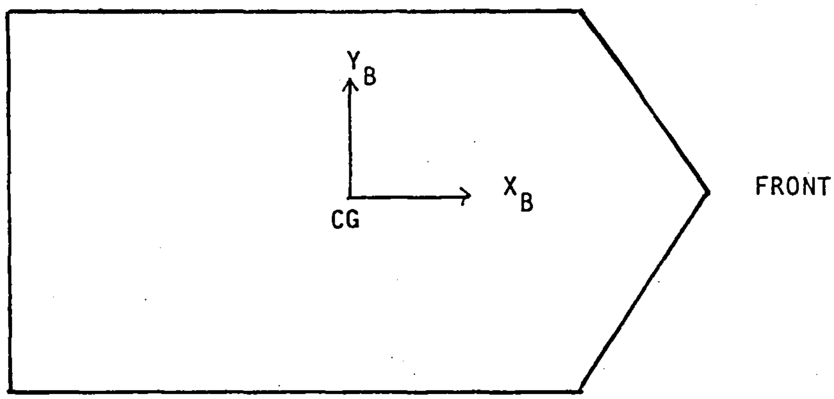


Fig 3. TOM_B Body Frame (B)

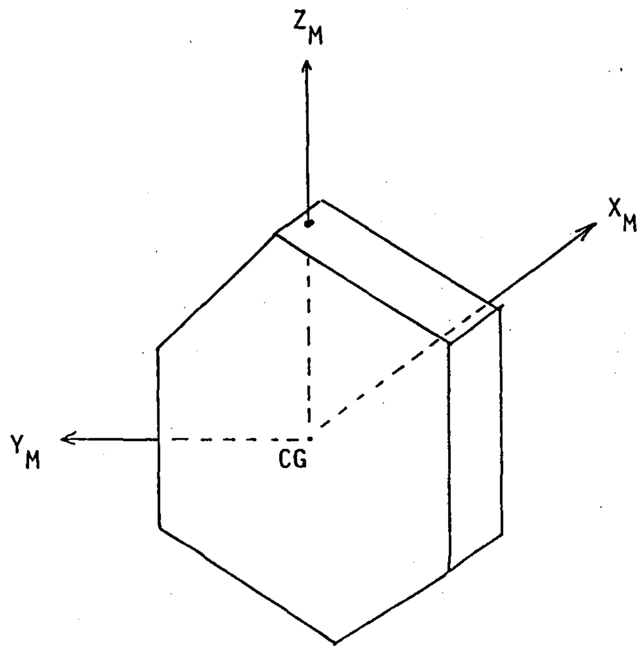


Fig 4. MOCK-UP MODULE BODY FRAME (B)

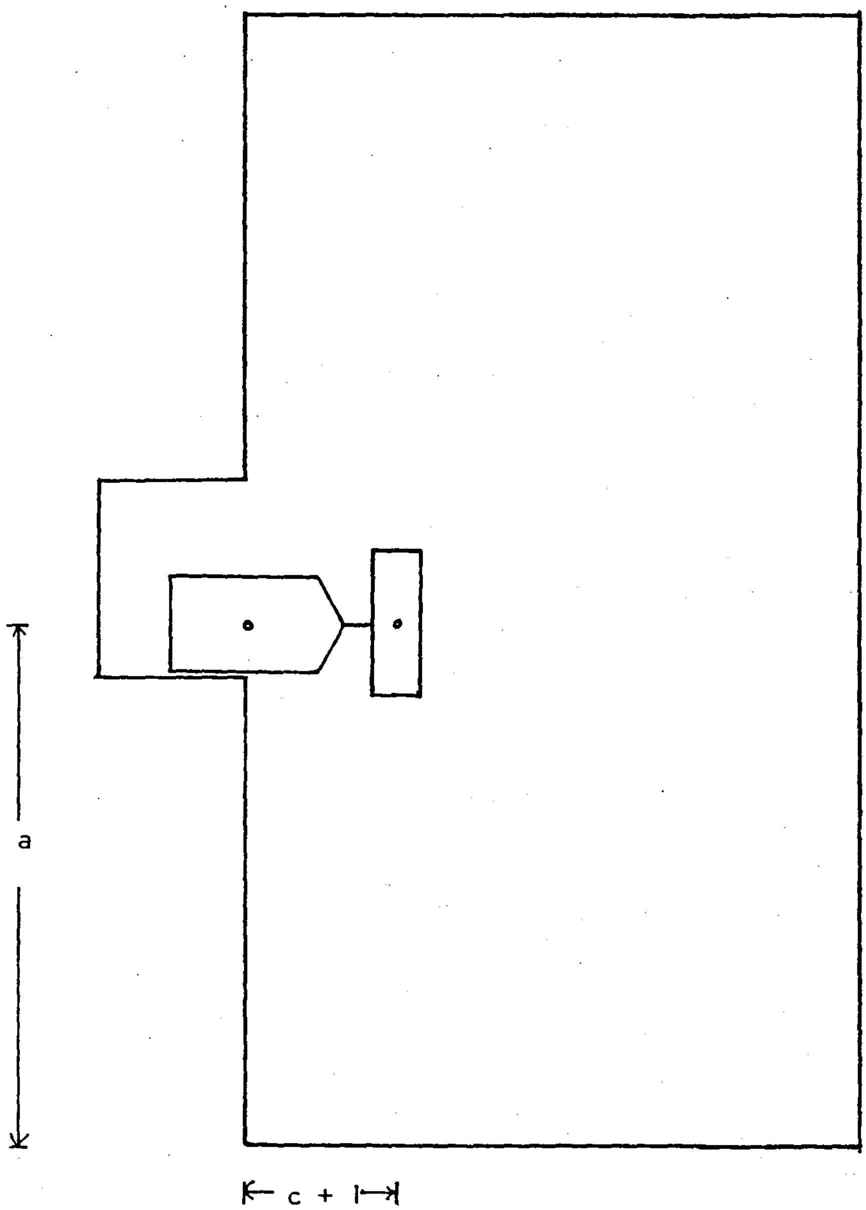


Fig 5 a) Initial position (top view)

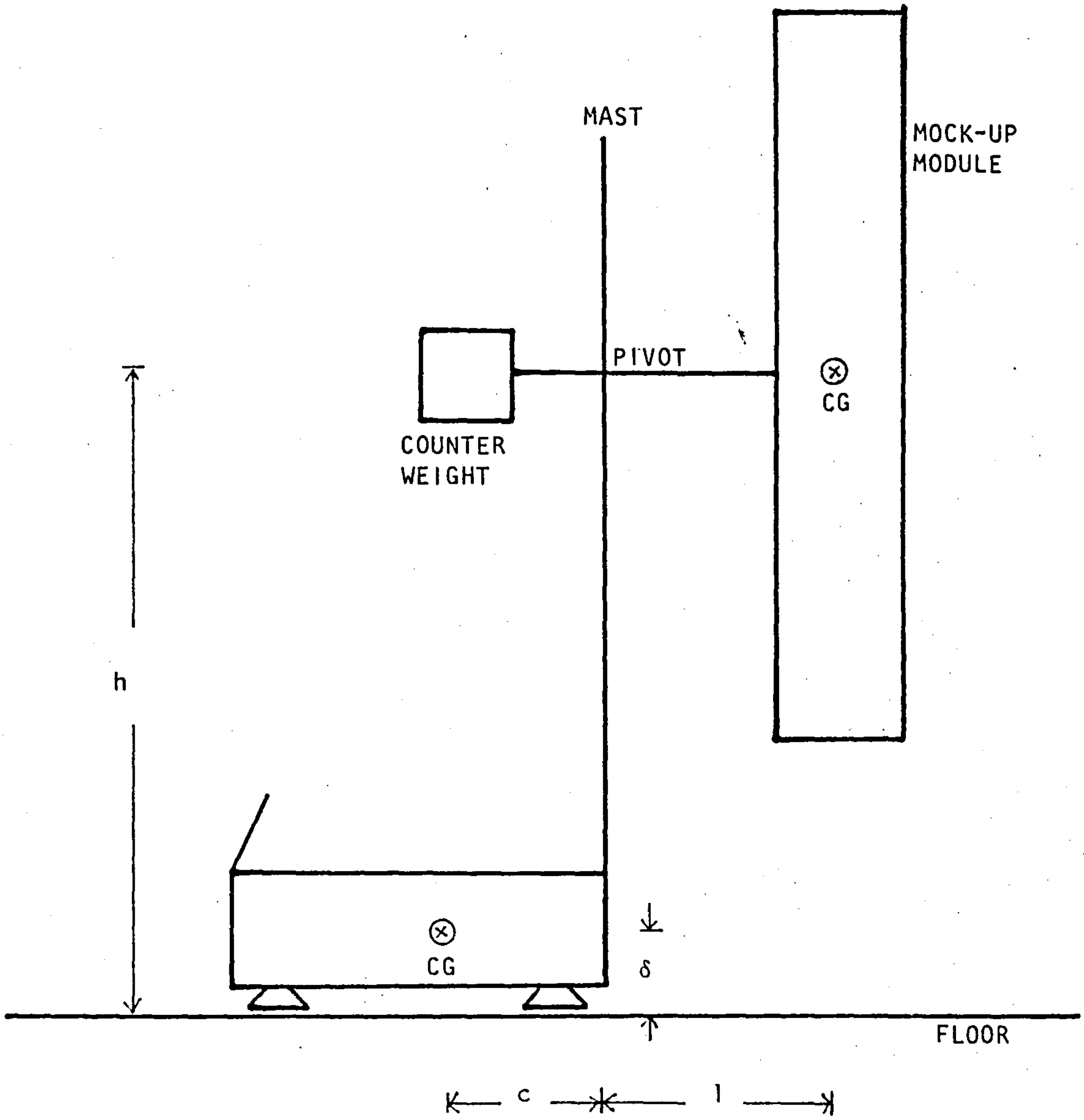


Fig 5 b)   Initial position   (side view)

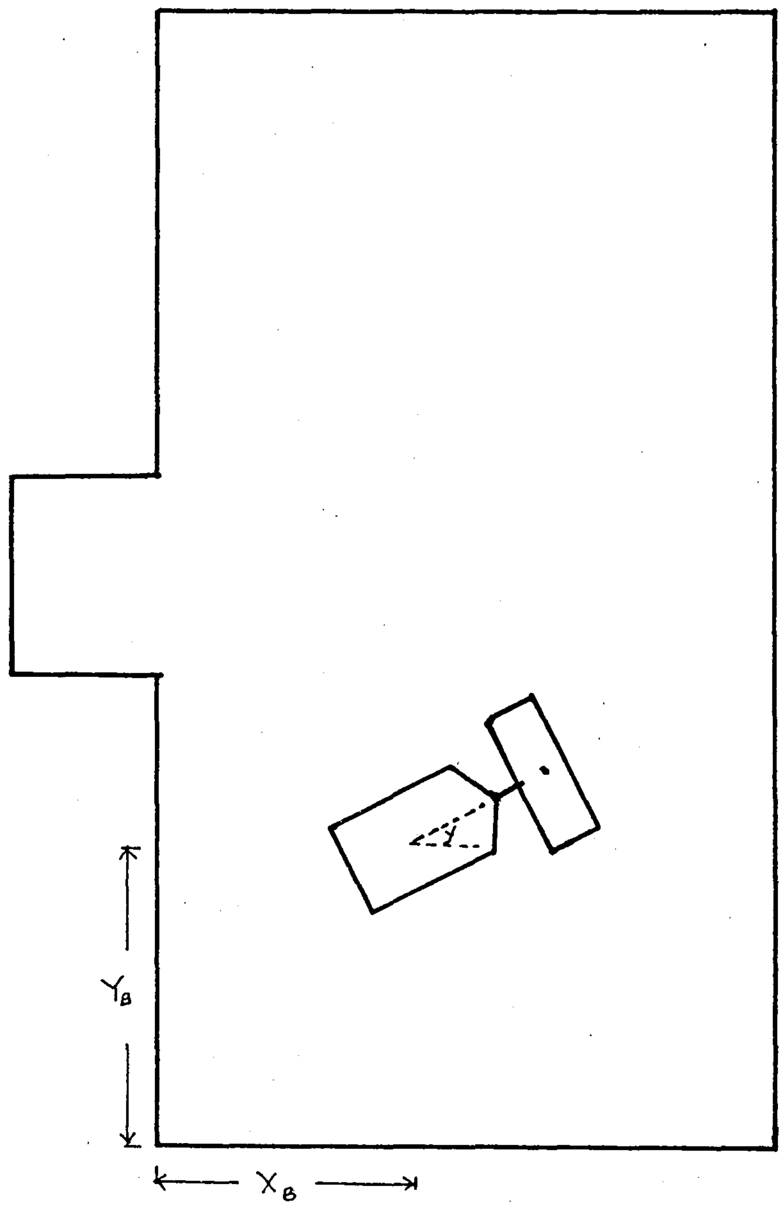


Figure 6 a) Position and yaw of TOM_B

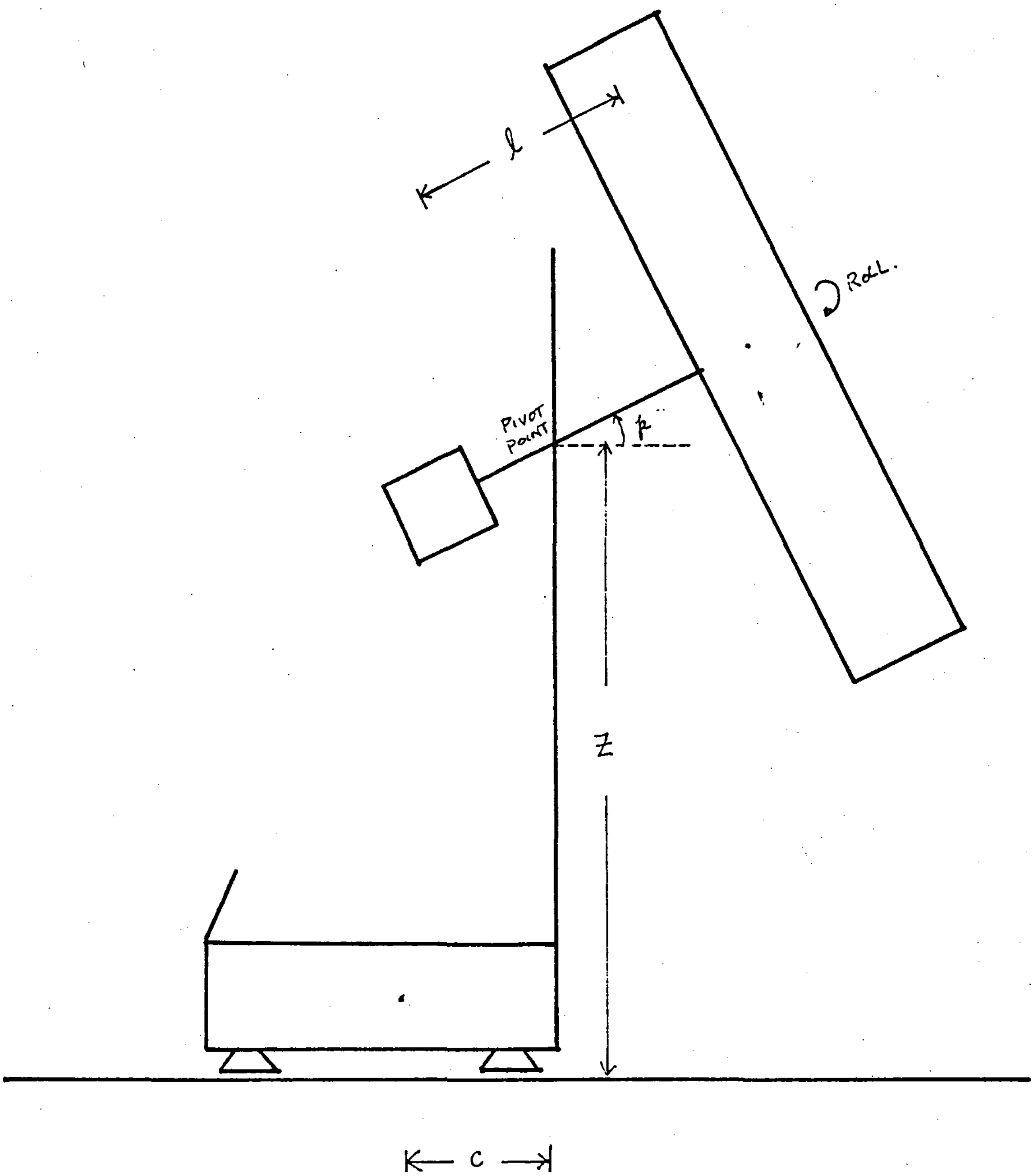


Figure 6 b) Pitch and roll of Mock-up Module

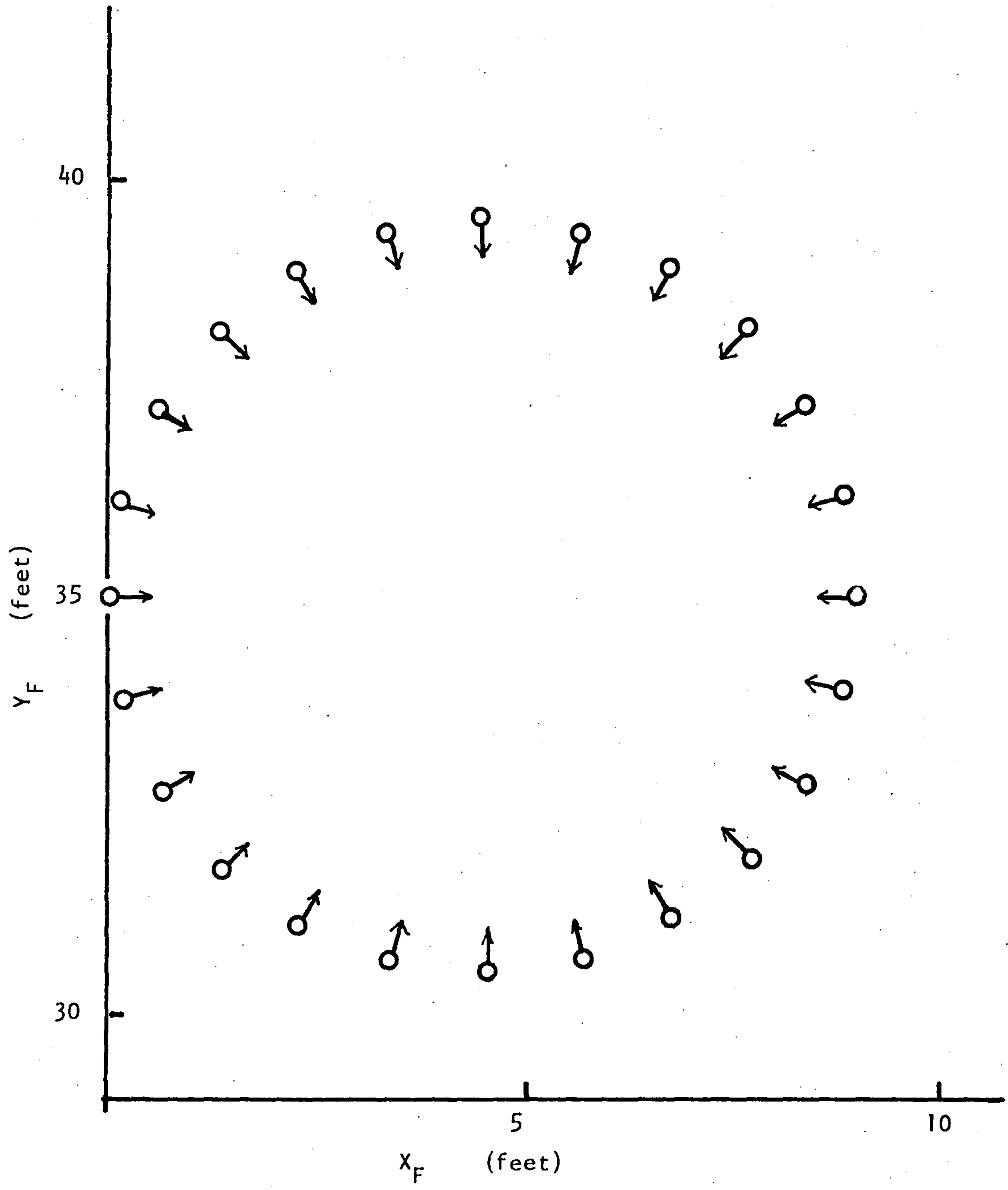


Figure 7 Position of TOM_B in floor coordinates

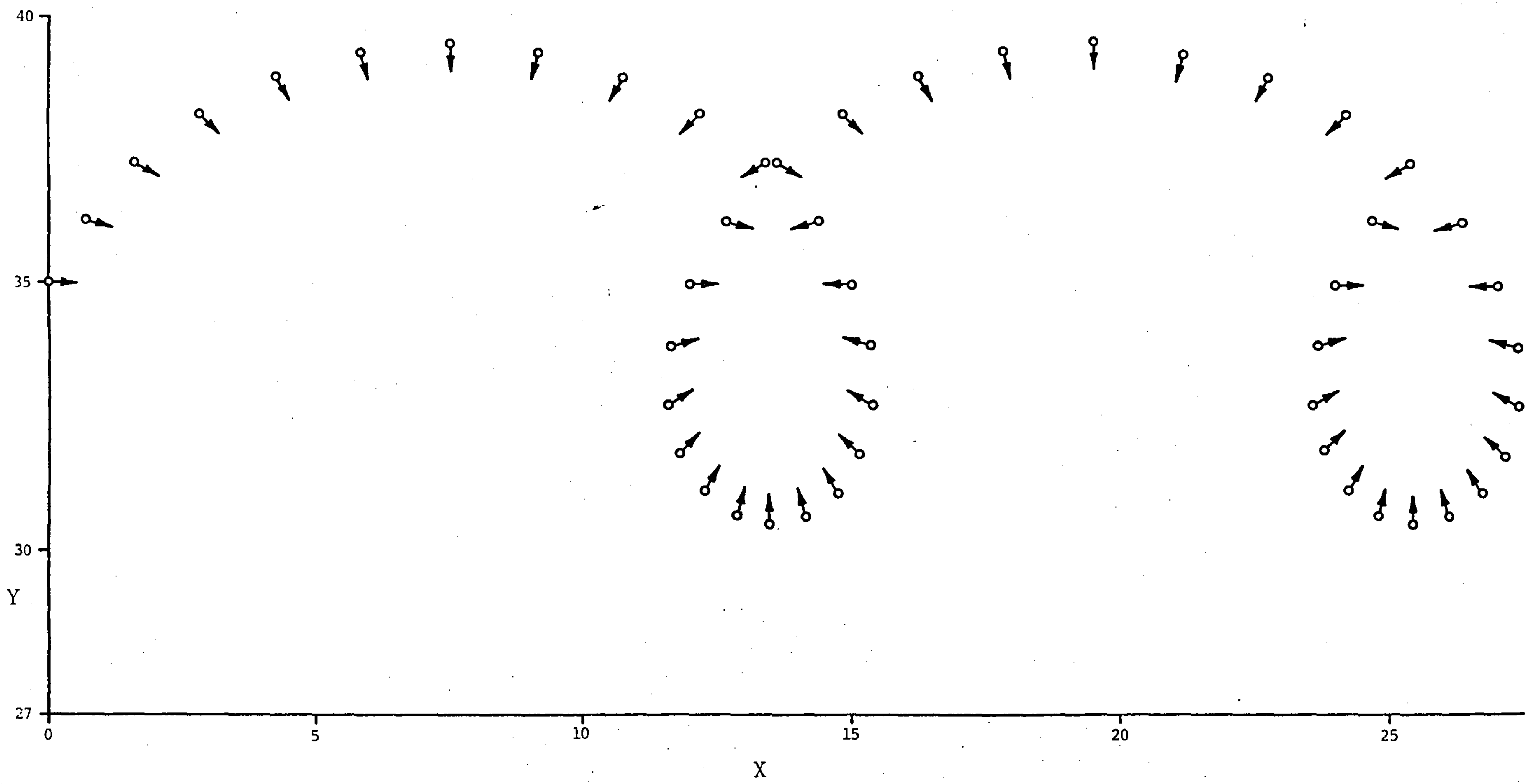


Fig. 8 Trajectory of TOM_B

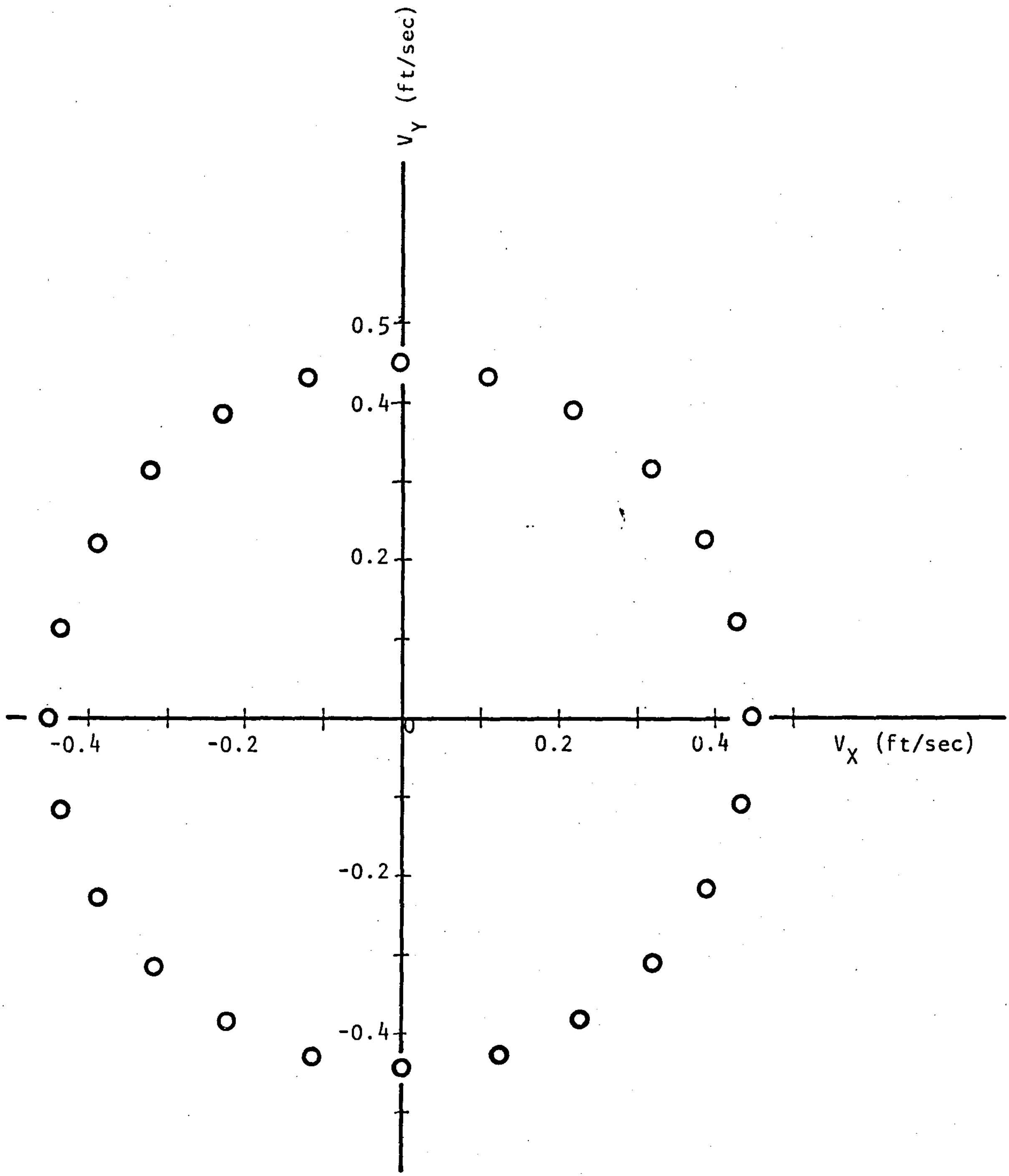


Figure 9   Velocity components of TOM_B

List of References

1. "Software Specification for Docking Simulations of the Orbital Maneuvering Vehicle OMV", J.D. Micheal, NASA internal communication.1984.

2. "Equations of Motion for Six Degree-of-freedom TRS Simulation", J.Galaboff, Systems Dynamics Lab, 1978.

3. "Development of an Autonomous Video Randezvous and Docking System", J.C. Tietz and T.E. Richardson, Martin Marieta Aerospace Phase 2 Final Report, 1983, NASA Contract # NAS8-34679.

4. "A New Method for Performing Digital Control System Attitude Computation Using Quaternions:, B.P. Ickes, AIAA Journal vol 8(1970)13.

5. "Design of a Terminal Pointer Hand Controller For Teleoperator Applications", E.L. Seanger & W.S. Woltosz, URS/Matrix Co. Final Report, 1973, NASA Contract # NAS8-28760.

```
$PAGESIZE: 56
$TITLE: '<<<   S V X   >>>'
C
C
C                 STATE VECTOR TRANSFORMATION MODULE  (SVX)
C
C                                    by
C
C
C                                 Dr. W. Teoh
C
C                                 U A H  1984
C
C-----------------------------------------------------------------------
C
C
            SUBROUTINE  SVX (S, CMDRAW, MODE)
C
C
C-----------------------------------------------------------------------
C
C     This is the state vector transformation module which accepts a
C     14 element  state vector S of the OMV as input and generates a
C     6-element command string  CMDRAW  as output. The argument MODE
C     conveys the following meaning :
C
C              MODE                 Meaning
C                0                rate control
C                1                position control
C           anything else         defaults to 1
C
C     Summary of the state vector components are as follows :
C
C        Component             Meaning
C
C            1        X       position of target vehicle from the
C            2        Y       chase vehicle in LVF
C            3        Z
C            4        VX      relative velocity of the two vehicles
C            5        VY      in LVF
C            6        VZ
C            7        LX      angular momentum vector in LVF
C            8        LY
C            9        LZ
C           10        Q1      attitude quaternions in body frame
C           11        Q2
C           12        Q3
C           13        Q4
C           14        M       instantaneous mass in kg.
```

```
C
C
C     Summary of command string components:
C
C         component    meaning          coord system
C             1          YAW          body frame
C             2           X           floor coordinate
C             3           Y           floor coordinate
C             4           Z           floor coordinate
C             5         PITCH         body frame
C             6          ROLL         body frame
C             7          MODE         integer
c
C
C     This module maintains a local counter to process initial
C     conditions at the start of the simulation.
C
C-----------------------------------------------------------------------
C
C
          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    LL, UL, UA, CC, AA, HH, QQ, TX, TY, Z
          REAL * 8    ROLL, PITCH, YAW, ROLDOT, PITDOT, YAWDOT
          REAL * 8    Q1, Q2, SY, CY, VX, VY, VZ
C
          INTEGER  CMDRAW(7), COUNT, MODE
C
C         ***  load-time initialization
C
          DATA     COUNT /0/
C
C         ***  decompose state vector and process it
C
          CALL  DECOMP (S, X, V, L, Q)
          IF (COUNT .NE. 0) GOTO 300
C
C            ***  initialization before start
C
             CALL ZERO (XO, 3)
C
C            ***  read parameters
C
             OPEN (1, FILE = 'SVXINT.DAT', STATUS = 'OLD')
             READ (1, 20) CC, LL, AA, HH
             READ (1, 20) IINV
```

```
C
C        ***  calculate inverse of moment of inertia tensor
C
         DO 50 K = 1, 3
            IINV(K) = 1.0 / IINV(K)
   50    CONTINUE
         CLOSE (1)
C
C        ***  set conversion factors
c
         UL = 10000.0
         UA = UL
         COUNT = COUNT + 1
C
C        ***  set transformation matrix elements to floor coord.
C
         E(1) = CC + LL - XO(1)
         E(2) = AA - XO(2)
         E(3) = HH - XO(3)
C
C        ***  initialize to home orientation
C
         CALL  ZERO (RPY, 3)
         COUNT = COUNT + 1
C
  300 IF (MODE .NE. 1) GO TO 400
C
C        ***  position commands
C
C        ***  update orientation and position
C
         CALL QTRPY  (Q, ROLL, PITCH, YAW)
         CALL UPDPOS (XM, X, XHOLD, E, 3)
C
C        ***  set orientation part of the command string
C
         CMDRAW(7) = 1
         CMDRAW(6) = JFIX(ROLL * UA)
         CMDRAW(5) = JFIX(PITCH * UA)
         CMDRAW(1) = JFIX(YAW * UA)
C
C        ***  transform to TOM_B position in floor coordinates
C
         QQ = CC + LL * DCOS(PITCH)
C
C        ***  X-component
C
         TX = XM(1) - QQ * DCOS(YAW)
         CMDRAW(2) = JFIX (TX * UL)
```

```
C
C           ***  Y-component
C
            TY = XM(2) - QQ * DSIN(YAW)
            CMDRAW(3) = JFIX (TY * UL)
C
C           ***  Z-component
C
            Z = XM(3) - LL * DSIN(PITCH)
            CMDRAW(4) = JFIX (Z * UL)
C
C           ***  This is a good place to call the I/O driver to
C           ***  transmit to TOM_B, but we won't for now
C
            RETURN
C
  400     IF (MODE .NE. 0) GO TO 900
C
C           ***  rate control
C
            CALL  QTRPY  (Q, ROLL, PITCH, YAW)
C
C           ***  form direction cosine matrix and calculate angular
C           ***  momentum in body frame
C
            CALL  DIRCOS (A, Q)
            CALL  MMUL (A, L, LB, 3)
C
C           ***  compute body rate
C
            ROLDOT = IINV(1) * LB(1)
            PITDOT = IINV(2) * LB(2)
            YAWDOT = IINV(3) * LB(3)
C
C           ***  construct orientation part of command string
C
            CMDRAW(7) = 0
            CMDRAW(6) = JFIX (ROLDOT * UA)
            CMDRAW(5) = JFIX (PITDOT * UA)
            CMDRAW(1) = JFIX (YAWDOT * UA)
C
C           ***  compute velocity of TOM_B in floor coordinates
C
            Q1 = LL * DSIN(PITCH) * PITDOT
            Q2 = (CC + LL * DCOS(PITCH)) * YAWDOT
            SY = DSIN(YAW)
            CY = DCOS(YAW)
C
C           ***  X-component of velocity in floor coordinate
```

```
C
                VX = V(1) + Q1 * CY + Q2 * SY
                CMDRAW(2) = JFIX (VX * UL)
C
C               *** Y-component of velocity in floor coordinate
C
                VY = V(2) + Q1 * SY - Q2 * CY
                CMDRAW(3) = JFIX (VY * UL)
C
C               *** Z-component
C
                VZ = V(3) - LL * DCOS(PITCH) * PITDOT
                CMDRAW(4) = JFIX (VZ * UL)
                RETURN
C
  900         CONTINUE
C
C               *** We have an un-recognizable code, default to 1 for
C               *** position control
C
                MODE = 1
                GO TO 300
C
   10         FORMAT (4F10.2)
   20         FORMAT ( F15.8)
              END
```

```
Name    Type          Offset P Class

A       REAL*8          466
AA      REAL*8          558
CC      REAL*8          542
CMDRAW  INTEGER*4         4 *
COUNT   INTEGER*4       538
CY      REAL*8          698
DCOS                           INTRINSIC
DSIN                           INTRINSIC
E       REAL*8          418
IH      REAL*8          566
IINV    REAL*8          442
K       INTEGER*4       574
L       REAL*8          370
LB      REAL*8          394
LL      REAL*8          550
MODE    INTEGER*4         8 *
PITCH   REAL*8          602
PITDOT  REAL*8          658
Q       REAL*8          154
Q1      REAL*8          674
```

```
2       REAL*8        682
P^T     REAL*8        210
(       REAL*8        618
W       REAL*8        242
OLDOT   REAL*8        650
( L     REAL*8        594
I I     REAL*8        186
        REAL*8          0 *
        REAL*8        690
        REAL*8        626
Y       REAL*8        634
        REAL*8        586
        REAL*8        578
        REAL*8         98
X       REAL*8        706
        REAL*8        714
        REAL*8        722
        REAL*8        122
        REAL*8          2
        REAL*8         26
HOLD    REAL*8         74
M       REAL*8         50
 N      REAL*8        610
AWDOT   REAL*8        666
        REAL*8        642

   223 $PAGE
```

```
D Line# 1      7                          Microsoft FORTRAN77 V3.13 8/05/83
      C
      C
              SUBROUTINE   DECOMP (S, X, V, L, Q)
      C
      C----------------------------------------------------------------------
      C
      C       This procedure decomposes the State vector S into its components
      C       which are also vectors. They have the following meaning :
      C
      C          Vector    Dimension          Meaning
      C            X           3        Position vector in LVF
      C            V           3        Velocity vector in LVF
      C            L           3        Angular momentum in LVF
      C            Q           4        Unit quaternion in body frame
      C
      C----------------------------------------------------------------------
      C
              REAL * 8     S(14), X(3), V(3), L(3), Q(4)
      C
              CALL   LD (S, X,  1, 3)
              CALL   LD (S, V,  4, 3)
              CALL   LD (S, L,  7, 3)
              CALL   LD (S, Q, 10, 4)
      C
              RETURN
              END

Name    Type        Offset P Class

L       REAL*8          12 *
Q       REAL*8          16 *
S       REAL*8           0 *
V       REAL*8           8 *
X       REAL*8           4 *

     $PAGE
```

.ine# 1   7   Microsoft FORTRAN77 V3.13 8/05/83

```
C
C
        SUBROUTINE   LD (A, B, M, N)
C
C-----------------------------------------------------------------------
C
C     This procedure copies N elements of vector A to vector B,
C     starting at the M-th element
C
C-----------------------------------------------------------------------
C
        REAL * 8  A(14), B(N)
        DO 100 K = 1, N
           B(K) = A(M + K - 1)
  100   CONTINUE
        RETURN
        END
```

ne	Type	Offset	P	Class
	REAL*8	0	*	
	REAL*8	4	*	
	INTEGER*4	750		
	INTEGER*4	8	*	
	INTEGER*4	12	*	

```
$PAGE
```

```
      C
      C
                SUBROUTINE  MMUL (A, B, C, N)
      C
      C-----------------------------------------------------------------
      C
      C     This procedure performs a matrix multiplication of an NxN
      C     matrix A to an  N-element column matrix  B to yield an N-
      C     element column matrix C
      C
      C-----------------------------------------------------------------
      C
                REAL * 8    A(N,N), B(N), C(N), S
      C
                DO 100 I = 1, N
 1                 S = 0.0
 1                 DO 200 J = 1, N
 2                    S = S + A(I,J) * B(J)
 2        200      CONTINUE
 1                 C(I) = S
 1        100   CONTINUE
                RETURN
                END
```

Name	Type	Offset	P	Class
A	REAL*8	0	*	
B	REAL*8	4	*	
C	REAL*8	8	*	
I	INTEGER*4	758		
J	INTEGER*4	774		
N	INTEGER*4	12	*	
S	REAL*8	766		

```
$PAGE
```

```
C
C
      SUBROUTINE  ZERO (A, N)
C
C------------------------------------------------------------------
C
C    This procedure initializes an N-element array A to zero at
C    run time
C
C------------------------------------------------------------------
C
      REAL * 8  A(N)
      DO 100 K = 1, N
         A(K) = 0.0
100   CONTINUE
      RETURN
      END
```

ne	Type	Offset	P	Class
	REAL*8	0	*	
	INTEGER*4	782		
	INTEGER*4	4	*	

```
$PAGE
```

```
C
C
      SUBROUTINE  UPDPOS (XM, X, XHOLD, E, N)
C
C-----------------------------------------------------------------------
C
C     This procedure updates the position of the OMV in local vertical
C     frame (XHOLD).
C
C     The new position of the module in floor coordinates is then com-
C     puted (XM)
C
C-----------------------------------------------------------------------
C
C
      REAL * 8   XM(N), X(N), XHOLD(N), E(N)
C
      DO 100 K = 1, N
         XHOLD(K) = X(K)
         XM(K)    = XHOLD(K) + E(K)
  100 CONTINUE
      RETURN
      END
```

Name	Type	Offset	P	Class
E	REAL*8	12	*	
K	INTEGER*4	790		
N	INTEGER*4	16	*	
X	REAL*8	4	*	
XHOLD	REAL*8	8	*	
XM	REAL*8	0	*	

```
$PAGE
```

```
C
C
      INTEGER FUNCTION JFIX (RR)
C
C----------------------------------------------------------------------
C
C     This procedure properly rounds a real number R to the nearest
C     integer.
C
C----------------------------------------------------------------------
C
      REAL * 8   RR
      REAL       R
      R  =  RR
      IF (R .GE. 0) THEN
         JFIX = IFIX (R + 0.5)
      ELSE
         JFIX = IFIX (R - 0.5)
      END IF
      RETURN
      END
```

```
me   Type        Offset P Class

FIX                        INTRINSIC
     REAL           798
     REAL*8           0 *
```

```
$PAGE
```

```
D Line# 1      7                         Microsoft FORTRAN77 V3.13 8/05/83
      C
      C
              SUBROUTINE   SETQ (QW, Q)
      C
      C----------------------------------------------------------------------
      C
      C     This procedure constructs a 4x4 transformation matrix QW  from
      C     the attitude quaternions Q
      C
      C     For reference, please see "Software Specifications For Docking
      C     Simulation Of The OMV" by J. Micheals, January, 1984.
      C
      C----------------------------------------------------------------------
      C
              REAL * 8    QW(4,4), Q(4)
      C
              DO 100 I = 1, 3
1                 DO 110  J = I+1, 4
2                    KK = I + J
2                    K = KK - (KK/4) * 4
2                    IF (K .EQ. 0) K = 2
2                    ISGNN = 1
2                    IF ((J .EQ. I+1) .AND. (J.NE. 4)) ISGNN = -1
2                    QW(I,J) = ISGNN * Q(K)
2       110       CONTINUE
1                 QW(I,I) = Q(4)
1       100   CONTINUE
              QW(4,4) = Q(4)
      C
              DO 200 I = 2, 4
1                KK = I - 1
1                DO 200 J = 1, KK
2                   QW(I,J) = -QW(J,I)
2       200   CONTINUE
              RETURN
              END
```

Name	Type	Offset	P	Class
I	INTEGER*4	802		
ISGNN	INTEGER*4	818		
J	INTEGER*4	806		
K	INTEGER*4	814		
KK	INTEGER*4	810		
Q	REAL*8	4	*	
QW	REAL*8	0	*	

393 $PAGE

```
Line# 1     7                                 Microsoft FORTRAN77 V3.13 8/05/83
C
C
         SUBROUTINE    DIRCOS (A, Q)
C
C-------------------------------------------------------------------------
C
C        This procedure takes the quaternion vector and generates
C        a 3 X 3 direction cosine matrix A
C
C-------------------------------------------------------------------------
C
C
         REAL * 8    Q(4), A(3,3), QKS, QRS, S1
C
         DO 100 K = 1, 3
C
C           ***   initialize diagonal elements
C
            A(K,K) = Q(4) ** 2
            DO 100 J = 1, 3
C
C              ***   fix up the diagonal elements
C
               A(K,K) = A(K,K) + DLTKRK(K,J) * Q(J) ** 2
C
C              ***   now do the off-diagonal elements
C
               IF ( J  .GT. K ) THEN
C
C                 ***   calculate index I <> J & K
C
                  I   = 6 / (J * K)
C
C                 ***   calculate the proper sign
C
                  S1  = QSIGN (K,J)
                  QKJ = Q(K) * Q(J)
                  QRS = Q(I) * Q(4) * S1
                  A(K,J) = 2.0 * (QKJ + QRS)
                  A(J,K) = 2.0 * (QKJ - QRS)
              END IF
   100   CONTINUE
         RETURN
         END
```

```
ame   Type          Offset P Class

      REAL*8            0 *
      INTEGER*4       838
```

```
D Line# 1      7
J       INTEGER*4      830
K       INTEGER*4      826
Q       REAL*8           4 *
QKJ     REAL           854
QKS     REAL*8       *****
QRS     REAL*8         858
S1      REAL*8         842

    438 $PAGE
```

```
C
C
      REAL    FUNCTION  DLTKRK (K,J)
C
C
C-------------------------------------------------------------------
C
C
      REAL              S
      INTEGER        K, J
      S = 1.0
      IF (K .NE. J) S = -1.0
      DLTKRK = S
      RETURN
      END
```

ame	Type	Offset	P	Class
	INTEGER*4	4	*	
	INTEGER*4	0	*	
	REAL	866		

```
$PAGE
```

```
D Line# 1      7                                       Microsoft FORTRAN77 V3.13 8/05/83
        C
        C
               REAL    FUNCTION QSIGN(K,J)
        C
        C
        C----------------------------------------------------------------------
        C
        C
               S = 1.0
               L = J + K
               IF (MOD(L,2) .EQ. 0) S = -1.0
               QSIGN = S
               RETURN
               END
```

Name	Type	Offset	P	Class
J	INTEGER*4	4	*	
K	INTEGER*4	0	*	
L	INTEGER*4	874		
MOD				INTRINSIC
S	REAL	870		

```
        $PAGE
```

```
C
C
      SUBROUTINE  QTRPY (Q, R, P, Y)
C
C
C     This subroutine calculates a reasonable set of roll,
C     pitch and yaw from the quaternion Q
C
C
      REAL * 8    Q(4), R, P, Y, M, THETA, CA, CB, CG
C
      M  = DSQRT (Q(1)**2 + Q(2)**2 + Q(3)**2)
C
C     calculate direction cosines CA, CB, CG
C
      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
C
C     calculate angle of rotation about Euler axis
C
      THETA = 2.0 * DACOS(Q(4))
C
C     now determine the roll, pitch and yaw
C
      R = CA * THETA
      P = CB * THETA
      Y = CG * THETA
      RETURN
      END
```

Name	Type	Offset	P	Class
A	REAL*8	886		
B	REAL*8	894		
CG	REAL*8	902		
ABS				INTRINSIC
ACOS				INTRINSIC
DSQRT				INTRINSIC
	REAL*8	878		
	REAL*8	8	*	
Q	REAL*8	0	*	
R	REAL*8	4	*	

```
D Line# 1      7
THETA   REAL*8              910
Y       REAL*8               12 *

    506 $PAGE
```

Name	Type	Size	Class
ECOMP			SUBROUTINE
IRCOS			SUBROUTINE
:KRK	REAL		FUNCTION
rIX	INTEGER*4		FUNCTION
D			SUBROUTINE
JL			SUBROUTINE
.IGN	REAL		FUNCTION
TRPY			SUBROUTINE
TQ			SUBROUTINE
{			SUBROUTINE
PDPOS			SUBROUTINE
ERO			SUBROUTINE

Pass One   No Errors Detected
             506 Source Lines

Appendix 3

OMVPLOT -- Source Listing

```
$PAGESIZE: 56
$TITLE:    '<<<  O M V   P L O T   >>>'
C
C---------------------------------------------------------------------
C
C
C                      Program  :      O M V P L O T
C
C
C                                  by
C
C
C                              Dr. W. Teoh
C
C
C                              U A H  1984
C
C
C
C
C
C---------------------------------------------------------------------
C
C
C        This is a graphical package that accepts a command string
C        and uses this information to generate and display the
C        position and orientation of TOM_B and the attached mock-
C        up module in two dimensions. One can choose to display
C        either the top or side view of the system.
C
C        This package is developed in FORTRAN 77 to run on an IBM
C        PC with at least 128K of RAM, and fitted with a TECMAR
C        GRAPHICS MASTER board.  An IBM Monochrome monitor is used
C        for the actual display.  The resolution in this work is
C        chosen to be 640 x 350.
C
C
C---------------------------------------------------------------------
C
C
C
C
         SUBROUTINE  SIDEVEW  (H, X, P)
C
C
C---------------------------------------------------------------------
C        This procedure produces a side view of TOM_B and the
C        attached mock-up module. The perspective is always in
C        the direction of +1 axis of the body fixed coordinate
```

```
D Line# 1      7                          Microsoft FORTRAN77 V3.13 8/05/83
C         system
C
C
C----------------------------------------------------------------------
C
C
          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
C
          COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
          COMMON /MF/ XFORM, SDFORM, VO, V1
          COMMON /ME/ EF, EEF, PRTFG
C
C
          N  = 10
          AA = 1.0
C
C         ***  define mock-up module shape at origin
C
          DO 100 K = 1, N
             V(3, K) = 1.0
  100     CONTINUE
C                                           << point  A >>
          V(1,1)  =  TT
          V(2,1)  = -DD
C                                           << point  B >>
          V(1,2)  = -TT
          V(2,2)  = -DD
C                                           << point  C >>
          V(1,3)  = -TT
          V(2,3)  =  DD
C                                           << point  D >>
          V(1,4)  =  TT
          V(2,4)  =  DD
C
C         ***  rotate it by P radians
C
          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)
C
C         ***  calculate translation
```

Line# 1 7 Microsoft FORTRAN77 V3.13 8/05/83

```
C
      PX = CC + LL * C
      PY = H +     LL * S
C
C     ***  move the rotated module out there
C
      CALL  NOTHNG (ROT, 3)
      ROT(1,3) = PX
      ROT(2,3) = PY
      CALL  XMUL (ROT, V, 4)
C
C     ***  now calculate the shape of the base
C
C     XX = X + CC
C                                    << point  E >>
      V(1,5) = CC
      V(2,5) = H
C                                    << point  F >>
      V(1,6) = CC
      V(2,6) = AA
C                                    << point  G >>
      V(1,7) = CC
      V(2,7) = 0.
C                                    << point  H >>
      V(1,8) = -RR
      V(2,8)  = 0.
C                                    << point  I >>
      V(1,9) = -RR
      V(2,9) = AA
C
      V(1,10) = PX
      V(2,10) = PY
C
C     ***  Transform to floor coordinates
C
      CALL  NOTHNG (FLOOR, 3)
      FLOOR(1,3) = X
      CALL  XMUL (FLOOR, V, N)
C
C     ***  transform to screen coordinates
C
      CALL  XMUL (SDFORM, V, N)
C
C     ***  erase old picture
C
      CALL  DRWFLR (VO)
      IF ((EF .EQ. 0) .AND. (EEF .NE. 0)) THEN
         CLR = 0
         CALL  SDRAW (V1, N, CLR)
```

```
          END IF
          CLR = 1
          CALL  SDRAW (V, N, CLR)
          CALL  MOVE (V, V1, N)
          EEF = 1
C
          RETURN
          END
```

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

$PAGE

```
C
C
      SUBROUTINE  SDRAW (V, N, CLR)
C
C
C------------------------------------------------------------------
C
C
C     This procedure draws the side view of TOM_B
C
C
C------------------------------------------------------------------
C
C
      REAL        V(3,10)
      INTEGER     N, CLR, X1, X2, Y1, Y2
C
C     ***  draw mobile base
C
      CALL  RCT (V, 5, CLR)
C
C     ***  draw linkage
C
      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)
C
C     ***  draw mock-up module
C
      CALL  RCT (V, 0, CLR)
      CALL  PURGE
      CALL  GRFRDY
C
      CALL  HOME
C
      RETURN
      END
```

```
ame   Type        Offset P Class

LR    INTEGER*4        8 *
      INTEGER*4        4 *
      REAL             0 *
1     INTEGER*4      238
```

D Line# 1   7   Microsoft FORTRAN77 V3.13 8/05/83

```
X2      INTEGER*4      246
Y1      INTEGER*4      242
Y2      INTEGER*4      250

    199 $PAGE
```

```
C
C
      SUBROUTINE  RCT  (V, OFF, CLR)
C
C
C-----------------------------------------------------------------------
C
C     This procedure draws a rectangle
C
C
C-----------------------------------------------------------------------
C
C
      REAL         V(3,10)
      INTEGER      OFF, CLR, X(10), Y(10)
C
      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
```

```
me    Type        Offset P Class

LR    INTEGER*4        8 *
      INTEGER*4      338
      INTEGER*4      334
)FF   INTEGER*4        4 *
      REAL             0 *
      INTEGER*4      254
      INTEGER*4      294
```

```
$PAGE
```

```
      SUBROUTINE  PLOT (CMD)
C
C
C-----------------------------------------------------------------------
C
C
C     This is the plot part of the graphical package, and can
C     be directly callable from OMV or SVX. The value of FLAG
C     obtained from the disk file named SIZE.DAT dictates one
C     of top or side view to be displayed.
C
C
C-----------------------------------------------------------------------
C
C
      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        V0(3,10), V1(3,10)
C
      COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
      COMMON /MF/ XFORM, SDFORM, V0, V1
C
      UL = 10000.0
      UA = UL
C
      IF (FLAG .EQ. 0) THEN
         T  = CMD(1) / UA
         X  = CMD(2) / UL
         Y  = CMD(3) / UL
         CALL  TOPVEW (X, Y, T)
      ELSE
         H = CMD(4) / UL
         X = CMD(2) / UL
         T = CMD(5) / UA
         CALL  SIDEVEW (H, X, T)
      END IF
C
      RETURN
      END
```

ame	Type	Offset	P	Class
CC	REAL	4		/MG /
CMD	INTEGER*4	0	*	
D	REAL	8		/MG /
FLAG	INTEGER*4	0		/MG /
H	REAL*8	382		
L	REAL	12		/MG /

```
R      REAL          16    /MG     /
FORM   REAL          36    /MF     /
       REAL*8       358
       REAL          24    /MG     /
       REAL*8       350
       REAL*8       342
       REAL          72    /MF     /
       REAL         192    /MF     /
       REAL          20    /MG     /
       REAL*8       366
FORM   REAL           0    /MF     /
       REAL*8       374

 265 $PAGE
```

```
C
C
      SUBROUTINE  TOPVEW (PX, PY, THETA)
C
C-----------------------------------------------------------------------
C
C
C     This procedure constructs the top view of TOM_B. No
C     correction to perspective distortion is implemented
C
C
C-----------------------------------------------------------------------
C
      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, V1(3,10)
      INTEGER     FLAG, N, CLR, EF, EEF, PRTFG
C
      COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
      COMMON /MF/ XFORM, SDFORM, VO, V1
      COMMON /ME/ EF, EEF, PRTFG
C
      N = 10
C
C     ***  get TOM_B shape at the origin
C
      CALL  ORGPOS (V, N)
C
C     ***  rotate by THETA if needed
C
C     IF  (THETA .NE. 0.0) THEN
C         ***  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
C         ***  rotate it
          CALL  XMUL (ROT, V, N)
C     END IF
C
C     ***  transform to floor coordinates
C
      CALL  NOTHNG (FLOOR, 3)
      FLOOR(1,3) = PX
      FLOOR(2,3) = PY
      CALL  XMUL (FLOOR, V, N)
```

Line# 1 7 Microsoft FORTRAN77 V3.13 8/05/83

```
C
C        ***  transform to screen coordinates
C
         CALL  XMUL (XFORM, V, N)
C
C        ***  get ready to draw, but first erase old picture
C
         CALL  DRWFLR (V1)
         IF ((EF .EQ. 0) .AND. (EEF .NE. 0)) THEN
             CLR = 0
             CALL  DRAW (VO, N, CLR)
         END IF
C
         CLR = 1
         CALL  DRAW (V, N, CLR)
         CALL  MOVE (V, VO, N)
         EEF = 1
C
         RETURN
         END
```

ne	Type	Offset	P	Class
	REAL*8	594		
	REAL	4		/MG /
R	INTEGER*4	602		
	REAL	8		/MG /
EF	INTEGER*4	4		/ME /
	INTEGER*4	0		/ME /
AG	INTEGER*4	0		/MG /
LOOR	REAL	546		
	REAL	12		/MG /
	INTEGER*4	582		
RTFG	INTEGER*4	8		/ME /
	REAL*8	0	*	
	REAL*8	4	*	
T	REAL	510		
R	REAL	16		/MG /
	REAL*8	586		
FORM	REAL	36		/MF /
IETA	REAL*8	8	*	
	REAL	24		/MG /
	REAL	390		
	REAL	72		/MF /
	REAL	192		/MF /
	REAL	20		/MG /
ORM	REAL	0		/MF /

```
$PAGE
```

```
      C
      C
                SUBROUTINE  MOVE (V, VO, N)
      C
      C
      C-------------------------------------------------------------------
      C
      C
      C         This procedure saves the shape vector V
      C
      C
      C-------------------------------------------------------------------
      C
      C
                REAL        V(3,10), VO(3,10)
      C
                DO 100 K = 1, N
 1                 DO 100 J = 1, 3
 2                    VO(J,K) = V(J,K)
 2        100   CONTINUE
      C
                RETURN
                END
```

Name	Type	Offset	P	Class
J	INTEGER*4	614		
K	INTEGER*4	606		
N	INTEGER*4	8	*	
V	REAL	0	*	
VO	REAL	4	*	

```
$PAGE
```

```
Line# 1      7                                Microsoft FORTRAN77 V3.13 8/05/83
      C
      C
                SUBROUTINE  NOTHNG (A, N)
      C
      C
      C-----------------------------------------------------------------------
      C
      C
      C         This procedure initializes an N x N matrix A to a
      C         unit matrix
      C
      C
      C-----------------------------------------------------------------------
      C
      C
                REAL        A(N,N)
      C
                DO 100 K = 1, N
                   DO 200 J = 1, N
                      A(K,J) = 0.0
        200        CONTINUE
                   A(K,K) = 1.0
        100     CONTINUE
      C
                RETURN
                END

ame   Type          Offset P Class

      REAL              0 *
      INTEGER*4       626
      INTEGER*4       618
      INTEGER*4         4 *


      $PAGE
```

```
D Line# 1      7                                Microsoft FORTRAN77 V3.13 8/05/83
          C
          C
                SUBROUTINE  XMUL (R, V, N)
          C
          C
          C-----------------------------------------------------------------
          C
          C
          C     This procedure uses a transformation matrix R and
          C     transforms the shape vector V having N columns
          C
          C
          C-----------------------------------------------------------------
          C
          C
                REAL        R(3,3), V(3,10), T(3), S
                INTEGER     ROW, COL
          C
                DO 100 COL = 1, N
 1                 DO 200 ROW = 1, 3
 2                    S = 0.0
 2                    DO 300 J = 1, 3
 3                       S = S + R(ROW,J) * V(J, COL)
 3            300     CONTINUE
 2                    T(ROW) = S
 2            200  CONTINUE
 1                 DO 400 L = 1, 3
 2                    V(L,COL) = T(L)
 2            400  CONTINUE
 1            100 CONTINUE
          C
                RETURN
                END
```

Name	Type	Offset	P	Class
COL	INTEGER*4	646		
J	INTEGER*4	662		
L	INTEGER*4	666		
N	INTEGER*4	8	*	
R	REAL	0	*	
ROW	INTEGER*4	654		
S	REAL	658		
T	REAL	634		
V	REAL	4	*	

```
420 $PAGE
```

```
C
C
      SUBROUTINE  ORGPOS (V, N)
C
C
C-----------------------------------------------------------------------
C
C
C     This procedure calculates the shape vector V of TOM_B
C     at the orgin. Only the top view is considered here.
C
C
C-----------------------------------------------------------------------
C
C
      REAL        V(3,10), XFORM(3,3), VO(3,10), W(2)
      REAL        V1(3,10)
      REAL        CC, DD, LL, RR, WW, CL, SDFORM(3,3)
      INTEGER     FLAG, CORNR(2,2), EF, EEF, PRTFG
C
      COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
      COMMON /MF/ XFORM, SDFORM, VO, V1
      COMMON /ME/ EF, EEF, PRTFG
C
C
      DO 100 K = 1, N
         V(3, K) = 1.0
100   CONTINUE
C
C     ***  set up shape matrix V
C
      CL    =  CC + LL
C                                    Corner  << A >>
      V(1, 1) = CC
      V(2, 1) =   0
C                                    Corner  << B >>
      V(1, 2) =  CC
      V(2, 2) = -WW
C                                    Corner  << C >>
      V(1, 3) = -RR
      V(2, 3) = -WW
C                                    Corner  << D >>
      V(1, 4) = -RR
      V(2, 4) =  WW
C                                    Corner  << E >>
      V(1, 5) =  CC
      V(2, 5) =  WW
C                                    Corner  << MM >>
      V(1, 6) =  CL
```

```
                V(2, 6) =   0
C                                       Corner  << F >>
                V(1, 7) =  CL + TT
                V(2, 7) = -DD
C                                       Corner  << G >>
                V(1, 8) =  CL - TT
                V(2, 8) = -DD
C                                       Corner  << H >>
                V(1, 9) =  CL - TT
                V(2, 9) =  DD
C                                       Corner  << I >>
                V(1,10) =  CL + TT
                V(2,10) =  DD
C
                RETURN
                END
```

Name	Type	Offset	P	Class	
CC	REAL	4		/MG	/
CL	REAL	702			
CORNR	INTEGER*4	678			
DD	REAL	8		/MG	/
EEF	INTEGER*4	4		/ME	/
EF	INTEGER*4	0		/ME	/
FLAG	INTEGER*4	0		/MG	/
K	INTEGER*4	694			
LL	REAL	12		/MG	/
N	INTEGER*4	4	*		
PRTFG	INTEGER*4	8		/ME	/
RR	REAL	16		/MG	/
SDFORM	REAL	36		/MF	/
TT	REAL	24		/MG	/
V	REAL	0	*		
V0	REAL	72		/MF	/
V1	REAL	192		/MF	/
W	REAL	670			
WW	REAL	20		/MG	/
XFORM	REAL	0		/MF	/

```
$PAGE
```

```
C
C
      SUBROUTINE  INITPL
C
C
C-----------------------------------------------------------------
C
C
C     This procedure initializes the system and calculates
C     all the necessary transformation matrices based on
C     the data obtained from SIZE.DAT
C
C
C-----------------------------------------------------------------
C
C
      REAL     V0(3,10),XFORM(3,3),SDFORM(3,3), W(2)
      REAL     CC, DD, LL, RR, WW, TT, V1(3,10)
C     REAL      CORNR(2,2), W(2)
      INTEGER  FLAG, EF, CORNR(2,2), EEF, CORNS(2,2), PRTFG
C
      COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
      COMMON /MF/ XFORM, SDFORM, V0, V1
      COMMON /ME/ EF, EEF, PRTFG
C
      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)
C
      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)
C
      READ (7,20) EF
      READ (7, 20) FLAG
      READ (7, 20) PRTFG
      CLOSE (7)
      FLG  = 1
C
C     ***  calculate floor shape
```

```
C
      JW = 30
      JL = 44
C
      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)
          V0(1,1) = J1 - JL
          V0(2,1) = L2 + 1
          V0(1,2) = J2 + JL
          V0(2,2) = L2 + 1
          V0(1,3) = -1000.0
          V0(2,3) = -1000.0
       END IF
C
      CALL  GRAFICS
      RETURN
10    FORMAT (F15.8)
20    FORMAT (I3)
      END
```

ame Type Offset P Class

D Line# 1 7 Microsoft FORTRAN77 V3.13 8/05/83

CC	REAL	4	/MG	/
ORNR	INTEGER*4	714		
ORNS	INTEGER*4	730		
DD	REAL	8	/MG	/
EF	INTEGER*4	4	/ME	/
F	INTEGER*4	0	/ME	/
FLAG	INTEGER*4	0	/MG	/
ELG	REAL	758		
	INTEGER*4	750		
J1	INTEGER*4	770		
J2	INTEGER*4	778		
J	INTEGER*4	786		
L	INTEGER*4	766		
JW	INTEGER*4	762		
	INTEGER*4	746		
1	INTEGER*4	774		
L2	INTEGER*4	782		
LL	REAL	12	/MG	/
RTFG	INTEGER*4	8	/ME	/
R	REAL	16	/MG	/
SDFORM	REAL	36	/MF	/
T	REAL	24	/MG	/
D	REAL	72	/MF	/
V1	REAL	192	/MF	/
	REAL	706		
W	REAL	20	/MG	/
XFORM	REAL	0	/MF	/

582 $PAGE

```
C
C
      SUBROUTINE  DRWFLR (V)
C
C
C-----------------------------------------------------------------------
C
C     This subroutine draws the floor portion of graphics
C
C
C-----------------------------------------------------------------------
C
C
      REAL          V(3,10)
      INTEGER       CT, X(10), Y(10)
C
      CT = 1
C
C     REPEAT
100      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
C     UNTIL  V(1,CT)  <   -100.0
C
      CALL  POLYGN (K, X, Y, 1)
      RETURN
      END
```

ame	Type	Offset	P	Class
	INTEGER*4	884		
	INTEGER*4	888		
	REAL	0	*	
	INTEGER*4	804		
	INTEGER*4	844		

```
$PAGE
```

```
C
C
      SUBROUTINE  DRAW (V, N, CLR)
C
C
C--------------------------------------------------------------------
C
C
C     This procedure actually draws the top view of TOM_B.
C
C     This procedure must be modified if different hardware
C     is used for the graphics display
C
C
C--------------------------------------------------------------------
C
C
      REAL        V(3, 10)
      INTEGER     X1, X2, Y1, Y2
      INTEGER     CLR
C
C     ***  draw mobile base
C
      CALL  RCT (V, 1, CLR)
C
C     ***  draw connecting line
C
      X1  = V(1,1)
      Y1  = V(2,1)
      X2  = V(1,6)
      Y2  = V(2,6)
      CALL  LINE (X1, Y1, X2, Y2, CLR)
C
C     ***  draw mocked-up
C
      CALL  RCT (V, 6, CLR)
C
      CALL  PURGE
      CALL  GRFRDY
      CALL  HOME
C
      RETURN
      END
```

Name	Type	Offset	P	Class
CLR	INTEGER*4	8	*	
N	INTEGER*4	4	*	
	REAL	0	*	

```
Line# 1      7
    INTEGER*4       892
    INTEGER*4       900
    INTEGER*4       896
    INTEGER*4       904

 656 $PAGE
```

```
C
C
      SUBROUTINE  CORDX (C, T, W)
C
C
C------------------------------------------------------------------
C
C
C     This procedure computes the necessary transformation
C     matrices from floor to screen coordinates
C
C
C------------------------------------------------------------------
C
C
      INTEGER     C(2,2)
      REAL        T(3,3), W(2)
C
C     ***  set up transformation matrix T
C
      T(1,3)  =  C(1,1)
      T(2,3)  =  C(2,2)
      T(3,3)  =  1.0
C
      T(1,1)  =  (C(2,1) - T(1,3)) / W(1)
      T(2,1)  =  (C(2,2) - T(2,3)) / W(1)
      T(3,1)  =  0.0
C
      T(1,2)  =  (C(1,1) - T(1,3)) / W(2)
      T(2,2)  =  (C(1,2) - T(2,3)) / W(2)
      T(3,2)  =  0.0
C
      RETURN
      END
```

ame	Type	Offset	P	Class
	INTEGER*4	0	*	
	REAL	4	*	
	REAL	8	*	

```
$PAGE
```

```
C
C     This is a graphics package for the TECMAR GRAPHICS MASTER board
C     written under Microsoft's FORTRAN 77.  To use this package, one
C     must include this package in the source file. A graphics master
C     must already be installed, or the software will hang.
C
C
      SUBROUTINE       PURGE
C
C     This procedure purges the graphics buffer and forces the board
C          to complete the drawing by closing the graphics channel.
C
      INTEGER          GRF
      CHARACTER        ESC
      COMMON    /GMBD/ GRF, ESC
      CLOSE (GRF)
      RETURN
      END
```

```
ne   Type         Offset P Class

SC   CHAR*1          4   /GMBD  /
F    INTEGER*4       0   /GMBD  /
```

```
$PAGE
```

```
D Line# 1      7                             Microsoft FORTRAN77 V3.13 8/05/83
C
C
              SUBROUTINE        GRFRDY
C
C             This procedure opens the graphics channel and sets it ready for
C                  communication
C
C
              INTEGER           GRF
              CHARACTER         ESC
              COMMON     /GMBD/ GRF, ESC
              OPEN (GRF, FILE = 'gm')
              RETURN
              END

Name    Type         Offset P Class

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

$PAGE
```

```
C
C
      SUBROUTINE       SETFB  (FG, BG)
C
C
C     This procedure sets the foreground color to FG and the background
C          color to BG. Both arguments must be of INTEGER type.
C
C
      INTEGER          GRF, FG, BG
      CHARACTER        ESC
      COMMON    /GMBD/ GRF, ESC
      WRITE (GRF, 10)  ESC, FG, BG
      RETURN
10    FORMAT (' ', A1, '[!', I2, ';', I2, 'c'\)
      END
```

```
ame   Type        Offset P Class

      INTEGER*4       4 *
SC    CHAR*1          4   /GMBD  /
G     INTEGER*4       0 *
F     INTEGER*4       0   /GMBD  /
```

```
742 $PAGE
```

D Line# 1 7 Microsoft FORTRAN77 V3.13 8/05/83

```
C
C
        SUBROUTINE      GRAFICS
C
C
        This procedure enters the GM graphics mode with a four-line text
            window at the bottom
C
C
        INTEGER         GRF
        CHARACTER       ESC
        COMMON    /GMBD/ GRF, ESC
C
        GRF = 9
        ESC = CHAR(27)
        CALL      GRFRDY
        WRITE (GRF, 10)  ESC
        WRITE (GRF, 20)  ESC
C       WRITE (GRF, 30)  ESC
        CALL      SETFB  (1, 0)
        CALL  HOME
        RETURN
C
 10     FORMAT (' ', A1, '[!0m'\)
 20     FORMAT (' ', A1, '[!640;352;2g'\)
 30     FORMAT (' ', A1, '[21;24r'\)
        END
```

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

```
$PAGE
```

```
Line# 1     7                              Microsoft FORTRAN77 V3.13 8/05/83
      C
      C
            SUBROUTINE  QUITGM
      C
      C
      C     This procedure gets one out of graphics mode and returns
      C     to text mode
      C
      C
            CHARACTER      CH, ESC
            INTEGER        GRF
            COMMON /GMBD/  GRF, ESC
      C
            CALL  HOME
            WRITE (GRF, 30)
            CALL  PURGE
            READ  (*, 10) CH
            CALL  GRFRDY
            CALL  TEXT
            RETURN
      10    FORMAT (A1)
      30    FORMAT ('Press <CR> to continue ... '\)
            END

me    Type         Offset P Class

      CHAR*1         1015
C     CHAR*1            4    /GMBD  /
      INTEGER*4         0    /GMBD  /

      $PAGE
```

```
 C
 C
          SUBROUTINE       TEXT
 C
 C
 C        This procedure returns the system to text mode
 C
 C
          INTEGER          GRF
          CHARACTER        ESC
          COMMON    /GMBD/ GRF, ESC
          WRITE (GRF, 10) ESC
          RETURN
 C
 10       FORMAT (' ', A1, '[!80;25;1a'\)
          END
```

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

```
 $PAGE
```

```
Line# 1      7                                    Microsoft FORTRAN77 V3.13 8/05/83
    C
    C
             SUBROUTINE       LINE  (X1, Y1, X2, Y2, COLOR)
    C
    C
    C        This procedure draws a line from (X1,Y1) to (X2,Y2) in COLOR
    C
    C
             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
```

```
ie    Type        Offset P Class

OLOR  INTEGER*4       16 *
      CHAR*1           4   /GMBD  /
      INTEGER*4        0   /GMBD  /
1     INTEGER*4        0 *
      INTEGER*4        8 *
      INTEGER*4        4 *
2     INTEGER*4       12 *
```

```
$PAGE
```

```
C
C
          SUBROUTINE       HIDELN (X1, Y1, X2, Y2, COLOR)
C
C
C         This procedure draws the line (X1,Y1) - (X2,Y2) but aborts drawi
C              before reaching target if a dot in a color other that BG is
C              encountered
C
C
          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'\)
          END
```

ame	Type	Offset	P	Class
COLOR	INTEGER*4	16	*	
SC	CHAR*1	4		/GMBD /
RF	INTEGER*4	0		/GMBD /
X1	INTEGER*4	0	*	
2	INTEGER*4	8	*	
1	INTEGER*4	4	*	
Y2	INTEGER*4	12	*	

```
$PAGE
```

```
C
C
      SUBROUTINE      POLYGN (N, X, Y, COLOR)
C
C
C     This procedure draws a closed polygon whose N vertices are store
C         in the arrays X and Y. The color to be used is COLOR
C
C
      INTEGER         GRF, X(N), Y(N), COLOR
      CHARACTER       ESC
      COMMON    /GMBD/ GRF, ESC
      WRITE (GRF, 10) ESC
      DO 100 K = 1, N
         WRITE (GRF, 20) X(K), Y(K)
100   CONTINUE
      WRITE (GRF, 30) COLOR
      RETURN
10    FORMAT (' ', A1, '[!'\)
20    FORMAT (     2(I3, ';')\)
30    FORMAT (      I3, 'p'\)
      END
```

```
ame   Type         Offset P Class

LOR   INTEGER*4       12 *
SC    CHAR*1           4    /GMBD  /
RF    INTEGER*4        0    /GMBD  /
      INTEGER*4     1160
      INTEGER*4        0 *
      INTEGER*4        4 *
      INTEGER*4        8 *
```

```
$PAGE
```

```
D Line# 1      7
      C
      C
              SUBROUTINE  HOME
      C
      C
      C       THIS SUBROUTINE HOMES THE CURSOR
      C
      C
              INTEGER       GRF
              CHARACTER     ESC
      C
              COMMON /GMBD/ GRF, ESC
      C
              WRITE (GRF, 10) ESC
              RETURN
        10    FORMAT (' ', A1, '[ 1;1 f'\)
              END
```

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

```
$PAGE
```

Name	Type	Size	Class
RDX			SUBROUTINE
W			SUBROUTINE
FLR			SUBROUTINE
BD		5	COMMON
AFIC			SUBROUTINE
RDY			SUBROUTINE
ELN			SUBROUTINE
ME			SUBROUTINE
TPL			SUBROUTINE
E			SUBROUTINE
		12	COMMON
		312	COMMON
		28	COMMON
VE			SUBROUTINE
THNG			SUBROUTINE
POS			SUBROUTINE
T			SUBROUTINE
LYGN			SUBROUTINE
GE			SUBROUTINE
TGM			SUBROUTINE
T			SUBROUTINE
DRAW			SUBROUTINE
FB			SUBROUTINE
DEVE			SUBROUTINE
INCOS			SUBROUTINE
XT			SUBROUTINE
PVEW			SUBROUTINE
MUL			SUBROUTINE

Pass One No Errors Detected
885 Source Lines

Appendix 4

OMV -- Data files

File : INITCON.DAT

This file contains all the needed initial conditions

```
0.0           POS(1) -- initial condition
0.0           POS(2) -- initial condition
0.0           POS(3) -- initial condition
0.00          VEL(1) -- initial condition
0.0           VEL(2) -- initial condition
0.0           VEL(3) -- initial condition
0.0           EUL(1) -- initial condition  ..  ROLL
0.0           EUL(2) -- initial condition  ..  PITCH
0.0           EUL(3) -- initial condition  ..  YAW
```

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

---

0.5588	CC	IN METERS
0.762	LL	IN METERS
11.668	AA	IN METERS
2.4384	HH	IN METERS
7048.37	IINV(1)	
3713.95	IINV(2)	
3713.95	IINV(3)	

File : HNDSGL.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
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

0.5588	CC : 22 inches
2.1336	DD : 84 inches
0.762	LL : 30 inches
1.016	RR : 40 inches
0.6096	WW : 24 inches
0.3048	TT : 12 inches
409	CORNR(1,1)
001	CORNR(1,2)
630	CORNR(2,1)
350	CORNR(2,2)
100	CORNR(1,1) SIDE VIEW
152	CORNR(1,2) SIDE VIEW
500	CORNR(2,1) SIDE VIEW
300	CORNR(2,2) SIDE VIEW
000	PLOT MODE : <> 0 MEANS NO CLEAR
000	VIEW : 0 = TOP VIEW, <> 0 = SIDE VIEW
001	PRTFG: 1-PLOT 2-PRINT 3-PLOT & PRINT

Appendix 5

OMV -- Source Listing

```
$LINESIZE:79
$PAGESIZE: 56
$TITLE: '<<<   O M V   >>>'
C
C                             OMV SIMULATION MODEL
C
C
C                                      by
C
C
C                                 Dr. W. Teoh
C
C                             U A H    Huntsville
C                                    1984
C
C-----------------------------------------------------------------------
C
C               This is a simplified version of a mathematical simulation
C           model of the OMV. In this model, the following simplifications
C           and assumptions are made :
C
C           1. The hand controllers provide  signals that are interpreted
C              as a force at the center of mass and/or a torque about the
C              center of mass to provide a rotation of constant angular
C              velocity.
C           2. The target vehicle is in a circular orbit; the altitude of
C              this orbit is inputted from the MDLPRM.DAT file.
C           3. Orbital mechanics is implemented, but smaller perturbation
C              effects are totally ignored.
C           4. Detailed placement of thrusters is not considered  (Please
C              see assumption 1. above)
C           5. Roll, pitch and yaw denote the  instantaneous  orientation
C              of the OMV.
C
C           A  14  component state vector is generated by this model, and
C           this state vector serves as input to the SVX module.
C
C
C-----------------------------------------------------------------------
C
            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
C
            COMMON  /MC/ III, MASS, CYCLE, MODE, STEP
            COMMON  /PC/ POS, VEL, EUL,  OMEGA
C
```

```
C     ***  system initialization
C
      IN = 2
      TIME = -1
      CALL  OMVMDL (IN)
      OPEN  (IN, FILE = 'HNDSGL.DAT')
C
C     ***  ***  Note : this invokes graphics routines, and can be
C                      elimiated if no graphics output.
C
      CALL  INITPL
C
C     ***  calculate the initial quaternions at the start of the
C     ***  simulation and read hand controller
C
      CALL  DETQ   (EUL, Q)
      CALL  HNDCTL (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
C
C     ***  main processing loop
C
C     WHILE  (FLAG = 0) DO
100       IF (FLAG .NE. 0) GOTO 900
C
C         ***  copy initial state into.work vectors and use these
C         ***  work vectors for solving the equations of motion
C
          CALL  MOTION (X, V, E, A, W, Q)
C
C         ***  update dynamic state
C
          CALL  MATCH  (E, X, V, EUL, POS, VEL, 3)
C
C         ***  calculate state vector and pass it on to the State
C         ***  Vector Transformation module
C
          CALL  STATE  (Q, S, W)
          CALL  SVX    (S, CMD, MODE)
          CALL  OUTPUT (A, W, X, V, E, Q, S, CMD, TIME)
C
C         ***  poll hand controller and get the next set of signals
C
          CALL  HNDCTL (IN, FLAG, A, W)
          GOTO 100
C     END WHILE
```

```
Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
      900    CONTINUE
             CLOSE (IN)
      C
      C      *** ***  This is also a call to the graphics package
      C
             CALL  QUITGM
      C
      C      ***  Grand exit, stage left
      C
             STOP
             END
```

ame	Type	Offset	P	Class
	REAL*8	242		
.J	INTEGER*4	266		
YCLE	REAL*8	32		/MC /
	REAL*8	74		
	REAL*8	48		/PC /
LAG	INTEGER*4	302		
II	REAL*8	0		/MC /
	INTEGER*4	294		
ASS	REAL*8	24		/MC /
ODE	INTEGER*4	40		/MC /
EGA	REAL*8	72		/PC /
S	REAL*8	0		/PC /
	REAL*8	98		
	REAL*8	130		
EP	INTEGER*4	44		/MC /
IME	INTEGER*4	298		
	REAL*8	26		
L	REAL*8	24		/PC /
	REAL*8	50		
	REAL*8	2		

```
$PAGE
```

```
C
C
      SUBROUTINE  OMVMDL (IN)
C
C-----------------------------------------------------------------------
C
C     This procedure obtains the necessary parameters of the OMV
C     by reading them from a disk file called  MDLPRM.DAT  after
C     getting the initial state of the  OMV  (from a file called
C     INITCON.DAT
C
C-----------------------------------------------------------------------
C
      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
C
      COMMON  /DC/  ACC,    WWB
      COMMON  /MC/  III,    MASS, CYCLE, MODE, STEP
      COMMON  /PC/  POS,    VEL, EUL,  OMEGA
C
C     ***  get initial conditions of the OMV
C
      OPEN  (IN, FILE = 'INITCON.DAT')
      CALL  VECTOR (IN, POS, 3)
      CALL  VECTOR (IN, VEL, 3)
      CALL  VECTOR (IN, EUL, 3)
      CLOSE (IN)
C
C     ***  read acceleration, angular rates and
C     ***  principal moments of inertia in body frame
C
      OPEN  (IN, FILE = 'MDLPRM.DAT')
      CALL  VECTOR (IN, ACC, 3)
      CALL  VECTOR (IN, WWB, 3)
      CALL  VECTOR (IN, III, 3)
C
C     ***  read mass characteristics & other parameters
C
      READ  (IN, 10)  MASS
      READ  (IN, 10)  CYCLE
      READ  (IN, 20)  MODE
      READ  (IN, 30)  STEP
      READ  (IN, 10)  ORBIT
      CLOSE (IN)
C
C     ***  calculate orbital frequency
C
```

```
Line# 1      7                                        Microsoft FORTRAN77 V3.13 8/05/83
             CALL  ANGFRE (ORBIT, OMEGA)
      C
      C
             RETURN
      10     FORMAT (F15.8)
      20     FORMAT (I1)
      30     FORMAT (I2)
             END
```

ame	Type	Offset	P	Class	
C	REAL*8	0		/DC	/
.CLE	REAL*8	32		/MC	/
UL	REAL*8	48		/PC	/
T	REAL*8	0		/MC	/
	INTEGER*4	0	*		
NV	REAL*8	306			
ASS	REAL*8	24		/MC	/
DE	INTEGER*4	40		/MC	/
MEGA	REAL*8	72		/PC	/
RBIT	REAL*8	330			
S	REAL*8	0		/PC	/
.EP	INTEGER*4	44		/MC	/
EL	REAL*8	24		/PC	/
B	REAL*8	24		/DC	/

```
$PAGE
```

```
C
C
      SUBROUTINE  ANGFRE(ORB, W)
C
C
C-----------------------------------------------------------------------
C
C     This  procedure calculates the orbital angular frequency
C     at a given altitude.  In this calculation,  the altitude
C     must be given in kilo-meters. This is necessary in order
C     for the calculations  to be carried out  without lossing
C     precision. The angular frequency W is in rad/second
C
C-----------------------------------------------------------------------
C
      REAL * 8  ORB
      REAL * 8  ALT, R3, W
C
      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
DSQRT                         INTRINSIC
ORB     REAL*8            0 *
R3      REAL*8          366
W       REAL*8            4 *
```

```
$PAGE
```

```
C
C
      SUBROUTINE  VECTOR (M, A, N)
C
C
C-----------------------------------------------------------------------
C
C
C     This procedure reads a vector A of N elements from input
C     unit M
C
C-----------------------------------------------------------------------
C
      INTEGER     M, N
      REAL * 8    A(N)
C
      DO 100  K = 1, N
          READ (M, 10) A(K)
  100 CONTINUE
      RETURN
   10 FORMAT (F15.8)
      END
```

```
ne   Type        Offset P Class

     REAL*8           4 *
     INTEGER*4      374
     INTEGER*4        0 *
     INTEGER*4        8 *
```

```
$PAGE
```

```
C
C
      SUBROUTINE  HNDCTL (IN, FLAG, A, W)
C
C--------------------------------------------------------------------------
C
C         Simulates hand controllers input by reading from a file
C     (called HNDSGL.DAT 12) integers to simulate a 12 bit output
C     of the hand controllers. Bit assignments are as follows :
C
C         bit          meaning (direction in body frame)
C         ===          ================================
C          1           Accelerate along   +1   axis
C          2           Accelerate along   -1   axis
C          3           Accelerate along   +2   axis
C          4           Accelerate along   -2   axis
C          5           Accelerate along   +3   axis
C          6           Accelerate along   -3   axis
C          7           Rotate about       +1   axis
C          8           Rotate about       -1   axis
C          9           Rotate about       +2   axis
C         10           Rotate about       -2   axis
C         11           Rotate about       +3   axis
C         12           Rotate about       -3   axis
C
C--------------------------------------------------------------------------
C
      REAL * 8          ACC(3), WWB(3)
      REAL * 8          A(3),   W(3)
      INTEGER           SL(6),  SA(6),  FLAG
      COMMON   /DC/     ACC,    WWB
C
      FLAG = 0
      READ (IN, 10, END = 90, ERR = 90) SL, SA
C
C     ***  no error, generate matrices A and W
C
      CALL  FUDGE (A, ACC, SL)
      CALL  FUDGE (W, WWB, SA)
      RETURN
90    CONTINUE
C
C     ***  error condition
C
      FLAG = 1
      RETURN
10    FORMAT (20I1)
      END
```

e	Type	Offset	P	Class
	REAL*8	8	*	
	REAL*8	0		/DC /
AG	INTEGER*4	4	*	
L	INTEGER*4	0	*	
	INTEGER*4	414		
	INTEGER*4	390		
	REAL*8	12	*	
	REAL*8	24		/DC /

264 $PAGE

```
D Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
      C
      C
            SUBROUTINE  FUDGE  (A, ACC, SL)
      C
      C-----------------------------------------------------------------------
      C
      C     ***  Sets appropriate components based on SL
      C
      C-----------------------------------------------------------------------
      C
            INTEGER     SL(6),  T, K, J
            REAL * 8    ACC(3), A(3), X
            DO 100 K = 1, 6, 2
1                J  =  (K + 1) / 2
1                X  =  0.0
1                T  =  SL(K) + SL(K+1)
1                IF (T .EQ. 1) THEN
1                    X  =  ACC(J)
1                    IF (SL(K) .EQ. 0) X = -X
1                END IF
1                A(J) = X
1     100   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
```

```
C
C
      SUBROUTINE  STATE (Q, S, W)
C
C----------------------------------------------------------------------
C
C         This procedure uses the dynamic quantities of the OMV and
C     constructs a  State Vector  of the OMV.  The 14 components of
C     this State Vector S are defined as follows
C
C         Components            Meaning
C       ==============   ======================================
C       S(1)  ..  S(3)   Relative displacement between  OMV and
C                        target
C       S(4)  ..  S(6)   Relative velocity between OMV & target
C       S(7)  ..  S(9)   Angular momentum vector of OMV in  LVF
C       S(10) .. S(13)   Attitude quaternions expressed in body
C                        frame, and
C       S(14)            Instantaneous mass,  assumed  constant
C                        throughout the simulation.
C
C
C----------------------------------------------------------------------
C
      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
C
      COMMON     /MC/   III, MASS, CYCLE, MODE, STEP
      COMMON     /PC/   POS, VEL, EUL,  OMEGA
C
C     ***  calculate angular momentum in body frame
C
      CALL  DOTPRD  (III, W, LB, 3)
C
C     ***  transforms it to local vertical frame
C
      CALL  DCSINV (Q, B)
      CALL  DMUL   (B, LB, LL, 3)
C
C     ***  Build state vector
C
      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)
```

D Line# 1 7 Microsoft FORTRAN77 V3.13 8/05/83

```
      S(14)  =  MASS
C
      RETURN
      END
```

Name	Type	Offset	P	Class
ı	REAL*8	642		
B	REAL*8	570		
CYCLE	REAL*8	32		/MC /
UL	REAL*8	48		/PC /
II	REAL*8	0		/MC /
L	REAL*8	546		
B	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 /
	REAL*8	0	*	
Q	REAL*8	466		
S	REAL*8	4	*	
STEP	INTEGER*4	44		/MC /
EL	REAL*8	24		/PC /
W	REAL*8	8	*	

```
$PAGE
```

```
C
C
      SUBROUTINE  PUT (N, S, A, M)
C
C-----------------------------------------------------------------------
C
C     ***  The procedure copies a vector A into a larger one S
C          starting at the N-th element of S
C
C-----------------------------------------------------------------------
C
      REAL * 8    S(14)
      REAL * 8    A(M)
C
      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	*	

```
$PAGE
```

```
D Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
          C
          C
                SUBROUTINE  DOTPRD (A, B, C, N)
          C
          C-----------------------------------------------------------------------
          C
          C     ***  This procedure calculates a vector C from two other
          C          vectors A and B such that
          C                C(I)  =  A(I) * B(I)
          C          for all i = 1 to N
          C
          C-----------------------------------------------------------------------
          C
                REAL * 8    A(N), B(N), C(N)
                DO 100 K = 1, N
1                   C(K) = A(K) * B(K)
1         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	*	

```
          $PAGE
```

```
C
C
      SUBROUTINE  DETQ (E, Q)
C
C-----------------------------------------------------------------
C
C     ***  calculates quaternions from the Euler angles
C          using expression given by Zack.
C
C-----------------------------------------------------------------
C
      REAL * 8    E(3), Q(4)
      REAL * 8    C1, S1, C2, S2, C3, S3, THETA
C
      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)
C
      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
C
      RETURN
      END
```

ame	Type	Offset	P	Class
1	REAL*8	750		
2	REAL*8	766		
3	REAL*8	782		
	REAL*8	0	*	
	REAL*8	4	*	
1	REAL*8	742		
2	REAL*8	758		
3	REAL*8	774		
HETA	REAL*8	734		

```
$PAGE
```

```
C
C
      SUBROUTINE SINCOS (THETA, S, C)
C
C
C-------------------------------------------------------------------
C
C     ***  this procedure returns the sine and cosine of an
C          angle THETA.
C
C-------------------------------------------------------------------
C
      REAL * 8    THETA, S, C, A
C
      C = DCOS(THETA)
      S = DSIN(THETA)
      RETURN
      END
```

Name	Type	Offset	P	Class
A	REAL*8	*****		
C	REAL*8	8	*	
DCOS				INTRINSIC
DSIN				INTRINSIC
S	REAL*8	4	*	
THETA	REAL*8	0	*	

```
$PAGE
```

```
C
C
      SUBROUTINE  MOTION (X, V, E, A, W, Q)
C
C-----------------------------------------------------------------------
C
C     ***  This procedure solves the equation of motion
C
C-----------------------------------------------------------------------
C
      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
C
      COMMON   /MC/     III, MASS, CYCLE, MODE, STEP
      COMMON   /PC/     POS, VEL, EUL, OMEGA
C
      H = CYCLE / FLOAT(STEP)
      N = STEP
      PI = 355.0 / 113.0
      TWO= PI * 2.0
C
C     ***  Divide 1 major cycle into N equal subintervals and
C     ***  determine the OMV state for each interval
C
      DO 100 KK = 1, N
C
C        ***   Update orientation
C
         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
C
C     ***  Calculate quaternion for this rotation, and transform
C     ***  it to local vertical frame with respect to initial frame
C
         CALL  DETQ( WW, QQ)
         CALL  UPDQ (Q, QQ)
C
C        ***  from the direction cosine matrix, calculate the
C        ***  acceleration vector in LVF and store it in the
C        ***  acceleration matrix AA
C
         CALL  DCSINV (Q, CIN)
```

```
D Line# 1      7                                Microsoft FORTRAN77 V3.13 8/05/83
1   482           CALL  DMUL   (CIN, A, B, 3)
1   483           CALL  STORE (B, AA, KK)
1   484  100   CONTINUE
    485 C
    486 C      ***  Solve the equation of motion using the Adam-Brashford
    487 C      ***  method
    488 C
    489        CALL  SOLVE (X, V, AA, N, H, OMEGA)
    490 C
    491        RETURN
    492        END

Name    Type        Offset P Class

A       REAL*8          12 *
AA      REAL*8         990
B       REAL*8        1230
C       REAL*8         918
CIN     REAL*8         846
CYCLE   REAL*8          32   /MC     /
E       REAL*8           8 *
EUL     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 *
QQ      REAL*8         814
STEP    INTEGER*4       44   /MC     /
TWO     REAL*8        1270
V       REAL*8           4 *
VEL     REAL*8          24   /PC     /
W       REAL*8          16 *
WW      REAL*8         790
X       REAL*8           0 *


    493 $PAGE
```

```
Line# 1      7                          Microsoft FORTRAN77 V3.13 8/05/83
      C
      C
            SUBROUTINE  MATCH (A, B, C, P, Q, R, N)
      C
      C
      C-------------------------------------------------------------------
      C
      C     ***  This procedure makes an exact duplicate B of a
      C          vector A of N elements
      C
      C-------------------------------------------------------------------
      C
            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
```

```
me   Type        Offset P Class

     REAL*8           0 *
     REAL*8           4 *
     REAL*8           8 *
     INTEGER*4     1290
     INTEGER*4       24 *
     REAL*8          12 *
     REAL*8          16 *
     REAL*8          20 *
```

```
$PAGE
```

```
D Line# 1      7                               Microsoft FORTRAN77 V3.13 8/05/83
       C
       C
                SUBROUTINE  STORE (AAA, AA, K)
       C
       C
       C-----------------------------------------------------------------------
       C
       C            This procedure takes an  instantaneous  acceleration vector
       C        AAA and stores it in the acceleration matrix AA which is needed
       C        by the numerical integration process
       C
       C-----------------------------------------------------------------------
       C
                REAL * 8    AA(3, 10)
                REAL * 8    AAA(3)
                DO 100 J = 1, 3
 1                  AA(J,K) = AAA(J)
 1          100 CONTINUE
                RETURN
                END

Name    Type         Offset P Class

AA      REAL*8            4 *
AAA     REAL*8            0 *
J       INTEGER*4      1294
K       INTEGER*4         8 *

       $PAGE
```

```
C
C
      SUBROUTINE SOLVE(X,V,A,N,H,W)
C
C-----------------------------------------------------------------------
C
C          This subroutine  produces the numerical solution to the
C      system of equations of motion using a 3 step Adam-Brashford
C      method.
C
C-----------------------------------------------------------------------
C
      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./
C
C     ***  pack user supplied nonhomomgenous part of DE
C     ***  into the higher part of AA
C
      DO 10 I = 1,10
         DO 10 K = 1,3
            AA(K,I+3) = A(K,I)
10    CONTINUE
C
C     ***  if this is the first call to solve (FLAG = T), then
C     ***  it is necessary to initialize some parameters
C
      IF (FLAG) THEN
         CALL INNIT(X,V,W,H)
         FLAG = .FALSE.
      END IF
C
C     ***  use the Adams-Brashford 3-step method to advance the
C     ***  solution H time units.  Place the solution back into
C     ***  X and V.
C
      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))
100   CONTINUE
      X(1) = U(1,N+3)
      V(1) = U(2,N+3)
      X(2) = U(3,N+3)
      V(2) = U(4,N+3)
      X(3) = U(5,N+3)
      V(3) = U(6,N+3)
```

```
D Line# 1      7                               Microsoft FORTRAN77 V3.13 8/05/83
        C
        C       ***  reset U and AA for the next call to SOLVE
        C
                DO 200 J = 1,6
 1                 DO 200 I = 1,3
 2                    U(J,I) = U(J,N+I)
 2                    IF (J .LE. 3) AA(I,J) = AA(I,N+J)
 2         200  CONTINUE
                RETURN
                END

Name    Type        Offset P Class

A       REAL*8           8 *
AA      REAL*8           0   /BLOCK /
F       REAL*8               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 *

        $PAGE
```

```
C
C
      SUBROUTINE INNIT(X,V,W,H)
C
C
C-----------------------------------------------------------------
C
C          This procedure initializes all the necessary  parameters
C     before solving the system of ordinary differential equations.
C     This procedure is invoked only once.
C
C-----------------------------------------------------------------
C
      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
C
      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
C        CONTINUE
100   CONTINUE
C
      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*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
300   CONTINUE
      RETURN
      END
```

Name	Type	Offset	P	Class
AA	REAL*8	0		/BLOCK /

```
CWT     REAL*8          1362
)BLE                           INTRINSIC
DCOS                           INTRINSIC
DSIN                           INTRINSIC
I       REAL              12 *
ID12    REAL*8           960   /BLOCK /
I       INTEGER*4       1350
J       INTEGER*4       1346
(       INTEGER*4       1342
SWT     REAL*8          1370
T       REAL*8          1354
J       REAL*8           312   /BLOCK /
/       REAL*8             4 *
W       REAL*8             8 *
VX2     REAL*8           936   /BLOCK /
VXW     REAL*8           944   /BLOCK /
WXWX3   REAL*8           952   /BLOCK /
X       REAL*8             0 *

    641 $PAGE
```

```
D Line# 1      7                         Microsoft FORTRAN77 V3.13 8/05/83
      C
      C
            FUNCTION F(J,I)
      C
      C
      C-----------------------------------------------------------------------
      C
      C
            REAL*8 AA(3,13),U(6,13),WX2,WXW,WXWX3,HD12,F
            COMMON /BLOCK/ AA,U,WX2,WXW,WXWX3,HD12
      C
            GO TO (10,20,30,40,50,60), J
      10    CONTINUE
               F = U(2,I)
               RETURN
      20    CONTINUE
               F = -WX2*U(6,I) + AA(1,I)
               RETURN
      30    CONTINUE
               F = U(4,I)
               RETURN
      40    CONTINUE
               F = -WXW*U(3,I) + AA(2,I)
               RETURN
      50    CONTINUE
               F = U(6,I)
               RETURN
      60    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
```

```
C
C
      SUBROUTINE  OUTPUT (A, W, X, V, E, Q, S, CMD, TIME)
C
C
C-----------------------------------------------------------------------
C
C          This is the output section of the system. Any further
C     modification of the output requirements of this model must
C     be done in this procedure.  In particular, if no output to
C     the CRT or printer is needed,  it is recommended that  C's
C     be inserted into column 1 of all the WRITE statments.  The
C     simulation clock is updated in this procedure.
C
C-----------------------------------------------------------------------
C
      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
C
      COMMON /ME/ EF, EEF, PRTFG
C
      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
C        WRITE (4, 10) A, W
         WRITE (4, 20) X, V
         WRITE (4, 30) E, W
         WRITE (4, 40) S
         WRITE (4, 50) CMD
         WRITE (4, 90)
         CLOSE (4)
  100 IF (PRTFG .NE. 2) CALL  PLOT (CMD)
C
      RETURN
   10 FORMAT (' A, W =', 3F10.6, 3X, 3F10.6)
   12 FORMAT (' ', 7I10)
   15 FORMAT (' TIME =', I6, '  Seconds')
   20 FORMAT (' X, V =', 3F10.6, 3X, 3F10.6)
   30 FORMAT (' E, W =', 3F10.6, 3X, 3F10.6/)
   40 FORMAT ('    S =', 3F10.6, 3X, 3F10.6/
     1          '       ', 3F10.3/
     2          '       ', 4F10.6, 3X,F10.3/)
   50 FORMAT ('  CMD =', 7I10)
   90 FORMAT (1H0)
      END
```

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	*	
T	INTEGER*4	1378		
TIME	INTEGER*4	32	*	
V	REAL*8	12	*	
W	REAL*8	4	*	
X	REAL*8	8	*	

```
723 $PAGE
```

```
C
C
      SUBROUTINE  DMUL (A, B, C, N)
C
C----------------------------------------------------------------------
C
C       This procedure performs a matrix multiplication of an NxN
C     matrix A to an N-element column matrix B to yield an N-element
C     column matrix C
C
C----------------------------------------------------------------------
C
      REAL * 8   A(N,N), B(N), C(N), S
C
      DO 100 I = 1, N
         S = 0.0
         DO 200 J = 1, N
            S = S + A(I,J) * B(J)
200      CONTINUE
         C(I) = S
100   CONTINUE
      RETURN
      END
```

ie	Type	Offset	P	Class
	REAL*8	0	*	
	REAL*8	4	*	
	REAL*8	8	*	
	INTEGER*4	1714		
	INTEGER*4	1730		
	INTEGER*4	12	*	
	REAL*8	1722		

```
$PAGE
```

```
C
C
      SUBROUTINE  UPDQ (Q, QQ)
C
C
C---------------------------------------------------------------------
C
C     This subroutine uses the previous quaternion and generates
C     the present quaternions with restpect to the local vertical
C     frame LVF. Quaternion algebra is used to deduce the needed
C     computation before hand to simplify the algorithm
C
C
C---------------------------------------------------------------------
C
C
      REAL * 8    Q(4), QQ(4), Q1, Q2, Q3, Q4
C
      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)
C
      Q(1) = Q1
      Q(2) = Q2
      Q(3) = Q3
      Q(4) = Q4
      RETURN
      END
```

```
Name   Type      Offset P Class

Q      REAL*8         0 *
Q1     REAL*8      1738
Q2     REAL*8      1746
Q3     REAL*8      1754
Q4     REAL*8      1762
QQ     REAL*8         4 *
```

```
$PAGE
```

```
C
C
      SUBROUTINE  DCSINV (Q, C)
C
C
C-----------------------------------------------------------------------
C
C     This subroutine takes the attitude quaternion Q and returns
C     the transpose of the direction cosine matrix
C
C-----------------------------------------------------------------------
C
C
      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
C
      Q1  = Q(1)
      Q2  = Q(2)
      Q3  = Q(3)
      Q4  = Q(4)
C
      Q11 = Q1 * Q1
      Q22 = Q2 * Q2
      Q33 = Q3 * Q3
      Q44 = Q4 * Q4
C
      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
      C(1,1) =  Q11 - Q22 - Q33 + Q44
      C(2,2) = -Q11 + Q22 - Q33 + Q44
      C(3,3) = -Q11 - Q22 + Q33 + Q44
C
      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
```

D Line# 1   7   Microsoft FORTRAN77 V3.13 8/05/83

Name	Type	Offset	P	Class
C	REAL*8	4	*	
Q	REAL*8	0	*	
Q1	REAL*8	1770		
Q11	REAL*8	1802		
Q12	REAL*8	1834		
Q13	REAL*8	1842		
Q14	REAL*8	1858		
Q2	REAL*8	1778		
Q22	REAL*8	1810		
Q23	REAL*8	1850		
Q24	REAL*8	1866		
Q3	REAL*8	1786		
Q33	REAL*8	1818		
Q34	REAL*8	1874		
Q4	REAL*8	1794		
Q44	REAL*8	1826		

826 $PAGE

ne	Type	Size	Class
GFRE			SUBROUTINE
OCK		968	COMMON
		48	COMMON
CSINV			SUBROUTINE
TQ			SUBROUTINE
JL			SUBROUTINE
OTPRD			SUBROUTINE
	REAL*8		FUNCTION
OGE			SUBROUTINE
NDCTL			SUBROUTINE
ITPL			SUBROUTINE
NIT			SUBROUTINE
AIN			PROGRAM
ATCH			SUBROUTINE
		48	COMMON
		12	COMMON
OTION			SUBROUTINE
VMDL			SUBROUTINE
TPUT			SUBROUTINE
C		80	COMMON
OT			SUBROUTINE
T			SUBROUTINE
UITGM			SUBROUTINE
INCOS			SUBROUTINE
LVE			SUBROUTINE
ATE			SUBROUTINE
TORE			SUBROUTINE
X			SUBROUTINE
DQ			SUBROUTINE
ECTOR			SUBROUTINE

Pass One No Errors Detected
826 Source Lines

Appendix 6

ADAM -- Source Listing

```
$PAGESIZE : 56
$TITLE: '          << A D A M >>'
C
C
C
C                  Program  :  A D A M
C
C
C                            by
C
C                      Dr. W. Teoh
C
C
C-----------------------------------------------------------------------
C
C           This program uses the Adam Brashforth method to solve
C           the equation of motion (homogeneous case) numerically
C           and compares the solution with the analytical results
C           such that both outputs are printed.
C
C
C-----------------------------------------------------------------------
C
C
        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/
C
C
C
        WRITE (*, 30)
        READ  (*,32)  W
C
C       get initial conditions
C
        CALL  GETINT (XO, VO, 3)
C
        DO 100 K = 1, 3
           X(K)  = XO(K)
           V(K)  = VO(K)
 100    CONTINUE
C
        DO 10 I = 1,36000
           T = 0.1*I
C
C          ***  calculate the analytical solution
C
           CALL EXACT(T,XE,VE,W,XO,VO)
```

```
D Line# 1      7                        Microsoft FORTRAN77 V3.13 8/05/83
1      C
1      C          ***  now get the numerical solution
1      C
1                 CALL SOLVE(X,V,A,N,H,W)
1      C
1      C          ***  output every 60 seconds
1      C
1                 JJ = (I / 600) * 600
1                 IF (JJ .EQ. I) THEN
1                    WRITE(*,20) T,XE,VE
1                    WRITE(*,20) T,X,V
1                    WRITE (*, 22)
1                 END IF
1      10      CONTINUE
       C
       20      FORMAT (F7.1, 6F12.6)
       30      FORMAT (' ORBITAL RATE  '\)
       22      FORMAT (1H )
       32      FORMAT (F15.8)
               STOP
               END
```

Name	Type	Offset	P	Class
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		
VO	REAL*8	122		
VE	REAL*8	26		
W	REAL*8	394		
X	REAL*8	50		
XO	REAL*8	74		
XE	REAL*8	2		

```
$PAGE
```

```
C
C
      SUBROUTINE EXACT(T,XE,VE,W,X,V)
C
C
C------------------------------------------------------------------
C
C     ** This subroutine calculates the exact solution
C        of the homogeneous ODEs
C
C------------------------------------------------------------------
C
C
C
      REAL*8 XE(3),VE(3),CWT,SWT,W, WT, X(3), V(3)
C
      WT    = W * T
      SWT   = DSIN(WT)
      CWT   = DCOS(WT)
C
      XE(1) = X(1) + (4 * SWT - 3*WT)*V(1)/W + 6*(SWT - WT)*X(3)
     1        + 2 * (CWT - 1) * V(3) /W
      XE(2) = CWT* X(2) + SWT * V(2) / W
      XE(3) = 2 * (1 - CWT) * V(1) / W + (4 - 3 * CWT) * X(3)
     1        - SWT * V(3) / W
      VE(1) = (4 * CWT -3) * V(1) + 6 * W * (CWT -1) * X(3)
     1        - 2 * SWT * V(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
```

```
me    Type       Offset P Class

VT    REAL*8        488
OS                          INTRINSIC
SIN                         INTRINSIC
WT    REAL*8        480
      REAL            0 *
.     REAL*8         20 *
/E    REAL*8          8 *
      REAL*8         12 *
T     REAL*8        472
X     REAL*8         16 *
E     REAL*8          4 *

$PAGE
```

```
  C
  C
             SUBROUTINE SOLVE(X,V,A,N,H,W)
  C
  C
  C-------------------------------------------------------------------
  C
  C
             **  This subroutine produces the numerical solution
                 to the system of equations of motion
  C
  C
  C-------------------------------------------------------------------
  C
  C
  C
             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./
  C
  C
                pack user supplied nonhomogeneous part of DE into
                the higher part of AA
  C
             DO 10 I = 1,10
1               DO 10 K = 1,3
2                  AA(K,I+3) = A(K,I)
2   10       CONTINUE
  C
                if this is the first call to solve (FLAG = T), then
                initialize
  C
             IF (FLAG) THEN
                CALL INNIT(X,V,W,H)
                FLAG = .FALSE.
             END IF
  C
                use the Adam-Brashford 3-step method to advance
                the solution h time units.  Place the solution
                back into X and V.
  C
             DO 100 I = 4,N+3
1               DO 100 J  = 1,6
2                  U(J,I) = U(J,I-1) +
2        +                  HD12*(23*F(J,I-1)-16*F(J,I-2)+5*F(J,I-3))
2   100      CONTINUE
             X(1) = U(1,N+3)
```

```
      V(1) = U(2,N+3)
      X(2) = U(3,N+3)
      V(2) = U(4,N+3)
      X(3) = U(5,N+3)
      V(3) = U(6,N+3)
C
C        reset U and AA for the next call to SOLVE
C
      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)
  200  CONTINUE
C     DO 300 I = 1,3
C        DO 300 K = 1,3
C           AA(K,I) = AA(K,N+I)
C300   CONTINUE
      RETURN
      END
```

ame	Type	Offset	P	Class
	REAL*8	8	*	
A	REAL*8	0		/BLOCK /
	REAL*8			FUNCTION
AG	LOGICAL*4	496		
	REAL	16	*	
D12	REAL*8	960		/BLOCK /
	INTEGER*4	500		
	INTEGER*4	512		
	INTEGER*4	504		
	INTEGER*4	12	*	
	REAL*8	312		/BLOCK /
	REAL*8	4	*	
	REAL*8	20	*	
2	REAL*8	936		/BLOCK /
XW	REAL*8	944		/BLOCK /
XWX3	REAL*8	952		/BLOCK /
	REAL*8	0	*	

```
$PAGE
```

```
D Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
          C
          C
                SUBROUTINE INNIT(X,V,W,H)
          C
          C
          C--------------------------------------------------------------------
          C
          C
          C       This is the initialization routine which is called only once
          C
          C
          C--------------------------------------------------------------------
          C
          C
                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
          C
                DO 100 I = 1,3
 1                 DO 100 J = 1,6
 2                    AA(J,I) = 0.0
 2        100   CONTINUE
                DO 200 K = 1,3
 1                 U(2*K-1,3) = X(K)
 1                 U(2*K  ,3) = V(K)
 1        200   CONTINUE
          C
                DO 300 I  = 1,2
 1                 T      = H*(I-3)
 1                 CWT    = DCOS(W*T)
 1                 SWT    = DSIN(W*T)
 1                 U(1,I) = X(1) + V(1)*(4*SWT-3*W*T)/W +
 1            +             6*X(3)*(SWT-W*T) + 2*V(3)*(CWT-1.0)/W
 1                 U(2,I) = V(1)*(4*CWT-3.0) + 6*W*X(3)*(CWT-1.0) -
 1            +             2*V(3)*SWT
 1                 U(3,I) = X(2)*CWT + V(2)*SWT/W
 1                 U(4,I) = -X(2)*W*SWT + V(2)*CWT
 1                 U(5,I) = 2*V(1)*(1.0-CWT)/W + X(3)*(4.0-3*CWT) +
 1            +             V(3)*SWT/W
 1                 U(6,I) = 2*V(1)*SWT + 3*X(3)*W*SWT + V(3)*CWT
 1        300   CONTINUE
                RETURN
                END
```

e	Type	Offset	P	Class
[illegible]	REAL*8	0		/BLOCK /
[illegible]	REAL*8	560		
E				INTRINSIC
OS				INTRINSIC
IN				INTRINSIC
	REAL	12	*	
12	REAL*8	960		/BLOCK /
	INTEGER*4	540		
	INTEGER*4	544		
	INTEGER*4	548		
NT	REAL*8	568		
	REAL*8	552		
	REAL*8	312		/BLOCK /
	REAL*8	4	*	
	REAL*8	8	*	
2	REAL*8	936		/BLOCK /
N	REAL*8	944		/BLOCK /
XWX3	REAL*8	952		/BLOCK /
	REAL*8	0	*	

220 $PAGE

```
C
C
      FUNCTION F(J,I)
C
C
C-----------------------------------------------------------------------
C
C
C     User supplied function
C
C
C-----------------------------------------------------------------------
C
C
      REAL*8 AA(3,13),U(6,13),WX2,WXW,WXWX3,HD12,F
      COMMON /BLOCK/ AA,U,WX2,WXW,WXWX3,HD12
C
      GO TO (10,20,30,40,50,60), J
10    CONTINUE
         F = U(2,I)
         RETURN
20    CONTINUE
         F = -WX2*U(6,I) + AA(1,I)
         RETURN
30    CONTINUE
         F = U(4,I)
         RETURN
40    CONTINUE
         F = -WXW*U(3,I) + AA(2,I)
         RETURN
50    CONTINUE
         F = U(6,I)
         RETURN
60    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
```

```
C
C
      SUBROUTINE  GETINT (X, V, N)
C
C
C-------------------------------------------------------------------
C
C
C     get initial conditions
C
C
C-------------------------------------------------------------------
C
C
      REAL * 8    X(N), V(N)
C
      OPEN (1, FILE = 'INITCON.DAT')
      DO 100 K = 1, N
         READ (1, 10) X(K)
100   CONTINUE
C
      DO 200 K = 1, N
         READ (1,10) V(K)
200   CONTINUE
      RETURN
10    FORMAT (F15.6)
      END
```

me	Type	Offset	P	Class
:	INTEGER*4	576		
	INTEGER*4	8	*	
	REAL*8	4	*	
:	REAL*8	0	*	

Name	Type	Size	Class
.OCK		968	COMMON
:XACT			SUBROUTINE
	REAL*8		FUNCTION
:TINT			SUBROUTINE
INNIT			SUBROUTINE
MAIN			PROGRAM
OLVE			SUBROUTINE

```
D Line# 1      7                          Microsoft FORTRAN77 V3.13 8/05/83
 Pass One      No Errors Detected
              285 Source Lines
```

End of Document