Teleoperator and Robotics System Analysis
Final Report

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

Prepared by
Dr. William Teoh*
Kenneth E. Johnson Research Center
University of Alabama in Huntsville
Huntsville, AL 35899

Date prepared: Sept. 30, 1987
Contract # NAS8-35670

*Present address: SPARTA, Inc., 4901 Corporate Drive, Huntsville, AL 35805
### FINANCIAL STATUS REPORT

**CONTRACT** 5-31257

<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Cumulative Costs incurred as of</td>
<td>October 15, 1984</td>
</tr>
<tr>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Estimate of cost to complete</td>
<td>$89,929.00</td>
</tr>
<tr>
<td>Estimated Percentage of Physical Completion</td>
<td>100%</td>
</tr>
</tbody>
</table>

Statement relating the Cumulative cost to the percentage of physical completion with explanation of any significant variance:

If you have questions concerning this statement, they may be addressed to Karen Allison, 895-6421.
Chapter 1
MSFC FLAT FLOOR FACILITY

1.1 INTRODUCTION

This is the final report prepared for NASA George Marshall Space Flight Center (MSFC) by The University of Alabama in Huntsville (UAH) as part of the deliverables of a contract (NAS8-35670) awarded to UAH in 1984. The initial period of performance was eight months, and two subsequent modifications to the contracts were made. The entire task terminated in May 1985. The scope of work entails the development of software to drive the flat floor simulation facility at MSFC.

At the conclusion of the contracted period, a final report was not submitted by UAH because it was not possible to verify the functionality of the control software. This was due to a series of hardware modifications of the facility. In January 1987, most of the hardware modifications and upgrades were completed, and the system was available for testing the software. The principal investigator, who, by that time had left UAH, worked with MSFC engineers at no cost to MSFC, conducted tests to demonstrated that the software was indeed working as expected. The mobile base was part in a closed-loop control in January 1987. This explains the delay in submitting the final report.

1.2 THE ORBITAL MANEUVERING VEHICLE (OMV)

The Orbital Maneuvering Vehicle (OMV) has been designed to operate as a remotely controlled space teleoperator. This vehicle will be deployed as a payload from the space shuttle. Control of the OMV will be from a ground station, or a control room located on the shuttle or the space station. The operator controlling the OMV is physically remote from the module and
exercises control over the vehicle. The main mission of the OMV will be to increase the level of space productivity without increasing human risk. The OMV will not only reduce risk in orbital activities, but also increase the capacity to perform strenuous orbital operations. It will drastically reduce the level of EVA for a given mission. Unlike EVA, the OMV will not be affected by prolonged operational durations; also, it will be able to operate at ranges beyond EVA capabilities. The OMV has been designed to handle significant masses on the order of 45,000 pounds. The design should give the OMV the capability to:

- Deploy satellites in orbits that are out of the shuttle's range
- Rendezvous and dock with existing orbital payloads
- Resupply payloads with fuel and other consumables
- Perform repair and service operations on orbital payloads when fitted with a flight telerobotics system (FTS)
- Transfer payloads to or from orbit to the orbitting shuttle or space station.

With these capabilities, the OMV will have a definite impact on the way orbital operations are carried out. Figure 1-1 shows an application overview. To assemble an accurate simulator, the preliminary design of the actual OMV was studied. This design was reported in the Preliminary Definition Study of the Teleoperator Maneuvering System (TMS), prepared by program development at MSFC [1]. This document is used to obtain critical specifications that are needed for simulator design. These specifications include the vehicle's size, shape, mass, docking mechanisms, and attitude control system. The preliminary design of the Orbital Maneuvering Vehicle is shown in Figures 1-2 through 1-4. The following is a list of key assumptions and guidelines for the MSFC reference design:
Figure 1-3. Propulsion Vehicle
- Payload placement/retrieval capability
- Shuttle orbiter based with LEO/GEO mission capability
- Minimum practical length and weight
- Minimum orbiter interfaces
- Installation capability at multiple locations in cargo bay
- Satisfaction of safety requirements of NASA
- Monitoring and safing capability from orbiter AFD
- Potential for being space based at either LEO or GEO
- Control from ground station
- Capability to accommodate add-on kits and/or modifications for future extended capability and unique mission activities
- Maintain modularity to extent practical to accommodate hardware replacement
- On-orbit serviceability should be a design consideration
- Use of existing/developed hardware to extent practical
- Redundancy in critical areas
- Degree of autonomy necessary to preclude continuous ground control
- Safe hold capability to survive a single failure
- Design for 10 year life with refurbishment.

In order to verify these operational concept, a simulator was needed to permit extensive testing, modeling, and evaluation of the parameters involved in orbital operations. These tests include docking mechanisms, target motion, and human factors. Flexibility is of key importance in developing a simulation of his type. Ease in reconfiguring the simulator is essential. This reconfiguration may be through a series of hardware upgrades, such as additional degrees of freedom, propulsion system changes, front end assemblies, etc., or through software enhancement in the OMV mathematical model.
1.3 FLAT FLOOR FACILITY

The overall simulation system is shown in Figures 1-5 and 1-6. The three major subsystems are:

(1) Control console equipped with hand controller and display units
(2) Mainframe containing the OMV response model, orbital mechanics, and state vector transformation
(3) Mobility vehicle (TOM-B) with the flat floor and dynamic target simulator

Each of these subsystems have been further subdivided into modular components to give added flexibility. Detailed implementation will be given for each of the major subsystems. The overall control flow in block diagram form is given in Figure 1-7. The function and responsibilities of each subsystem can be summarized in the following:

**Control Room** - The control room is to serve as the man-machine interface. This interface consists of a command station (hand controllers) and sensory feedback devices (video monitors, status screens, etc.). The commands are then sent to the mainframe subsystem.

**Mainframe Subsystem** - The mainframe subsystem is to accept the hand controller commands, process these commands with respect to the OMV mathematical model, then generate and transmit the appropriate mobility base commands.

**Mobility Base** - The mobility base subsystem is to execute the generated commands from the mainframe. This consists of a number of vehicle movements to achieve the intent of the hand controller input.

1.4 SUBSYSTEM DESCRIPTION
Figure 1-6. MSFC Flat Floor Facility
Figure 1-7. Control Flow
1.4.1 Control Room Subsystem

The control room is the center of all teleoperation activities. This control room may be located on the ground, in the space shuttle or within the space station. The simulator control room is located adjacent to the flat floor, and its interior is depicted in Figure 1-8. From this location, an operator can control the simulator vehicle and cognitively sense, through various feedback methods, the overall operation. This is the idea of telepresence. The degree of telepresence is a function of the sensory feedback. This section will outline both the current and proposed types of sensory feedback. The degree of telepresence necessary for successful OMV man-machine interfacing is not well defined at this point in time. Critical human factors design is, however, beyond the scope of this work.

The main feedback element is direct video from cameras mounted on the vehicle. The video feedback is displayed on the screens in front of the pilot, as shown in Figure 1-8. Each screen will give a different view relevant to the operation to be performed. As currently configured, this is the only sensory feedback available to the operator. Modifications to be made to the control room include adding a status screen so that the operator will have pertinent data such as range, range rate, fuel depletion, force/torque, etc. NASA is at present evaluating the use of stereoscopic vision systems and 3-D displays. This would allow the operator to observe one main screen as opposed to correlating the views from several screens. Other modifications may include optical proximity sensing for collision detection, tracking, and centering operations. Touch screens, menu driven subsystems, and a mouse may be used. These feedback devices recreate a realistic scenario of the workspace within the control room. This technique will allow effective remote servicing capability.
The other major function of the control room subsystem is to accept operator control inputs. Control authority must be fast and efficient to permit teleoperation. The central input devices used are two 3 DOF hand controllers.

In the present implementation two hand controllers are used. One is used to control the translational axes - X, Y, Z, while the other controls rotational motion - roll, pitch, and yaw. These hand controllers give the operator full control over the vehicle. Full detail of the hand controller hardware and software will be given later.

The communication system that connects the control station to the OMV is a very critical component in the subsystem. This system defines the feedback and control limitations involved in teleoperation. Specifications for the OMV include communication via the Tracking and Data Relay Satellite System (TDRSS). All communication will be processed through this link. Because of the inherent time delay constraint involved when transmitting over large distances, NASA has chosen to incorporate this time delay into the OMV simulation. This will allow testing of variable time delays and their effects on the command and control that the operator will experience. The data rate limit for TDRSS is 1 Mbps down and 10 Kbps up, which requires that the standard video data rate be reduced to lower frame rates, lower pixel resolution, and adaptive encoding [2]. Figure 1-9 gives the overall communication data flow.

1.4.2 Mainframe Subsystem

The mainframe subsystem is responsible for accepting inputs from the control room subsystem and generating the appropriate commands to the mobility base subsystem. The mainframe subsystem hardware is composed of a
Figure 1-9. Communication Data Flow

* A MOCM IS REQUIRED FOR LENGTHS > 100 FEET

** to be completed August 1986
mainframe digital computer and the communication equipment. Connection between the mainframe subsystem and the mobility base subsystem is achieved via the communications network. The mainframe subsystem software consists of a module code named OMM, which is a mathematical model of the actual OMV. Each of the primary hardware and software components will be described in the subsequent chapters.

The computer used to process all off-vehicle computations is Digital VAX 11/750 minicomputer. This computer will be central in processing and controlling the data flow between the control room and the mobility base. The VAX 11/750 will process the hand controller inputs and generate the appropriate commands to the mobility base. This computer is responsible for generating the major cycle interrupts.

1.4.3 Mobility Base Subsystem

The third major component of the OMV simulation is the mobility base subsystem. This subsystem receives commands from the mainframe subsystem via a telemetry link. The responsibility of the mobility base subsystem is to execute these commands. The mobility base subsystem contains both hardware and software components. The hardware includes the mobile base vehicle (code named TOM-6), the Orbital Maneuvering Vehicle mockup module, the flat floor, and the target motion simulator. A description of the mobile base (TOM-6) and its associated subsystems will be given in full detail. The software component of this subsystem consists of the on-board processing logic of TOM-6; it's design, implementation, and verification will also be given in complete detail. The flat floor on which the mobile base traverse measures 86 feet by 44 feet and is shown in Figure 1-10. The floor was constructed in 1982 to test vehicles with air bearings. It is
within .001 inch between any adjacent square foot and has an overall flatness of .032 inches in the plane. It has a reinforced concrete foundation with a special epoxy resin surface for low friction.

The Orbital Maneuvering Vehicle mockup is shown in Figure 1-11. This mockup module was constructed according to the actual Orbital Maneuvering Vehicle specifications[1]. This mockup was mounted on the front of the mobile base. Figure 1-12 shows this arrangement. This arrangement facilitates realistic simulation of hardware-related operations such as docking, camera placement, etc.

A target motion simulator was constructed to replicate the motion of an orbiting target. Since the Orbital Maneuvering Vehicle will have many diverse tasks, a general purpose target was constructed, that is, the target was constructed with a standard docking mechanism mounted on its front. The target is mounted on the end of a robot arm. This robot arm, built by Kadar Corp., has a 20 foot reach with a 1000 pound payload capability. With appropriate software, this robot can emulate spin and precession motions which are common in orbiting satellites. The target motion simulator is mentioned because it is part of the overall Orbital Maneuvering Vehicle simulation, and will be used in testing. The detailed design and implementation is beyond the scope of this paper and will not be presented here. This robot arm, unique because of its size and performance specifications, is shown along with the mounted target in Figure 1-13. The mobility base has been code named TOM-B and will be referenced as such. TOM-B is a vehicle with air bearings that floats on the flat floor. The vehicle has six degrees of freedom. The vehicle is capable of translational and rotational motion. The X and Y translational and yaw motion is accomplished through the air bearing pads. The Z axis is driven by a DC
Figure 1-12. TOM-B with Docking Mechanism
Figure 1-13. Target Motion Simulator
drive motor and associated gear train. Similarly, the rotational motion of pitch and roll is fulfilled by DC drive motors and gear trains. X and Y translation is confined by the dimensions of the flat floor, which is 96 feet diagonally. Z motion is restricted to plus or minus 20 inches from the center of the drive train. Pitch is limited to plus or minus 20 degrees referenced from the horizontal center line. The other rotational axes, yaw and roll, are continuous. By executing appropriate motions, realistic Orbital Maneuvering Vehicle motions can be achieved. Note that the commands received from the mainframe subsystem emulate orbital motion. Thus, the motion of TOM-B is not necessarily that of the mockup module. For example, if the mockup module were to execute a yaw about it's Z axis, the output of the mainframe subsystem would generate a sequence of commands to TOM-B to execute a translation plus a rotation. The characteristics of TOM-B are shown in Table 1-1.

<table>
<thead>
<tr>
<th>TABLE 1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approximate mass:</td>
</tr>
<tr>
<td>Moment of Inertia:</td>
</tr>
<tr>
<td>Mass of fuel:</td>
</tr>
<tr>
<td>Number of thrusters:</td>
</tr>
<tr>
<td>Thrust developed:</td>
</tr>
</tbody>
</table>

Two sets of three thrusters, each of which is capable of delivering 13.2 Newtons of force, are mounted on each corner of the vehicle. Cold compressed air at 3500 psi is used as propellent. Six propellents tanks are used, four of which are used for thruster firings and two for the air bearing pads, as shown in Figure 1-14. Note that the thrusters are non-throttleable. The translation in the X and Y axes, as well as yaw motion of
Figure 1-14. TOM-B Showing the Six Degrees of Freedom & Thruster Assembly
the vehicle, are obtained by firing the appropriate thrusters. Translation along the Z axis, as well as pitch and roll are carried out using stepping motors fitted with resolvers. The overall hardware organization is given in Figure 1-15.

The computer and associated electronics are mounted on the rear of the vehicle. The control electronics include the A/D, D/A, modem, and sensor processing boards. The vehicle is fitted with X and Y accelerometers. To give orientation feedback a gyroscope is used. In the initial hardware configuration the accelerometers are used for measuring velocity and position, and the gyro is used to measure orientation of the vehicle. The gyro installed on TOM-B has a total error rate of $5 \times 10^{-6}$ degree/sec, which is more than adequate to provide feedback information on angular velocity and displacement. The accuracy is not present with the accelerometers[27].

The large error arises from the facts that:

1) The sensor has a high drift rate.
2) The signals from the sensors must be integrated numerically to obtain the translational displacement.
3) The errors are cumulative and propagate with time.

The accelerometers and gyro are actually designed to measure accelerations and angular velocities, respectively. When the signals must be integrated to get displacement, the following steps must be carried out:

1) They must be sampled frequently within every major cycle.
2) The signals must be conditioned and corrected for bias, scaling, offset, and drift.
3) To provide reliable displacement, sophisticated integration algorithms must be used.
Figure 1-15. TOM-B Control Hardware Organization
All these factors contribute to large computational overhead. In addition, there is no meaningful method for correcting the drift, other than using some external reference scheme.

Since position control is used in the present system, it is mandatory to have an accurate navigation system. This effectively rules out the use of accelerometers to provide positional feedback. An alternative, simpler navigation system is needed. This new navigation system does not replace the accelerometers; they are still needed to provide the rate feedback. The navigation system works on the principle that reflectors are mounted around the perimeter of the floor. A positionable distance meter, mounted on the vehicle, detects these reflectors. The system has two of these devices mounted on the front of the vehicle. It is estimated that a positional accuracy of several millimeters can easily achieve in this way. More importantly, the computation is relatively straightforward, fast, and the error does not propagate with time. This navigation system provides position feedback necessary for control of the vehicle. A detailed discussion of the design and implementation of the navigation system will not be presented here.

1.5 SOFTWARE DESCRIPTION

The current task as mentioned primarily is to develop suitable software as part of the flat floor simulation system so that it can be used to realistically study the behavior of the OMV. The software is made up of three major modules: a) the OMV mathematical model (OMM) which accepts operator input from the control station and compute the state of the OMV, b) the State Vector Transformation Module (SVX) which translates the OMV state vector into a set of commands for the mobility base, and c) mobility
base control logic TOM-B. When these commands are executed, the mobility base would have moved in such a manner that the OMV mockup mounted on it would have replicated the motion of the OMV. Figure 1-16 depicts the connectivity of these components.

Chronologically, SVX was developed first, followed by TOM-B, and OMM was developed last. However, OMM and SVX was tested and verified first, as these two modules are hardware independent, while TOM-B was test verified last. For the purpose of this report, the OMV mathematical model OMM will be describes in Chapter 2, the State Vector Transformation module SVX will be described in Chapter 3, and TOM-B in Chapter 4. A summary of testing procedures and conclusions will be presented in Chapter 5, together with the test date obtained.
Figure 1-16. Flat Floor Facilities -- Software Architecture
Chapter 2
OMV Mathematical Model (OMM)

2.1 INTRODUCTION

This report discusses the design and implementation of OMM - a mathematical model of the Orbital Maneuvering Vehicle [3]. The Orbital Maneuvering Vehicle (OMV) 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 \([X,Y,Z]^T\), orientation \([r,p,y]^T\) and their rates \([r,p,y]^T\) 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 OMM simulates the motion of the Orbital Maneuvering Vehicle in space. OMM 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 OMM (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 motion of the mobile base on the flat floor. The over-all relation is as shown in Figure 2-1 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 will be discussed in the next chapter. Throughout this report, it is important to bear in mind that the
OMM simulates the motion of the Orbital Maneuvering vehicle but otherwise has no physical relationship with the Orbital Maneuvering Vehicle. The mobility base on the flat floor will attempt to move in such a manner that a mockup 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 OMM. Otherwise the mobile base is not related to the OMV. The mockup 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 [4].

2.2 THE OMV MODEL

This section 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 2-2. The position, velocity as well as orientation of the second vehicle are described in LVF and is therefore relative to the
Figure 2-2. Local Vertical Frame (L)
orbiting vehicle. Throughout this work, it is assumed 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 2-3, 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 2-1. These figures are taken from the MSFC Preliminary Definition Studies.

2. The OMV is manipulated using signals from a set of hand controllers [5]. 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,
Figure 2-3. OMV Body Frame
<table>
<thead>
<tr>
<th>Dynamic Variable</th>
<th>Value</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass M</td>
<td>3282.75</td>
<td>kg</td>
</tr>
<tr>
<td>I_{11}</td>
<td>7048.37</td>
<td>kg m(^2)</td>
</tr>
<tr>
<td>I_{22}</td>
<td>3713.95</td>
<td>kg m(^2)</td>
</tr>
<tr>
<td>I_{33}</td>
<td>3713.95</td>
<td>kg m(^2)</td>
</tr>
</tbody>
</table>

Table 2-1. OMV Mass Characteristics
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 within 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.

2.3 ROTATIONAL EQUATIONS OF MOTION

The rotational equation of motion can be written as:

\[ \tau = \dot{L} \]

where \( \mathbf{L} = \mathbf{I}\omega \) is the angular momentum vector and \( \tau \) is the applied torque. \( \mathbf{I} \) is the moment of inertia tensor and \( \omega \) 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 [6,7], that is:

\[
\mathbf{I} = \begin{bmatrix}
I_{11} & 0 & 0 \\
0 & I_{22} & 0 \\
0 & 0 & I_{33}
\end{bmatrix}
\]
Remember that \( w = [w_1, w_2, w_3]^T \) is obtained from the hand controller signals. The solution of the rotational equations of motion yields \( \phi \), \( \theta \) and \( \psi \) the three Euler angles. The order and sense of rotation is chosen in the conventional manner [8], that is:

\[
[\phi_1 \theta_2 \psi_3]^T
\]

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 representatives are exactly equivalent [9]. A quaternion \( q \) 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 \( \alpha \) about an Euler axis \( E = [E_1, E_2, E_3]^T \). Theory has shown that this is the shortest angular path[10] in the sense that \( \alpha \) is less than the algebraic sum of \( \phi \), \( \theta \) and \( \psi \). The angle \( \alpha \) and the Euler axis can be expressed in terms of the quaternion \( q \) as:

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

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 \(E\) [10]. In fact, the following relation holds:

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

Quaternion algebra leads to further computational economy when successive rotations need to be calculated. Let say, at any instant, the attitude of the OMV is specified by the quaternion \(q_1\) relative to some non-rotating frame. Suppose further that an instant later, the vehicle's attitude has changed, having rotated by \(\phi, \theta\) and \(\psi\). These angular displacements are measured relative to the rotated body frame. If the new attitude is described by a second quaternion \(q_2\), the attitude of the vehicle, relative to the non-rotating frame [11,12] is then given by

\[ q = q_1q_2 \]

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:

\[ q = q_0q_1q_2 \ldots q_n \]

Of course, the attitude of the vehicle after the \(n+1\)-th rotation is \(q = q_nq_{n+1}\). Thus, the attitude of the vehicle can be computed from the pre-
vious 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.

2.4 EQUATIONS OF MOTION

The translational equations of motion [8] 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:

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

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

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

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

\[
C^{-1} = \begin{bmatrix}
q_4 + q_1 - q_2 - q_3 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\
2(q_1q_2 + q_3q_4) & q_4 - q_1 + q_2 - q_3 & 2(q_2q_3 - q_1q_4) \\
2(q_1q_3 - q_2q_4) & 2(q_2q_3 + q_1q_4) & q_4 - q_1 - q_2 + q_3
\end{bmatrix}
\]
It is obviously impractical to seek an analytical solution to the translational equations of motion. Numerical methods must be used. In the present work, the Adam-Bashforth method is used. For this purpose, each major cycle is subdivided into N (normally 10, but see later section) subintervals, 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 \( \dot{w} = [w_1, w_2, w_3]^T \) is known, one can calculate \( L_B \) in body frame using the relation

\[
L_B = I \dot{w} \\
L = C^{-1} L_B
\]

where \( C^{-1} \) 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 and will be described in Chapter 3.

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

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

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

Ideally, the hand controllers signals should be sensed and updated every minor cycle. But because of timing considerations they will be sensed once every major cycle, and it is explicitly assumed that the bit
<table>
<thead>
<tr>
<th>bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acceleration along +1 direction</td>
</tr>
<tr>
<td>2</td>
<td>Acceleration along -1 direction</td>
</tr>
<tr>
<td>3</td>
<td>Acceleration along +2 direction</td>
</tr>
<tr>
<td>4</td>
<td>Acceleration along -2 direction</td>
</tr>
<tr>
<td>5</td>
<td>Acceleration along +3 direction</td>
</tr>
<tr>
<td>6</td>
<td>Acceleration along -3 direction</td>
</tr>
<tr>
<td>7</td>
<td>+ roll; CCW rotation about 1-axis</td>
</tr>
<tr>
<td>8</td>
<td>- roll; CW rotation about 1-axis</td>
</tr>
<tr>
<td>9</td>
<td>+ pitch; CCW rotation about 2-axis</td>
</tr>
<tr>
<td>10</td>
<td>- pitch; CW rotation about 2-axis</td>
</tr>
<tr>
<td>11</td>
<td>+ yaw; CCW rotation about 3-axis</td>
</tr>
<tr>
<td>12</td>
<td>- yaw; CW rotation about 3-axis</td>
</tr>
</tbody>
</table>

Table 2-2. Hand Controller Bit Assignments
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 neighborhood 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 \( \mathbf{a} \) must be expressed in LVF before it can be used in solving the equations of motion. In the OMV software, this is carried out as mentioned previously by:

a) Calculating the inverse of the direction cosine matrix \( C^{-1} \),
b) Transforming the vector \( \mathbf{a} \) to \( \mathbf{A} \) in LVF, and
c) Placing \( \mathbf{A} \) in an acceleration matrix \( \mathbf{AA} \).

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

B) Numerical Solutions:

A three step Adam-Bashforth method [15] 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 positions and velocity vectors. A subroutine called STATE is then invoked to assemble the state vector.
C) Output Section:

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

A fairly simple graphics package called PLOT is implemented to provide graphics output. This package is developed for the initial software checking only; namely to provide to operator with some form of visual output and is not construed as a deliverable. It must be emphasized that this package is hardware dependent, and is not compatible with the PDP 11/34 mini-computer. The present graphics package runs on an IBM Personal Computer fitted with a TECMAR GRAPHICS MASTER board and an IBM monochrome monitor. A resolution of 640 by 352 is used for the package, although the system has a potential resolution of 720 by 700 pixels [16]. 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 2. 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 [14]. 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 3 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 4, and a hierarchal chart is shown in Figure 2-4.

2.6 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 and VAX 750 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
Note 1: Hardware incompatible graphics package.

Thus, the equations of motion reduce to:

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

This set of equations can be solved numerically using the Adam-Bashforth method. Further, if \( x_1, x_2, x_3 \) and \( v_1, v_2, v_3 \) are the initial conditions, it can be shown that the analytical solutions are:

\[
\begin{align*}
x(t) &= x_1 - \frac{(3\alpha t - 4\sin\alpha t)}{\alpha} v_1 - 6(\alpha t - \sin\alpha t) x_3 - \frac{(1 - \cos\alpha t)}{\alpha} v_3 \\
\dot{x}(t) &= -3 - 4\cos\alpha t v_1 - 6\alpha(1 - \cos\alpha t) x_3 - 2(\sin\alpha t) v_3 \\
y(t) &= \cos\alpha t x_2 + \frac{\sin\alpha t}{\alpha} v_2 \\
\dot{y}(t) &= -\alpha(\sin\alpha t) x_2 + (\cos\alpha t) v_2 \\
z(t) &= \frac{2(1 - \cos\alpha t)}{\alpha} v_1 + (4 - 3\cos\alpha t) x_3 + \frac{\sin\alpha t}{\alpha} v_3 \\
\dot{z}(t) &= 2(\sin\alpha t) v_1 + 3\alpha(\sin\alpha t) x_3 + (\cos\alpha t) v_3
\end{align*}
\]

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

\[
\alpha = \frac{GM_e}{(R_o + H)^3}
\]
where $G$ is the universal gravitation constant, $M_e$ is the mass of the earth, $R_0$ is the mean earth radius and $H$ is the altitude. Note that at higher orbits, $\Omega$ approaches 0 and the equations of motion approach

\[
\begin{align*}
\dddot{x} & \rightarrow 0 \\
\dddot{y} & \rightarrow 0 \\
\dddot{z} & \rightarrow 0 
\end{align*}
\]

and better agreement between numerical and analytical results are expected for high altitudes than lower orbits. A computer program called ADAM has been developed that would, given a set of initial conditions, calculate both the numerical and analytical solutions to the equations of motion. The source listing of ADAM is shown in Appendix 5. In the present set of tests, an altitude of 200 kilometers ($\Omega = 0.00118 \text{ rad/sec}$) is used throughout. This altitude represents the lowest design orbit of the Orbital Maneuvering Vehicle. Table 2-3 shows a comparison between the analytical and numerical solutions at this altitude, using the initial conditions:

\[
\begin{align*}
x_1 &= 0, \quad x_2 = x_3 = 0 \\
v_1 &= 0.05, \quad v_2 = v_3 = 0 
\end{align*}
\]

The results shows that the two solutions agree to better than $3 \times 10^{-8}$ in 60 minutes, or about 0.03 millimeters. This figure is well below the expected accuracy of the flat floor simulation system. This suprisingly small error comes from the fact that the angular velocity $\Omega$ is quite small. When $\Omega = 1.0$ is used, (this angular frequency does not make sense physically, as it represents an orbit well below the earth's surface, but...
<table>
<thead>
<tr>
<th>Time in Minutes</th>
<th>X (meters)</th>
<th>Z (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Numerical</td>
<td>Analytical</td>
</tr>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>5</td>
<td>13.746736</td>
<td>13.746736</td>
</tr>
<tr>
<td>10</td>
<td>20.161917</td>
<td>20.161917</td>
</tr>
<tr>
<td>15</td>
<td>12.828963</td>
<td>12.828962</td>
</tr>
<tr>
<td>20</td>
<td>-12.952950</td>
<td>-12.952952</td>
</tr>
<tr>
<td>25</td>
<td>-59.582227</td>
<td>-59.582233</td>
</tr>
<tr>
<td>30</td>
<td>-126.855533</td>
<td>-126.855544</td>
</tr>
<tr>
<td>35</td>
<td>-211.993176</td>
<td>-211.993191</td>
</tr>
<tr>
<td>40</td>
<td>-309.986003</td>
<td>-309.986022</td>
</tr>
<tr>
<td>45</td>
<td>-414.220544</td>
<td>-414.220565</td>
</tr>
<tr>
<td>50</td>
<td>-517.304365</td>
<td>-517.304388</td>
</tr>
<tr>
<td>55</td>
<td>-611.988644</td>
<td>-611.988666</td>
</tr>
<tr>
<td>60</td>
<td>-692.072815</td>
<td>-692.072834</td>
</tr>
</tbody>
</table>

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

Table 2-3. Comparison Between Analytical and Numerical Solutions
constitutes a valid situation mathematically), the errors propagate quite fast as to render the comparison meaningless after 10 minutes.

A second test was carried out at the same altitude, using null initial conditions:

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

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 2-4. A plot of the relevant dynamic variables as a function of time is shown in Figure 2-5. 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 \(X\) 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 favorably with the numerical results at \(t = 4\) seconds.

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

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

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

\[ x_1 = x_2 = x_3 = 0 \quad \text{and} \quad v_1 = v_2 = v_3 = 0 \]

Note: All quantities are expressed in Local Vertical Frame.

### Table 2-4. OMV Acceleration Along +X Direction

<table>
<thead>
<tr>
<th>Time in Seconds</th>
<th>X in meters</th>
<th>Y in meters</th>
<th>Z in meters</th>
<th>Z in meters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.5</td>
<td>0.002940</td>
<td>0.012125</td>
<td>0.000001</td>
<td>0.000007</td>
</tr>
<tr>
<td>1.0</td>
<td>0.012128</td>
<td>0.024625</td>
<td>0.000009</td>
<td>0.000029</td>
</tr>
<tr>
<td>1.5</td>
<td>0.027565</td>
<td>0.037125</td>
<td>0.000032</td>
<td>0.000065</td>
</tr>
<tr>
<td>2.0</td>
<td>0.049253</td>
<td>0.049625</td>
<td>0.000077</td>
<td>0.000117</td>
</tr>
<tr>
<td>2.5</td>
<td>0.077190</td>
<td>0.062125</td>
<td>0.000152</td>
<td>0.000183</td>
</tr>
<tr>
<td>3.0</td>
<td>0.111377</td>
<td>0.074624</td>
<td>0.000263</td>
<td>0.000264</td>
</tr>
<tr>
<td>3.5</td>
<td>0.151814</td>
<td>0.087124</td>
<td>0.000418</td>
<td>0.000360</td>
</tr>
<tr>
<td>4.0</td>
<td>0.198501</td>
<td>0.099624</td>
<td>0.000625</td>
<td>0.000471</td>
</tr>
</tbody>
</table>
Figure 2-5. Translation Along X-Axis
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 2-5 shows the relevant commands for the mobile
base. As verified by the graphics display, the mock up module mounted on
the mobile base executes a pure pitch at the same rate as the OMV, while
the mobile base has to translate along the +X direction. In addition, the
pivot point is progressively lowered as expected. This test shows that the
modules OMV and SVX are properly interfaced, and that correct results are
produced. The command strings as outputted by the system to the flat floor
is shown in Figure 2-6.

To further ascertain that the system is functioning properly, the
hand controller signals corresponding to a translation along L-axis and a
yaw is generated. The relevant commands to the flat floor system is shown
in Table 2-6. A pictorial representation of the mobile base and mock up is
as shown in Figure 2-7. Note that the path of the center of mass of the
mock up exactly duplicates that of the OMV.

In summary, various tests have shown that the OMV-SVX system func-
tions 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 2-8.
There is just one area that needs further investigation, namely timing con-
siderations. 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

\[ \begin{align*}
  r &= y = 0, \quad \text{and} \quad p = w_2 = 0
\end{align*} \]
<table>
<thead>
<tr>
<th>Time (Sec)</th>
<th>Pitch (Rad)</th>
<th>X (meters)</th>
<th>Z (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>5.0000</td>
<td>2.4384</td>
</tr>
<tr>
<td>4</td>
<td>0.0698</td>
<td>5.0010</td>
<td>2.3852</td>
</tr>
<tr>
<td>8</td>
<td>0.1396</td>
<td>5.0074</td>
<td>2.3324</td>
</tr>
<tr>
<td>12</td>
<td>0.2094</td>
<td>5.0167</td>
<td>2.2800</td>
</tr>
<tr>
<td>16</td>
<td>0.2793</td>
<td>5.0295</td>
<td>2.2284</td>
</tr>
<tr>
<td>20</td>
<td>0.3491</td>
<td>5.0460</td>
<td>2.1778</td>
</tr>
<tr>
<td>24</td>
<td>0.4189</td>
<td>5.0659</td>
<td>2.1285</td>
</tr>
</tbody>
</table>

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

Table 2-5. OMV--Pure Pitch Motion at 0.017453 rad/sec
Figure 2.6. Pure Pitch Motion at 0.07453 rad/sec
<table>
<thead>
<tr>
<th>Time (Sec)</th>
<th>$X$ (meters)</th>
<th>$Y$ (meters)</th>
<th>$Z$ (meters)</th>
<th>Yaw (rad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>11.6680</td>
<td>2.4384</td>
<td>0.0000</td>
</tr>
<tr>
<td>4</td>
<td>0.2752</td>
<td>11.2418</td>
<td>2.4390</td>
<td>0.3470</td>
</tr>
<tr>
<td>8</td>
<td>1.0709</td>
<td>11.0039</td>
<td>2.4433</td>
<td>0.6940</td>
</tr>
<tr>
<td>12</td>
<td>2.2919</td>
<td>11.1199</td>
<td>2.4545</td>
<td>1.0410</td>
</tr>
<tr>
<td>16</td>
<td>3.7925</td>
<td>11.7135</td>
<td>2.4750</td>
<td>1.3880</td>
</tr>
<tr>
<td>20</td>
<td>5.3934</td>
<td>12.8512</td>
<td>2.5062</td>
<td>1.7350</td>
</tr>
<tr>
<td>24</td>
<td>6.9035</td>
<td>14.5350</td>
<td>2.5480</td>
<td>2.0820</td>
</tr>
</tbody>
</table>

Table 2-6. Motion of the Mobile Base Under Constant Acceleration of $(0.025, 0, 0)^T$ and Constant Yaw at $0.08675$ rad/sec
Figure 2-7. Trajectory of Mobile Base When OMV is Executing a Translation & Yaw
Figure 2-8. OMV Pure Yaw Motion
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 2-7 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 [13] took most of the computation time. Table 2-8 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 2-9 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.

The series of tests conducted, some of which are not reported here, shows that the simplified mathematical of the Orbital Maneuvering Vehicle
| 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 2-7. OMV Time Test
Table 2-8. Optimized OMV Execution Times Per Major Cycle as a Function of Number of Steps $N$

<table>
<thead>
<tr>
<th>$N$</th>
<th>Execution time (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.068</td>
</tr>
<tr>
<td>5</td>
<td>0.079</td>
</tr>
<tr>
<td>6</td>
<td>0.090</td>
</tr>
<tr>
<td>7</td>
<td>0.100</td>
</tr>
<tr>
<td>8</td>
<td>0.111</td>
</tr>
<tr>
<td>9</td>
<td>0.122</td>
</tr>
<tr>
<td>10</td>
<td>0.132</td>
</tr>
<tr>
<td>Time in Minutes</td>
<td>Analytic</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>0.000000</td>
</tr>
<tr>
<td>5</td>
<td>13.746736</td>
</tr>
<tr>
<td>10</td>
<td>20.161917</td>
</tr>
<tr>
<td>15</td>
<td>12.828962</td>
</tr>
<tr>
<td>20</td>
<td>-12.952953</td>
</tr>
<tr>
<td>25</td>
<td>-59.582233</td>
</tr>
<tr>
<td>30</td>
<td>-126.855544</td>
</tr>
<tr>
<td>35</td>
<td>-211.993191</td>
</tr>
<tr>
<td>40</td>
<td>-309.986022</td>
</tr>
<tr>
<td>45</td>
<td>-414.220565</td>
</tr>
<tr>
<td>50</td>
<td>-517.304388</td>
</tr>
<tr>
<td>55</td>
<td>-611.988666</td>
</tr>
<tr>
<td>60</td>
<td>-692.072834</td>
</tr>
</tbody>
</table>

Table 2-9. Comparison Test Between N = 4 and N = 10 Steps
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.
Chapter 3

STATE VECTOR TRANSFORMATION MODULE (SVX)

3.1 INTRODUCTION

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

References [14,17] are reports that pertain to the various aspects of the OMV. From these reports, the various components that make up the state vector can be deduced and are presented below:

<table>
<thead>
<tr>
<th>Component</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X$</td>
<td>Position of the target vehicle relative to the OMV in local vertical frame LVF</td>
</tr>
<tr>
<td>2</td>
<td>$Y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$Z$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$V_X$</td>
<td>Relative velocity of the chase vehicle</td>
</tr>
<tr>
<td>5</td>
<td>$V_Y$</td>
<td>in LVF</td>
</tr>
<tr>
<td>6</td>
<td>$V_Z$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$L_X$</td>
<td>Angular momentum vector in LVF</td>
</tr>
<tr>
<td>8</td>
<td>$L_Y$</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$L_Z$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$q_1$</td>
<td>Attitude quaternions in body frame</td>
</tr>
</tbody>
</table>
It is often more convenient to consider the state vector to be made up of the following four vectors: \( \mathbf{X} = [X, Y, Z]^T \), \( \mathbf{V} = [V_X, V_Y, V_Z]^T \), \( \mathbf{L} = [L_X, L_Y, L_Z]^T \) 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:

<table>
<thead>
<tr>
<th>Component</th>
<th>Position Control</th>
<th>Rate Control</th>
<th>Coord. System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( y ) yaw of TOM-B</td>
<td>( \dot{y} ) yaw rate</td>
<td>body frame</td>
</tr>
<tr>
<td>2</td>
<td>( X ) position of TOM-B</td>
<td>( V_X ) velocity of LVF</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( Y ) TOM-B</td>
<td>( V_Y ) TOM-B</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( Z ) pos of pivot</td>
<td>( V_Z ) vel of pivot</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( p ) pitch angle</td>
<td>( \dot{p} ) pitch rate</td>
<td>body frame</td>
</tr>
<tr>
<td>6</td>
<td>( r ) roll angle</td>
<td>( \dot{r} ) roll rate</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 pos. control</td>
<td>0 rate control</td>
<td></td>
</tr>
</tbody>
</table>

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

3.2 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 the 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 3.1.

A subscript $L$ will be used to indicated quantities defined in this coordinate system.

B. The Floor Coordinate (F)

The floor coordinates has its origin at one corner of the flat floor as shown in Figure 3.2. 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 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 3.3.
Fig. 3-1. Local Vertical Frame (L)
Figure 3-2. Floor Coordinates (F)
Figure 3-3. TOM-B Body Frame (B)
D. The Mockup Module Body Frame (M)

It assume that the mockup 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 3.4.

3.3 ANALYSIS

It is obvious that the position and attitude from the state vector are relative quantities. Thus, initial conditions at the start of the simulation must be known. Figures 3.5 and 3.6 shows the initial state of the mobile base and mockup 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 $[X_0,Y_0,Z_0]^T$ 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:

$$\begin{bmatrix}
X_L \\
Y_L \\
Z_L
\end{bmatrix} = \begin{bmatrix}
S_1 \\
S_2 \\
S_3
\end{bmatrix}$$
Figure 3-4. Mock-Up Module Body Frame (B)
Figure 3-5. Initial Position (top view)
Figure 3-6. Initial Position (side view)
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 mockup module in floor coordinates using the equation:

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

Equation [I] governs the transformation of the position vector of the OMV in LVF to a position vector for the mockup 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 $[r_M, \phi_M, z_M]^T$ as shown in Figure 3-7 and 3-8 the position of TOM-B $[X_F, Y_F, Z_F]^T$ in floor coordinates is given by:

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

Note that $Z_F$ is the height of the center of mass of TOM-B from the floor (a constant quantity), and is not of interest here. Instead, the quantity of interest is $Z$, which is the height of the pivot point from the floor as shown in Figure 3.6, and

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

It follows that the velocity of TOM-B and the pivot point is given by
Figure 3-7. Position and Yaw of TOM-B
Figure 3-8. Pitch and Roll of Mock-Up Module
\[
\begin{bmatrix}
X_F \\
Y_F \\
Z
\end{bmatrix}
= \begin{bmatrix}
X_M + (c + 1 \cos(p)) \sin(p) y + l \sin(p) \cos(y) p \\
Y_M - (c + 1 \cos(p)) \cos(p) y + l \sin(p) \sin(y) p \\
Z_M - l \cos(p) p
\end{bmatrix}
\]

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

The quaternions \( q_1, q_2, q_3, q_4 \) from the state vector specify the OMV's attitude in body frame, as discussed in References [18,19]. At any instant, its orientation is given by [10]:

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

where

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

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

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

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

\[
L_B = A L
\]

here \( A \) is the direction cosine matrix which can be constructed from the attitude quaternions \( q_1, q_2, q_3, \) and \( q_4 \)
Knowing the moment of inertia tensor $\mathbf{I}$, one can calculate the angular rates

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

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

3.4 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 $\text{MODE} <> 0$ (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 and round off to the nearest integer (16-bit word) and is the sole output of the SVX module.

Rate information is not of interest when the system is in position control,
and is therefore not transmitted. Throughout this module, the scale factors for all angular and displacement quantities are $10^4$ and $10^3$ respectively.

Case 2 $\text{MODE} <> 0$ (rate control)

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

$$[y, x_B, y_B, z, p, r, 0]^T$$

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

Case 3 $\text{MODE} <> 0$ and $\text{MODE} <> 1$

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

3.5 IMPLEMENTATION

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

The subroutine is implemented in FORTRAN 77, and the usual programming practices are adhered to. Most of the major steps are either properly documented in the form of COMMENT statements or implemented as subprograms, following a modular design approach. Whenever possible, structured codes are used unless severe degradation of execution speed may result.
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 6 shows a complete listing of this module. A more detailed description of the testing procedure will be presented later in this section.

A local counter (COUNT) is initialized at load time, and updated during execution to enable SVX to determine the initial state on start up. During this period, other tasks are carried out as an integral part of the initialization process. This includes reading a file (SVXINT.DAT) for the values of c, l, a, h and o, as well as the inverse of the moment of inertia tensor I⁻¹.

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

\[ E = [E_1, E_2, E_3]^T \]
\[ = [c + 1 - X_0, a - Y_0, h - Z_0]^T \]

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

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

When MODE is non-zero, position control is assumed. SVX invokes subroutines QTRPY and UPDPOS to calculate the desired orientation and position of the OMV. A transformation is then made to determine the required position (of the mobile base TOM-B in floor coordinates) and orientation
(of the mockup 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 the 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 display the results in two dimensions. By way of example, a sequence of 60 state vectors of the form:

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

is generated. This set of state vectors simulates 50 seconds of run time in which position control is used. The meaning of this state vector is that the OMV is to remain stationary, but executes a yaw at a rate of 15° per major cycle (0.1 second). Here, we have assumed that the OMV is a disk shaped object having a uniform mass distribution and a constant mass of 1500 pounds. Note that in case of position control, the angular momentum vector is inconsequential, so a null vector is used. These figures may not be very realistic, but they are adequate for testing the SVX module.

Figure 3-9 shows the result of a portion of the output command string. In this and subsequent figures, a circle or dot indicates that the position of the center of mass of TOM-B in floor coordinates, while an attached arrow shows its yaw. This figure depicts that TOM-B moves in a circular path and its yaw is changing at a rate of 15° per major cycle. It is noted that the radius of the circular path is equal to the distance between the centers of mass of TOM-B and the mock-up module. Thus, the mock-up module would be spinning about its $Z_M$ 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 3-10. In this figure, TOM-B attempts to move in a circular path with a net displacement
Figure 3-9. Position of TOM-B in Floor Coordinates
Figure 3-10. Trajectory of TOM-B
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.

3.6 RESULTS

Other similar tests have been conducted. For example, the state vector in the beginning of this section has been input for rate control, and the result is plotted in Figure 3-11. This and similar results have demonstrated that the module SVX is functioning properly and that correct command strings are obtained. One must remember that the outputs of this module are commands to TOM-B, indicating the desired position, (or velocity) and attitude (or angular rates). The proper interpretation, and subsequent execution, of these commands are performed by the TOM-B Executive, and is outside the scope of the SVX module.
Figure 3-11. Velocity Components of TOM-B
4.1 INTRODUCTION

TOM-B is the control software that drives the mobility base. A description of the mobility base has been given in Chapter 1, and will not be repeated here.

TOM-B is designed to perform position or rate control over the mobility base. During development and testing, position control was used. The command structure coming from six could consist of a sequence of 6 numbers, each of which specifies the desired position and orientation of the vehicle when the command is executed but because of communication bandwidth, the command string consists of a positional increment, which must be added to the current position to yield the desired position. Further, the most efficient mode of transmission is in integer format and this format is adopted here. It is understood that for positional quantities (such as X, Y, and Z) the unit used is 0.001 inch, while for the remaining quantities (angular), a unit of 0.1 degree is used. By way of example, the command string:

10 0 20 0 0 0

is interpreted such that TOM-B move along X axis 0.01 inches from the current position, and rotate by 2 degrees about its Z axis. All other axes remain unchanged. Symbolic names are used to represent each of these quantities in the command string, the command transmitted to TOM-B is of the form:

CMD-X, CMD-Y, CMD-THETA, CMD-Z, CMD-P, CMD-R

Essentially, based on the desire position/orientation and the current
position/orientation, one can calculate the required impulses $f_x$ and $f_y$. This is the required impulse that moves TOM-B from the present position to the desired position, and is expressed most conveniently in floor coordinates. This impulse is translated into the corresponding impulses $F_X$ and $F_Y$, which are impulses that must be exerted by TOM-B. This is necessary because at any particular moment, the body-centered coordinate system defined with respect to TOM-B may not be lined up with the floor coordinates. Once $F_X$ and $F_Y$ are known, the individual impulses $F_X1$, $F_X2$, $F_Y1$, and $F_Y2$ to be exerted by the appropriate thrusters are determined. From these impulses, one can calculate the firing times of these thrusters, since they cannot be throttled. The firing times are then suitably scaled, and the appropriate numbers loaded into the corresponding down counter. A control signal is then sent to fire the thrusters, as shown in Figure 4-1.

Figure 4-2 shows the hypothetical position and orientation of TOM-B when the position and orientation of TOM-B is given by the vector $(x, y, \theta)$ determined from the navigation system. Here $\theta$ is the orientation of the vehicle. The desired position and orientation is dictated by the command string $(X_{CMD}, Y_{CMD}, \theta_{CMD})$ such that the vehicle will be at this position at the end of the current major cycle. The required impulse to accomplish this is given by:

$$f_x = \text{mag}(X, X_{CMD}, V_{OX})$$
$$f_y = \text{mag}(Y, Y_{CMD}, V_{OY})$$

where $f_x$, $f_y$ are the required impulses along $X$ and $Y$ directions in floor coordinates. $V$ is the velocity of the vehicle, also expressed in floor coordinates. It is noted that $V_{OX}$, and $V_{OY}$ are obtained from the accelerometer readings $V_x$ and $V_y$ using the transformation:
Figure 4-1. TOM-B Thruster Control Signal
Figure 4-2. TOM_B Orientation Angle $\theta$. 
\[
\begin{bmatrix}
V_{ox} \\
V_{oy}
\end{bmatrix} =
\begin{bmatrix}
-sin \theta & cos \theta \\
cos \theta & sin \theta
\end{bmatrix}
\begin{bmatrix}
V_x \\
V_y
\end{bmatrix}
\]

and the function \( g \), is given by:

\[
g(X, X_{CMD}, V_{ox}) = T - \lambda \quad \text{if } \lambda^2 \text{ is non-negative}
\]

\[
g(X, X_{CMD}, V_{ox}) = -V_{ox} + \frac{(V_{ox} + 2a(X_{CMD} - X))^{1/2}}{a}
\]

where

\[
\lambda^2 = \frac{T^2 - 2(X_{CMD} - X - V_{ox}T)}{a}
\]

Here, \( a \) is the magnitude of the acceleration produced when one pair of thrusters is fired simultaneously in the same direction, and is approximately equal to 0.1 ft/sec\(^2\). \( T = 0.1 \) is the major period. Note that the impulses \( f_x \) and \( f_y \) are defined relative to the floor coordinates. To determine the actual impulses \( F_x, F_y \) that TOM-B must exert to produce the same displacement, we use the transformation:

\[
\begin{bmatrix}
F_x \\
F_y
\end{bmatrix} =
\begin{bmatrix}
-sin \theta & cos \theta \\
cos \theta & sin \theta
\end{bmatrix}
\begin{bmatrix}
f_x \\
f_y
\end{bmatrix}
\]

where \( \theta \) is the orientation of the vehicle as determined by the navigation system.
4.2 CONTROL LAW

Once the impulses $F_x$ and $F_y$ are known, then the individual impulses $F_{xl}, F_{x2}, F_{yl},$ and $F_{y2}$ that each thruster must produce can be calculated [13]. The notation as shown in Figure 4-3. Wherever a negative quantity is encountered, the directly opposite thruster will be used. Obviously, one must have the relation:

\[
F_x = F_{x1} + F_{x2} \\
F_y = F_{y1} + F_{y2}
\]

Note that not only must the impulses produce the required translational displacement, but also must produce the necessary angular displacement. We define the required torque $T_0$ by the relation:

\[
T_0 = 2J_{zz}(\theta_{CMD} - \theta) / T^2
\]

where $T$ is the major period and $J_{zz}$ is the principle moment of inertia about Z-axis of TOM-B [5]. It is prudent to consider the following two cases.

Case 1. $F_x \leq F_y$

In this case:

\[
F_{y1} = F_y / 2 + T_0 / (2L_y) \\
F_{y2} = F_y - F_{y1}
\]

If one defines a quantity $F_X$ to be

\[
F_X = (T_0 + (F_{y2} - F_{y1})L_y) / (2L_x)
\]

then
Figure 4-3. TOM-B Thruster Impulses
Case 2. $F_X > F_Y$

In this case,

$$F_{x1} = F_X / 2 + F_X$$
$$F_{x2} = F_X - F_{x1}$$

If one defines another quantity $T'_0$ such that:

$$T'_0 = T_0 + (F_{x2} - F_{x1})L_x$$

then,

$$F_{y1} = F_Y / 2 + T'_0 / (2L_Y)$$
$$F_{y2} = F_Y - F_{y1}$$

These impulses must be converted into the corresponding firing times $T_{x1}$, $T_{x2}$, $T_{y1}$, and $T_{y2}$, respectively because the thrusters are not throttleable. These can be accomplished using the formula:

$$T_{xj} = F_{xj} / ma$$
$$T_{yj} = F_{yj} / ma$$

for $j = 1, 2$. Here, $ma$ (mass times acceleration) is the thrust developed by each thruster. Recall that a negative $T_{xj}$ means that opposite thrusters will be used.

4.3 TOM-B PROCESSING LOGIC--ALGORITHM

In this section, the high level control logic is discussed fully. The name of the software is TOMC. This is to differentiate between the
hardware TOM-B. The code is written in FORTRAN and MACRO-II (Appendix 5). A top down design is used throughout.

The main program of the control logic is shown in Figure 4-4. The initialization procedure consists of the following steps:

a) A routine is used to set up a schedule to interrupt the system ten times every second. The interrupt service routine must:
   1) Interrupt the incoming command string,
   2) Determine the present position and orientation of TOM-B using the navigation system,
   3) Get the buffers containing the accelerometer and gyro readings. Note that the position for the other three axes (Z, pitch and roll) will also be determined by this service routine.

Thus, updated information is always available in any given major cycle.

b) Static quantities (such as physical dimensions of the vehicle which are not expected to change) are initialized.

c) A data file is opened and accessed so that dynamic quantities such as mass of fuel, number of thruster pairs per side, thrust that will be developed by each thruster, calibration data, scale factors, etc., are initialized. This is an efficient design, as the system may be subject to further modifications, or the experimental condition may change (e.g., a different module may be mounted, causing a change in the mass of the vehicle). Under this circumstance, the data file is modified offline, without having to change and recompile the entire software.

After the initialization phase, the balance of the main program
Figure 4-4. Control Software - Main Program
involves intercepting the command string once every 0.1 second, and executing this command string until a command to stop is encountered. When this happens, preliminary shutdown procedures (such as turning off all thrusters) is carried off before the final system shutdown.

The processing of a major cycle is carried out in a procedure called MAJOR, as shown in Figure 4-5. On entering this procedure, appropriate memory locations are accessed and the current position and orientation of TOM-B are determined. The command string is examined first to see if any thrusters must be activated. A separate routine called THRUSTER performs the necessary thruster logic. When this subtask is completed, the balance of the command string is examined to see if it is necessary to move any of the stepping motors which control the remaining axes (Z, pitch and roll). The procedure MOTOR performs the necessary stepping motor control logic. A waiting procedure is implemented to place the processor in a dormant state until the next command string is intercepted. A higher priority is assigned to thruster logic. This is deliberately done because of the nature of the thruster hardware logic. An appropriate number is placed in the corresponding down counter and a control signal is issued to fire a thruster. The hardware fires the thruster and decrements the counter until its contents are zero, after which the thruster shuts down. During this interval the processor performs other tasks, and need not wait until the firing cycle is completed. For this reason alone, thruster logic is processed first is procedure MAJOR.

4.4 TESTING AND VERIFICATION

Verification of TOM-B was accomplished by a series of measurements and tests conducting using the mobility base. These series of measurements
Figure 4-5. Control Software - Major Cycle
were lengthy and involved interaction with the hardware. Although handling and adjusting hardware components were outside the scope of the contract, UAH has provided personnel to perform these minor operations, under the supervision of MSFC personnel, to expedite the testing procedure. Because of the frequent hardware modification/upgrade and because of concurrent time needed by ESSEX for their measurements, it was not possible for UAH to have a reasonable block of uninterrupted machine time for testing purposes. Frequently, it is necessary to schedule our tests between ESSEX's runs. More frequently, our tests have to be suspended because of hardware unavailability or failures.

A series of tests were conducted initially to ensure that the TOM-B initialization procedure was corrected. This was done by modifying the code to display all critical parameters such as scale factors/orbits of the gyro and accelerometers, firing table, etc. The interrupt routines were also thoroughly tested on-line. The result was that several parameters have to be tuned, but this was easily accomplished since all critical parameters were placed in data files, and as such they are easy to modify without disturbing the code.

Several of the components on the mobility base must be calibrated in order to obtain some of the parameters. These include the gyro and the accelerometers. For proper operation, the precise scale factor and offsets of these components must be obtained in order to correlate the outputs of these sensors to actual vehicle parameter (position, speed and orientation in the appropriate units). Figure 4-6 shows the gyro/accelerometer package [20]. An optimal place to mount these packages would be at the e.g. of the mobility base. This was accomplished mostly by trial and error method, and special software was developed for this purpose.
Figure 4-6. Accelerometer & Gyroscope Mounting
The gyroscope and the accelerometer have been bench tested, but our runs showed that an on-vehicle calibration is necessary. The gyroscope was calibrated in the conventional manner. A separate calibration progress called ACE was developed to permit date acquisition and analysis. The procedure developed is as follows:

a) Allow the system to warm up to operating temperature.

b) All air handles to the facilities have been disabled so that drafts would not cause any extraneous motion. This turned out to be an important consideration, especially when a full scale OMV mockup was mounted on the mobility base.

c) ACE was commanded to fire an appropriate set of thrusts, causing the mobility base to execute a pure rotational motion about its Z-axis. The firing time was recorded.

d) The angular displacement in radians during the thruster firing was recorded.

e) When the thrusters ceased firing, the angular displacement and time required until the mobility base ceased rotation were also needed.

From these data, it is possible to deduce the kinetic coefficient of rotational friction (which turned out to be quite small) and the proper scale factor (and offset) of the gyroscope. Thus, one can correlate the gyro output to angular displacement. Figure 4-7 shows one such set of calibration data.

Similar procedures were used to calibrate the two accelerometers mounted along the X and Y axis of the mobility base respectively. In this instance, however, the appropriate thrusters were selected to produce pure translation along a single axis instead. Several interesting phenomenon
were observed:

a) The e.g. does not line along the symmetrical axis of the vehicle. It was necessary to counter-balance the mobility base with lead bricks in order to obtain translational motion without a rotational component.

b) The kinetic coefficient of friction was quite large. The test procedure was to enable the thrusters for two seconds, measure the displacement $d_1$, after which time the mobility base was permitted to coast to a stop. The displacement $d_2$ and time $t_2$ were recorded. In 60% of the trials, $d_2$ was no greater than $d_1$.

c) The floor is not flat. With the air handles off, there was no significant air current in the facility. When the mobility base was put in certain areas of the floor, it had a tendency to drift in a consistent direction, but the drift rate although observable is very small.

Figure 4-8 shows the a typical calibration curve of one of the accelerometers. Both accelerometers behave quite identically so that this figure is quite typical. Immediately several problems are evident.

a) The signal to noise ratio is unacceptable, as can be estimated from this diagram. Remember time $t = 0$ was the time when the thrusters commenced firing.

b) A slope change was always observed approximately $\frac{1}{2}$ seconds after time $t = 0$. This change of slope represents the fact that the thrust level drops after $\frac{1}{2}$ seconds of firing. This is further substantiated by a change in the pitch and is detectable by hearing. This drop in level is an indication that the phlenum is
not able to supply air at the designed rates to the thrusters. This could be a result of an engineering design change in which additional thrusters have been added to the phlenum.

c) The data shows a lot of scattering. This is due to the excessive vibrations transmitted to the accelerometer when the thrusters are enabled.

The combination of poor signal to noise ratio, a drop in thrust level after 0.5 seconds and noise means that at best, one may extract marginal rate data from the accelerometer outputs, and would entail the use of various smoothing, fitting and integration techniques. Thus, attempting to obtain position data by further integration would be counter productive. These observed problems, as well as a recommendation for an independent position feedback subsystem, was reported to MSFC.

It is at this point in time that the contractual period was up, and the facilities was scheduled to shut down for major hardware modification.

4.5 RESULTS

Although we were not able to complete testing the software, several important tasks have been accomplished. First, attitude control using gyro output was completed. During some tests, we were able to point the mobility base in any desired direction and maintaining this direction. This indicated that the software is exercising positive control for this axis. Since the accelerometer data are processed in the same fashion, all that would be needed to close the loop was to implant a position feedback subsystem. This was completed in January of this year. Mr. Ralph Kissei of MSFC wrote the necessary software to control this sensor as well as the analysis logic to process the data. These models were integrated to TOM-B
and the system tested. Two methodologies were used to obtain the rate data. The first was to use the accelerometer output, while the second method was to compute the rate by computing the time derivatives of the position data.

After integrating the position feedback system, TOM-B works as expected. In a test run, the mobility base was instructed to translate along its x-axis by 5 cm, execute yaw of $30^\circ$ and then hold that position and orientation. The mobility base did just that, indicating that the software does indeed exercise closed-loop control over the mobility base. One disturbing observation is that to execute this maneuver, most of the thrusters are firing, an indication that further optimization of the control logic may be needed. The complete listing of TOM-B is given in Appendix 7.
List of References


APPENDIX 1

OMV Translational Equations of Motion
OMV Translational Equations Of Motion

Consider a target vehicle orbiting the earth with an angular velocity $\omega$ and an orbit radius of $R_0$. We can define a local vertical frame (LVF) at the center of gravity of this vehicle as shown in the figure below:

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

\[ M_c \ddot{R} = \mathbf{F}_g + F_c \]  \hspace{1cm} (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 \( F_c \) is the control force exerted on the vehicle from the on-board thrusters and jets. The objective of this exercise is to derive the equation of motion in terms of \( r \) and its time derivatives. Namely, we wish to express the motion of the chase vehicle (OMV) in local vertical frame. This choice turns out to be very convenient for docking maneuvers.

From the above figure, it is obvious that

\[ \mathbf{R} = \mathbf{R}_o + \mathbf{r}_E \]  \hspace{1cm} (2)

it follows that

\[ \ddot{\mathbf{R}} = \ddot{\mathbf{R}}_o + \ddot{\mathbf{r}}_E \]  \hspace{1cm} (3)

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

\[ \{ \frac{d}{dt} \}_E = \{ \frac{d}{dt} + \mathbf{w} \times \}_L \]

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

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

and

\[ \ddot{r}_E = \frac{d}{dt} (\dot{r}_L + \mathbf{w} \times r_L) + \mathbf{w} \times (\dot{r}_L + \mathbf{w} \times r_L) \]
\[ = \ddot{r}_L + \mathbf{w} \times \dot{r}_L + \mathbf{w} \times \dot{r}_L + \mathbf{w} \times (\mathbf{w} \times r_L) \]
\[ = \ddot{r}_L + 2\mathbf{w} \times \dot{r}_L + \mathbf{w} \times (\mathbf{w} \times r_L) \]  \hspace{1cm} (4)
From equations (3) and (4), we have:

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

Furthermore, for a circular orbit,

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

therefore,

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

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

\[ R = R_o + r \]
\[ R^2 = (R_o + r) \cdot (R_o + r) \]
\[ = R_o^2 + r^2 + 2R_o \cdot r \]
\[ = R_o^2 + 2R_o \cdot r \]
\[ = R_o^2 \left(1 + \frac{2(R_o \cdot r)}{R_o^2}\right) \]
so that

\[ R^{-3} = R_o^{-3} \left(1 + \frac{2(R_o \cdot r)}{R_o^2}\right)^{-3/2} \]
\[ \Rightarrow R^{-3} \left(1 - \frac{3(R_o \cdot r)}{R_o^2}\right) \]

Thus,

\[ F_g = -(G M_e M_c / R^3) R \]
\[ = -(G M_e M_c / R_o^3) (R_o + r) \left(1 - \frac{3(R_o \cdot r)}{R_o^2}\right) \]
\[ = -w^2 M_c (R_o + r) \left(1 - \frac{3(R_o \cdot r)}{R_o^2}\right) \]
\[ \approx -w^2 M_c (R_o + r - 3(R_o \cdot r / R_o^2) R_o) \]  \( \text{(6)} \)

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

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

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

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

which, after re-arranging, gives:

\[\ddot{r} = A - 2w \times \dot{r} - w^2 r - w \times (w \times r) + 3w^2 (R_o \cdot r/R_o^2) R_o\] (7)

Now, we shall state \(r, R_o\) and \(w\) 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,

\[r = [X, Y, Z]^T\]
\[R_o = [0, 0, R_o]^T\]
\[w = [0, w, 0]^T\] and
\[A = [A_x, A_y, A_z]^T\]

and it can easily be shown that:

\[2w \times \dot{r} = [2w \dot{Z}, 0, -2w \dot{X}]^T\]
\[w \times (w \times r) = [-w^2 X, 0, -w^2 Z]^T\]
\[3w(R_o \cdot r/R_o^2) R_o = [0, 0, 3w^2 Z]^T\] and
\[w^2 r = [w^2 X, w^2 Y, w^2 Z]^T\]

and substituting into equation (7) yields

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

\[ \ddot{x} = A_x - 2w\dot{y} \]
\[ \ddot{y} = A_y - w^2 y \]
\[ \ddot{z} = A_z + 2w\dot{x} + 3w^2 z \]

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

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

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

SUBROUTINE SIDEVIEW (H, X, P)

This procedure produces a side view of TOM_B and the attached mock-up module. The perspective is always in the direction of +1 axis of the body fixed coordinate.
(<< O M V P L O T >>)

D Line# 1 7

50 C system
51 C
52 C
53 C-----------------------------------------------
54 C
55 C
56 REAL * 8 H, X, P, C, S
57 REAL XFORM(3,3), SDFORM(3,3), VO(3,10), V(3,10)
58 REAL ROT(3,3), FLOOR(3,3), Vl(3,10)
59 REAL CC, DD, LL, RR, WW, TT
60 INTEGER FLAG, N, CLR, EF, EEF, PRTFG
61 C
62 COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
63 COMMON /MF/ XFORM, SDFORM, VO, VI
64 COMMON /ME/ EF, EEF, PRTFG
65 C
66 C
67 N = 10
68 AA = 1.0
69 C
70 C *** define mock-up module shape at origin
71 C
72 DO 100 K = 1, N
73 100 V(3, K) = 1.0
74 100 CONTINUE
75 C
76 V(1,1) = TT
77 V(2,1) = -DD
78 C
79 V(1,2) = -TT
80 V(2,2) = -DD
81 C
82 V(1,3) = -TT
83 V(2,3) = DD
84 C
85 V(1,4) = TT
86 V(2,4) = DD
87 C
88 C *** rotate it by P radians
89 C
90 CALL SINCOS (P, S, C)
91 CALL NOTHING (ROT, 3)
92 ROT(1,1) = C
93 ROT(1,2) = -S
94 ROT(2,1) = S
95 ROT(2,2) = C
96 CALL XMUL (ROT, V, 4)
97 C
98 C *** calculate translation
PX = CC + LL * C  
PY = H + LL * S

*** move the rotated module out there

CALL NOTHNG (ROT, 3)
ROT(1,3) = PX
ROT(2,3) = PY
CALL XMUL (ROT, V, 4)

*** now calculate the shape of the base

XX = X + CC
V(1,5) = CC
V(2,5) = H
V(1,6) = CC
V(2,6) = AA
V(1,7) = CC
V(2,7) = 0.
V(1,8) = -RR
V(2,8) = 0.
V(1,9) = -RR
V(2,9) = AA
V(1,10) = PX
V(2,10) = PY

*** Transform to floor coordinates

CALL NOTHING (FLOOR, 3)
FLOOR(1,3) = X
CALL XMUL (FLOOR, V, N)

*** transform to screen coordinates

CALL XMUL (SDFORM, V, N)

*** erase old picture

CALL DRWFILR (VO)

IF ((EF .EQ. 0) .AND. (EEF .NE. 0)) THEN
CLR = 0
CALL SDRAW (V1, N, CLR)
END IF
CALL SDRAW (V, N, CLR)
CALL MOVE (V, V1, N)
EEF = 1
RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>REAL*8</td>
<td>198</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>REAL*8</td>
<td>218</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>REAL</td>
<td>4</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>INTEGER*4</td>
<td>234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>REAL</td>
<td>8</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>EEF</td>
<td>INTEGER*4</td>
<td>4</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>EF</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>FLAG</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>FLOOR</td>
<td>REAL</td>
<td>158</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>REAL*8</td>
<td>0</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>202</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>REAL</td>
<td>12</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>INTEGER*4</td>
<td>194</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>REAL*8</td>
<td>8</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>PRTFG</td>
<td>INTEGER*4</td>
<td>8</td>
<td>/ME</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>REAL</td>
<td>226</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PY</td>
<td>REAL</td>
<td>230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROT</td>
<td>REAL</td>
<td>122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>REAL</td>
<td>16</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>REAL*8</td>
<td>210</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>SDFORM</td>
<td>REAL</td>
<td>36</td>
<td>/MF</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>REAL</td>
<td>24</td>
<td>/MF</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>REAL</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>REAL</td>
<td>72</td>
<td>/MF</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>REAL</td>
<td>192</td>
<td>/MF</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>REAL*8</td>
<td>4</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>XFORM</td>
<td>REAL</td>
<td>0</td>
<td>/MF</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE SDRAW (V, N, CLR)

This procedure draws the side view of TOM_B

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

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

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

*** draw mock-up module
CALL RCT (V, 0, CLR)
CALL PURGE
CALL GRFRDY
CALL HOME
RETURN
END
<table>
<thead>
<tr>
<th>D Line#</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X2</td>
<td>INTEGER*4</td>
<td>246</td>
</tr>
<tr>
<td>Y1</td>
<td>INTEGER*4</td>
<td>242</td>
</tr>
<tr>
<td>Y2</td>
<td>INTEGER*4</td>
<td>250</td>
</tr>
</tbody>
</table>

199 SPAGE
SUBROUTINE RCT (V, OFF, CLR)

This procedure draws a rectangle

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

DO 100 K = 1, 4
   J = K + OFF
   X(K) = V(1,J)
   Y(K) = V(2,J)
100 CONTINUE

CALL POLYGON(4, X, Y, CLR)
RETURN
END
SUBROUTINE PLOT (CMD)

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

INTEGER CMD(7), FLAG
REAL * 8 X, Y, T, UL, UA, H
REAL XFORM(3,3), SDFORM(3,3), CC, LL, DD, RR, WW, TT
REAL VO(3,10), V1(3,10)

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

UL = 10000.0
UA = UL

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

RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>REAL</td>
<td>4</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>CMD</td>
<td>INTEGER*4</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>REAL</td>
<td>8</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>FLAG</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>REAL*8</td>
<td>382</td>
<td>/MG</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>REAL</td>
<td>12</td>
<td>/MG</td>
<td></td>
</tr>
</tbody>
</table>
```
< O M V P L O T >>>

\*ne# 1 7
  REAL   16 /MG /  
  NEOM  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     
  ORM   REAL   0 /MF /  
  REAL*8 374     

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

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

REAL * 8 PX, PY, THETA, S, C
REAL V(3,10), VO(3,10), SDFORM(3,3)
REAL ROT(3,3), FLOOR(3,3), XFORM(3,3)
REAL CC, DD, LL, RR, WW, TT, V1(3,10)
INTEGER FLAG, N, CLR, EF, EEF, PRTFG

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

N = 10

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

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

*** transform to floor coordinates
CALL NOTHNG (FLOOR, 3)
FLOOR(1,3) = PX
FLOOR(2,3) = PY
CALL XMUL (FLOOR, V, N)
*** transform to screen coordinates
CALL XMUL (XFORM, V, N)
*** get ready to draw, but first erase old picture
CALL DRWFLR (V1)
IF ((EF .EQ. 0) .AND. (EEF .NE. 0)) THEN
CLR = 0
CALL DRAW (VO, N, CLR)
END IF
CLR = 1
CALL DRAW (V, N, CLR)
CALL NOVE (V, VO, N)
EEF = 1
RETURN
END
SUBROUTINE MOVE (V, VO, N)

This procedure saves the shape vector V

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

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

RETURN
END
SUBROUTINE NOTHING (A, N)

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

REAL A(N,N)

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

RETURN
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>626</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>618</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE XMUL (R, V, N)

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

DO 100 COL = 1, N

DO 200 ROW = 1, N

S = 0.0

DO 300 J = 1, 3

S = S + R(ROW,J) * V(J, COL)

CONTINUE

T(ROW) = S

CONTINUE

DO 400 L = 1, 3

V(L, COL) = T(L)

CONTINUE

RETURN

END

Name    Type               Offset  P Class

ROW    INTEGER*4          646
J       INTEGER*4          662
I       INTEGER*4          666
J       INTEGER*4          668
I       INTEGER*4          670
i       INTEGER*4          672
J       REAL              654
I       REAL              658
J       REAL              634
I       REAL              638

ORIGINAL PAGE IS OF POOR QUALITY
SUBROUTINE ORGPOS (V, N)

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

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

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

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

*** set up shape matrix $V$

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

Name          Type     Offset P Class
CC    REAL        4       /MG       /
CL    REAL        702       /MG       /
CORN    INTEGER*4  678       /MG       /
DD    REAL        8       /MG       /
EEF   INTEGER*4   4       /ME       /
EF    INTEGER*4   0       /ME       /
FLAG  INTEGER*4   0       /MG       /
K     INTEGER*4   694       /MG       /
LL    REAL        12       /MG       /
V     INTEGER*4   4       *         /
PRTFG INTEGER*4   8       /ME       /
RR    REAL        16       /MG       /
SDFORM REAL      36       /MF       /
TT    REAL        24       /MG       /
V     REAL        0       *         /
VO    REAL        72       /MF       /
/1    REAL        192      /MF       /
V     REAL        670       /MG       /
W     REAL        20       /MG       /
FORM  REAL        0       /MF       /
SUBROUTINE INITPL

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

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

EEF = 0
OPEN (7, FILE = 'SIZE.DAT')
READ (7, 10) CC, DD, LL, RR, WW, TT
DO 200 K = 1, 2
   READ (7, 20) (CORNR(K,J), J=1, 2)
200 CONTINUE
W(1) = 12.2
W(2) = 24.4
CALL CORDX (CORNR, XFORM, W)
DO 300 K = 1, 2
   READ (7, 20) (CORNS(K,J), J=1, 2)
300 CONTINUE
W(1) = 12.2
W(2) = 6.096
CALL CORDX (CORNS, SDFORM, W)
READ (7, 20) EF
READ (7, 20) FLAG
READ (7, 20) PRTFG
CLOSE (7)
FLG = 1
*** calculate floor shape
536 C
537 JW = 30
538 JL = 44
539 C
540 IF (FLAG .EQ. 0) THEN
541 J1 = CORNR(1,1)
542 L1 = CORNR(1,2)
543 J2 = CORNR(2,1)
544 L2 = CORNR(2,2)
545 JJ = (L2 - L1 + 1) / 2
546 V1(1,1) = J1
547 V1(2,1) = L1
548 V1(1,2) = J2
549 V1(2,2) = L1
550 V1(1,3) = J2
551 V1(2,3) = L2
552 V1(1,4) = J1
553 V1(2,4) = L2
554 V1(1,5) = J1
555 V1(2,5) = L2 + JW - JJ
556 V1(1,6) = J1 - JL
557 V1(2,6) = L2 + JW - JJ
558 V1(1,7) = J1 - JL
559 V1(2,7) = L2 - JL - JJ
560 V1(1,8) = J1
561 V1(2,8) = L2 - JL - JJ
562 V1(1,9) = -1000.0
563 V1(2,9) = -1000.0
564 ELSE
565 J1 = CORNS(1,1)
566 L1 = CORNS(1,2)
567 J2 = CORNS(2,1)
568 L2 = CORNS(2,2)
569 VO(1,1) = J1 - JL
570 VO(2,1) = L2 + 1
571 VO(1,2) = J2 + JL
572 VO(2,2) = L2 + 1
573 VO(1,3) = -1000.0
574 VO(2,3) = -1000.0
575 END IF
576 C
577 CALL GRAFICS
578 RETURN
579 10 FORMAT (F15.8)
580 20 FORMAT (I3)
581 END
<table>
<thead>
<tr>
<th>Line#</th>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CC</td>
<td>REAL</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>ORNR</td>
<td>INTEGER*4</td>
<td>714</td>
</tr>
<tr>
<td></td>
<td>ORNS</td>
<td>INTEGER*4</td>
<td>730</td>
</tr>
<tr>
<td></td>
<td>DD</td>
<td>REAL</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>EF</td>
<td>INTEGER*4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>INTEGER*4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>FLAG</td>
<td>INTEGER*4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>FLG</td>
<td>REAL</td>
<td>758</td>
</tr>
<tr>
<td></td>
<td>J1</td>
<td>INTEGER*4</td>
<td>770</td>
</tr>
<tr>
<td></td>
<td>J2</td>
<td>INTEGER*4</td>
<td>773</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>INTEGER*4</td>
<td>786</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>INTEGER*4</td>
<td>766</td>
</tr>
<tr>
<td></td>
<td>JW</td>
<td>INTEGER*4</td>
<td>762</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTEGER*4</td>
<td>746</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>INTEGER*4</td>
<td>774</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>INTEGER*4</td>
<td>782</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>REAL</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>RTFG</td>
<td>INTEGER*4</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>REAL</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>SDFORM</td>
<td>REAL</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>REAL</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>REAL</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>/1</td>
<td>REAL</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>REAL</td>
<td>706</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>REAL</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>.FORM</td>
<td>REAL</td>
<td>0</td>
</tr>
</tbody>
</table>

ORIGINAL PAGE IS OF POOR QUALITY
SUBROUTINE DRWFLR (V)

This subroutine draws the floor portion of graphics

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

CT = 1

REPEAT
  K = CT
  X(K) = V(1,K)
  Y(K) = V(2,K)
  CT = CT + 1
  IF (V(1,CT) .GE. -100.0) GO TO 100
UNTIL V(1,CT) < -100.0

CALL POLYGN (K, X, Y, 1)
RETURN
END
SUBROUTINE DRAW (V, N, CLR)

This procedure actually draws the top view of TOM_B.

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

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

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

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

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

RETURN
END
<table>
<thead>
<tr>
<th>Line#</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INTEGER*4 892</td>
</tr>
<tr>
<td></td>
<td>INTEGER*4 900</td>
</tr>
<tr>
<td></td>
<td>INTEGER*4 896</td>
</tr>
<tr>
<td></td>
<td>INTEGER*4 904</td>
</tr>
</tbody>
</table>

656 $PAGE$
SUBROUTINE CORDX (C, T, W)

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

**::

*** set up transformation matrix T

INTEGER C(2,2)
REAL T(3,3), W(2)

T(1,3) = C(1,1)
T(2,3) = C(2,2)
T(3,3) = 1.0

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

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

RETURN
END

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

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

**SUBROUTINE PURGE**

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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR*1</td>
<td>4</td>
<td>/GMBD</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>/GMBD</td>
</tr>
</tbody>
</table>
SUBROUTINE GRFRDY

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

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
SUBROUTINE SETFB (FG, BG)

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

INTEGER GRF, FG, BG
CHARACTER ESC
COMMON /GMBD/ GRF, ESC
WRITE (GRF, 10) ESC, FG, BG
RETURN
FORMAT (' ', A1, '!c', I2, ';', I2, 'c')
END

name    Type        Offset P Class

INTEGER*4  4  *
C: INTEGER*4  0  *
SC INTEGER*4  4 /GMBD /
F INTEGER*4  0 /GMBD /

742 PAGE
SUBROUTINE GRAFICS

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

INTEGER GRF
CHARACTER ESC
COMMON /GMBD/ GRF, ESC

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

FORMAT (' ', A1, '[[10m\"
FORMAT (' ', A1, '[[640;352;2g\"
FORMAT (' ', A1, '[[21;24r\"
END

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

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

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

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

FORMAT (A1)
FORMAT ('Press <CR> to continue ... '

TYPE Offset P Class

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

PAGE
SUBROUTINE TEXT

This procedure returns the system to text mode

INTEGER GRF
CHARACTER ESC
COMMON /GMBD/ GRF, ESC
WRITE (GRF, 10) ESC
RETURN
FORMAT (' ', A1, '[:150;25;1a\n')
END

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

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

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

INTEGER GRF, X1, Y1, X2, Y2, COLOR
CHARACTER ESC
COMMON /GMBD/ GRF, ESC
WRITE (GRF, 10) ESC, X1, Y1, X2, Y2, COLOR
FORMAT (' ', Ai, '!', 4(I3,';'), I3, '1"
END
SUBROUTINE HIDE LN (X1, Y1, X2, Y2, COLOR)

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

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

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

FORMAT (' ', A1, '[]', 4(I3, ':'), I3, 'S\')
SUBROUTINE POLYGN (N, X, Y, COLOR)

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

INTEGER GRF, X(N), Y(N), COLOR
CHARACTER ESC
COMMON /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
FORMAT (', A1, '!', ')
FORMAT ( '(', 2(I3, ';'), ')
FORMAT ( 'I3, ', 'p')
END
D Line 1
868 C
869 C
870 SUBROUTINE HOME
871 C
872 C
873 C THIS SUBROUTINE HOMES THE CURSOR
874 C
875 C
876 INTEGER GRF
877 CHARACTER ESC
878 C
879 COV/ISION /GMBD/ GRF, ESC
880 C
881 WRITE (GRF, 10) ESC
882 RETURN
883 10 FORMAT (' ', A1, '[ 1;1 f\')
884 END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC</td>
<td>CHAR*1</td>
<td>4 /GMBD /</td>
</tr>
<tr>
<td>GRF</td>
<td>INTEGER*4</td>
<td>0 /GMBD /</td>
</tr>
<tr>
<td>Type</td>
<td>Size</td>
<td>Class</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td>312</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBROUTINE</td>
</tr>
</tbody>
</table>

Pass One  No Errors Detected
885 Source Lines
APPENDIX 3

OMV Data Files Used During Development
File: INITCON.DAT

This file contains all the needed initial conditions

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

This file contains all the model parameters needed by OMV

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

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

<table>
<thead>
<tr>
<th>CC</th>
<th>IN METERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5588</td>
<td></td>
</tr>
<tr>
<td>0.762</td>
<td></td>
</tr>
<tr>
<td>11.668</td>
<td></td>
</tr>
<tr>
<td>2.4384</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LL</th>
<th>IN METERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7048.37</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AA</th>
<th>IN METERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3713.95</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HH</th>
<th>IN METERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3713.95</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IINV(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IINV(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IINV(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
File: HNDSSL.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
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 4

OMV Mathematical Model (OMM) Source Listing
OMV SIMULATION MODEL

by

Dr. W. Teoh

U A H
Huntsville
1984

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

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

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

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

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

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

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

REAL * 8  X(3), V(3), E(3), A(3), W(3), Q(4)
REAL * 8  POS(3), VEL(3), EUL(3), OMEGA
REAL * 8  III(3), S(14), MASS, CYCLE
INTEGER  CMD(7), IN, FLAG, MODE, STEP
INTEGER * 4  TIME
COMMON  /MC/ III, MASS, CYCLE, MODE, STEP
COMMON  /PC/ POS, VEL, EUL, OMEGA
Line#    7  
50 C    *** system initialization
51 C
52 IN = 2
53 TIME = -1
54 CALL OMVMDL(IN)
55 OPEN (IN, FILE = 'HNDSGL.DAT')
56 C
57 *** *** Note : this invokes graphics routines, and can be eliminated if no graphics output.
58 C
59 CALL INITPL
60 C
61 *** calculate the initial quaternions at the start of the simulation and read hand controller
62 C
63 CALL DETQ (EUL, Q)
64 CALL HNDCTL(IN, FLAG, A, W)
65 CALL MATCH(EUL, POS, VEL, E, X, V, 3)
66 CALL STATE(Q, S, W)
67 CALL SVX(S, CMD, MODE)
68 CALL OUTPUT(A, W, X, V, E, Q, S, CMD, TIME)
69 TIME = 0
70 C
71 *** main processing loop
72 C
73 WHILE (FLAG = 0) DO
74 IF (FLAG .NE. 0) GOTO 900
75 C
76 *** copy initial state into work vectors and use these work vectors for solving the equations of motion
77 C
78 CALL MOTION(X, V, E, A, W, Q)
79 C
80 *** update dynamic state
81 C
82 CALL MATCH(E, X, V, EUL, POS, VEL, 3)
83 C
84 *** calculate state vector and pass it on to the State Transformation module
85 C
86 CALL STATE(Q, S, W)
87 CALL SVX(S, CMD, MODE)
88 CALL OUTPUT(A, W, X, V, E, Q, S, CMD, TIME)
89 C
90 *** poll hand controller and get the next set of signals
91 C
92 CALL HNDCTL(IN, FLAG, A, W)
93 C
94 GOTO 100
95 C
96 END WHILE
### Line 99-109

```
CONTINUE C
*** This is also a call to the graphics package
CALL QUITGM C
*** Grand exit, stage left
STOP C
END
```

### Variable Information

<table>
<thead>
<tr>
<th>Line</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>REAL*8</td>
<td>242</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>266</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>REAL*8</td>
<td>32</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>74</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>REAL*8</td>
<td>48</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>302</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>REAL*8</td>
<td>0</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>294</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>REAL*8</td>
<td>24</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>40</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>REAL*8</td>
<td>72</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>98</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>REAL*8</td>
<td>130</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>26</td>
<td>MC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>50</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>2</td>
<td>PC</td>
<td></td>
</tr>
</tbody>
</table>

110 $PAGE
SUBROUTINE OMMDL (IN)

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

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

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

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

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

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

*** calculate orbital frequency
CALL ANGFRE (ORBIT, OMEGA)
RETURN
CALL ANGFRE (F15.8)
CALL ANGFRE (I1)
CALL ANGFRE (I2)
END

name  Type Offset P Class
C REAL*8 0 /DC /
V REAL*8 32 /MC /
U REAL*8 48 /PC /
I REAL*8 0 /MC /
IN INTEGER*4 0 *
V REAL*8 306 /MC /
S REAL*8 24 /MC /
D INTEGER*4 40 /MC /
E REAL*8 72 /PC /
R REAL*8 330 /PC /
E REAL*8 0 /PC /
L INTEGER*4 44 /MC /
E REAL*8 24 /PC /
B REAL*8 24 /DC /
SUBROUTINE ANGFR(E(ORB, W))

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

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

ALT = ORB * 0.001
R3 = (6.370 + ALT)**3
W = DSQRT (398.86 / R3) * 0.001
RETURN
END
SUBROUTINE VECTOR (M, A, N)

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

INTEGER M, N
REAL * 8 A(N)

DO 100 K = 1, N
       READ (M, 10) A(K)
100 CONTINUE
RETURN
10 FORMAT (F15.8)
END

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

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

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

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

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

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

*** no error, generate matrices A and W
CALL FUDGE (A, ACC, SL)
CALL FUDGE (W, WWB, SA)
RETURN
CONTINUE

*** error condition
FLAG = 1
RETURN
FORMAT (20I1)
END
<table>
<thead>
<tr>
<th>Line#</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>0</td>
<td>/DC</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>INTEGER*4</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>414</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>390</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>12</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>24</td>
<td>/DC</td>
<td></td>
</tr>
</tbody>
</table>

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

*** Sets appropriate components based on SL

INTEGER SL(6), T, K, J
REAL  ACC(3), A(3), X

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

D Line# 1 7
339 C    S(14) = MASS
340 C    RETURN
342 C    END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>642</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>570</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CYCLE</td>
<td>REAL*8</td>
<td>32</td>
<td>/MC</td>
</tr>
<tr>
<td>UL</td>
<td>REAL*8</td>
<td>48</td>
<td>/PC</td>
</tr>
<tr>
<td>II</td>
<td>REAL*8</td>
<td>0</td>
<td>/MC</td>
</tr>
<tr>
<td>L</td>
<td>REAL*8</td>
<td>546</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>REAL*8</td>
<td>498</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>REAL*8</td>
<td>522</td>
<td></td>
</tr>
<tr>
<td>MASS</td>
<td>REAL*8</td>
<td>24</td>
<td>/MC</td>
</tr>
<tr>
<td>MODE</td>
<td>INTEGER*4</td>
<td>40</td>
<td>/MC</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>714</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEGA</td>
<td>REAL*8</td>
<td>72</td>
<td>/PC</td>
</tr>
<tr>
<td>POS</td>
<td>REAL*8</td>
<td>0</td>
<td>/PC</td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>REAL*8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>INTEGER*4</td>
<td>44</td>
<td>/MC</td>
</tr>
<tr>
<td>EL</td>
<td>REAL*8</td>
<td>24</td>
<td>/PC</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

343 SPACE

PRECEDING PAGE BLANK NOT FILMED
SUBROUTINE PUT (.., S, A, M)

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

REAL*8   S(14)
REAL*8   A(M)

DO 100 K = 1, M
    N = N + 1
    S(N) = A(K)
100 CONTINUE
RETURN
END
SUBROUTINE DOTPRD (A, B, C, N)

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

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

Name  Type    Offset  P  Class
A    REAL*8    0    *
B    REAL*8    4    *
C    REAL*8    8    *
K    INTEGER*4   726  *
N    INTEGER*4   12  *
SUBROUTINE DETQ (E, Q)

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

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

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

Q(1) = S1 * C3 * C2 + C1 * S3 * S2
Q(2) = S1 * S3 * C2 + C1 * C3 * S2
Q(3) = C1 * S3 * C2 - S1 * C3 * S2
Q(4) = C1 * C3 * C2 - S1 * S3 * S2

RETURN
END
SUBROUTINE SINCOS (THETA, S, C)

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

REAL * 8    THETA, S, C, A

C = DCOS(THETA)
S = DSIN(THETA)
RETURN
END
REAL * 8   POS(3), VEL(3), EUL(3), OMEGA
REAL * 8   X(3), V(3), E(3), A(3), W(3), Q(4)   
REAL * 8   CIN(3,3), C(3,3), AA(3,10), B(3), QQ(4)
REAL * 8   WW(3), PI, TWO
REAL * 8   III(3), MASS, CYCLE
INTEGER     MODE, STEP
COMMON /MC/   III, MASS, CYCLE, MODE, STEP
COMMON /PC/   POS, VEL, EUL, OMEGA
H = CYCLE / FLOAT(STEP)
N = STEP
PI = 355.0 / 113.0
TWO = PI * 2.0
*** Divide 1 major cycle into N equal subintervals and
*** determine the OMV state for each interval
DO 100 KK = 1, N
*** Update orientation
DO 200 J = 1, 3
    WW(J) = W(J) * H
    E(J) = E(J) + WW(J)
    IF (E(J) .GT. TWO) E(J) = E(J) - TWO
CONTINUE
*** Calculate quaternion for this rotation, and transform
*** it to local vertical frame with respect to initial frame
CALL DETQ( WW, QQ)
CALL UPDQ (Q, QQ)
*** from the direction cosine matrix, calculate the
*** acceleration vector in LVF and store it in the
*** acceleration matrix AA
CALL DCSINV (Q, CIN)
D Line# 1 7
1 482 CALL DMUL (CIN, A, B, 3)
1 483 CALL STORE (B, AA, KK)
1 484 100 CONTINUE
1 485 C
1 486 C *** Solve the equation of motion using the Adam-Brashford
1 487 C *** method
1 488 C
1 489 CALL SOLVE (X, V, AA, N, H, OMEGA)
1 490 C
1 491 RETURN
1 492 END

Name Type Offset P Class
A REAL*8 12 *
AA REAL*8 990
B REAL*8 1230
E REAL*8 918
C 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
M ASS 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
SUBROUTINE MATCH (A, B, C, P, Q, R, N)

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

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

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

RETURN
END

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

SPACE
SUBROUTINE STORE (AAA, AA, K)

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>AAA</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>1294</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

535 $PAGE$
SUBROUTINE SOLVE(X,V,A,N,II,W)

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>REAL*8</td>
<td>0</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>REAL*8</td>
<td>FUNCTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLAG</td>
<td>LOGICAL*4</td>
<td>1298</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>REAL</td>
<td>16</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>HD12</td>
<td>REAL*8</td>
<td>960</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>INTEGER*4</td>
<td>1302</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>1314</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>1306</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>INTEGER*4</td>
<td>12</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>REAL*8</td>
<td>312</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>REAL*8</td>
<td>20</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>WX2</td>
<td>REAL*8</td>
<td>936</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>WXW</td>
<td>REAL*8</td>
<td>944</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>WXWX3</td>
<td>REAL*8</td>
<td>952</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

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

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

```fortran
REAL * 8 X(3), V(3), AA(3,13), U(6,13), WX2, WXW, WXWX3
REAL * 8 CWT, SWT, T, W, HD12
COMMON /BLOCK/ AA, U, WX2, WXW, WXWX3, HD12
WXW = W*W
WXWX3 = 3*WXW
WX2 = 2*W
HD12 = DBLE(H)/12.0
DO 100 K = 1, 3
   U(2*K-1,3) = X(K)
   U(2*K ,3) = V(K)
   CONTINUE
DO 300 I = 1, 2
   T = H*(I-3)
   CWT = DCOS(W*T)
   SWT = DSIN(W*T)
   U(1,I) = X(1) + V(1)*(4*CWT-3.0)/W + 6*X(3)*(SWT-W*T) + 2*V(3)*(CWT-1.0)/W + 2*V(3)*SWT
   U(2,I) = V(1)*(4*CWT-3.0) + 6*W*X(3)*(CWT-1.0) - 2*V(3)*SWT
   U(3,I) = X(2)*CWT + V(2)*SWT/W
   U(4,I) = -X(2)*W*SWT + V(2)*CWT
   U(5,I) = 2*V(1)*(1.0-CWT)/W + X(3)*(4.0-3*CWT) + V(3)*SWT/W
   U(6,I) = 2*V(1)*SWT + 3*X(3)*W*SWT + V(3)*CWT
   CONTINUE
RETURN
END
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>/BLOCK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D Line# 1  
1  7

CWT  REAL*8  1362

DBLE  INTRINSIC

DCOS  INTRINSIC

DSIN  INTRINSIC

I    REAL   12 *

ID12  REAL*8  960 /BLOCK /

I    INTEGER*4  1350

J    INTEGER*4  1346

K    INTEGER*4  1342

SWT  REAL*8  1370

T    REAL*8  1354

J    REAL*8  312 /BLOCK /

V    REAL*8  4 *

W    REAL*8  8 *

\sqrt{X}  REAL*8  936 /BLOCK /

\sqrt{XW}  REAL*8  944 /BLOCK /

\sqrt{WXW}  REAL*8  952 /BLOCK /

X    REAL*8  0 *
FUNCTION F(J,I)

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

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

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

CONTINUE

F = U(2,I)
RETURN

CONTINUE

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

CONTINUE

F = U(4,I)
RETURN

CONTINUE

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

CONTINUE

F = U(6,I)
RETURN

CONTINUE

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

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

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

COMMON /ME/ EF, EEF, PRTFG

TIME = TIME + 1
T = (T / 10) * 10 - TIME
IF ((T .NE. 0) .OR. (PRTFG .EQ. 0)) RETURN
IF (PRTFG .EQ. 1) GO TO 100
OPEN (4, FILE = 'LPT1:'), WRITE (4, 15) TIME / 10
WRITE (4, 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)
IF (PRTFG .NE. 2) CALL PLOT (CMD)

RETURN
FORMAT (' A, W =', 3F10.6, 3X, 3F10.6)
FORMAT (' ', 7I10)
FORMAT (' TIME =', I6, ' Seconds')
FORMAT (' X, V =', 3F10.6, 3X, 3F10.6)
FORMAT (' E, W =', 3F10.6, 3X, 3F10.6/
FORMAT (' S =', 3F10.6, 3X, 3F10.6/
FORMAT (' ', 3F10.3/
FORMAT (' ', 4F10.6, 3X,F10.3/
FORMAT (' CMD =', 7I10)
FORMAT (1HO)
END
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>CMD</td>
<td>INTEGER*4</td>
<td>28</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>REAL*8</td>
<td>16</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>EEF</td>
<td>INTEGER*4</td>
<td>4</td>
<td>/ME/</td>
<td></td>
</tr>
<tr>
<td>EF</td>
<td>INTEGER*4</td>
<td>0</td>
<td>/ME/</td>
<td></td>
</tr>
<tr>
<td>PRTFG</td>
<td>INTEGER*4</td>
<td>8</td>
<td>/ME/</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>20</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>REAL*8</td>
<td>24</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>INTEGER*4</td>
<td>1378</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>INTEGER*4</td>
<td>32</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>REAL*8</td>
<td>12</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

100 CONTINUE
RETURN
END
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>1714</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>1730</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>12</td>
<td>*</td>
</tr>
<tr>
<td>REAL*8</td>
<td>1722</td>
<td></td>
</tr>
</tbody>
</table>

747 SPAGE
SUBROUTINE UPDQ (Q, QQ)

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

```
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)
```

RETURN

END

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>REAL*8</td>
<td>1738</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>REAL*8</td>
<td>1746</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>REAL*8</td>
<td>1754</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>REAL*8</td>
<td>1762</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

$PAGE
SUBROUTINE DCSINV(Q, C)

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

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

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

Q11 = Q1 * Q1
Q22 = Q2 * Q2
Q33 = Q3 * Q3
Q44 = Q4 * Q4

Q12 = 2.0 * Q1 * Q2
Q13 = 2.0 * Q1 * Q3
Q23 = 2.0 * Q2 * Q3
Q14 = 2.0 * Q1 * Q4
Q24 = 2.0 * Q2 * Q4
Q34 = 2.0 * Q3 * Q4

C(1,1) = Q11 - Q22 - Q33 + Q44
C(2,2) = -Q11 + Q22 - Q33 + Q44
C(3,3) = -Q11 - Q22 + Q33 + Q44

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

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

826 $PAGE$
<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFRE</td>
<td>968</td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>DCK</td>
<td>48</td>
<td>COMMON</td>
</tr>
<tr>
<td>SINV</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>JL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>XPRD</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>REAL*8</td>
<td></td>
<td>FUNCTION</td>
</tr>
<tr>
<td>DGE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>NDCTL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TPL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>INIT</td>
<td></td>
<td>PROGRAM</td>
</tr>
<tr>
<td>INCH</td>
<td>48</td>
<td>COMMON</td>
</tr>
<tr>
<td>VTION</td>
<td>12</td>
<td>COMMON</td>
</tr>
<tr>
<td>VMDL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TPUT</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>D</td>
<td>80</td>
<td>COMMON</td>
</tr>
<tr>
<td>DOT</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>HITCM</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>NCOS</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>VLE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>VATE</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>FORR</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>TV</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>PDQ</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td>VECTR</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
</tbody>
</table>

Pass One  No Errors Detected
826 Source Lines

ORIGINAL PAGE IS
OE POOR QUALITY
APPENDIX 5

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

Real*8 XE(3), VE(3), X(3), V(3), A(3,10), W
Real*8 XO(3), VO(3)
DATA A/30*0.0/
DATA N,H /10, 0.01/

WRITE (*, 30)
READ (*,32) W
get initial conditions
CALL GETINT (XO, VO, 3)

DO 100 K = 1, 3
  X(K) = XO(K)
  V(K) = VO(K)
100 CONTINUE

DO 10 I = 1, 36000
  T = 0.1*I

*** calculate the analytical solution
CALL EXACT(T,XE,VE,W,X0,VO)
*** now get the numerical solution
CALL SOLVE(X,V,A,N,H,W)

*** output every 60 seconds
JJ = (I / 600) * 600
IF (JJ .EQ. I) THEN
WRITE(*,20) T,XE,VE
WRITE(*,22) T,X,V
WRITE (*, 22)
END IF
CONTINUE

FORMAT (F7.1, 6F12.6)
FORMAT (' ORBITAL RATE '")
FORMAT (1H )
FORMAT (F15.8)
STOP
END
SUBROUTINE EXACT(T, XE, VE, W, X, V)

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

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

87 C
88 WT = W * T
89 SWT = DSIN(WT)
90 CWT = DCOS(WT)

92 XE(1) = X(1) + (4 * SWT - 3 * WT) * V(1) / W + 6 * (SWT - WT) * X(3)
93 1 + 2 * (CWT - 1) * V(3) / W
94 XE(2) = CWT * X(2) + SWT * V(2) / W
95 XE(3) = 2 * (1 - CWT) * V(1) / W + (4 - 3 * CWT) * X(3)
96 1 - SWT * V(3) / W

97 VE(1) = (4 * CWT - 3) * V(1) + 6 * W * (CWT - 1) * X(3)
98 1 - 2 * SWT * V(3)
99 VE(2) = CWT * V(2) - W * SWT * X(2)
100 VE(3) = 2 * SWT * V(1) + 3 * W * SWT * X(3) + CWT * V(3)
101 RETURN
102 END
** This subroutine produces the numerical solution to the system of equations of motion.

```fortran
LOGICAL FLAG
REAL*8 X(3), V(3), A(3,10), AA(3,13), U(6,13)
REAL*8 WX2, WXW, WXWX3, HD12, F, W
COMMON /BLOCK/ AA, U, WX2, WXW, WXWX3, HD12
DATA FLAG / .TRUE. /
pack user supplied nonhomogeneous part of DE into the higher part of AA
DO 10 I = 1,10
   DO 10 K = 1,3
      AA(K,I+3) = A(K,I)
   CONTINUE
   if this is the first call to solve (FLAG = T), then initialize
IF (FLAG) THEN
   CALL INNIT(X,V,W,H)
   FLAG = .FALSE.
   END IF
use the Adam-Brashford 3-step method to advance the solution h time units. Place the solution back into X and V.
DO 100 I = 4,N+3
   DO 100 J = 1,6
      U(J,I) = U(J,I-1) + HD12*(23*F(J,I-1)-16*F(J,I-2)+5*F(J,I-3))
   CONTINUE
   X(1) = U(1,N+3)
```
reset U and AA for the next call to SOLVE

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

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

RETURN
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td></td>
<td>FUNCTION</td>
<td></td>
</tr>
<tr>
<td>LOGICAL*4</td>
<td>496</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>960</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>504</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>312</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>936</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>944</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>952</td>
<td>/BLOCK</td>
<td></td>
</tr>
<tr>
<td>REAL*8</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE INIT(X,V,W,H)

This is the initialization routine which is called only once

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

WXW = W*W
WXWX3 = 3*WXW
WX2 = 2*W
HD12 = DBLE(H)/12.0

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

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

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

User supplied function

REAL*8 AA(3,13),U(6,13),WX2,WWX,WWX3,HD12,F
COMMON /BLOCK/ AA,U,WWX,WWX3,HD12
GO TO (10,20,30,40,50,60), J
CONTINUE
F = U(2,I)
RETURN
CONTINUE
F = -WX2*U(6,I) + AA(1,I)
RETURN
CONTINUE
F = U(4,I)
RETURN
CONTINUE
F = -WXW*U(3,I) + AA(2,I)
RETURN
CONTINUE
F = U(6,I)
RETURN
CONTINUE
F = WX2*U(2,I) + WWX3*U(5,I) + AA(3,I)
RETURN
END

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td>0</td>
<td>/BLOCK</td>
</tr>
<tr>
<td>REAL*8</td>
<td>960</td>
<td>/BLOCK</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>REAL*8</td>
<td>312</td>
<td>/BLOCK</td>
</tr>
<tr>
<td>REAL*8</td>
<td>936</td>
<td>/BLOCK</td>
</tr>
<tr>
<td>REAL*8</td>
<td>944</td>
<td>/BLOCK</td>
</tr>
<tr>
<td>REAL*8</td>
<td>952</td>
<td>/BLOCK</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>258 SPACE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Line# 1  7
Pass One  No Errors Detected
285 Source Lines

Microsoft FORTRAN77 V3.13 8/05/83

PRECEDING PAGE BLANK NOT FILMED
APPENDIX 6

State Vector Transformation (SVX) Source Listing
STATE VECTOR TRANSFORMATION MODULE (SVX)
by

Dr. W. Teoh
U A H 1984

-------------------------------------------------------------

SUBROUTINE SVX (S, CMDRAW, MODE)

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

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

Summary of the state vector components are as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X position of target vehicle from the chase vehicle in LVF</td>
</tr>
<tr>
<td>2</td>
<td>Y relative velocity of the two vehicles in LVF</td>
</tr>
<tr>
<td>3</td>
<td>Z angular momentum vector in LVF</td>
</tr>
<tr>
<td>4</td>
<td>VX attitude quaternions in body frame</td>
</tr>
<tr>
<td>5</td>
<td>VY instantaneous mass in kg.</td>
</tr>
<tr>
<td>6</td>
<td>VZ</td>
</tr>
<tr>
<td>7</td>
<td>LX</td>
</tr>
<tr>
<td>8</td>
<td>LY</td>
</tr>
<tr>
<td>9</td>
<td>LZ</td>
</tr>
<tr>
<td>10</td>
<td>Q1</td>
</tr>
<tr>
<td>11</td>
<td>Q2</td>
</tr>
<tr>
<td>12</td>
<td>Q3</td>
</tr>
<tr>
<td>13</td>
<td>Q4</td>
</tr>
<tr>
<td>14</td>
<td>M</td>
</tr>
</tbody>
</table>
Summary of command string components:

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

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

```fortran
REAL * 8 S(14), 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
```

INTEGER CMDRAW(7), COUNT, MODE

*** load-time initialization

DATA COUNT /0/

*** decompose state vector and process it

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

*** initialization before start

CALL ZERO (XO, 3)

*** read parameters

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

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

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

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

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

IF (MODE .NE. 1) GO TO 400

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

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

*** transform to TOM_B position in floor coordinates
QQ = CC + LL * DCOS(PITCH)

*** X-component
TX = XM(1) - QQ * DCOS(YAW)
CMDRAW(2) = JFIX (TX * UL)
*** Y-component

TY = XM(2) - QQ * DSIN(YAW)
CMDRAW(3) = JFIX(TY * UL)

*** Z-component

Z = XM(3) - LL * DSIN(PITCH)
CMDRAW(4) = JFIX(Z * UL)

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

RETURN

IF (MODE .NE. 0) GO TO 900

*** rate control

CALL QTRPY (Q, ROLL, PITCH, YAW)

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

CALL DIRCOS (A, Q)
CALL MMUL (A, L, LB, 3)

*** compute body rate

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

*** construct orientation part of command string

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

*** compute velocity of TOM_B in floor coordinates

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

*** X-component of velocity in floor coordinate
D Line# 1 7
197 C
198 VX = V(1) + Q1 * CY + Q2 * SY
199 CMDRAW(2) = JFIX (VX * UL)
200 C
201 C
202 C
203 VY = V(2) + Q1 * SY - Q2 * CY
204 CMDRAW(3) = JFIX (VY * UL)
205 C
206 C
207 C
208 VZ = V(3) - LL * DCOS(PITCH) * PITDOT
209 CMDRAW(4) = JFIX (VZ * UL)
210 RETURN
211 C
212 900 CONTINUE
213 C
214 C
215 *** We have an un-recognizable code, default to 1 for
216 C
217 *** position control
218 C
219 MODE = 1
220 10 FORMAT (4F10.2)
221 20 FORMAT (F15.8)
222 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      REAL*8  418
DSIN      REAL*8  566
INV       REAL*8  442
L          INTEGER*4  574
LL        REAL*8  370
LB        REAL*8  394
LLE       REAL*8  550
MODE      INTEGER*4  8 *
ITCH      REAL*8  602
ITDOT     REAL*8  658
J          REAL*8  154
J1        REAL*8  674
<table>
<thead>
<tr>
<th>Line#</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REAL*8</td>
<td>682</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>618</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>242</td>
</tr>
<tr>
<td></td>
<td>D OT REAL*8</td>
<td>650</td>
</tr>
<tr>
<td></td>
<td>L REAL*8</td>
<td>594</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>690</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>626</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>634</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>586</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>578</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>706</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>714</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>722</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>OLD REAL*8</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>d REAL*8</td>
<td>610</td>
</tr>
<tr>
<td></td>
<td>w DOT REAL*8</td>
<td>666</td>
</tr>
<tr>
<td></td>
<td>REAL*8</td>
<td>642</td>
</tr>
</tbody>
</table>
This procedure decomposes the State vector S into its components which are also vectors. They have the following meaning:

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

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

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

RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>REAL*8</td>
<td>12</td>
<td>*</td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>16</td>
<td>*</td>
</tr>
<tr>
<td>S</td>
<td>REAL*8</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>V</td>
<td>REAL*8</td>
<td>8</td>
<td>*</td>
</tr>
<tr>
<td>X</td>
<td>REAL*8</td>
<td>4</td>
<td>*</td>
</tr>
</tbody>
</table>
SUBROUTINE LD (A, B, M, N)

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

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

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

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

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

END
SUBROUTINE ZERO (A, N)

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

REAL*8  A(N)
DO 100 K = 1, N
A(K) = 0.0
100 CONTINUE
RETURN
END
SUBROUTINE UPDPOS (XM, X, XHOLD, E, N)

This procedure updates the position of the OMV in local vertical frame (XHOLD).
The new position of the module in floor coordinates is then computed (XM).

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

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

RETURN
END

Name Type Offset P Class

REAL*8 12 *
INTEGER*4 790
INTEGER*4 16 *
REAL*8 4 *
XHOLD REAL*8 8 *
"M REAL*8 0 *

334 $PAGE
This procedure properly rounds a real number R to the nearest integer.

```
346       REAL*8   RR
347       REAL    R
348       R = RR
349       IF (R .GE. 0) THEN
350          JFIX = IFIX (R + 0.5)
351       ELSE
352          JFIX = IFIX (R - 0.5)
353       END IF
354       RETURN
355       END
```
SUBROUTINE SETQ (QW, Q)

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

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

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

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

QW(I,I) = Q(4)

CONTINUE
QW(4,4) = Q(4)

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

RETURN

END

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

393 $PAGE
SUBROUTINE DIRCOS (A, Q)

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

REAL * 8 Q(4), A(3,3), QKS, QRS, S1

DO 100 K = 1, 3

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

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

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

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

*** calculate the proper sign
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
CONTINUE
RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL*8</td>
<td></td>
<td>0 *</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td></td>
<td>838</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>830</td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>826</td>
</tr>
<tr>
<td>Q</td>
<td>REAL*8</td>
<td>4 *</td>
</tr>
<tr>
<td>QKJ</td>
<td>REAL</td>
<td>854</td>
</tr>
<tr>
<td>QKS</td>
<td>REAL*8</td>
<td>****</td>
</tr>
<tr>
<td>QRS</td>
<td>REAL*8</td>
<td>858</td>
</tr>
<tr>
<td>S1</td>
<td>REAL*8</td>
<td>842</td>
</tr>
</tbody>
</table>

438 $PAGE$
REAL FUNCTION DLTKRK (K, J)

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER*4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL</td>
<td>866</td>
<td></td>
</tr>
</tbody>
</table>

$PAGE
REAL FUNCTION QSIGN(K,J)

S = 1.0
L = J + K
IF (MOD(L,2) .EQ. 0) S = -1.0
QSIGN = S
RETURN
END

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>P Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>INTEGER*4</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>K</td>
<td>INTEGER*4</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>L</td>
<td>INTEGER*4</td>
<td>874</td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>INTEGER*4</td>
<td>874</td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>S</td>
<td>REAL</td>
<td>870</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE QTRPY (Q, R, P, Y)

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

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

M = DSQRT (Q(1)**2 + Q(2)**2 + Q(3)**2)
calculate direction cosines CA, CB, CG

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

calculate angle of rotation about Euler axis

THETA = 2.0 * DACOS(Q(4))

now determine the roll, pitch and yaw

R = CA * THETA
P = CB * THETA
Y = CG * THETA
RETURN
END
D Line# 1 7
THETA REAL*8  910
Y    REAL*8  12 *

506 $PAGE
<table>
<thead>
<tr>
<th>Line</th>
<th>Type</th>
<th>Size</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TCOMP</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>TRCOS</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>JX</td>
<td>INTEGER*4</td>
<td>FUNCTION</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>JL</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>IGN</td>
<td>REAL</td>
<td>FUNCTION</td>
</tr>
<tr>
<td></td>
<td>TRPY</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>TTO</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>PDPOS</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>ERO</td>
<td></td>
<td>SUBROUTINE</td>
</tr>
</tbody>
</table>

Pass One  No Errors Detected
506 Source Lines
APPENDIX 7

Mobility-base Control Logic (TOM_C) Source Listing
TOM-B EXECUTIVE

This is the main program for the OMV on-board processing logic.
The following steps are carried out:
1. Performs a system initialization
2. Set up an infinite loop to process each major cycle until both CMDRAW(1) & CMDRAW(2) are <= -99.
   During each major cycle, a subroutine PMAJOR performs all the necessary functions. It then waits for the next cycle. The start of the next cycle is indicated when FLAG is cleared.

Each major cycle has a period of 0.1 sec; this value is input from disk during system initialization. Since cycle execution is tracked by using this variable, the period may be altered by changing its value on disk.

Absolute commands will be used throughout.

It is assumed that a routine SETUP sets an interrupt schedule and performs all the necessary services.

---

INTEGER * 4 FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
INTEGER * 4 CYCLE
COMMON /CHMD/, CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
COMMON /CYCLE/, CYCLE, JSTF1

*** system initialization ***
CALL INITOM
CALL GOHOME

*** MONITOR CYCLE PROCESS ***

C 100 WHILE ((CMDRAW(3).GT.-99) .AND. (CMDRAW(2).GT.-99)) DO
   IF((CMDRAW(3).LT.-99).AND.(CMDRAW(2).LT.-99))GOTO 900
   *** PROCESS A MAJOR CYCLE ***
   CALL PMAJOR
   *** WAIT UNTIL NEXT CYCLE & CONTINUE ***
   CALL WAIT
GO TO 100
END WHILE
CONTINUE

*** perform house cleaning before quitting ***

CALL GOHOM
STOP
END

SUBROUTINE INITOM

This procedure performs a system initialization.
1. A disk data file called INITOM is accessed for the pertinent information.
2. Power to disk drive may then be disconnected
3. Press (CR) to continue.
4. INIT calls SETUP to establish an interrupt schedule.

INTEGER * 4 FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
INTEGER DOF
INTEGER * 2 FRITBLX(20,2), FRITBLY(20,2), JETBUF(40)
REAL LX, LY, MASS, MAJOR, JZZ
REAL MTRVRD(6), MTRVCL(6), MTRVOF(6)
REAL NAUVAL(3), NAVCAL(3), NAVOFF(6)
REAL MTRPRD(6), MTRPCCL(6), MTRPOF(6)
COMMON /CMD/ CMDRET, CMDRAW, CMDUAL(9), CMDMOD, FLAG
COMMON /DACO/ DACRDG(6), DACCL(6), DACCOF(6)
COMMON /DYNA/ THRU, ACC(2), LX, LY, DOF
COMMON /JETS/ MTHR, MTHRY, FRITBLX, FRITBLY, JETBUF, SCL, SCL'
COMMON /MTR/ MTRPRD, MTRPCCL, MTRPOF
COMMON /NAV/ NAUVAL, NAVCAL, NAVOFF
COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD
COMMON /POSN/ POSTN(9), OPOSTN(9)
COMMON /RATE/ VCKET(9), OLDVEL(9)
COMMON /SNSR/ SNRR(3), SNRC(3), SNRB(3)
COMMON /PRCN/ EPSL, EPSA, UL, UA
COMMON /DELY/ DV(3)

When

WRITE (*,39)
FORMAT (' in INITOM')

Implementation notes :

<table>
<thead>
<tr>
<th>PHYS. QTY</th>
<th>STANDARD</th>
<th>MKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASS</td>
<td>77.64 SLUG (2500 LB)</td>
<td>1132.77 KG</td>
</tr>
<tr>
<td>MAJOR</td>
<td>0.1 SEC</td>
<td>0.1 SEC</td>
</tr>
<tr>
<td>JZZ</td>
<td>334.17 SLUG-FT-FT</td>
<td>452.95 KG-M-M</td>
</tr>
<tr>
<td>THRUST</td>
<td>3 LB</td>
<td>13.345 NT</td>
</tr>
<tr>
<td>LX</td>
<td>32 IN</td>
<td>0.787 M</td>
</tr>
<tr>
<td>LY</td>
<td>31 IN</td>
<td>0.762 M</td>
</tr>
<tr>
<td>ACC</td>
<td>0.0773 FT/SEC/SEC</td>
<td>0.02356 M/SEC/SEC</td>
</tr>
<tr>
<td>WZ</td>
<td>0.0478 RAD/SEC</td>
<td>0.04786 RAD/SEC</td>
</tr>
</tbody>
</table>

- Both ACC and WZ are used in the model OM2
OPEN (LG, FILE = 'INITOM.DAT', STATUS = 'OLD')

READ (LG, 10) MASS
READ (LG, 10) MAJOR
READ (LG, 10) JZZ
READ (LG, 10) THRUST
READ (LG, 10) LX
READ (LG, 10) LY
READ (LG, 10) EPSL
READ (LG, 10) EPSA
READ (LG, 10) UL
READ (LG, 10) UA

READ (LG, 20) NTHRX
READ (LG, 20) NTHRY
READ (LG, 20) DOF

READ (LG, 10) SCLX
READ (LG, 10) SCY

DO 100 K = 1, 3
READ (LG, 10) SNRC(K), SNRB(K)
CONTINUE

DO 110 K = 1, 3
READ (LG, 10) NAVCAL(K), NAVOFF(K)
CONTINUE

DO 120 K = 1, 3
READ (LG, 10) MTRPCL(K), MTRPOF(K)
CONTINUE

DO 130 K = 1, 3
READ (LG, 30) MTRVCL(K), MTRVOF(K)
CONTINUE

DO 140 K = 1, 3
READ (LG, 30) DACCAL(K), DACCOF(K)
CONTINUE

DO 200 K = 1, DOF
READ (LG, 10) POSTN(K)
OPSTN(K) = POSTN(K)
VLCTY(K) = 0.0
OLDVEL(K) = 0.0
IF (K LE 3) DV(K) = 0.0
CONTINUE

NN = NTHRX * 4
DO 300 K = 1, NN
READ (LG, 20) FRTBLX(K, 1)
CONTINUE

NN = NTHRY * 4
DO 350 K = 1, NN
READ (LG, 20) FRTBLY(K, 1)
CONTINUE

Compute other quantities

PI = 355.0 / 113.0
CYPAR = 180.0 / PI
This procedure synchronizes TOM_B EXECUTIVE to the interrupt service routine.

This procedure:
A. Transmits the current position & orientation to the main-framed computer &
B. waits until interrupt service routine is completed when FLAG is cleared.

Note that
FLAG = 0 means system is OK. TOM_B EXECUTIVE should proceed in the normal manner.
FLAG = -1 means there is a hardware failure of some sort. In this case, the main frame is notified and the mission aborted.
FLAG = 1 means not ready. Wait some more.

There is no provision to halt and power down TOM_B in case of hardware failure from software at this time.

REAL LX, LY
INTEGER 4 FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
INTEGER DOF
COMMON /CMMD/ CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
COMMON /DYNA/ THRUST, ACC(2), LX, LY, DOF

*** Report position ***
CALL XMIT

*** Wait until ready ***

WHILE (FLAG .GT. 0) DO
100 IF (FLAG .LE. 0) GO TO 200
   GO TO 100
END WHILE

*** See if there is any hardware failures ***
200 IF (FLAG .GE. 0) RETURN

*** We have hardware failure ***

DO 300 K=1,DOF
   CMDRET(K) = -99
   CONTINUE
300

*** Tell mainframe & abort mission ***
CALL SENDIT
SUBROUTINE XMIT

This procedure takes the current TMB position and places it in a buffer. An I/O driver SENDIT is called to transmit this information to the main frame. All lengths are expressed in meters, while all angular quantities are expressed in radians. All must be scaled before sending.

REAL LX, LY
INTEGER DOF
INTEGER * 4 CMDRET(9), FLAG, CMDMOD, CMDRAH(9)
COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD
COMMON /DYNA/ THRUST, ACC(2), LX, LY, DOF
COMMON /PRCN/ EPSR, EPSA, UL, UA
COMMON /POSN/ POSTN(9), OPOSTN(9)
COMMON /CMMD/ CMDRET, CMDRAH, CMDVAL(9), CMDMOD, FLAG
COMMON /RATE/ VLCTY(9), OLVEL(9)

DO 100 K=1, DOF
   FACTOR = UL
   IF ((K.GT. 1) .AND. (K.LT. 5)) FACTOR = UL
   TMP = POSTN(K) * FACTOR
   CMDRET(K) = IFIX(TMP + 0.5)
100 CONTINUE

CALL SENDIT

RETURN
END

SUBROUTINE PMAJOR

This procedure processes a major cycle by:
A. determine its current position.
B. determine its current velocity.
C. decode the command sequence.
D. decide if it needs to adjust its position/velocity based on the value of FIRFLG:
   1 : FIRFLG = 0 ; no adjustment needed.
   2 : FIRFLG = 10 ; use thrusters
   3 : FIRFLG = 1 ; use motors
   4 : FIRFLG = 11 ; use both thrusters & motors
E. In case when both thrusters & motors need to be used, the thrusters are fired first.
This procedure updates the position and velocities of all the six axis of the mobile base, after having saved its current state. The axes assignment is as follows:

<table>
<thead>
<tr>
<th>Axis</th>
<th>Dynamic quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yaw of mobile base</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>Z</td>
</tr>
<tr>
<td>5</td>
<td>pitch</td>
</tr>
<tr>
<td>6</td>
<td>roll</td>
</tr>
</tbody>
</table>

Release notes:
- Triangulation navigation system is not ready. Position X and Y are calculated in NAVCN instead of measured.
- Motor rate feedback is unreliable, but position feedback is. Thus, motor rates are derived from the position feedback data by differentiation, until hardware is rectified.

```fortran
SUBROUTINE UPDATE

This procedure updates the position and velocities of all the six axis of the mobile base, after having saved its current state. The axes assignment is as follows:

<table>
<thead>
<tr>
<th>Axis</th>
<th>Dynamic quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yaw of mobile base</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>Z</td>
</tr>
<tr>
<td>5</td>
<td>pitch</td>
</tr>
<tr>
<td>6</td>
<td>roll</td>
</tr>
</tbody>
</table>

Release notes:
- Triangulation navigation system is not ready. Position X and Y are calculated in NAVCN instead of measured.
- Motor rate feedback is unreliable, but position feedback is. Thus, motor rates are derived from the position feedback data by differentiation, until hardware is rectified.
```
REAL THETA, V(3), JG, W(2), VV(3)

COMMON /Dyna/ THrust, ACC(2), LX, LY, DOF
COMMON /Phys/ MASS, MAJOr, JZZ, PIrAd
COMMON /posn/ POSTN(9), OPOSTN(9)
COMMON /Rate/ VLCTY(9), OLDVEL(9)
COMMON /Motr/ MTRPRD, MTRPCL, MTRPOF
COMMON /Motv/ MTRVRD, MTRVCL, MTRVOF
COMMON /Buff/ GYRBuf, NAVBUF, MTRBUF, MTRVBUF, SNRBUF, DACBUF
COMMON /snr/ SNRR(3), SNC(3), SNRb(3)

DO 100 K = 1, DOF
   OPOSTN(K) = POSTN(K)
   OLDVEL(K) = VLCTY(K)
   CONTINUE

   THETA = POSTN(1)

W(1) = VLCTY(2)
W(2) = VLCTY(3)
CALL FTB (W, THETA, VV)
V(1) = VLCTY(1)
V(2) = VV(1)
V(3) = VV(2)
DO 200 K = 1, 3
   KK = (K-1) * 6
   JG = GYRBuf(KK+1)
   DO 220 J = 2, 6
      JG = JG + GYRBuf(KK+J)
   CONTINUE

SNRBUF(K) = JG / 100000.0
V(K) = V(K) + JG/100000.0

CONTINUE

transform to floor coordinates
VLCTY(1) = V(1)
V(1) = V(2)
V(2) = V(3)
CALL FTB (V, THETA, W)
VLCTY(2) = W(1)
VLCTY(3) = W(2)

CALL NAVCN (MAJOR, GYRBuf, 18)

*** Find position & velocity of motors (axes 4..6)
rates are obtained by differentiation

KK = DOF - 3
IF (KK .LE. 0) GO TO 900
DO 400 K = 1, KK
   MTRPRD(K) = MTRBUF(K) + MTRPCL(K) + MTRPOF(K)
   JJ = K + 3
   POSTN(JJ) = MTRPRD(K)
   VLCTY(JJ) = (POSTN(JJ) - OPOSTN(JJ)) / MAJOR
CONTINUE

CONTINUE

RETURN

END

SUBROUTINE FTB (F, THETA, B)
This subroutine takes a vector \( F(2) \) as expressed in flat floor coordinates and transforms it to body coordinates through a rotation of \( \theta \) radians. The transformed vector is placed in the array \( B \).

\[
\begin{align*}
\text{REAL} & \quad F(2), B(2) \\
C &= \cos(\theta) \\
S &= \sin(\theta) \\
B(1) &= F(1) \times C + F(2) \times S \\
B(2) &= -F(1) \times S + F(2) \times C
\end{align*}
\]

RETURN
END

SUBROUTINE BTF (B, THETA, F)

This subroutine takes a body vector and transforms it to flat floor coordinates via a pure rotation by \( \theta \) radians.

\[
\begin{align*}
\text{REAL} & \quad B(2), F(2) \\
C &= \cos(\theta) \\
S &= \sin(\theta) \\
F(1) &= B(1) \times C - B(2) \times S \\
F(2) &= B(1) \times S + B(2) \times C
\end{align*}
\]

RETURN
END

SUBROUTINE NAVGN (PERIOD, JBUF, N)

This is a temporary procedure to determine absolute position & orientation of TOM_B by using the rate information to allow for system checkout. This effectively by-passes the triangulation navigation system.

This procedure must be replaced ultimately by an appropriate on
THETA = BODE6(JBUF, 6, 0.0, 0.1)
POSTN(1) = THETA

DO 100 K = 1, 3
   DELTA = (VLCTY(K) + OLDVEL(K)) * PERIOD / 2.0
   POSTN(K) = OPOSTN(K) + DELTA
CONTINUE

RETURN
END

REAL * 8 FUNCTION BODE6 (F, N, A, B)

This subroutine uses simpson's rule to perform a simple integration to obtain THETA

INTEGER F(N)
REAL * 8 H, SUM

WRITE (*,39)
FORMAT (' '' in BODE'

H = (B - A) / FLOAT(N - 1)
SUM = 19.0 * (FLOAT(F(1)) + FLOAT(F(6)))
   + 75.0 * (FLOAT(F(2)) + FLOAT(F(5)))
   + 50.0 * (FLOAT(F(3)) + FLOAT(F(4)))
BODE6 = 5.0 * H * SUM / 288.0
RETURN
END

SUBROUTINE CMDFIX

This procedure processes transmitted commands in CMDRAW and calculate their actual values and places them in CMDVAL.
It is assumed that absolute (and not delta) commands will be used. Depending on the value of CMDMOD, rate or position commands are implemented:
   CMDMOD = 0 means rate control
   CMDMOD = 1 means positional control
System of units used in TOM_B EXECUTIVE is MKS.

According to TOM BRYAN, delta commands will never be used, but this procedure can be modified if & when delta commands are desired.

Command index assignment:

<table>
<thead>
<tr>
<th>INDEX</th>
<th>AXIS</th>
<th>TYPE</th>
<th>MODE=0</th>
<th>MODE=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>v</td>
<td>COM+</td>
<td></td>
<td>Tompl</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>COM+</td>
<td></td>
<td>Tompl</td>
</tr>
</tbody>
</table>
**ORIGINAL PAGE IS OF POOR QUALITY**

```
2   X   length    X^*     X
3   Y   length    Y^*     Y
4   Z   length    Z^*     A
5   PITCH   angular    P^*     P
6   ROLL   angular    R^*     R

------------------------------------------------------------
INTEGER * 2  FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
INTEGER  DOF
REAL  LX, LY, MASS, MAJOR, JZZ, PIRAD
COMMON /PHYS/  MASS, MAJOR, JZZ, PIRAD
COMMON /DYNA/  THRUST, ACC(2), LX, LY, DOF
COMMON /CMD/  CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
COMMON /PRCN/  EPSL, EPSA, UL, UA

*** CONVERT AHOY! ***

DO 100 K=1,DOF
   FACTOR = UA
   IF ((K.GT.1) .AND. (K.LT.5)) FACTOR = UL
   RDG = FLOAT(CMDRAW(K)) / FACTOR
   CMDVAL(K) = RDG
CONTINUE

CMDMOD = CMDRAW(7)

RETURN
END

SUBROUTINE DECISION(FIRFLG)

This procedure decides whether or not corrective action needs to be taken by setting and returning a flag FIRFLG:
A. FIRFLG = 0; No action needed
B. 0<FIRFLG<10; Need to move DC motors
C. FIRFLG =10; Need to fire thrusters
D. FIRFLG =11; Need to do both

Decision is made based on the comparison between the command sequence and current TOM_B dynamic quantities, remembering that the system at this instance is under either position or rate control, and that the commands are absolute commands.

------------------------------------------------------------
INTEGER * 4  DOF, FIRFLG, FG
INTEGER * 2  FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
REAL  LX, LY
COMMON /DYNA/  THRUST, ACC(2), LX, LY, DOF
COMMON /PRCN/  EPSL, EPSA, UL, UA
COMMON /CMD/  CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG

*** Check motor section ***

CALL CHKCMD(4,DOF,EPSL,EPSA,FG)
   FIRFLG = FG

*** Check thrusters section ***

IF (CMDMOD NE 0) GOTO 100
```
This procedure checks the absolute command against the vehicle's position or velocity to determine if any corrective action needs to be taken. If it does, the flag FG will be set. FG is either 0 or 1 on return from this subroutine.

*** initialize loop parameters ***

FG = 0
K = FIRST
EPSLN = EP1

*** check between FIRST & LAST inclusive ***

REPEAT
  T = ABS(POSTN(K))
  IF (CMDMOD .EQ. 0) T = ABS(VLCTY(K))
  X = ABS(CMDVAL(K))
  IF (ABS(X - T) .GT. EPSLN) FG = 1
  EPSLN = EP2
  K = K + 1
  IF ((K .LE. LAST) .AND. (FG .EQ. 0)) GOTO 100
UNTIL K > LAST OR FG = 1

RETURN
END

REAL FUNCTION FSIGN(X)

This procedure returns the sign of a REAL variable as +1.0 or -1.0.

REAL X

IF (X .EQ. 0.0) 100, 200, 200
100 FSIGN = -1.0
RETURN
200 FSIGN = 1.0
SUBROUTINE ITABLE

This procedure initializes all entries of both firing tables to zero.

**Subroutine ITABLE**

```
INTEGER * 2 FRTLX(20, 2), FRTLY(20, 2), JETBUF(40)
COMMON /JETS/ NTHRX, NTHRY, FRTLX, FRTLY, JETBUF, SCLX, SCLY

*** initialize X- firing table ***
NX = NTHRX * 4
NY = NTHRY * 4

DO 100 K = 1, NX
   FRTLX(K, 2) = 0
100 CONTINUE

*** Now take care of Y- firing table ***
DO 200 K = 1, NY
   FRTLY(K, 2) = 0
200 CONTINUE

RETURN
END
```

SUBROUTINE TABLE(Fl, F2, NT, TBL, SCALE, NDXR)

This procedure sets up the appropriate firing table by:
A. determining the appropriate # of thrusters to be used
B. calculate the corresponding firing times.
C. load the information in the firing table buffer.

To ensure stability of the vehicle, F1 & F2 must be symmetrized (if such a word exists at all).

**Subroutine TABLE**

```
REAL T(2), TIME(2), LX, LY, MASS, MAJOR, JZZ
INTEGER BASE(2), N(2), DOF
INTEGER * 2 TBL(20, 2)
COMMON /DYNAX/ THRUST, ACC(2), LX, LY, DOF
COMMON /PHYS/ MASS, MAJOR, JZZ, PI RAD

*** Calculate firing times & make them symmetric when possible
Firing times are in seconds

T(1) = F1 / THRUST
T(2) = F2 / THRUST
```
Same EPS as in TSTFIR

EPS = 0.001 * MAJOR
CALL SYMM(T,EPS)

*** set base indeces & actual firing times ***

DO 100 K=1,2
   BASE(K) = (K-1) * NT + 1
   TM = T(K)
   IF (TM .LT. 0) BASE(K) = BASE(K) + 2 * NT
   *** calculate # of thrusters to be used ***
   TM = ABS(TM)
   CALL NMTHR(TM,NN,NT)
   N(K) = NN
   *** NOTE: NN is the # of thrusters to be used ***
   TIME(K) = (TM / FLOAT(NN)) * SCALE / MAJOR
100 CONTINUE

*** Symmetrize TIME(1) & TIME(2) ***
CALL SYMM(TIME,EPS)

*** fill up the firing table buffer ***

DO 200 K=1,2
   NN = N(K)
   DO 200 J=1,NN
      INDEX = BASE(K) + J - 1
      JM = IFIX (ABS(TIME(K)) + 0.5)
      TBL(INDEX,2) = JM
200 CONTINUE

RETURN
END

SUBROUTINE SYMM(T,EPSLN)

This procedure symmetrizes two forces T(1), T(2) acting along the same line, but can be in opposite directions.

When the magnitudes of the two forces has an absolute difference less than the required precision EPSLN the two magnitudes are made to be identical.

This procedure is implemented hopefully to take care of minor truncation errors since all computations are carried out in single precision.

REAL T(2)

*** Calculate magnitudes & signs of each force

T1 = T(1)
AT1 = ABS(T1)
S1 = FSIGN(T1)

T2 = T(2)
AT2 = ABS(T2)
** = FSIGN(T2)
SUBROUTINE NMTHR(T,NN,NT)
---------------------------------------------------------------

This procedure calculates the optimal number of thrusters to
be used on each side.

T : Firing time in major cycles
NN: # of thrusters to be used
NT: Total # of thrusters available on l side.

At present, it is decided that an ad hoc limit of 5 major cy-
cles will be used.

E.G. If it takes 1 thruster for 6 seconds,
we will use 2 thrusters for 3 seconds.

Thus, the # of thrusters on each side that is needed is:

NN = FIRING TIME/5

Once NN is decided, the new firing times must be readjusted to
reflect the change. This is done in the calling procedure TA-
BLE.

It is necessary that 1 <= NN <= NT
---------------------------------------------------------------

REAL /MASS, MAJOR, JZZ
COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD

TX = ABS(T)
NN = IFIX(TX/MAJOR+0.5)
IF (NN .LE. 0) NN = 1
IF (NN .GT. NT) NN = NT

RETURN
END

SUBROUTINE LOADIT(K,TAB,JB)
---------------------------------------------------------------

This procedure takes the contents of a firing table & loads
them into the JET buffer. This feature is implemented for
easy future expansion when more thrusters will be added.

Here, JB(40) is the jet buffer
TAB(K,2) is the appropriate firing table
SUBROUTINE FIRE

This procedure loads firing times from firing tables into JETBU and then invokes the I/O driver LDCTR to fire the appropriate thrusters. NOTE: LDCTR will only load the non-zero table entries.

**SUBROUTINE LDBUF (N, T, J)**

This subroutine takes the contents of a firing table and performs a "this is a good place for a stick up" and places the corresponding firing times into JETBU.
SUBROUTINE MOTORS

This procedure calculates the required DC motor rates, converts them into DAC values & sends them out to the corresponding DAC. An I/O driver is then called on to move the motors.

The logic depends on the command mode (Rate or positional control).

It is explicitly assumed that:
A. The DC motors are rate driven. Therefore, the DAC outputs dictate the rate.
B. Each DAC is 12 bit and is wired for bi-polar output.
C. When position commands are used, a DC motor rate based on a three-cycle period is used. The choice of 3 is arbitrary, and can be adjusted in the final testing.

REAL MASS, MAJOR, JZZ, LX, LY
INTEGER DOF, F
INTEGER * 4 FLAG, CMDMOD, CMDRAH(9), CMDRET(9)
INTEGER * 2 GYRBUF(18)
INTEGER * 2 NAVBUF(3), MTRBUF(6), MTVBUF(6), SNRBUF(3), DACBUF(6)
COMMON /PHYS/ MASS, MAJOR, JZZ, PI
COMMON /DYNA/ THRUST, ACC(2), LX, LY, DOF
COMMON /CMD/ CMDRET, CMDRAH, CMDVAL(9), CMDMOD, FLAG
COMMON /POSN/ POSTN(9), OPOSTN(9)
COMMON /RATE/ VLCTY(9), OLVEL(9)
COMMON /DAC/ DACRDG(6), DACCAL(6), DACCOF(6)
COMMON /BUFF/ GYRBUF, NAVBUF, MTRBUF, MTVBUF, SNRBUF, DACBUF

*** When i ***

KK = DOF - 3
DO 100 MOTOR = 1, KK
M = MOTOR
M3 = M + 3
XCMD = CMDVAL(M3)
C *** Estimate required rate based on mode ***
Q = XCMD
IF (CMDMOD .NE. 0) Q = (XCMD - POSTN(M3)) / (3.0 * MAJOR)
C *** Convert to DAC count ***
R = Q * DACCAL(M) + DACCOF(M)
IR = IFIX (R + 0.5)
SR = FSIGN (R)
C *** Make sure there is no sudden change in direction ***
X = VLCTY(M3)
C IX = IFIX (X * 100 + 0.5)
C IF (CMDMOD .EQ. 0) THEN
   IF (IX .NE. 0) GOTO 200
   X = 0.0
   GOTO 300
C ELSE IF (FSIGN (X) * SR .LT. 0) THEN
   IF (FSIGN (X) * SR .GE. 0) GOTO 300
   IR = 0
   SR = -1
C
GOTO 100
ENDIF

### Make sure DAC count is within limits ###

300
JR = IABS(IR)
IF (JR .GT. 2047) JR = 2047
RR = JR * SR
IR = IFIX(RR + 0.5)

### This is a good place to stick up ###

DACRDG(M) = RR
DACBUF(M) = IR
100 CONTINUE

### Move the motors ###

CALL MTRDRV(DACBUF,KK)
RETURN
END

SUBROUTINE THRSTR

This procedure handles thruster logic.

REAL FF(2), F(2), A(2), T(3)
REAL MASS, MAJOR, JZZ, LX, LY

INTEGER DOF
INTEGER * 2 FRTBLX(20,2), FRTBLY(20,2), JETBUF(40)
INTEGER * 4 FLAG, CMDMOD, CMDRAW(9), CMDVAL(9)
COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD
COMMON /Dyna/ TMBUST, ACC(2), LX, LY, DOF
COMMON /RATE/ VLCTY(9), SDLVEL(9)
COMMON /POST/ POSTN(9), OPOSTN(9)
COMMON /JETS/ NTHR, NTHY, FRTBLX, FRTBLY, JETBUF, SCLX, SCLY
COMMON /CMMD/ CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG

transform acceleration vector ACC to floor coordinates
THETA = POSTN(1)
CALL BTF (ACC, THETA, A)

### calculate required impulses. This is mode dependent ###

IF (CMDMOD .EQ. 0) THEN

IF (CMDMOD .NE. 0) GOTO 100
FF(1) = MASS * (CMDVAL(2) - VLCTY(2)) / 2
FF(2) = MASS * (CMDVAL(3) - VLCTY(3)) / 2
TORG = JZZ * (CMDVAL(1) - VLCTY(1)) / 2
GO TO 120

ELSE CONTINUE

100 DO 150 K = 1, 3
V = VLCTY(K)
P = POSTN(K)
C = CMDVAL(K)

...
IF (K .GT. 1) GOTO 130
   AX = 2 * THRUST * LX / JZZ
   AA = AX
   IF ((C-P) .LT. 0.0) AA = -AX
   GO TO 135
C
ELSE
130
   AA = A(K-1)
C
END IF
135
C
WRITE (*,10) V, P, C, AA
C
10 FORMAT (' ', 4E15.8)
   T(K) = G(V, P, C, AA)
C
150
CONTINUE
   T1 = T(2)
   T2 = T(1)
   TORQ = 0.0
   IF (ABS(TQ) .LT. 0.0001) GOTO 200
   TORQ = THRUST * LX * TQ
200
   FF(1) = T1 * MASS * A(1)
   FF(2) = T2 * MASS * A(2)
C
END IF
C
120
CONTINUE

*** Transform force from floor coordinates to TOM_B coords ***

CALL FTB (FF, THETA, F)
   FX = F(1)
   FY = F(2)

*** Use control laws to calculate force along X & Y directions of TOM_B ***

CALL CTRLW(TORQ, FX, FY, FX1, FX2, FY1, FY2)

*** Convert to firing times and put into firing tables ***

CALL ITABLE
   CALL TABLE(FX1, FX2, NTHRX, FRTBLX, SCLX, 2)
   CALL TABLE(FY1, FY2, NTHRY, FRTBLY, SCLY, 3)

*** Fire them thrusters ***

CALL FIRE
RETURN
END

SUBROUTINE CTRLW(TORQ, FX, FY, FX1, FX2, FY1, FY2)

This procedure calculates FX1, FX2 from FX & FY1, FY2 from FY & TORQ.
It also checks that each FX1, FX2, FY1, FY2 does not exceed the maximum developed thrust on TOM_B.
C
S = 1.0
IF (FX .LE. FY) THEN
  IF (FX .LT. FY) GOTO 100
  FY1 = FY / 2.0 + TORQ / (2.0 * LY)
  FY2 = FY - FY1
  CALL CHECK(FY1, FY2, NTHR, THRUST, TORQ)
  IF ((FY1 .LT. 0.0) .AND. (FY2 .LT. 0.0)) CALL SWAP(FY1, FY2, S)
  DF = (TORQ + S*(FY2 - FY1) * LY) / (2 * LX)
  FX1 = FX / 2.0 + DF
  FX2 = FX - FX1
  CALL CHECK(FX1, FX2, NTHR, THRUST, TORQ)
  IF ((FX1 .LT. 0.0) .AND. (FX2 .LT. 0.0)) CALL SWAP(FX1, FX2, S)
ELSE
  CALL CHECK(FX, FY, NTHR, THRUST, TORQ)
  IF ((FX .LT. 0.0) .AND. (FY .LT. 0.0)) CALL SWAP(FX, FY, S)
ENDIF
END

SUBROUTINE SWAP (X, Y, S)
------------------------------------------------------------------------
This subroutine exchanges X and Y
------------------------------------------------------------------------

REAL T
S = -1.0
T = X
X = Y
Y = T
RETURN
END

SUBROUTINE CHECK (F1, F2, NTHR, THRUST, TORQ)
------------------------------------------------------------------------
This procedure ensures that the thrust required does not exceed the maximum thrust that TOM_B can deliver.
------------------------------------------------------------------------

REAL LIMIT
REAL MASS, MAJOR, JZZ, PIRAD
INTEGER FG
COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD

FX = NTHR * THRUST * LIMIT
REAL FUNCTION G (VO, XO, CMDX, AC)

---

This procedure calculates the optimum firing time for thrusters in a direction when position control is used. A distinction is made between a firing time (≈ 1 major cycle) and an angle.
If a firing time < 1/20 of a major cycle, (5 MS) it is set to zero.

NOTE: all dynamic variables are in floor coordinates !!! and time is expressed in seconds.

------------------------------------------------------------------------------------------------------------------

REAL MASS, MAJOR, JZ, PIRAD
COMMON /PHYS/ MASS, MAJOR, JZ, PIRAD

DX = CMDX - X0
SV = FSIGN(V0)
SD = FSIGN(DX)
IF ((X0 .LT. 0.) .AND. (CMDX .LT. 0.)) GOTO 32
IF (DX .GE. 0.) GOTO 31
XX0 = CMDX
CMD = X0
GOTO 38
ELSE
XX0 = X0
CMD = CMDX
GOTO 38
31 CONTINUE
IF (DX .GE. 0.0) GOTO 33
XX0 = X0
CMD = CMDX
GOTO 38
33 CONTINUE
XX0 = CMDX
CMD = X0
END IF
38 CONTINUE
D = ABS (DX)
V = ABS (V0)
A = ABS (AC)
IF (SD * SV .GT. 0.0) GO TO 50

DX and V0 are anti-parallel
T1 = V / A
RA = T1 ^ 2 + 2 * D / A
T2 = SQRT (RA)
G = SD * (T1 + T2)
RETURN

DX and V0 are parallel
50 CONTINUE
T = MAJOR
X = ABS (XX0)
X1 = ABS (XX0) + V * T
X2 = X1 + A * T * T / 2.0
XC = ABS (CMD)
DO CASE
1: XC = X1
IF (XC .GT. X1) GO TO 200
RA = T ^ 2 - 2.0 * (X1 - XC) / A
IF (RA .LT. 0.0) GO TO 250
G = SD + T * SQRT (RA)
RETURN
ELSE
RA = V * V - 2 * A * (XC - X)
G = -SD * (V + SQRT(RA))
RETURN
END IF

2: X2 > XC > X1
CONTINUE
IF (XC .GT. X2) GO TO 300
RA = T * T + 2 * (X1 - XC) / A
IF (RA .LT. 0.0) GO TO 300
TF = T - SQRT(RA)
G = SD * TF
RETURN
END IF

3: XC > X2
CONTINUE
TF = (SQRT(V * V + 2.0 * A * D) - V) / A
G = SD * TF
RETURN

END CASE

END