(NASA-CR-179220) TELEOPERATOR AND ROBOTICS SYSTEM ANALYSIS Final Report (Alabama Univ.) 243 p Avail: NTIS HC A11/MF A01
CSCL 13I
G3/37
N88-12105
Unclas
0109776

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

Total Cumulative Costs incurred as of October 15, 1984
date

Estimate of cost to complete $89,929.00

Estimated Percentage of Physical Completion 100%

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:

APPLICATIONS OVERVIEW

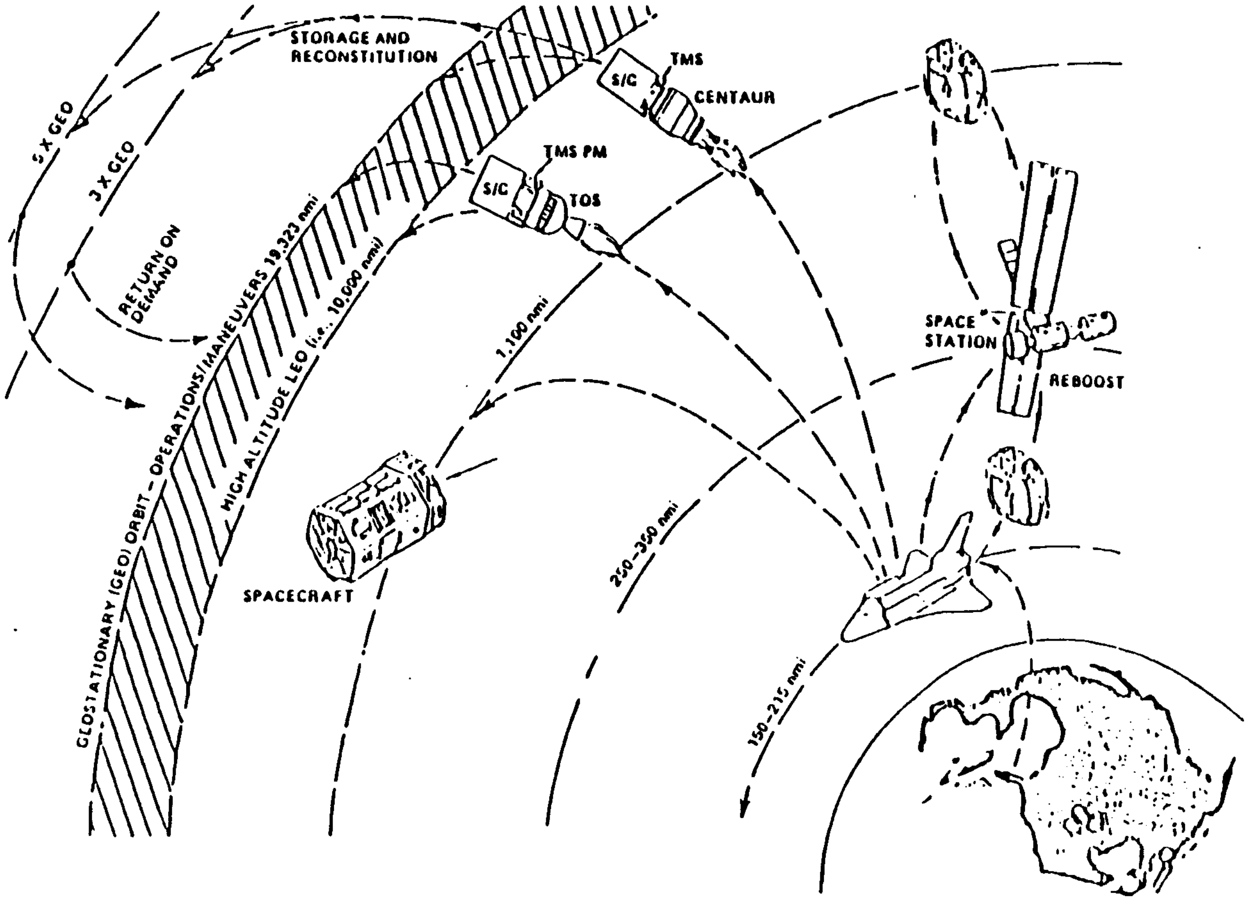
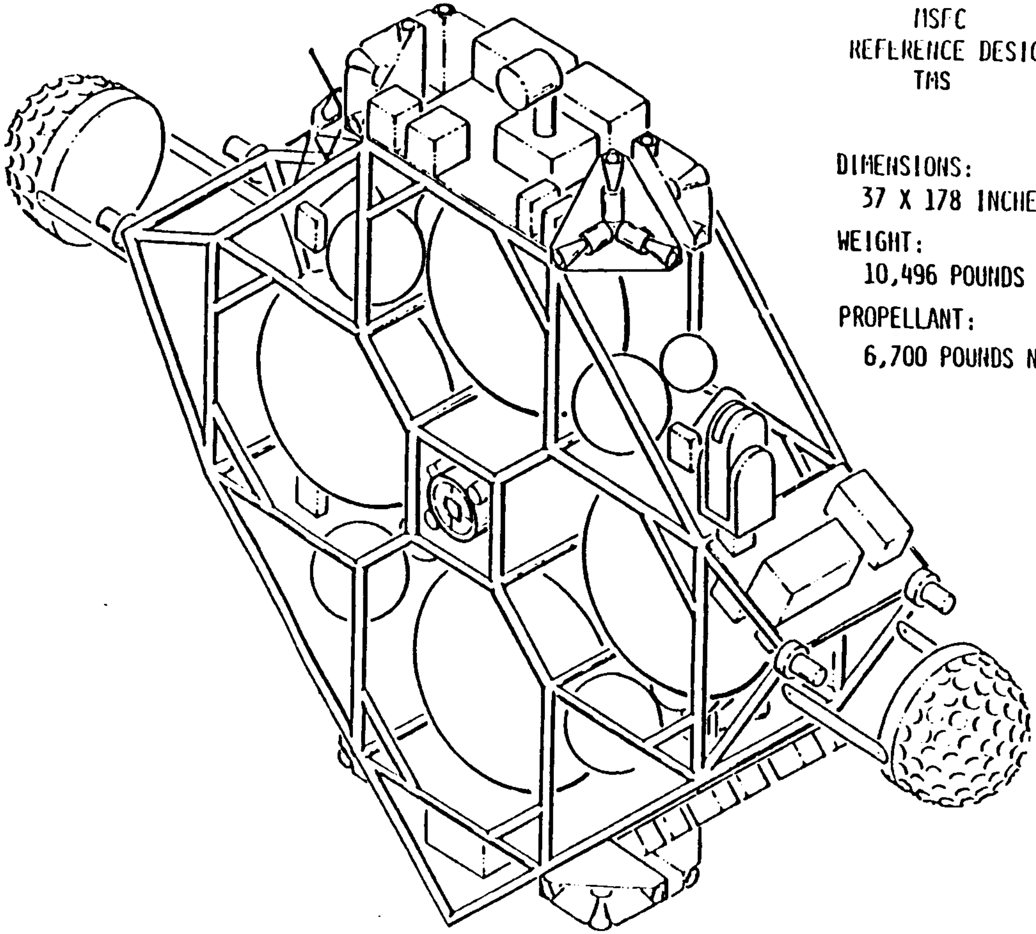


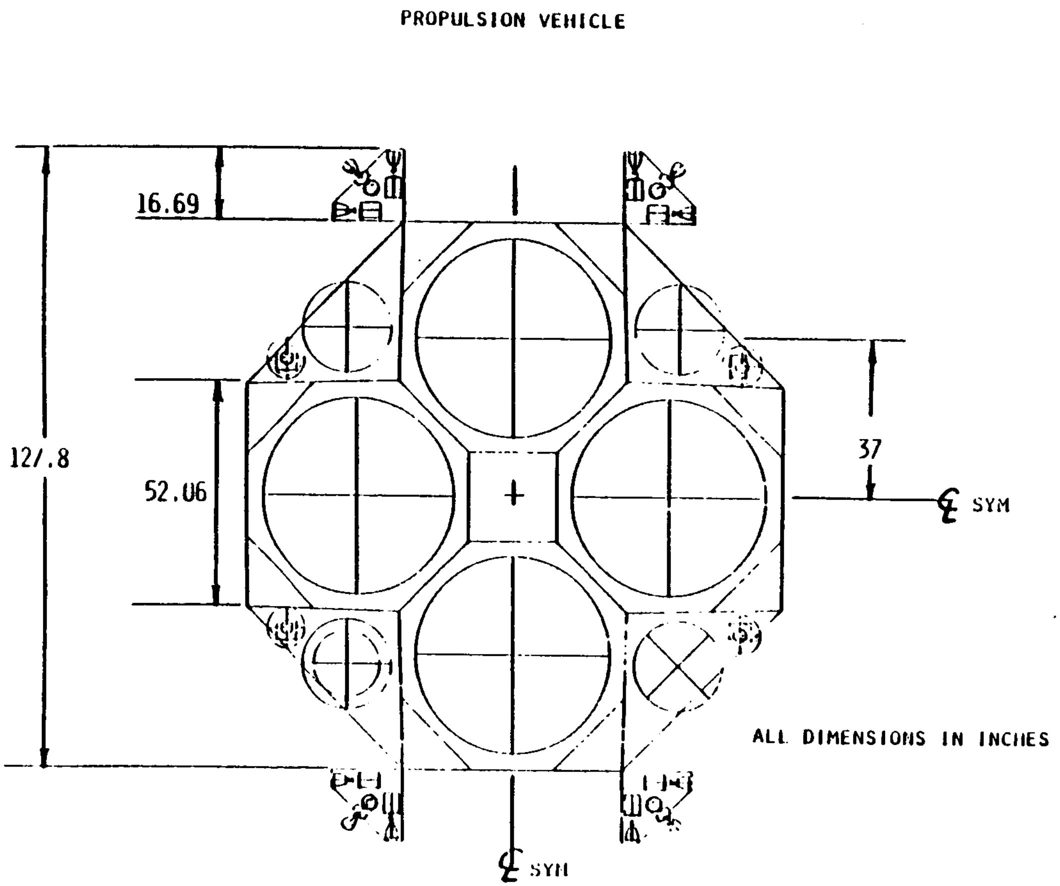
Figure 1-1. Applications Overview



MSFC
REFERENCE DESIGN
TMS

DIMENSIONS:
37 X 178 INCHES
WEIGHT:
10,496 POUNDS (LOADED)
PROPELLANT:
6,700 POUNDS NTO/MMH

Figure 1-2. MSFC Reference Design TMS



ALL DIMENSIONS IN INCHES

Figure 1-3. Propulsion Vehicle

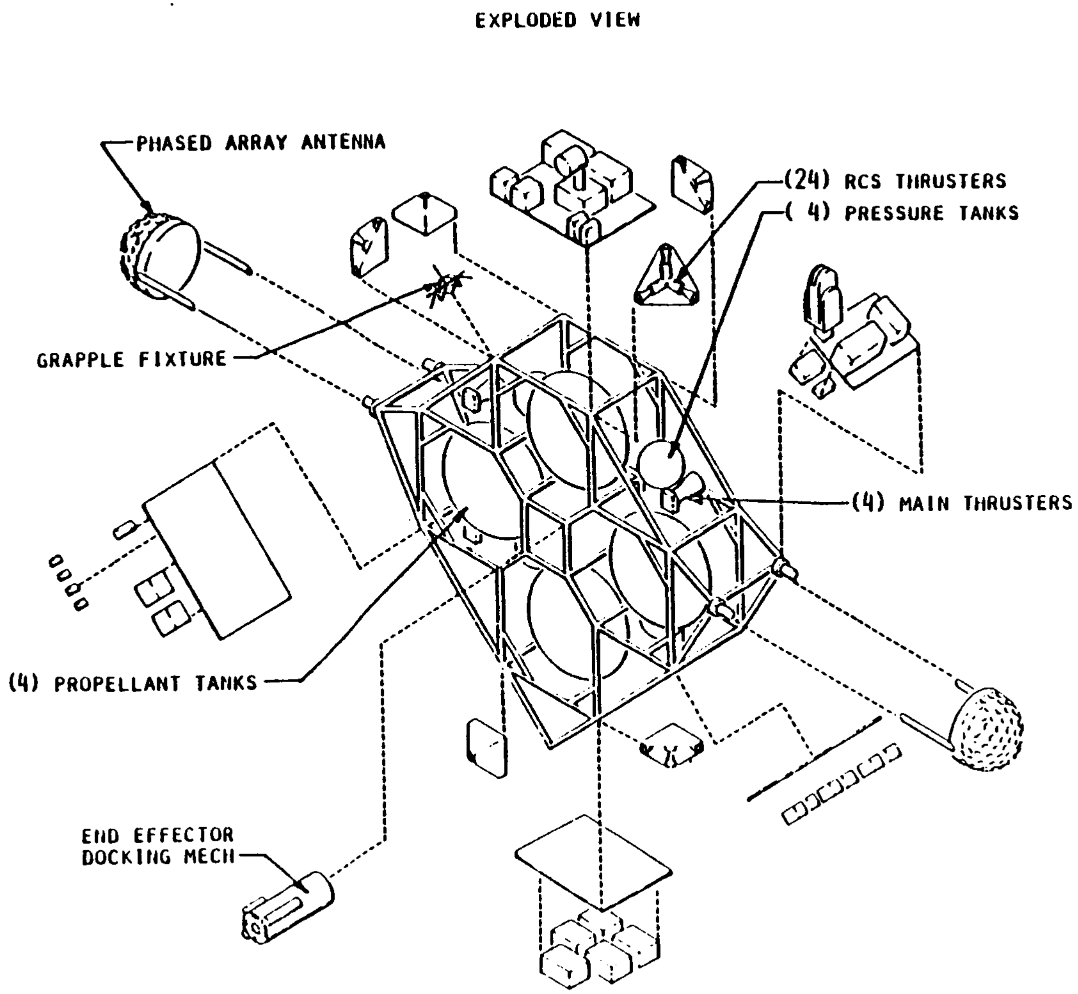


Figure 1-4. Exploded View

- 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 accomodate 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

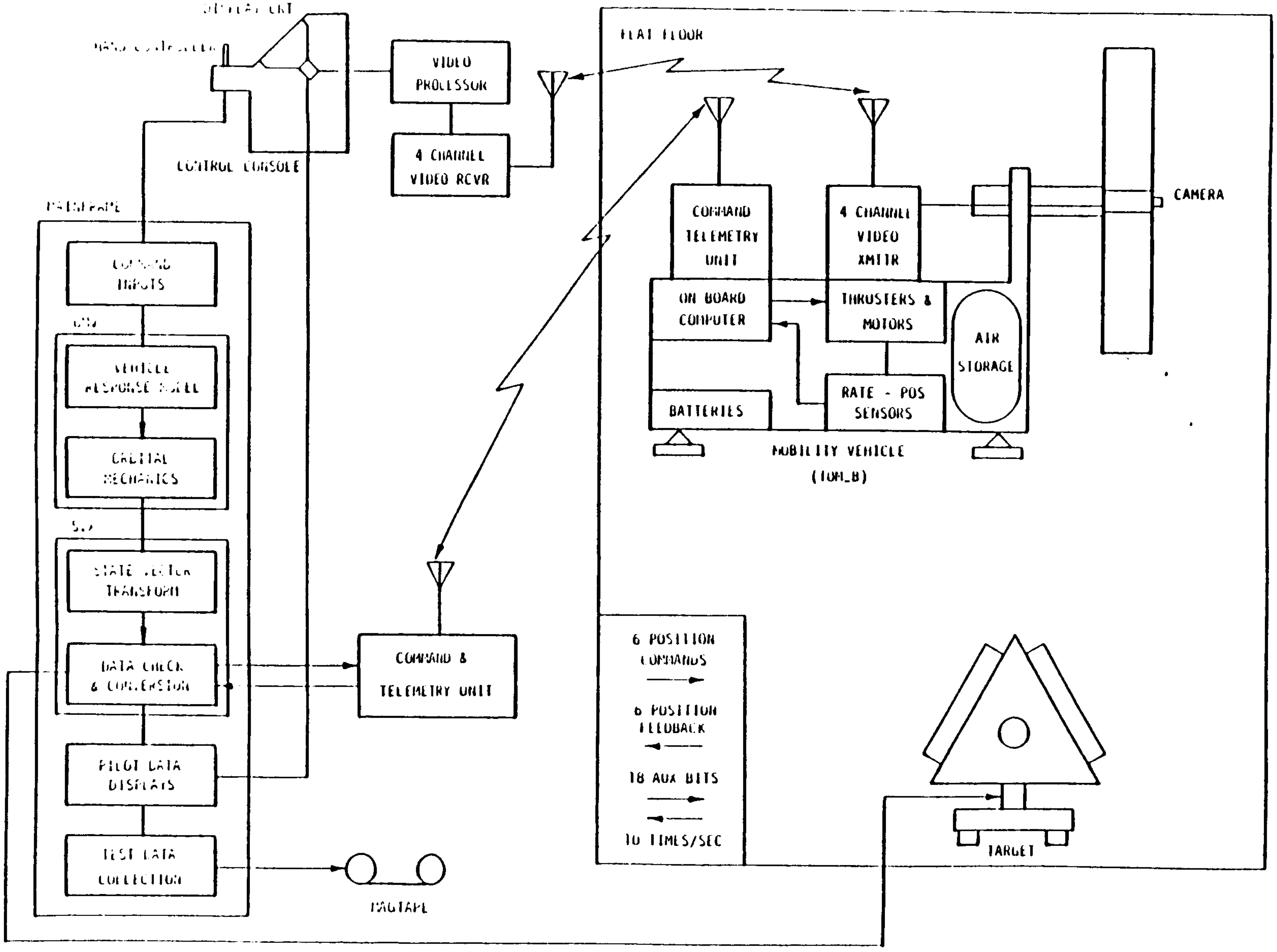


Figure 1-5. MSFC Flat Floor Simulation System

ORIGINAL PAGE IS
OF POOR QUALITY

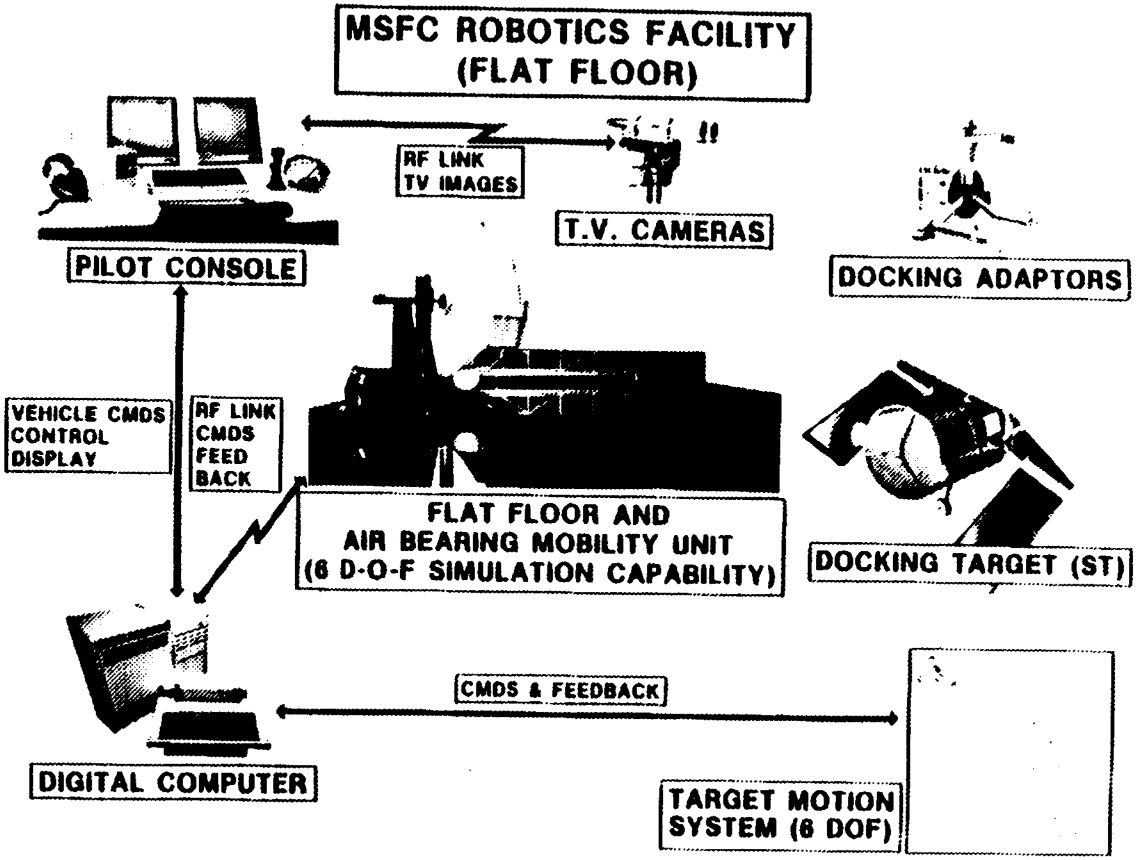
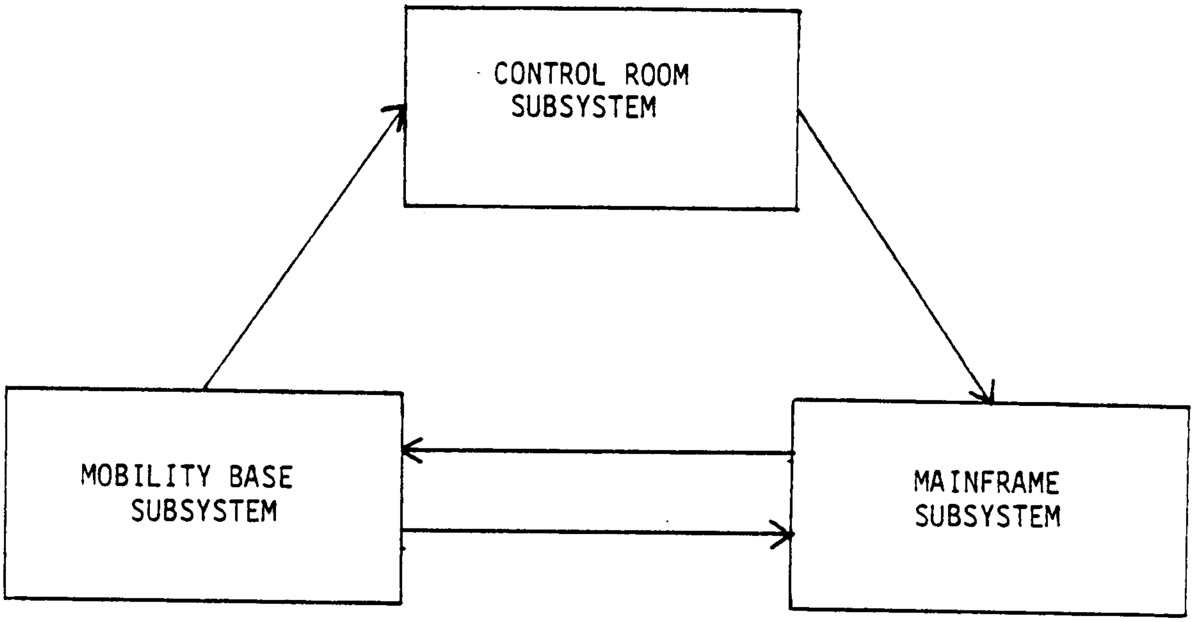


Figure 1-6. MSFC Flat Floor Facility



ARROWS SHOW COMMUNICATION FLOW

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.

ORIGINAL PAGE IS
OF POOR QUALITY

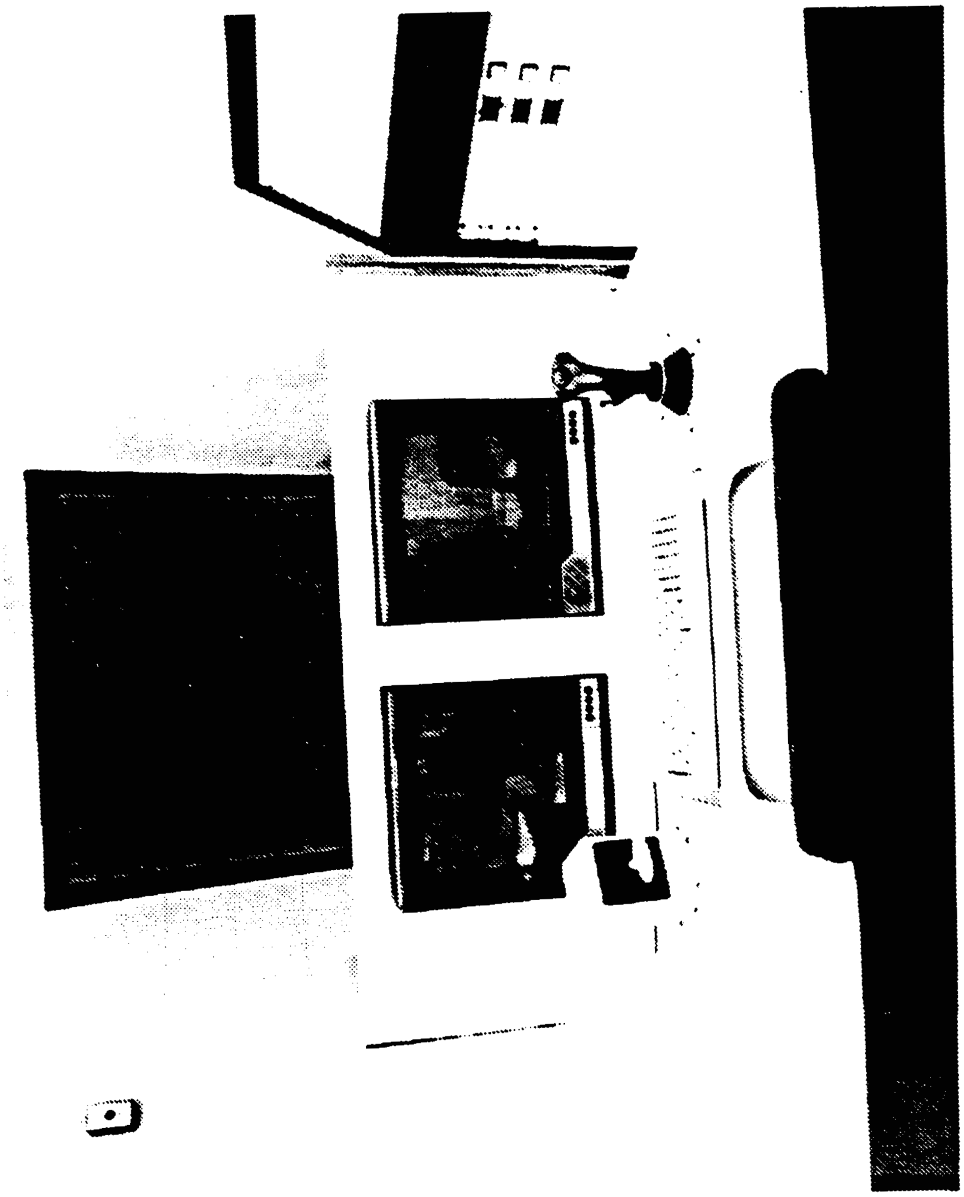


Figure 1-8. Control Room

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.

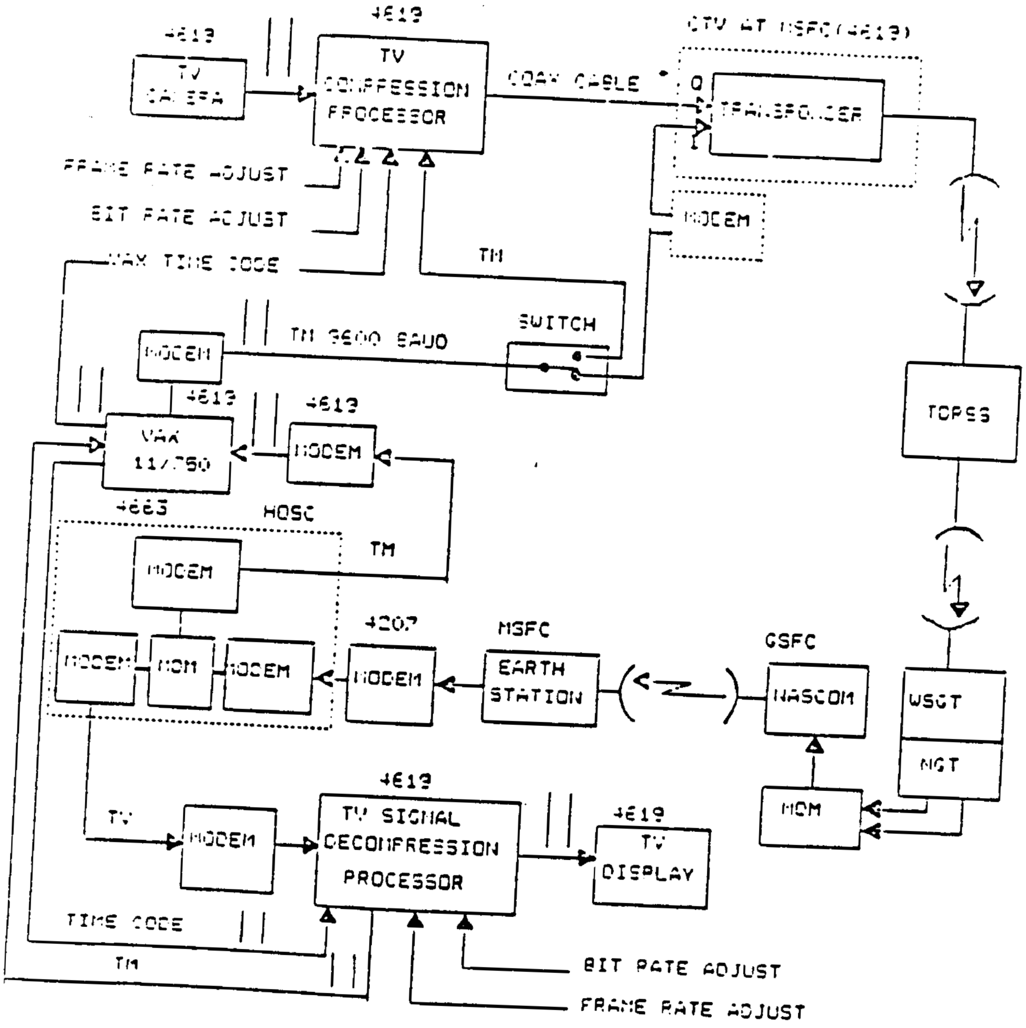
1.4.2 Mainframe Subsystem

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

PRELIMINARY**

7-12-85

MSFC FLAT FLOOR TV/TM RETURN LINK



* A MODEM IS REQUIRED FOR LENGTHS > 100 FEET

|| INTERCONNECT POINTS FOR 6 DOF (4663)

** to be completed August 1986

Figure 1-9. Communication Data Flow

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-B), the Orbital Maneuvering Vehicle mockup module, the flat floor, and the target motion simulator. A description of the mobile base (TOM-B) 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-B; 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

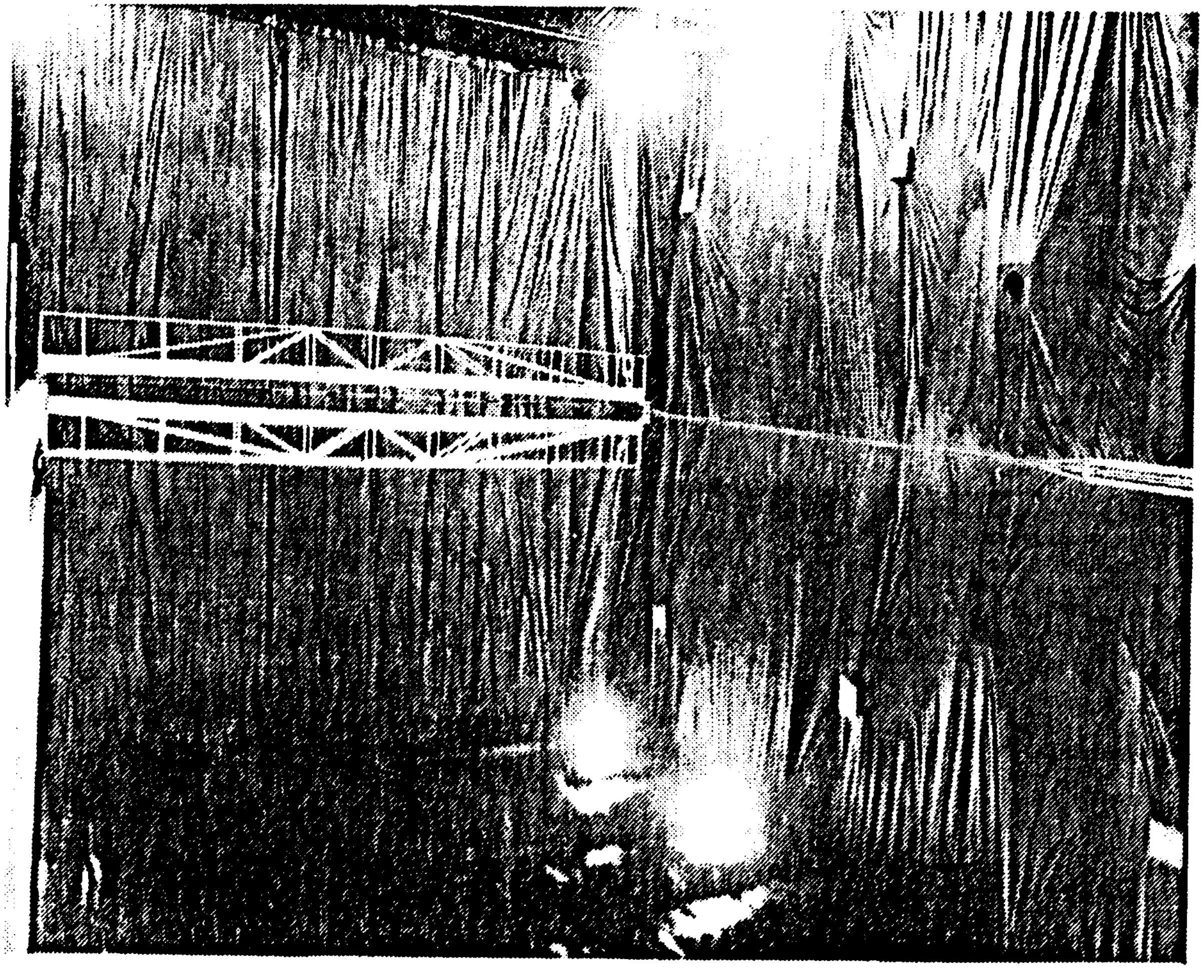


Figure 1-10. Flat Floor

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

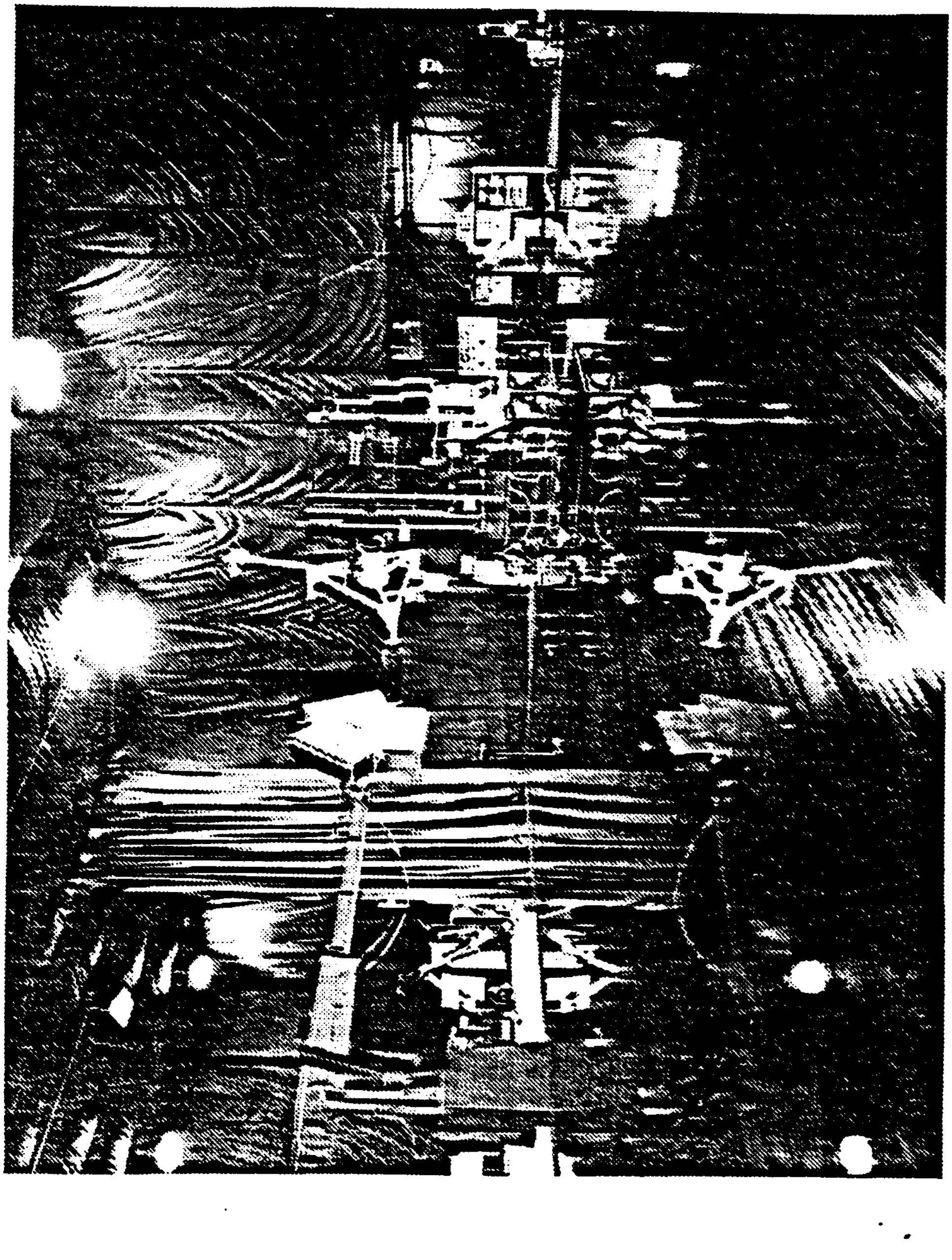


Figure 1-11. TOM-B Simulating Docking

ORIGINAL PAGE IS
OF POOR QUALITY

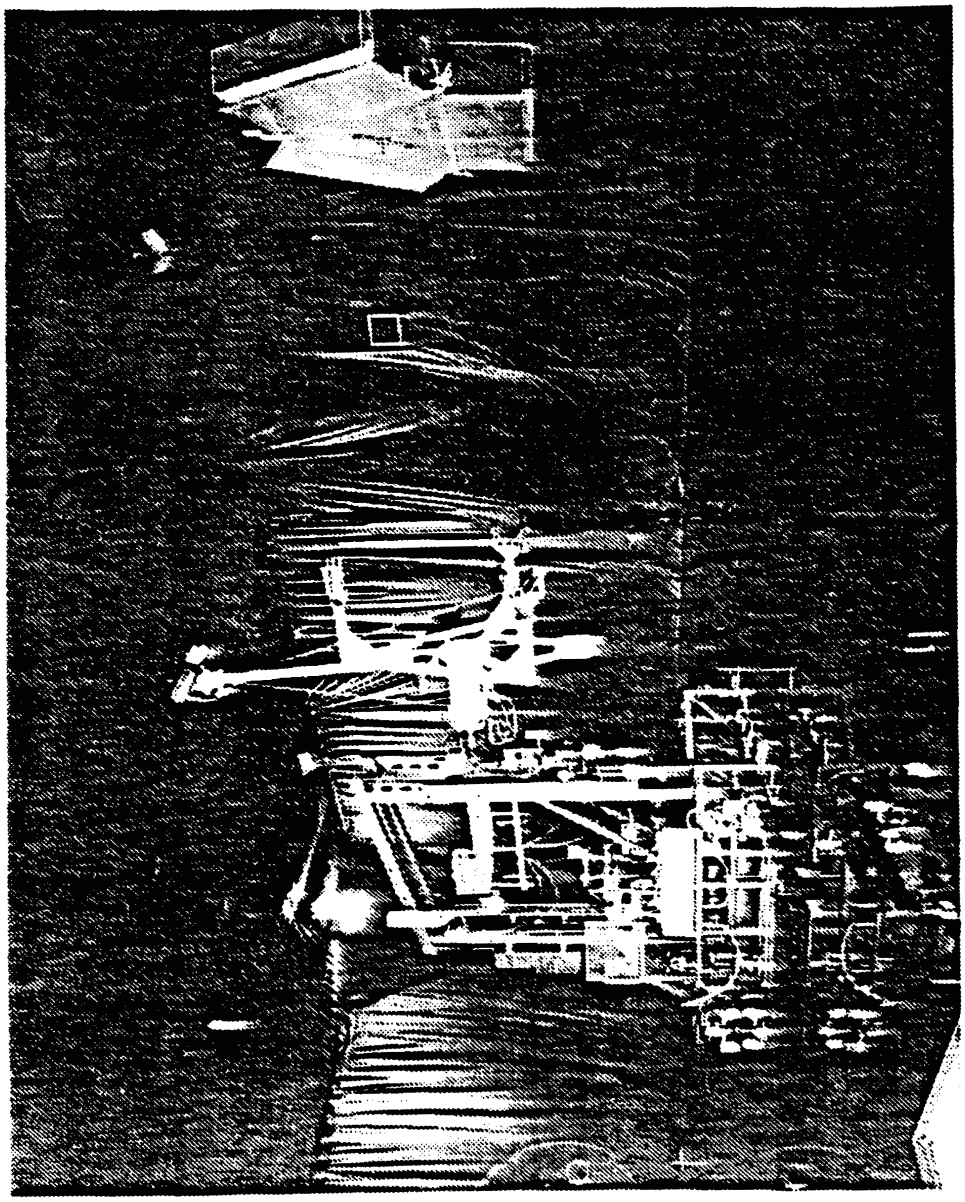


Figure 1-12. TOM-B with Docking Mechanism

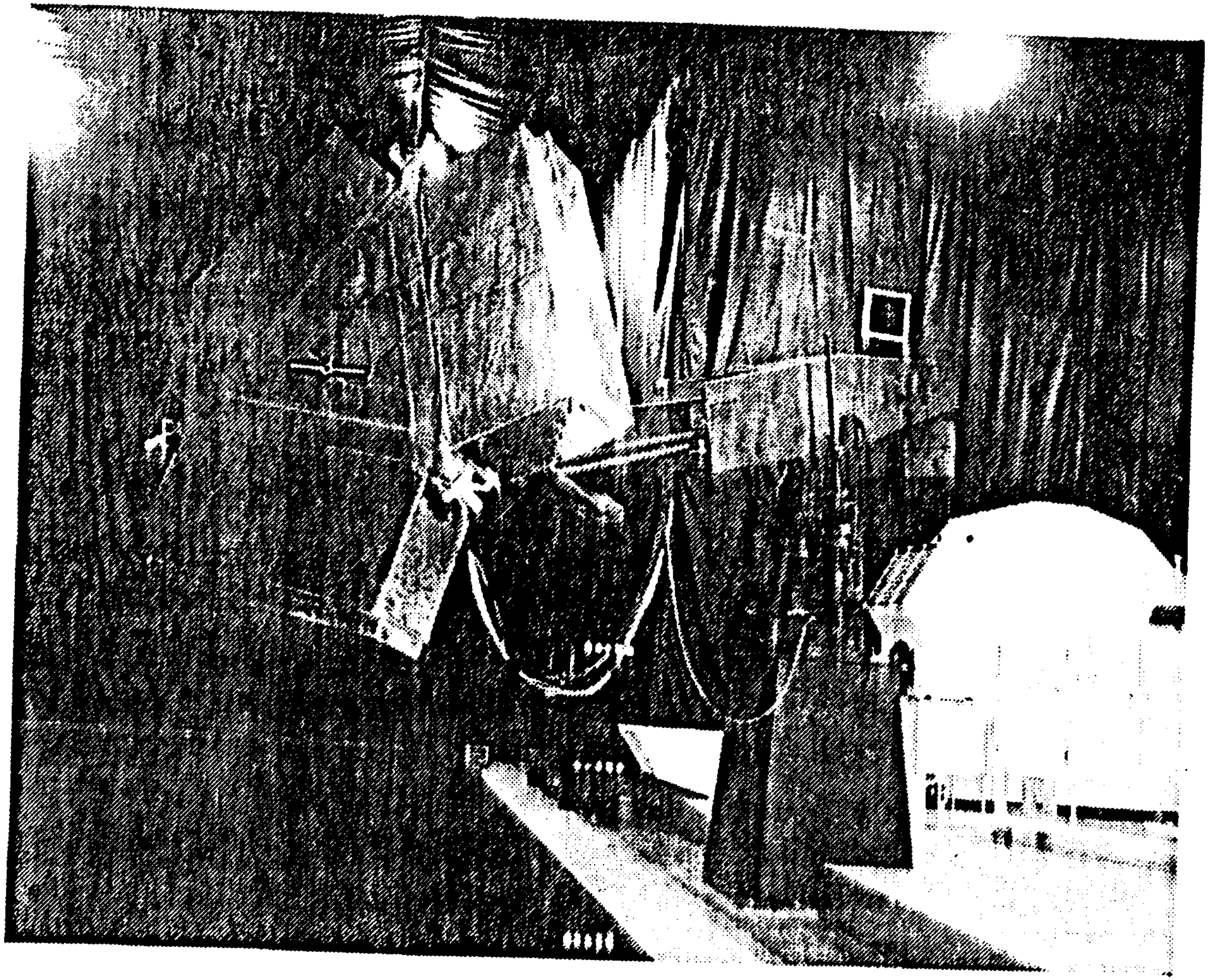


Figure 1-13. Target Motion Simulator

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 1-1

Approximate mass:	1360.5 kg
Moment of Inertia:	100 kg-m^2
Mass of fuel:	136.4 kg
Number of thrusters:	24
Thrust developed:	13.2 Newtons/thrust

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-throttable. The translation in the X and Y axes, as well as yaw motion of

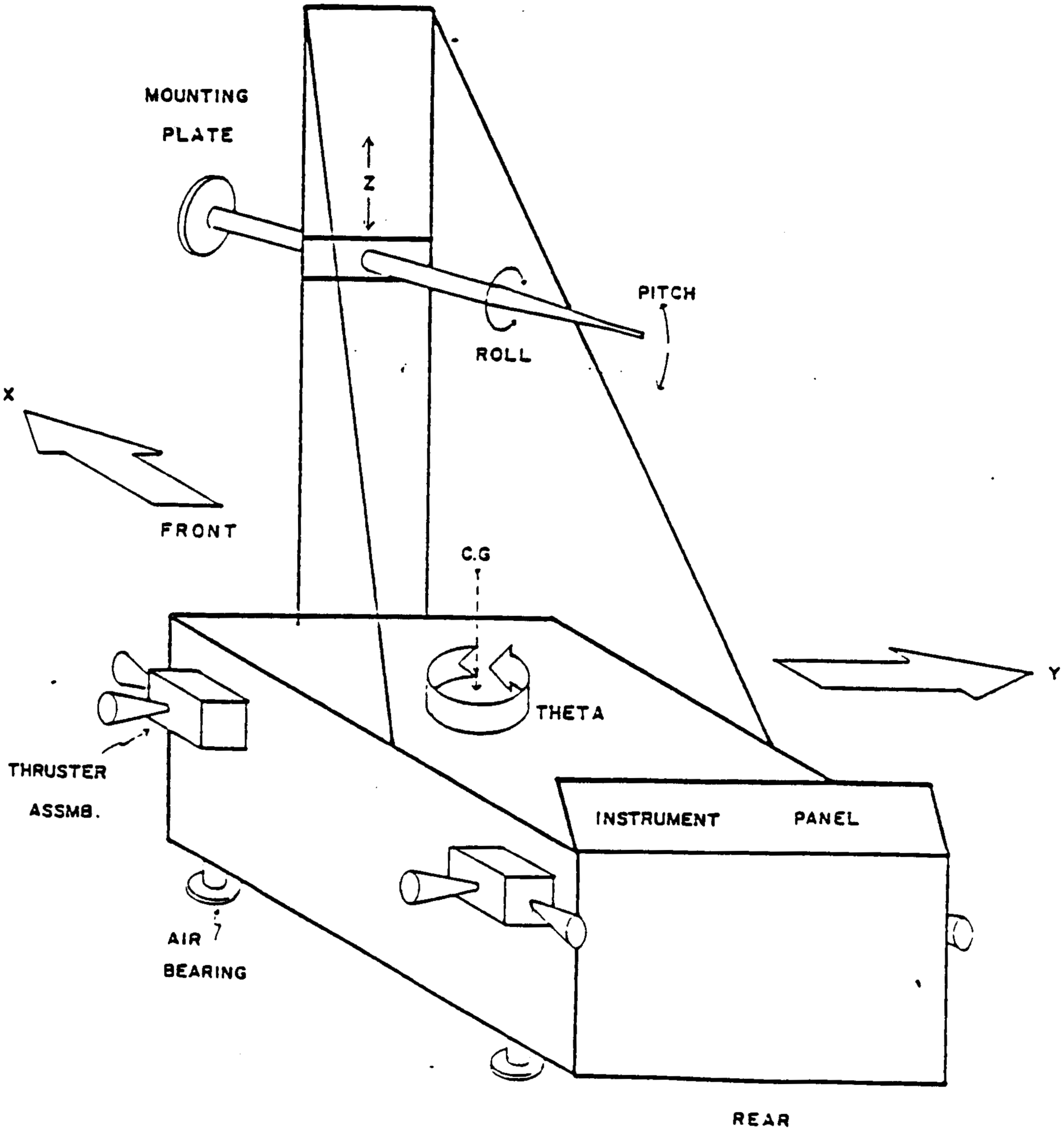


Figure 1-14. TOM-B Showing the Six Degrees of Freedom & Thruster Assembly

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 x 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.

ORIGINAL PAGE IS
OF POOR QUALITY

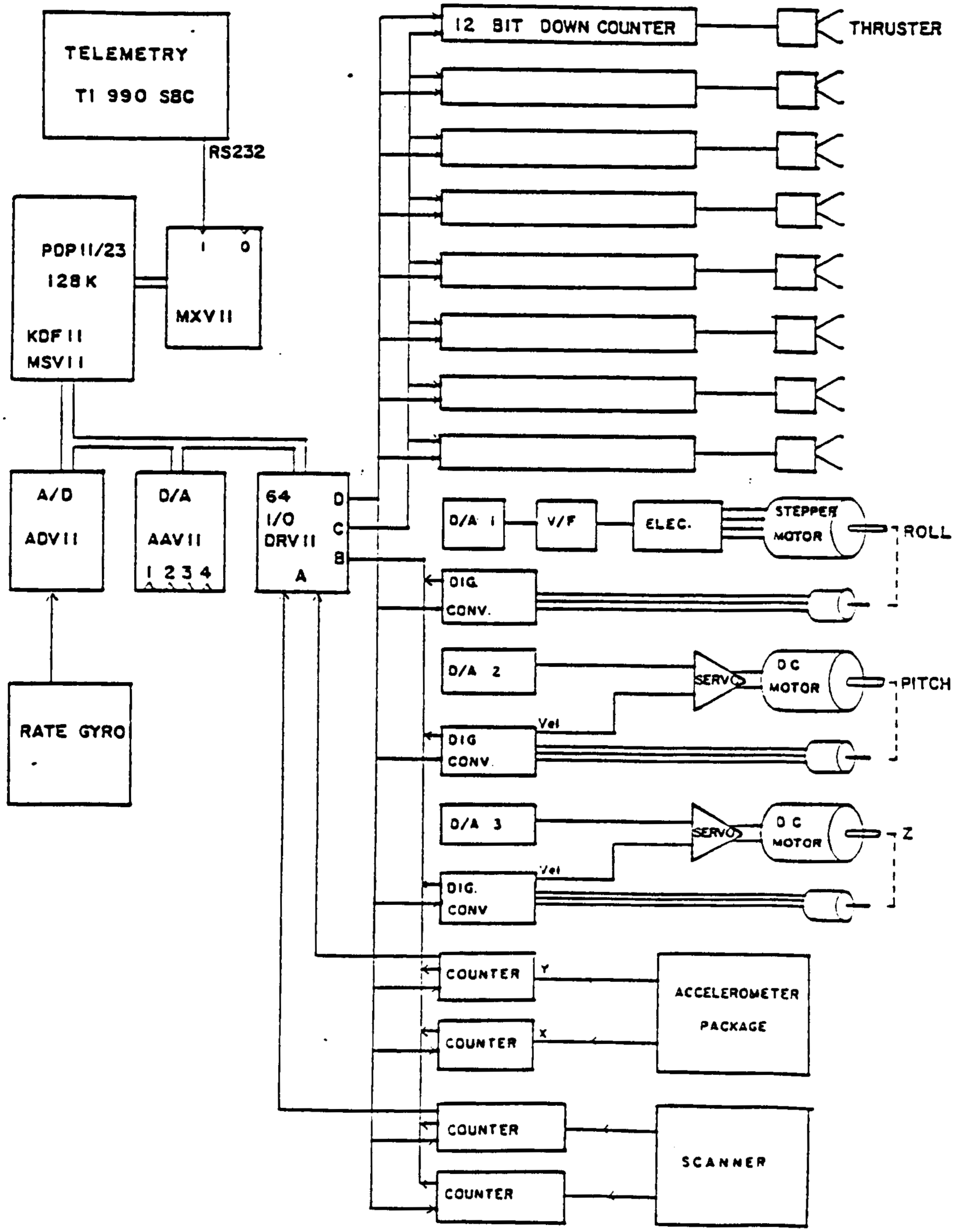


Figure 1-15. TOM-B Control Hardware Organization

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.

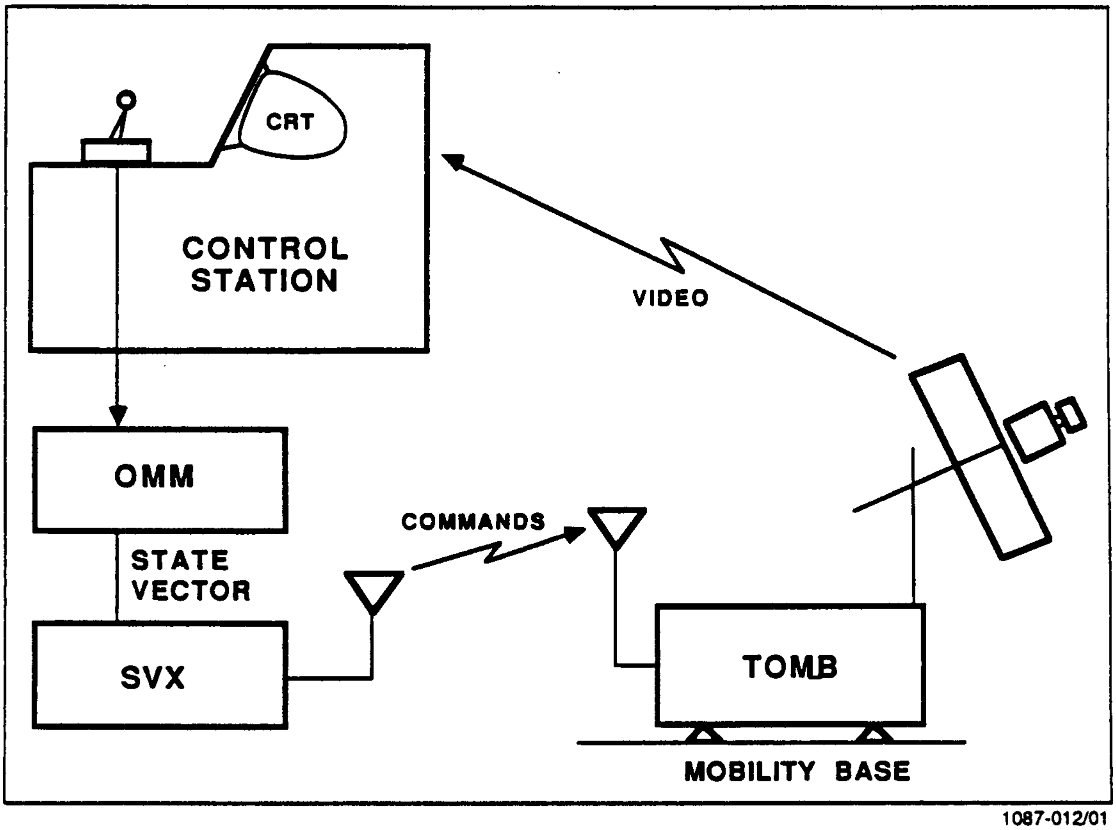


Figure 1-16. Flat Floor Facilities -- Software Architecture

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

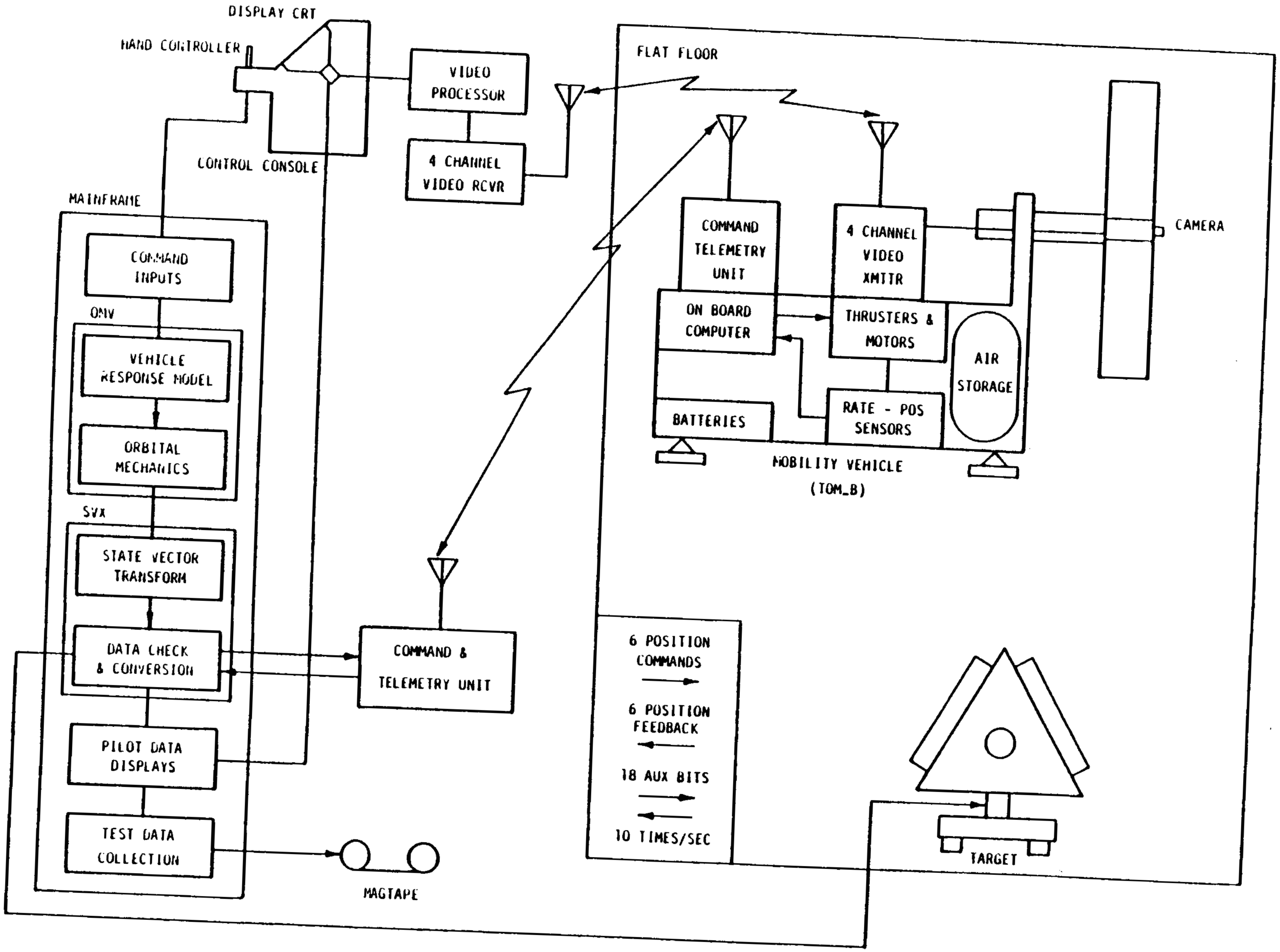


Figure 2-1. MSFC Flatfloor Simulation System

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

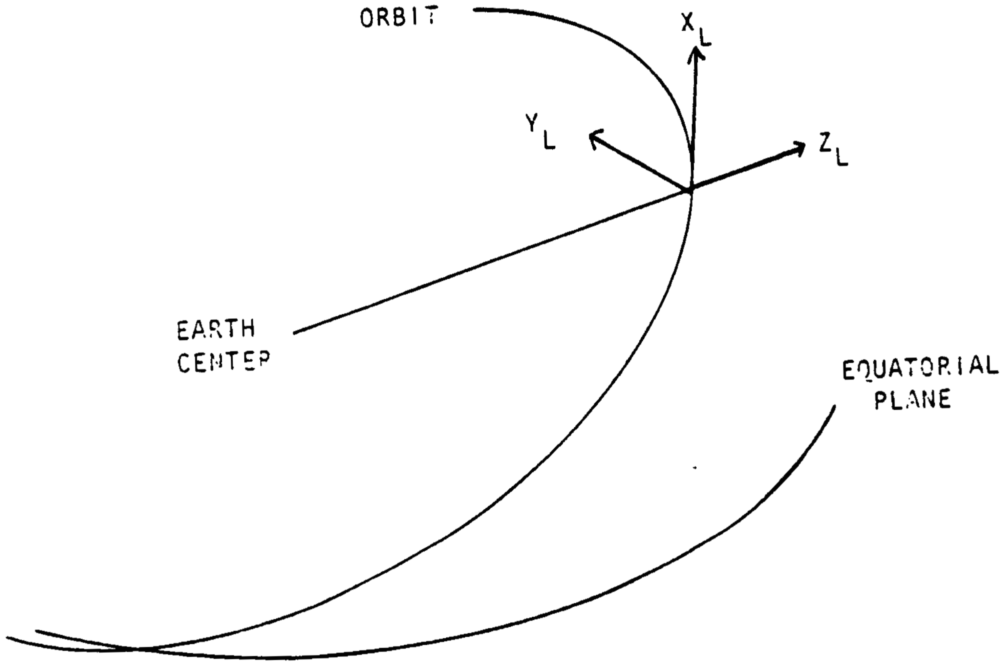


Figure 2-2. Local Vertical Frame (L)

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,

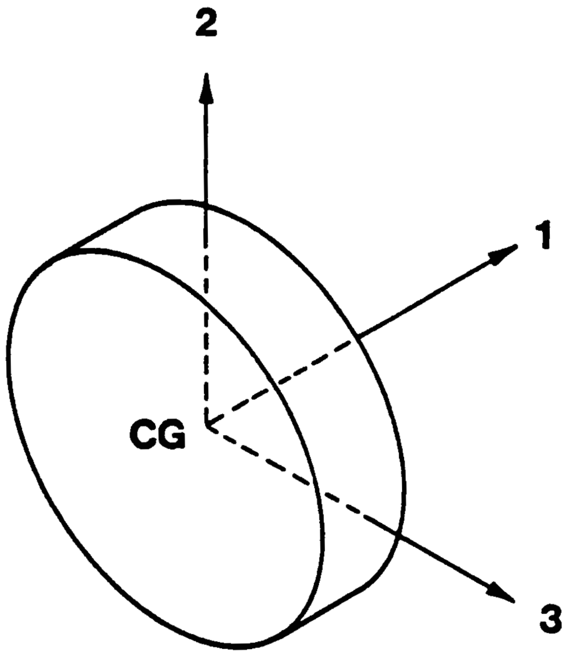


Figure 2-3. OMV Body Frame

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

Table 2-1. 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 $L = Iw$ is the angular momentum vector and τ is the applied torque. I is the moment of inertia tensor and w is the body rate. The solution can be drastically simplified by choosing the body axes 1, 2 and 3 such that I is diagonal [6,7], that is:

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

Remember that $w = [w_1,w_2,w_3]^T$ is obtained from the hand controller signals. The solution of the rotational equations of motion yields ϕ , θ and ψ the three Euler angles. The order and sense of rotation is chosen in the conventional manner [8], that is:

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

To reduce computational overhead, quaternions are used to specify the attitude of the OMV rather than the Euler angles themselves. It has been proven that the two 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 α 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 α is less than the algebraic sum of ϕ , θ and ψ. The angle α and the Euler axis can be expressed in terms of the quaternion q as:

$$\cos\frac{\alpha}{2} = q_4$$

$$\mathbf{E} \quad = \quad (\mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3) \; / \; (q_1 + q_2 + q_3)^{\frac{1}{2}}$$

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 <u>relative to some non-rotating frame</u>. Suppose further that an instant later, the vehicle's attitude has changed, having rotated by ϕ, θ and ψ. These angular displacements are measured relative to the rotated body frame. If the new attitude is described by a second quaternion q_2, the attitude of the vehicle, <u>relative to the non-rotating frame</u> [11,12] is then given by

$$q = q_1 q_2$$

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

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

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

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:

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

Here, the position and velocity vectors $[X,Y,Z]^T$ and $[X,Y,Z]^T$ refer to the position and velocity of the OMV relative to the target vehicle, as expressed in Local Vertical Frame. ω is the orbital velocity, and $A = [A_x,A_y,A_z]^T$ 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 $w = [w_1, w_2, w_3]^T$ is known, one can calculate L_B in body frame using the relation

$$L_B = Iw$$

$$L = C^{-1} L_B$$

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

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

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

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

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

B) Numerical Solutions:

A three step Adam-Bashforth method [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 compatable 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.

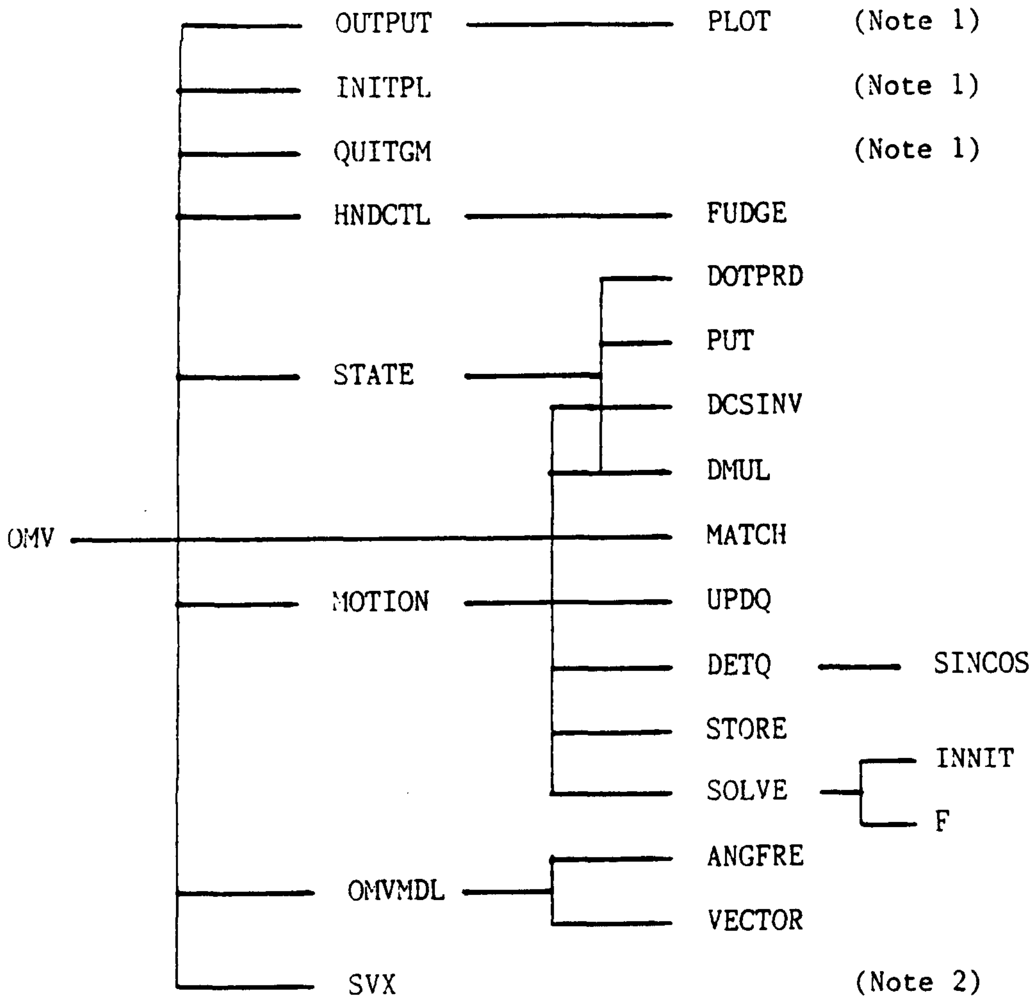
2.6 TESTING AND RESULTS

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

Figure 2-4. OMV Heirarchial Chart



Note 1 : Hardware incompatible graphics package.

Note 2 : Vector Transformation Module. See Reference 1.

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

Thus, the equations of motion reduce to:

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

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

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

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

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

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

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

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

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

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

where G is the universal gravitation constant, M_e is the mass of the earth, R_0 is the mean earth radius and H is the altitude. Note that at higher orbits, Ω approaches 0 and the equations of motion approach

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

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

$$X_1 = 0, \quad X_2 = X_3 = 0$$
$$V_1 = 0.05, \quad V_2 = V_3 = 0$$

The results shows that the two solutions agree to better than 3×10^{-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 Ω is quite small. When Ω = 1.0 is used, (this angular frequency does not make sense physically, as it represents an orbit well below the earth's surface, but

Time in Minutes	X (meters)		Z (meters)	
	Numerical	Analytical	Numerical	Analytical
0	0.000000	0.000000	0.000000	0.000000
5	13.746736	13.746736	5.271240	5.271240
10	20.161917	20.161917	20.427114	20.427114
15	12.828963	12.828962	43.576117	43.576178
20	-12.952950	-12.952952	71.829442	71.829444
25	-59.582227	-59.582233	101.660919	101.660923
30	-126.855533	-126.855544	129.347660	129.347664
35	-211.993176	-211.993191	151.434377	151.434380
40	-309.986003	-309.986022	165.164663	165.164664
45	-414.220544	-414.220565	168.824984	168.824985
50	-517.304365	-517.304388	161.958539	161.958536
55	-611.988644	-611.988666	145.422253	145.422248
60	-692.072815	-692.072834	121.279843	121.279843

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

Table 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:

$$X_1 = X_2 = X_3 = 0$$
$$V_1 = V_2 = V_3 = 0$$

The hand controller signals were chosen to yield a constant acceleration along the X-axis in the LVF, that is $a = [0.025,0,0]^T$, and the orientation of the OMV is chosen to be aligned to the LVF at $t = 0$. The result after 4 seconds of simulation is shown in Table 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:

$$X_1 = X_2 = X_3 = 0$$
$$V_1 = V_2 = V_3 = 0$$
$$r = p = y = 0$$

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

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

Initial conditions :

$X_1 = X_2 = X_3 = 0$ and

$V_1 = V_2 = V_3 = 0$

Note : All quantities are expressed in Local Vertical Frame.

Table 2-4. OMV Acceleration Along +X Direction

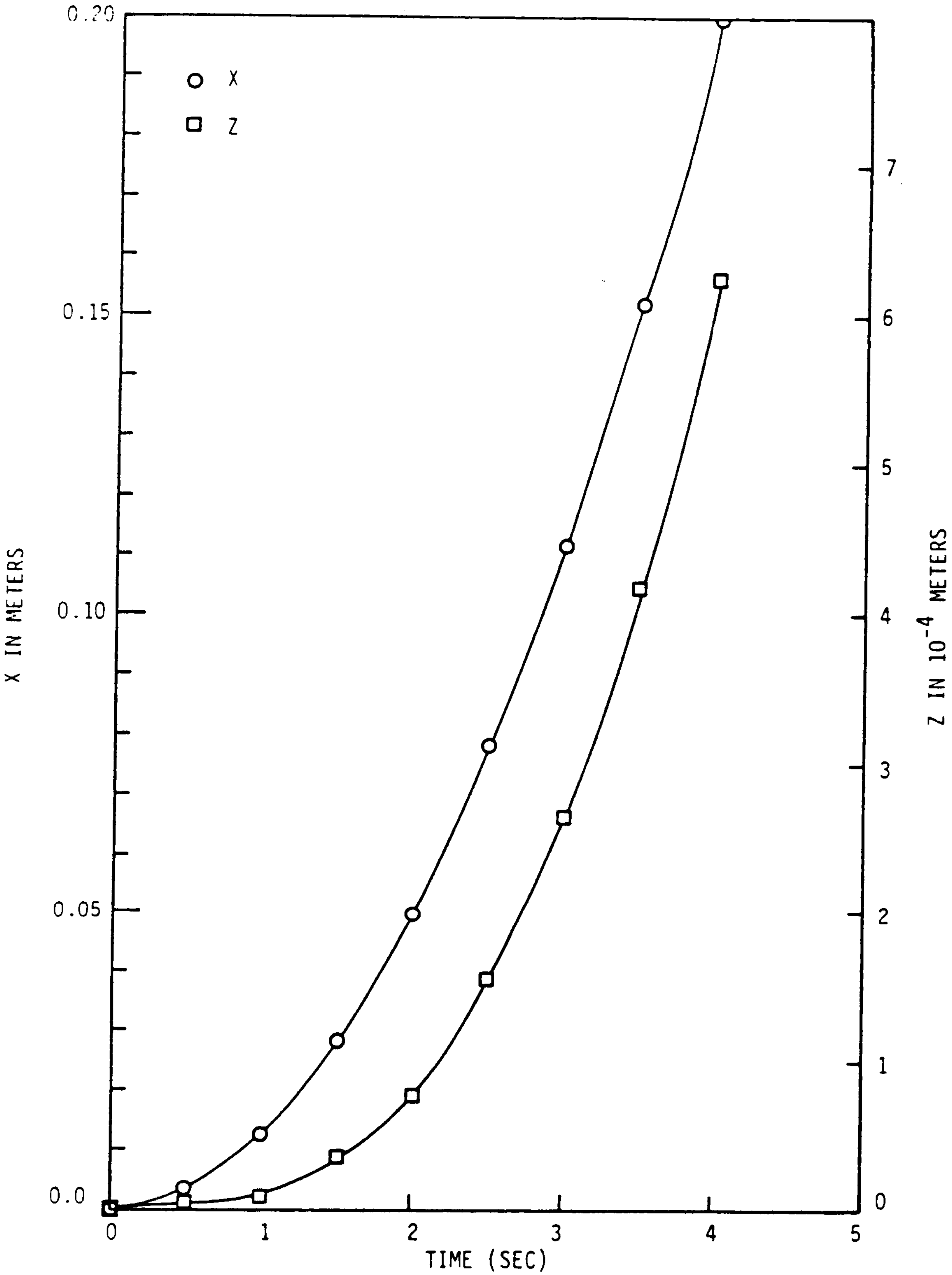


Figure 2-5. Translation Along X-Axis

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

When the OMV is executed in this mode, the state vectors are fed into the SVX module, with the result that the state vector is translated into a sequence of commands CMD. This sequence of commands is to be transmitted to the flat floor. Table 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 1-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 functions properly. By way of example, a pure yaw motion of the OMV demands that the mobile base describes a circular path as shown in Figure 2-8. There is just one area that needs further investigation, namely timing considerations. This system must be able to complete all the computation within 0.1 second -- a major cycle. When the system is uploaded to the PDP 11/34, it was discovered that the computer took more than 0.1 seconds to complete one major cycle of computation. At this juncture, one can take

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

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

Table 2-5. OMV--Pure Pitch Motion at 0.017453 rad/sec

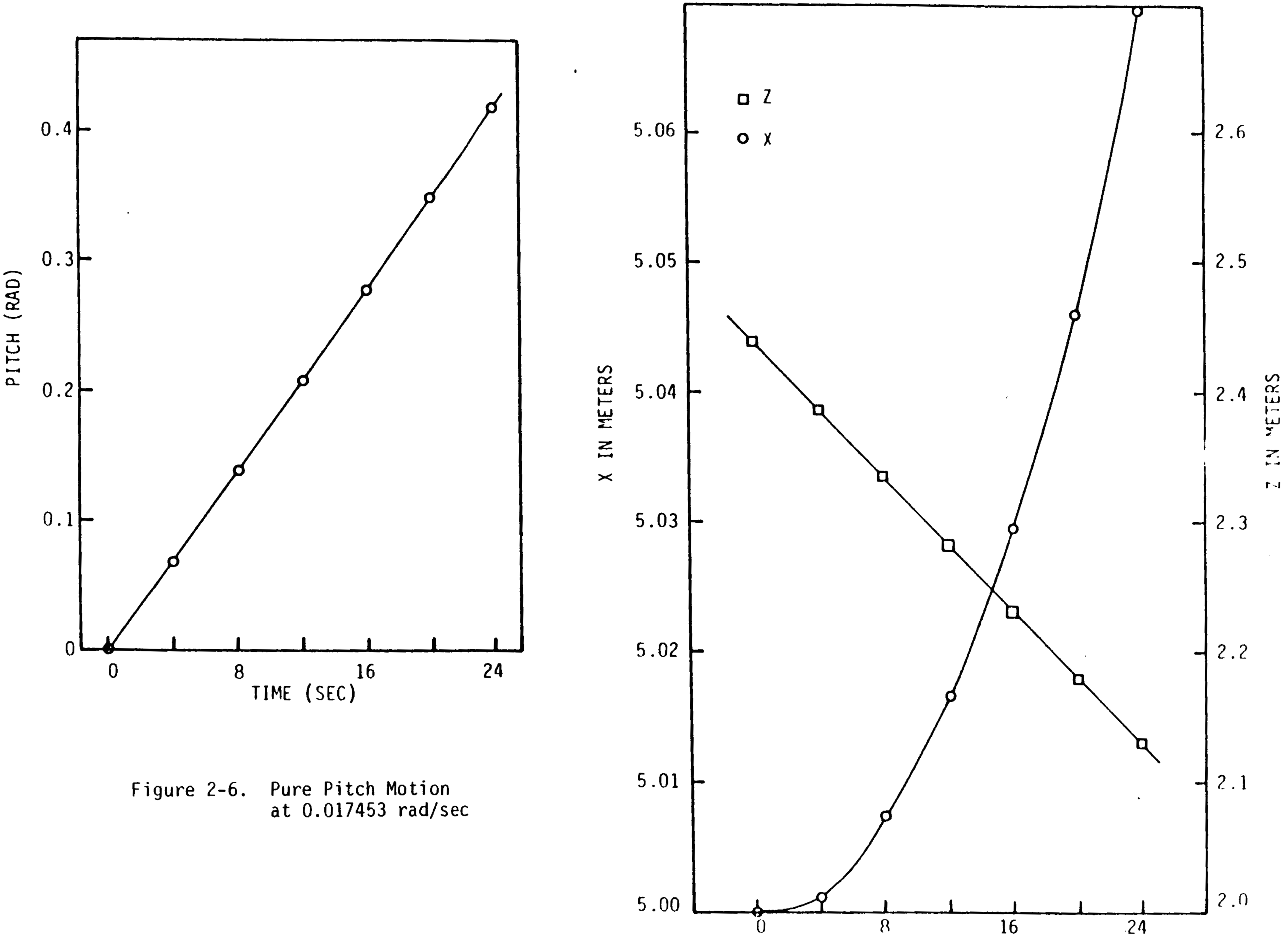


Figure 2-6. Pure Pitch Motion at 0.017453 rad/sec

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

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

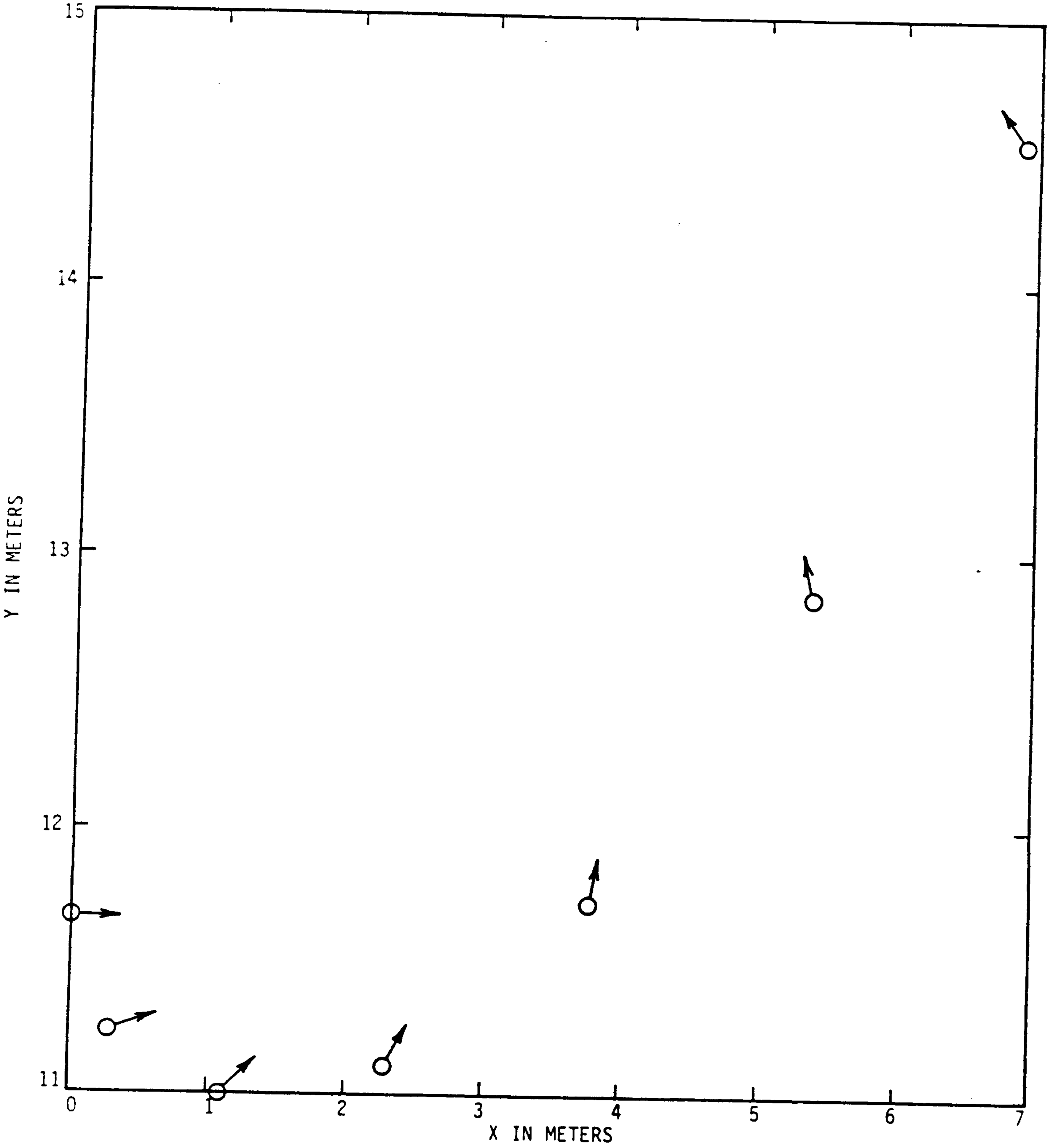


Figure 2-7. Trajectory of Mobile Base When OMV is Executing a Translation & Yaw

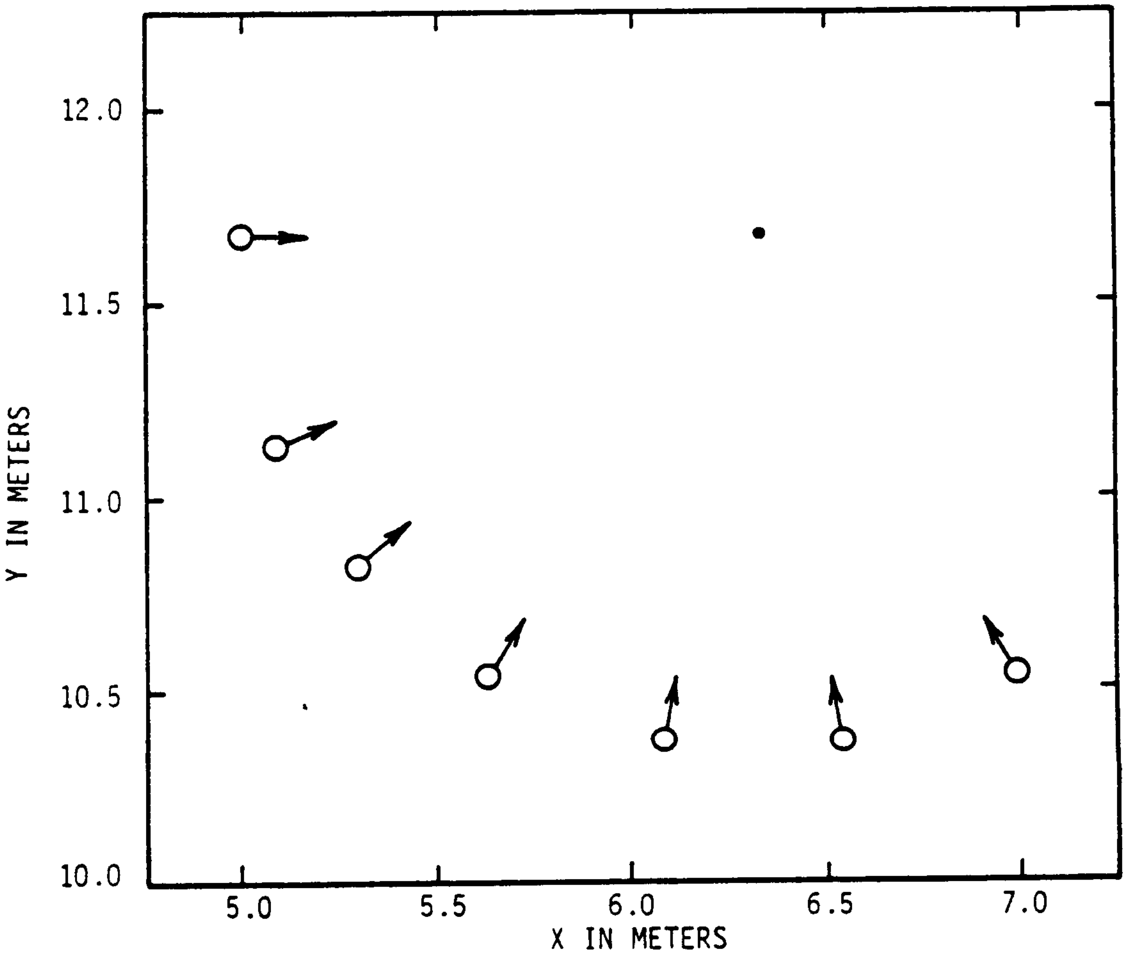


Figure 2-8. OMV Pure Yaw Motion

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

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

Table 2-8. Optimized OMV Execution Times Per Major Cycle as a Function of Number of Steps N

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

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:

Component	Symbol	Meaning
1	X	Position of the target vehicle relative
2	Y	to the OMV in local vertical frame LVF
3	Z	
4	V_x	Relative velocity of the chase vehicle
5	V_y	in LVF
6	V_z	
7	L_x	Angular momentum vector in LVF
8	L_y	
9	L_z	
10	q_1	Attitude quaternions in body frame

11	q_2	
12	q_3	
13	q_4	
14	m	Mass of OMV

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

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

Component	Position Control		Rate Control		Coord. System
	Symbol	Meaning	Symbol	Meaning	
1	y	yaw of TOM-B	$\dot{y}$	yaw rate	body frame
2	X	position of	V_x	velocity of	LVF
3	Y	TOM-B	V_y	TOM-B	
4	Z	pos of pivot	V_z	vel of pivot	
5	p	pitch angle	$\dot{p}$	pitch rate	body frame
6	r	roll angle	$\dot{r}$	roll rate	
7	1	pos. control	0	rate control	

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

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.

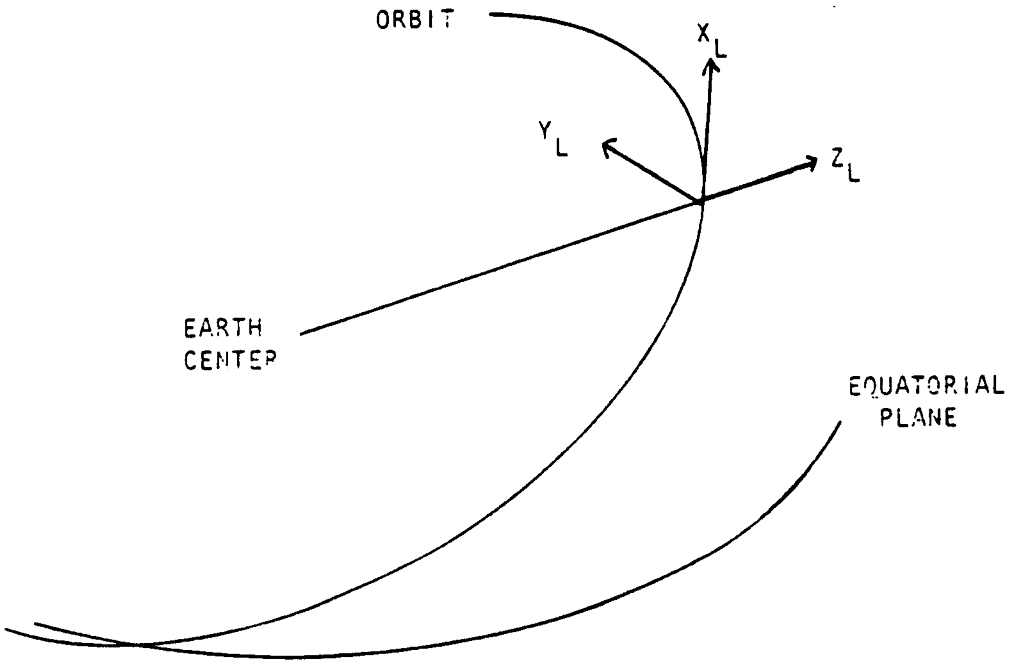


Fig. 3-1. Local Vertical Frame (L)

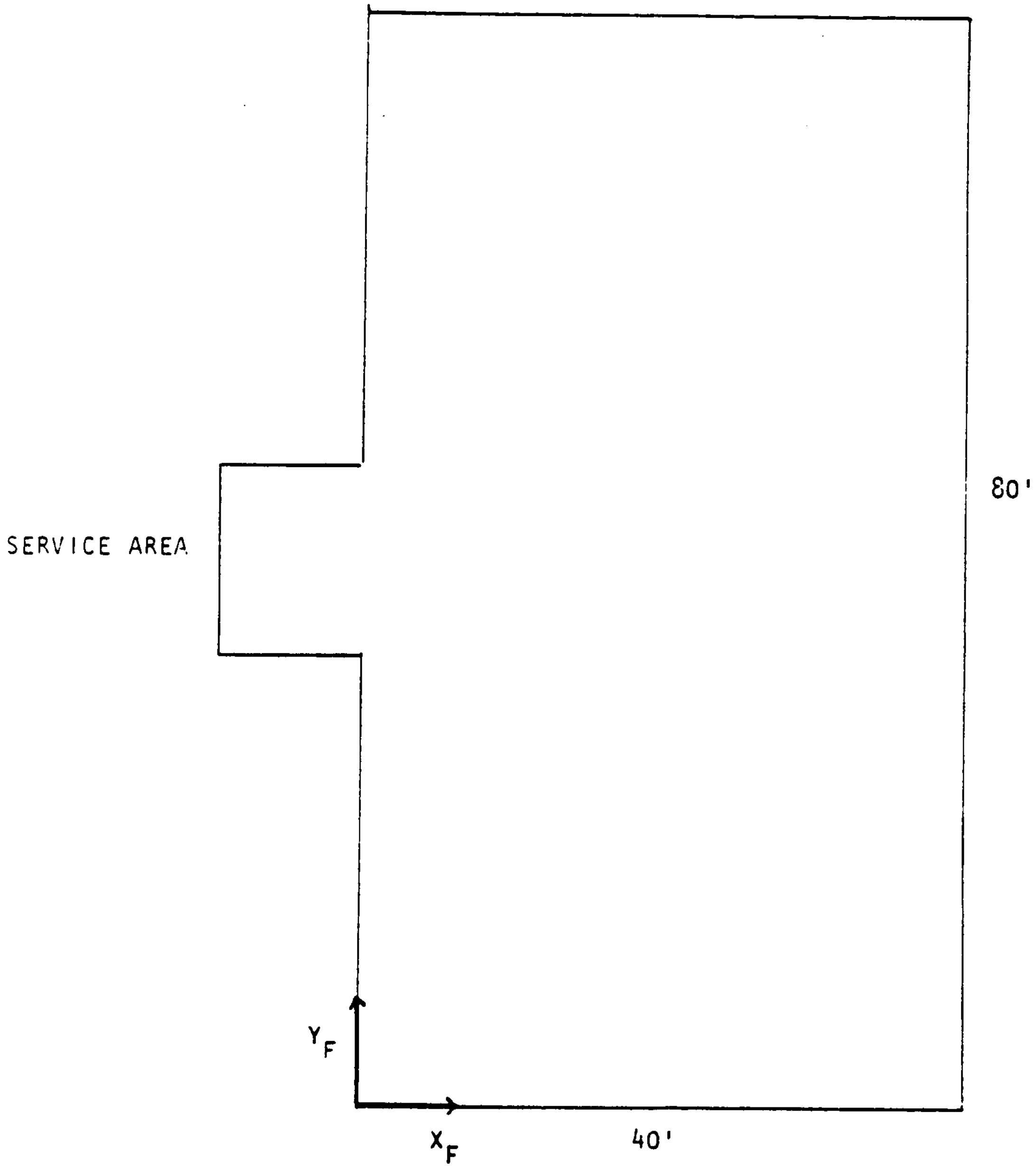


Figure 3-2. Floor Coordinates (F)

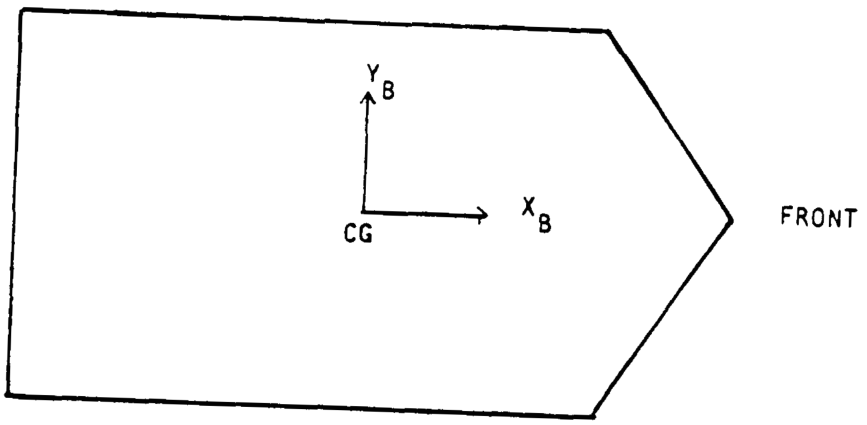


Figure 3-3. TOM-B Body Frame (B)

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, 1, 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:

$$\begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix}$$

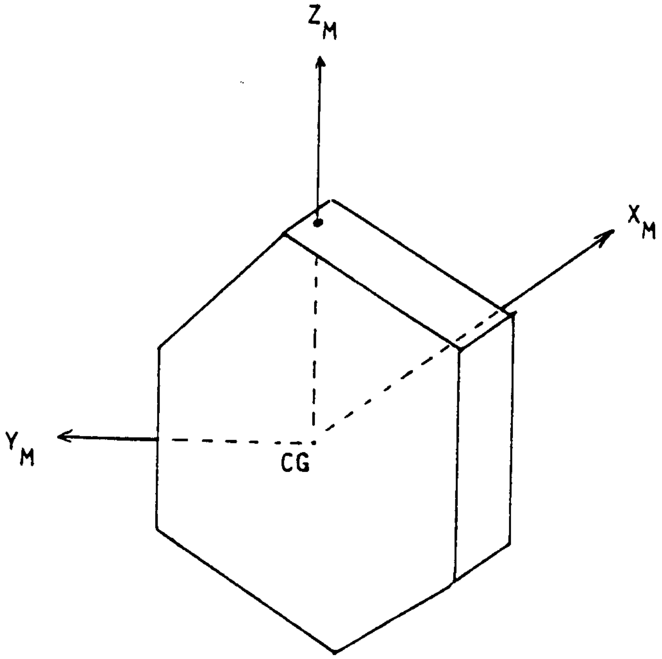


Figure 3-4. Mock-Up Module Body Frame (B)

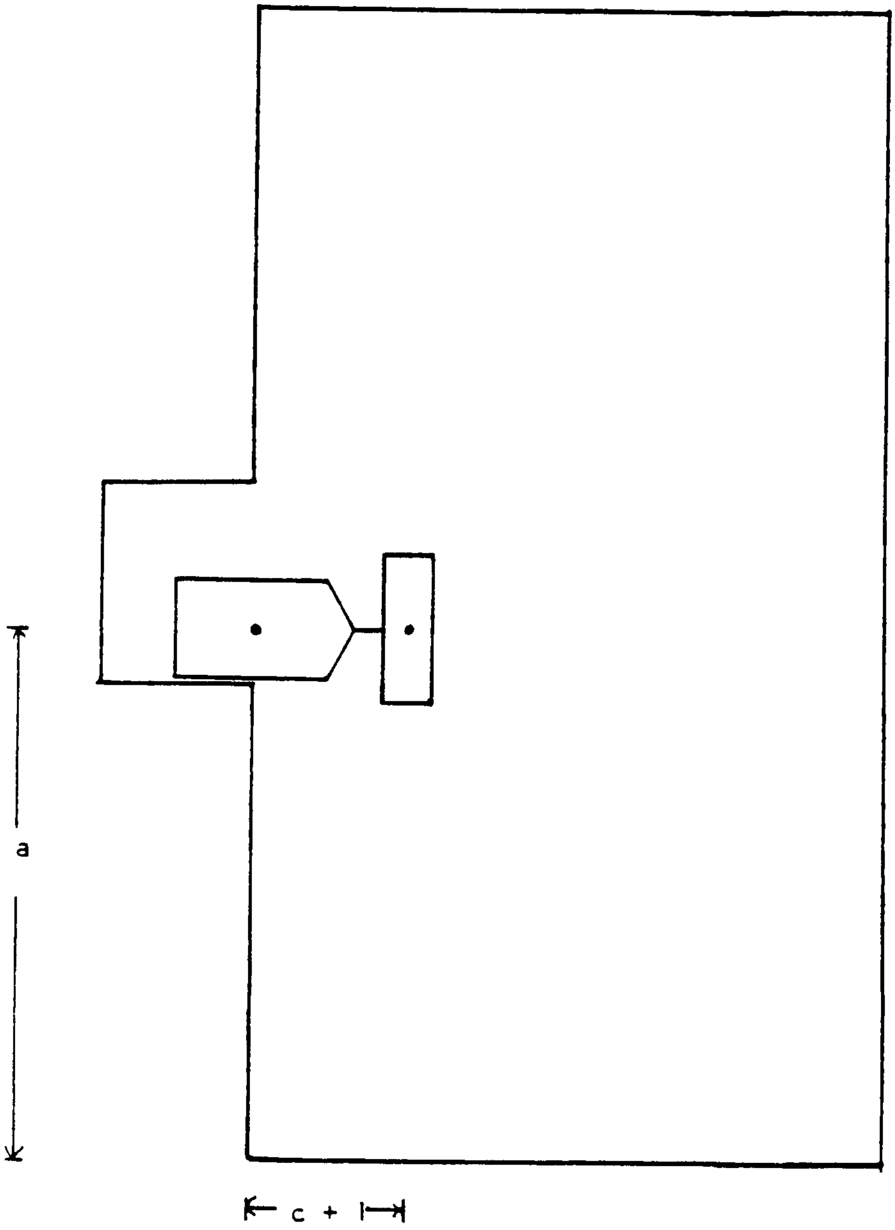


Figure 3-5. Initial Position (top view)

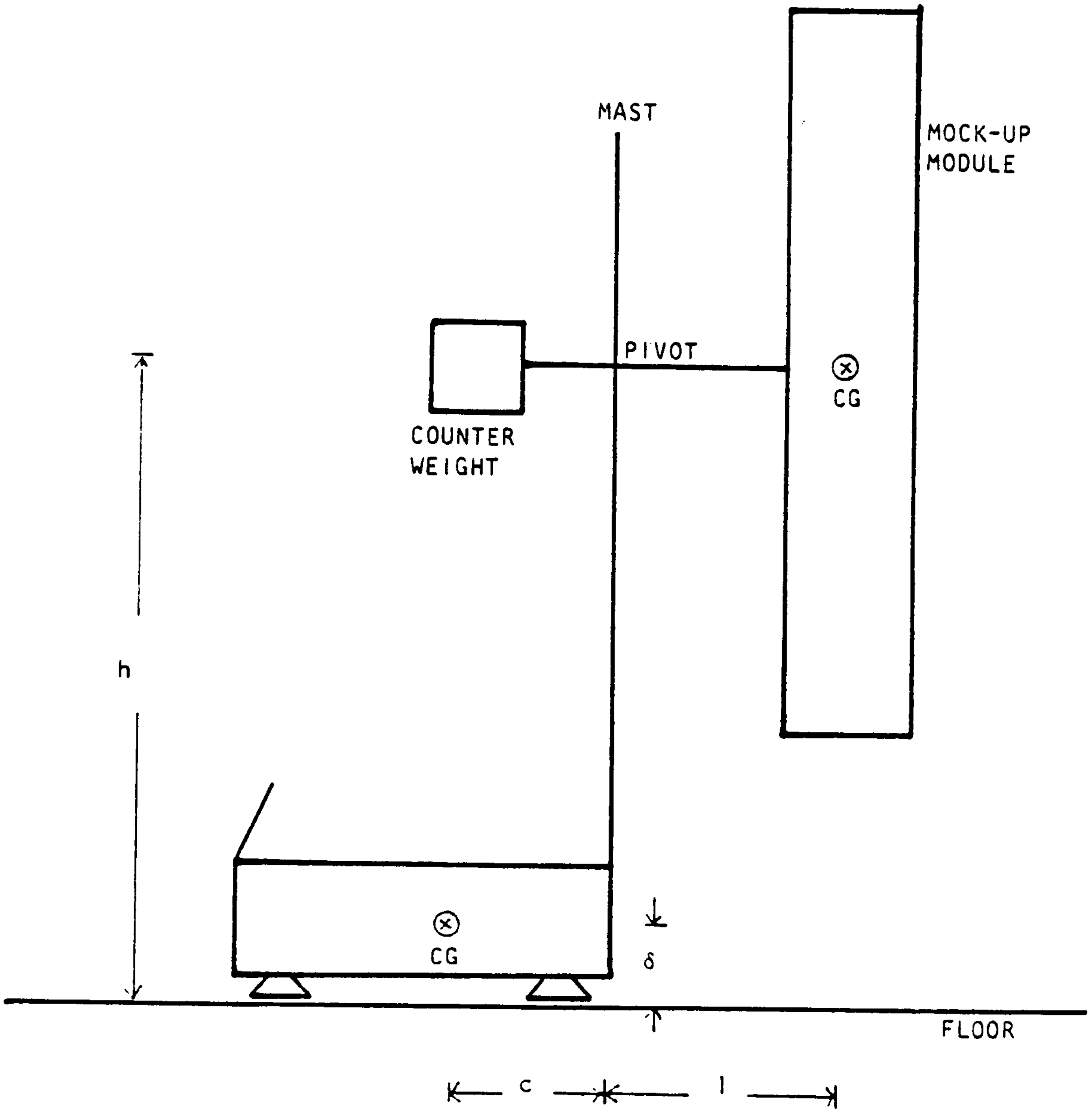


Figure 3-6. Initial Position (side view)

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:

$$\begin{bmatrix} X_M \\ Y_M \\ Z_M \end{bmatrix} = \begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} + \begin{bmatrix} c + l - X_0 \\ a - Y_0 \\ h - Z_0 \end{bmatrix} \qquad [I]$$

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 $[r_M, p_M, r_M]^T$ 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:

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

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

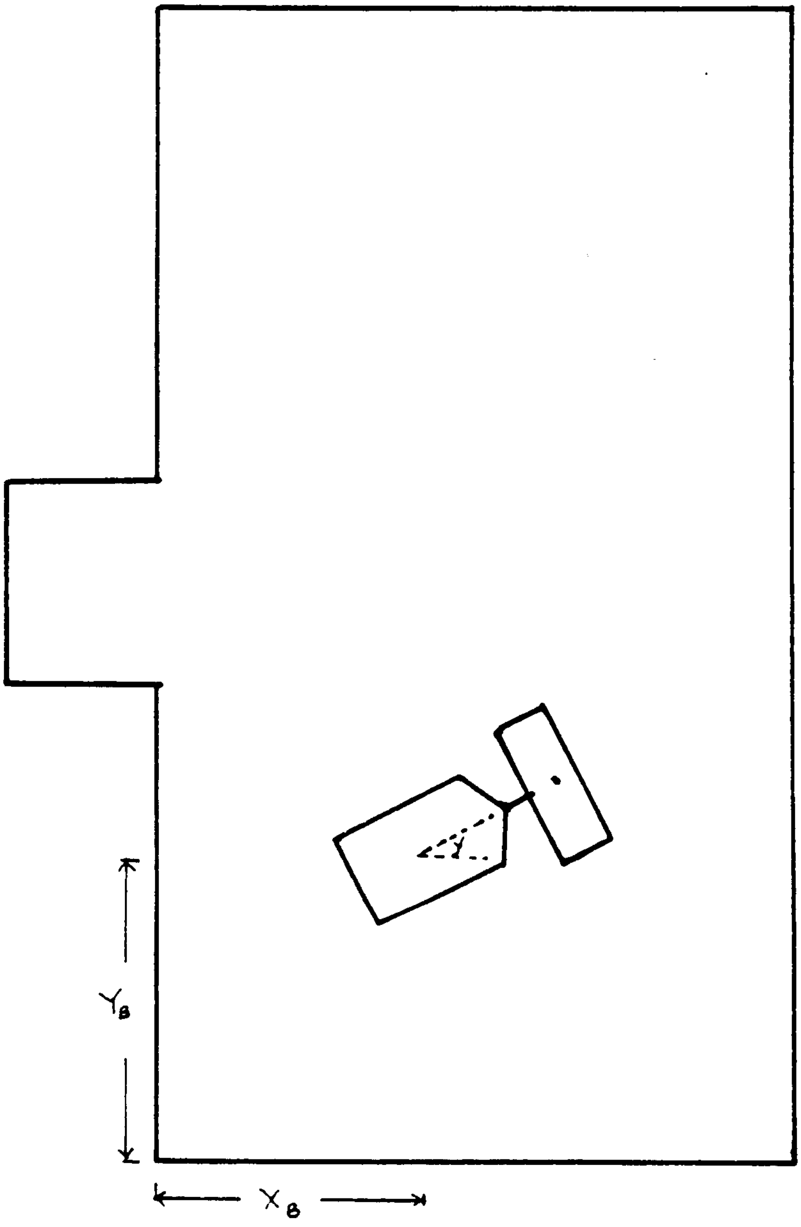


Figure 3-7. Position and Yaw of TOM-B

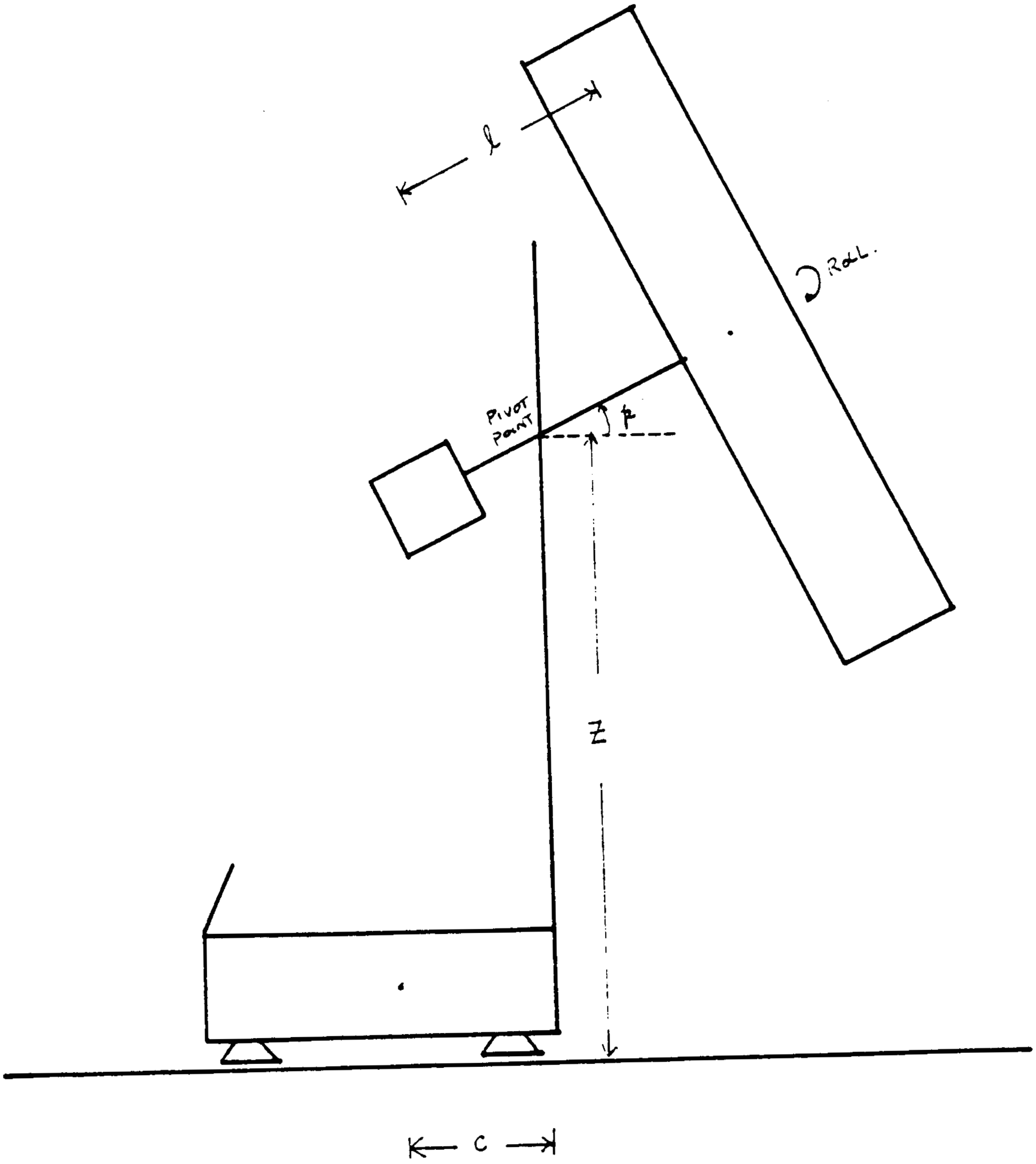


Figure 3-8. Pitch and Roll of Mock-Up Module

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

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

The quaternions q_1, q_2, q_3, q_4 from the state vector specifies 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

$$\alpha = 2\cos^{-1}(q_4)$$
$$[\ O_x,\ O_y,\ O_z\ ]^T = (\mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3)\ /\ (q_1 + q_2 + q_3)^{0.5}$$

while their rates are $w_B = [w_1, w_2, w_3]^T$ 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

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

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

$$\begin{aligned} w_B &= [w_1, w_2, w_3]^T \\ &= I^{-1} L_B \quad = I^{-1} (A\ L) \end{aligned}$$

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 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 MODE <> 0 (rate control)

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

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

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

Case 3 MODE <> 0 and MODE <> 1

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

3.5 IMPLEMENTATION

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

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

SVX is compiled and tested using a 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

$$\begin{aligned} E &= [E_1, E_2, E_3]^T \\ &= [c + l - X_0, a - Y_0, h - Z_0]^T \end{aligned}$$

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 displays the results in two dimensions. By way of example, a sequence of 60 state vectors of the form:

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

is generated. This set of state vectors simulates 50 seconds of run time in which position control is used. The meaning of this state vector is that the OMV is to remain stationary, but executes a yaw at a rate of 15° 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

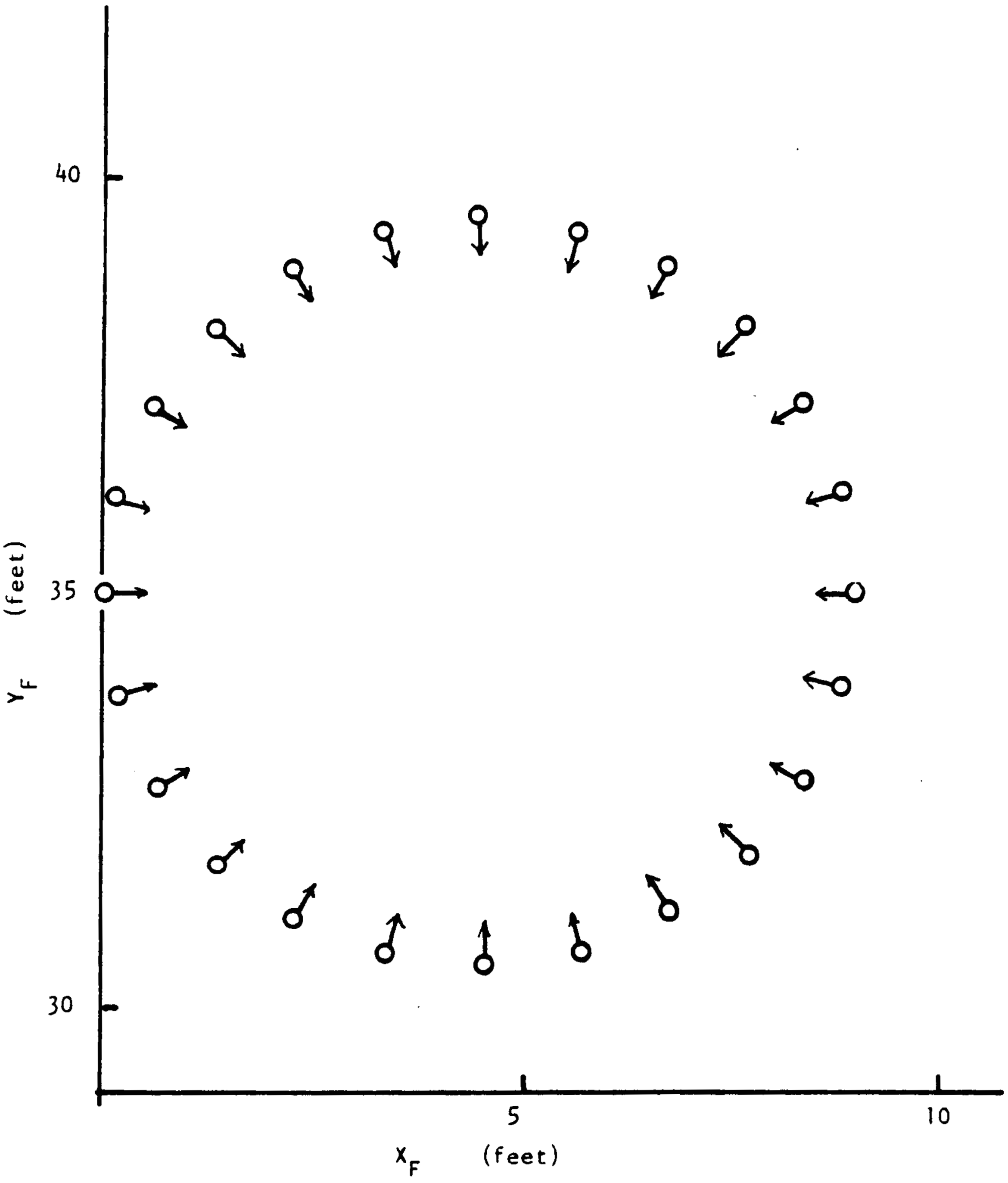


Figure 3-9. Position of TOM-B in Floor Coordinates

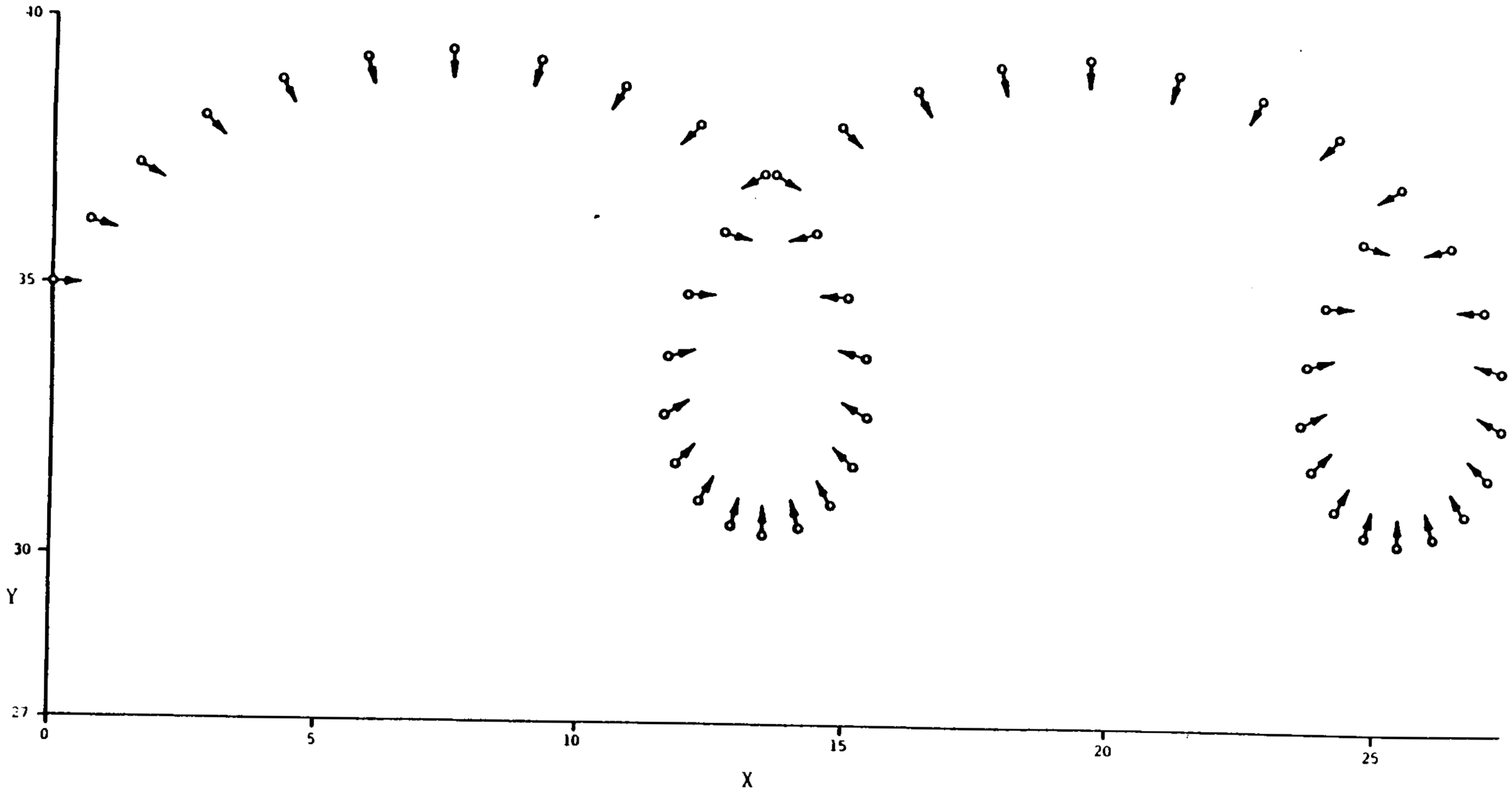


Figure 3-10. Trajectory of TOM-B

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 simular results have demonstrated that the module SVX is functioning properly and that correct command strings are obtained. One must remember that the outputs of this module are commands to TOM-B, indicating the desired position, (or velocity) and attitude (or angular rates). The proper interpretation, and subsequent execution, of these commands are performed by the TOM-B Executive, and is outside the scope of the SVX module.

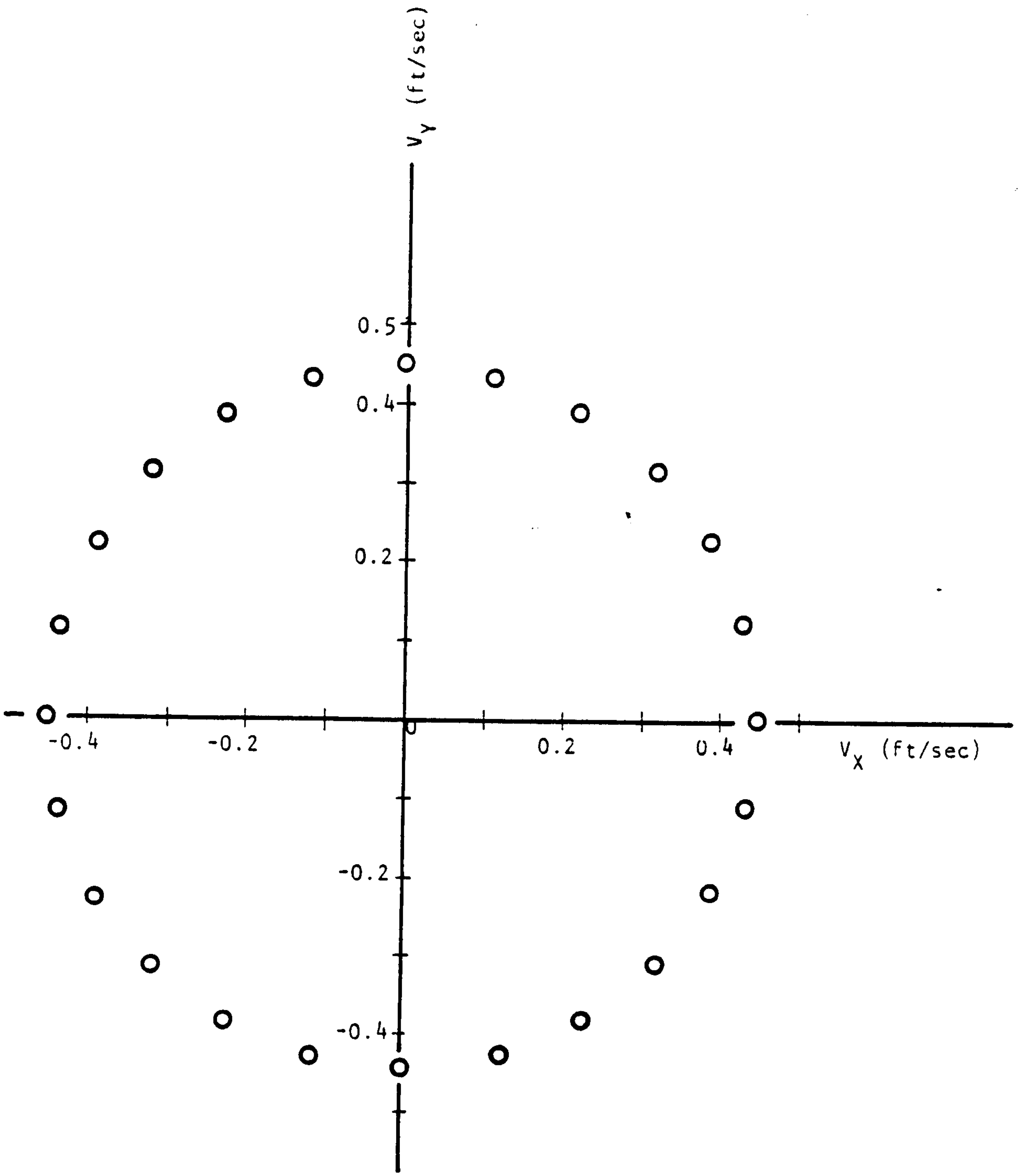


Figure 3-11. Velocity Components of TOM-B

Chapter 4

MOBILITY BASE ON-BOARD CONTROL LOGIC
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 FX and FY, 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 FX and FY are known, the individual impulses FX1, FX2, FY1, and FY2 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,) determined from the navigation system. Here is the orientation of the vehicle. The desired position and orientation is dictated by the command string (X_{CMD}, Y_{CMD}, $_{CMD}$) 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:

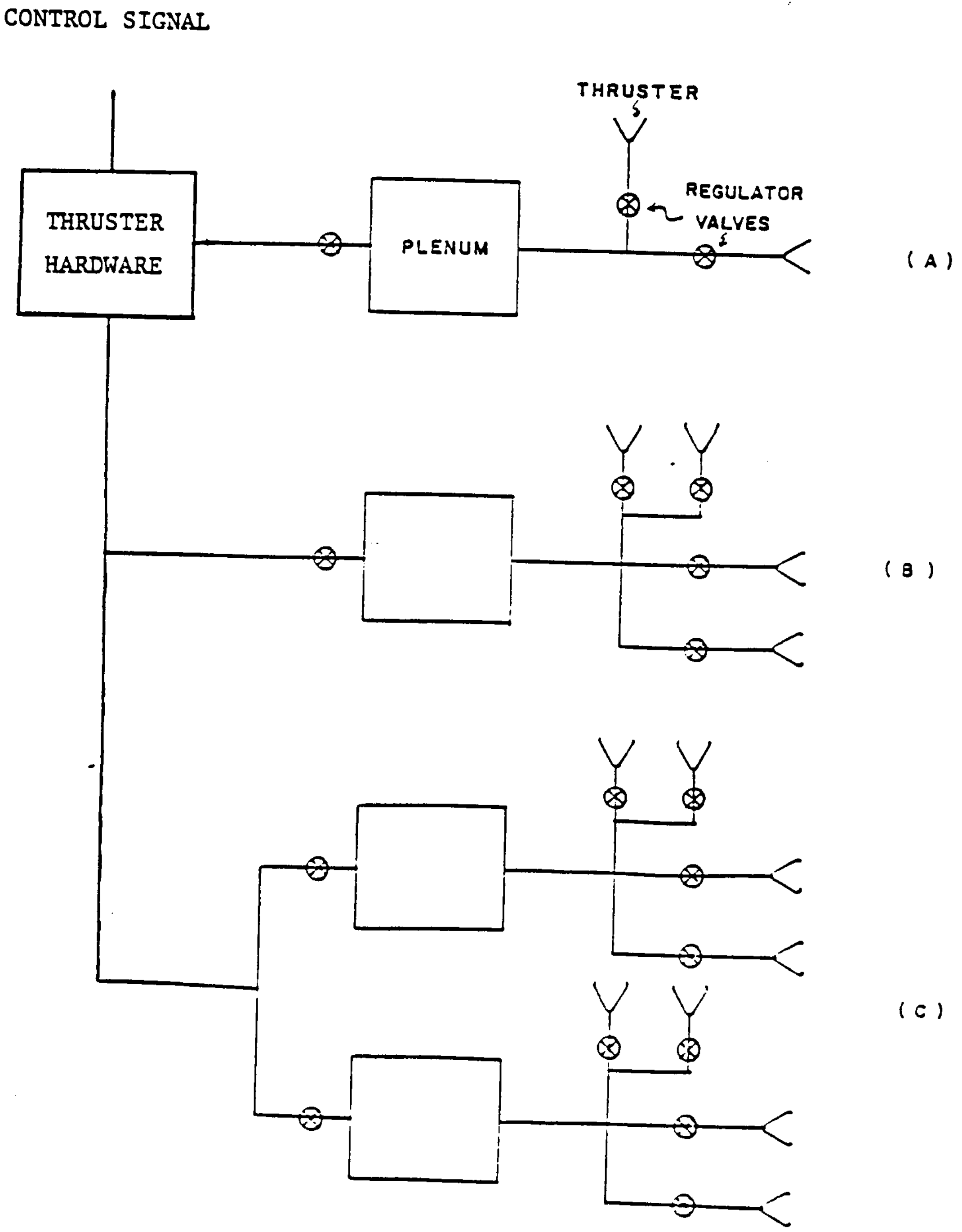
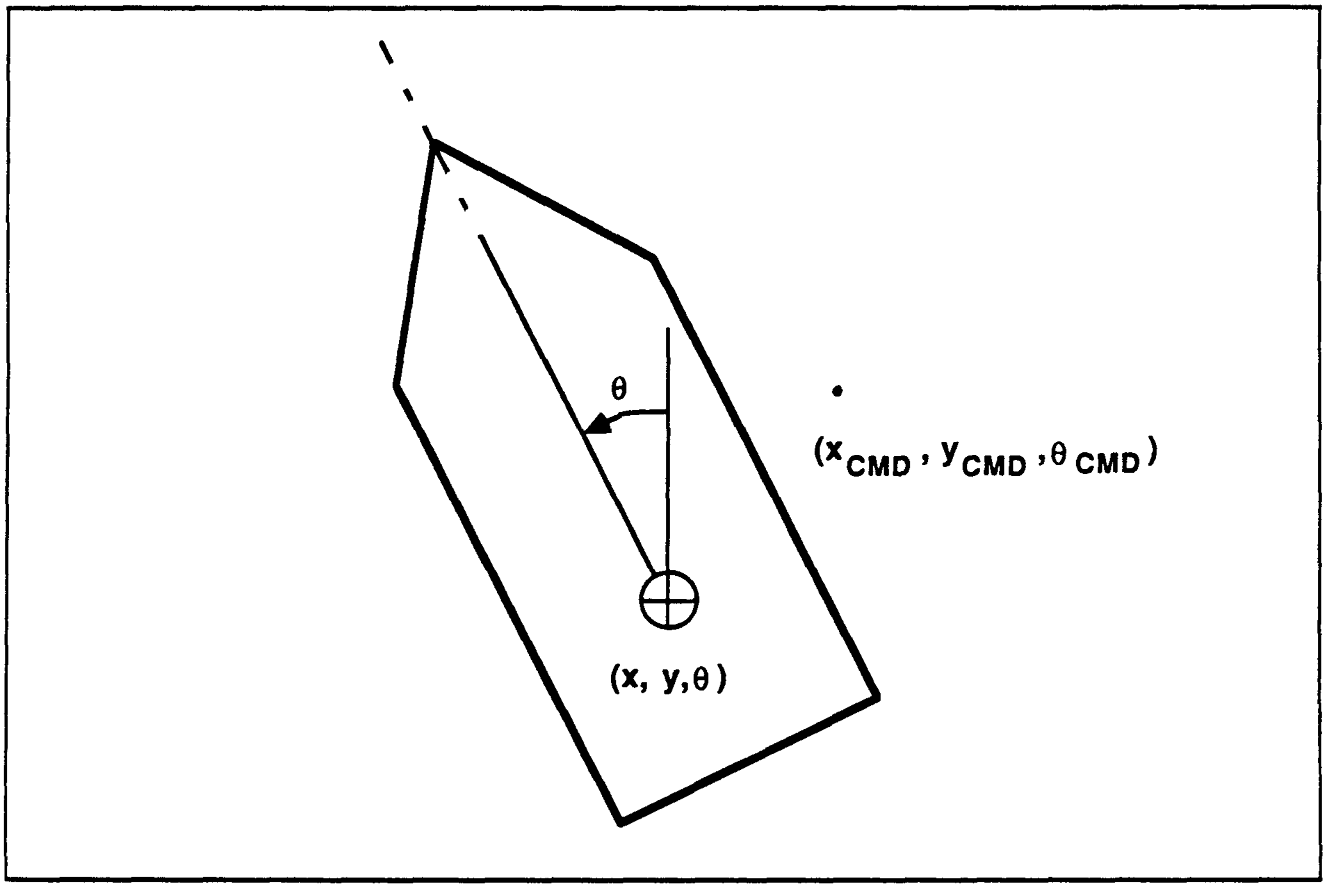


Figure 4-1. TOM-B Thruster Control Signal



1087-012/02

Figure 4-2. TOM_B Orientation Angle θ.

$$\begin{bmatrix} V_{ox} \\ V_{oy} \end{bmatrix} \begin{matrix} \dot{=} \\ = \end{matrix} \begin{bmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} V_x' \\ V_y' \end{bmatrix}$$

and the function g, is given by:

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

$$g(X, X_{CMD}, V_{ox}) = \frac{-V_{ox} + (V_{ox} + 2a(X_{CMD} - X))^{1/2}}{a} \quad \text{otherwise}$$

where

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

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:

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

where θ 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_{x1}, F_{x2}, F_{y1}, 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

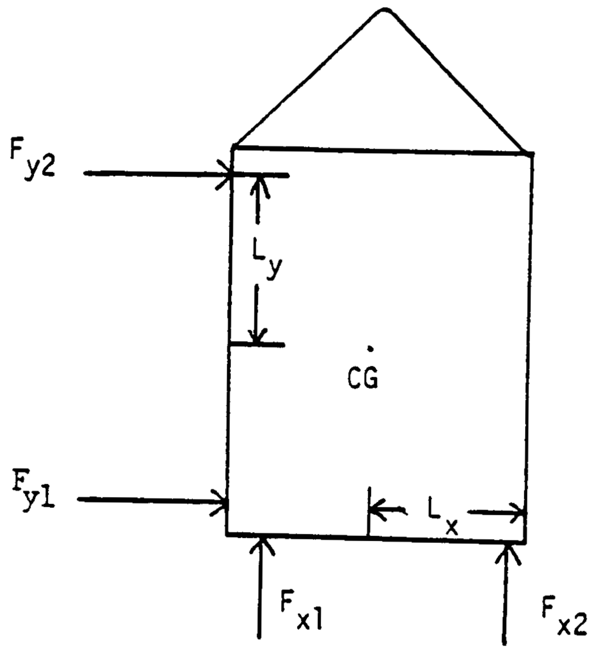


Figure 4-3. TOM-B Thruster Impulses

$$F_{x1} = F_x / 2 + \quad F_x$$

$$F_{x2} = F_x - F_{x1}$$

Case 2. $F_x > F_y$

In this case,

$$F_{x1} = F_x / 2 + T_0 / (2L_x)$$

$$F_{x2} = F_x - F_{x1}$$

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

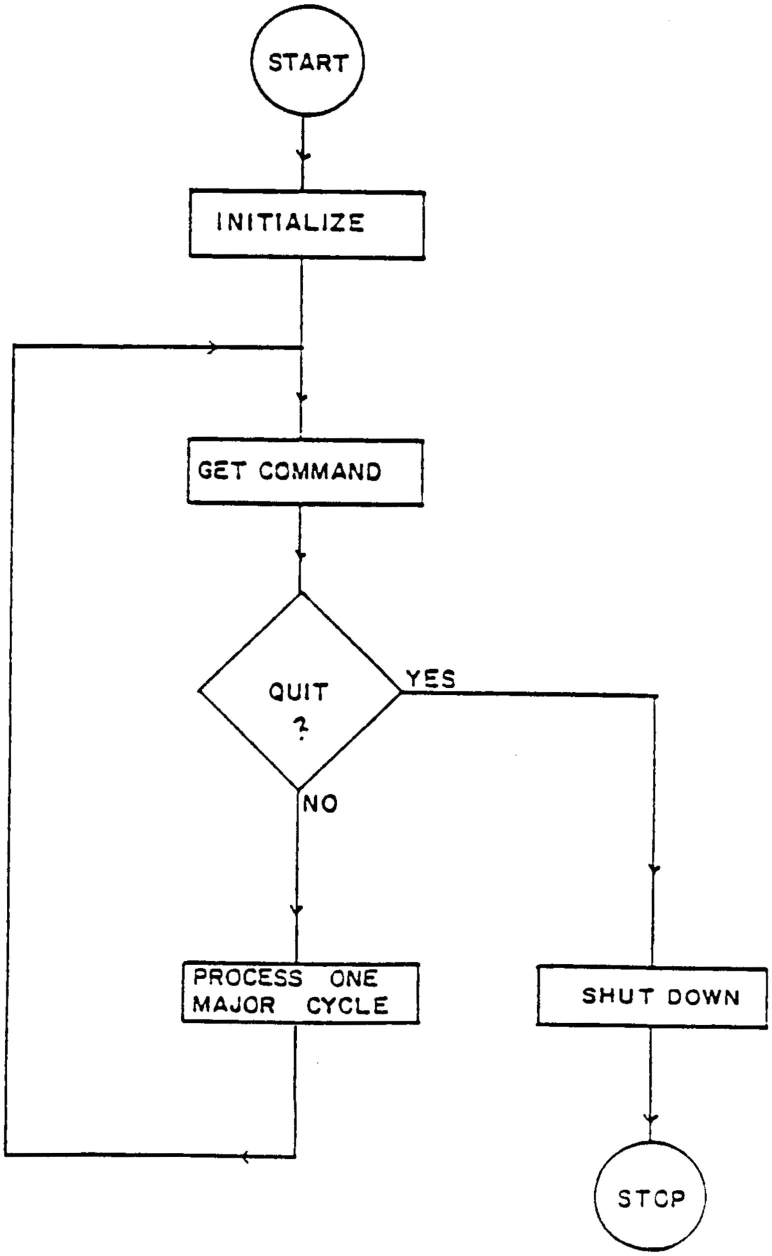


Figure 4-4. Control Software - Main Program

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

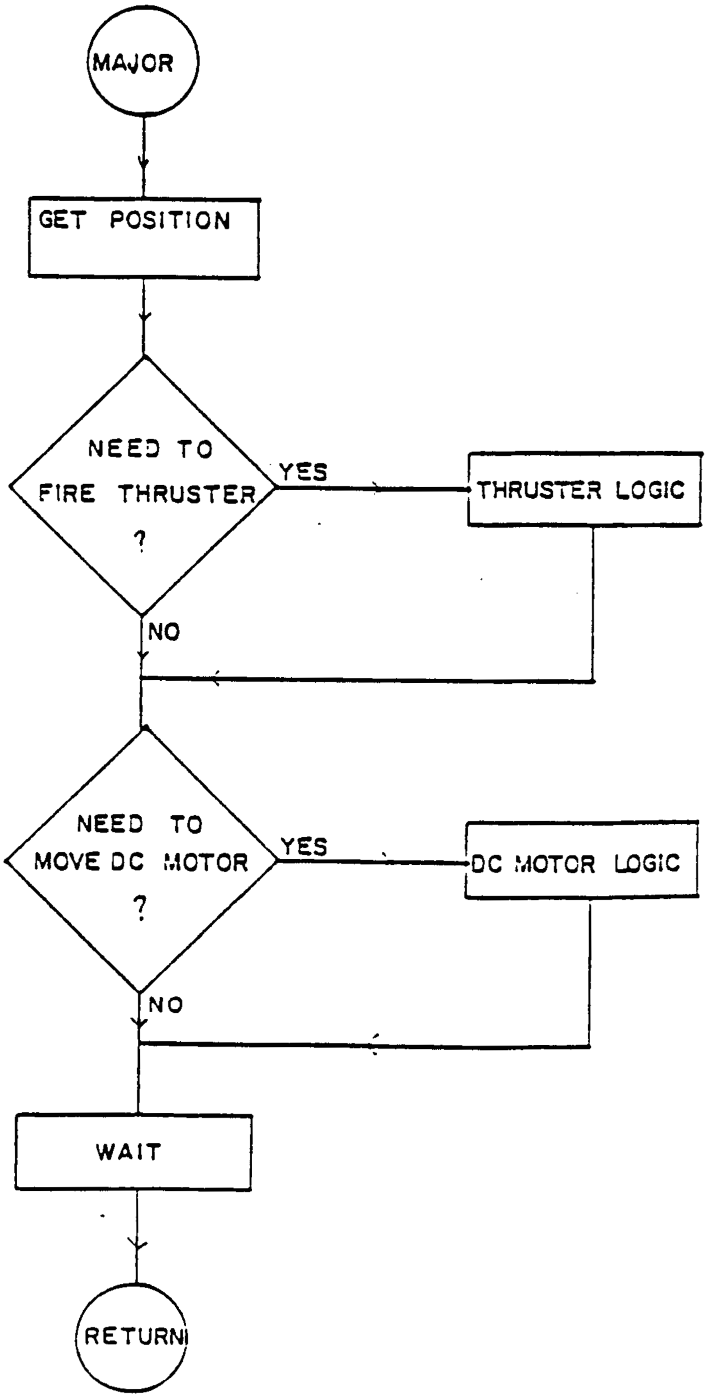


Figure 4-5. Control Software - Major Cycle

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 uninterupted 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 end 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.

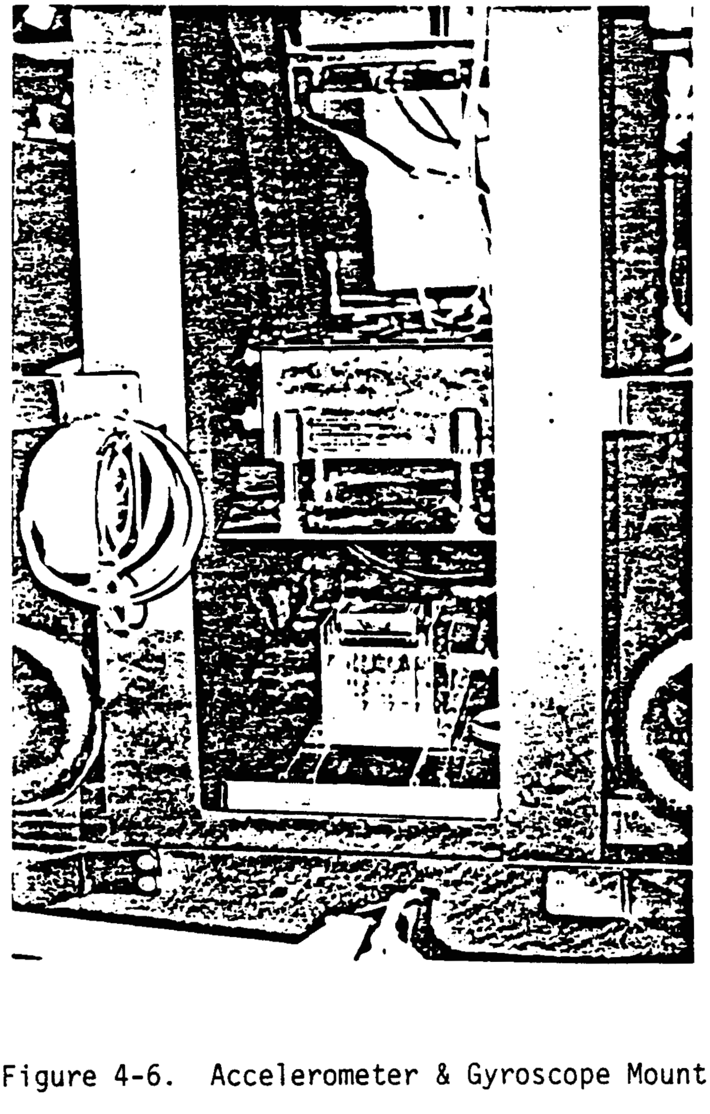


Figure 4-6. Accelerometer & Gyroscope Mounting

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 seperate 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

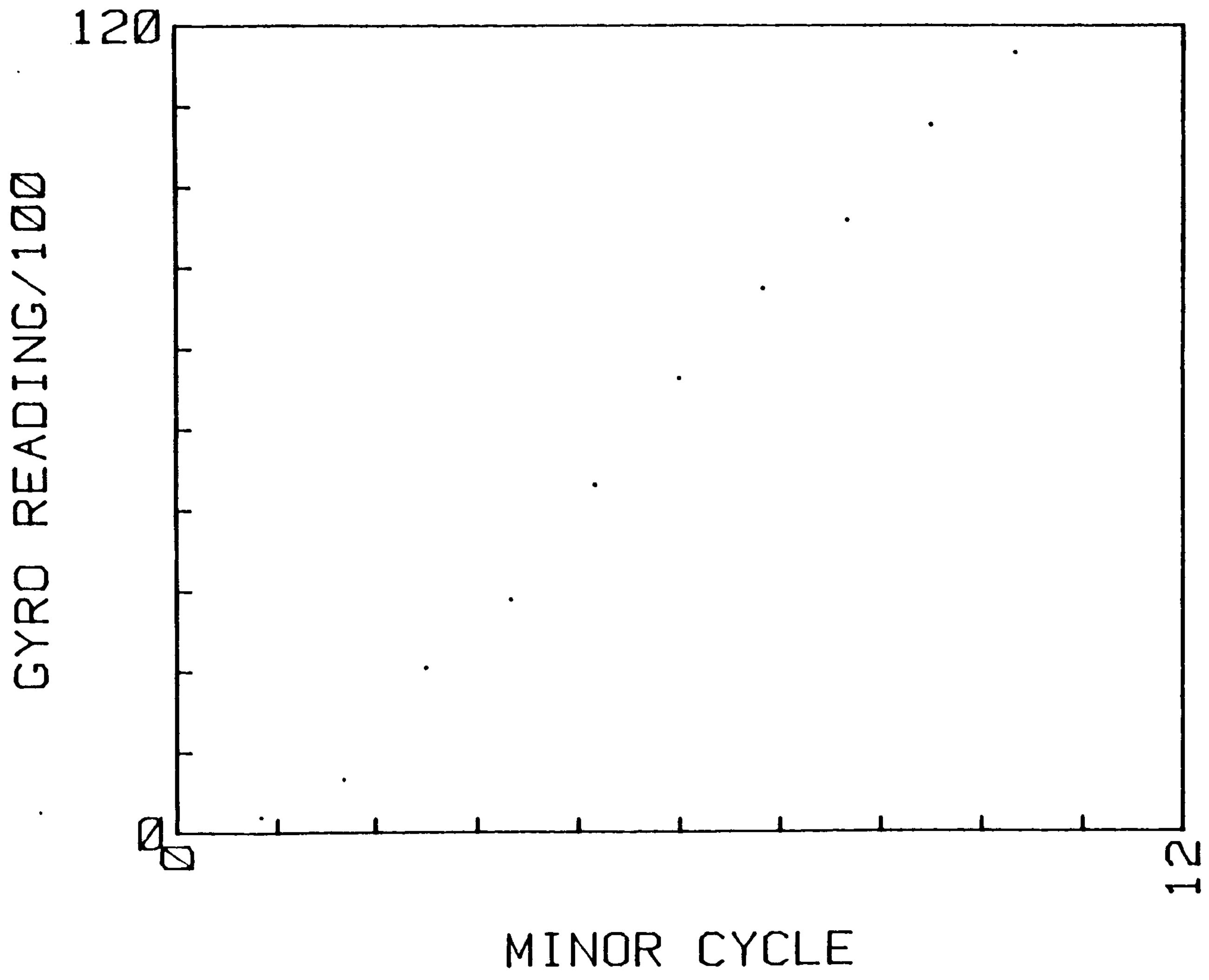


Figure 4-7. Gyroscope Reading

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 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 ½ seconds after time $t = 0$. This change of slope represents the fact that the thrust level drops after ½ 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

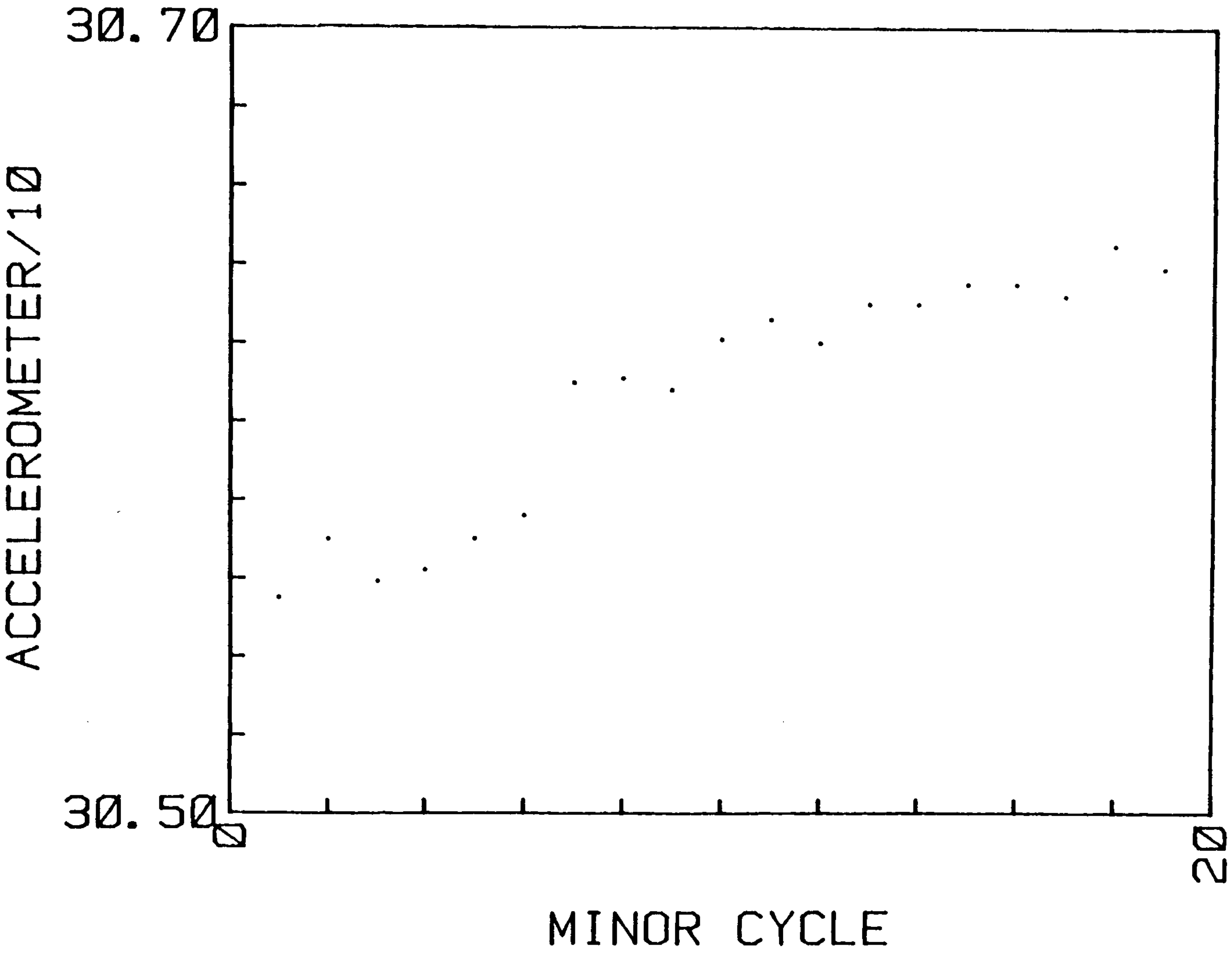


Figure 4-8. Accelerometer Reading

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 a 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 feed back subsystem. This was completed in January of this year. Mr. Ralph Kissel 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 feed back 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° 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

[1] "Teleoperator Maneuvering System Preliminary Definition Study," NASA Technical Report, May 9, 1984. Contract # NAS8-35670.

[2] Smith, E.C., Personal Communication, 1985.

[3] Teoh, W. "OMV--A Simplified Mathematical Model of the Orbital Maneuvering Vehicle," Interim Report, 1985.

[4] Teoh, W., Walls, J.L.S., Roe, F., et. al., "Orbital Maneuvering Vehicle - A Teleoperated Module," ROBOTS 9 Conference, June 1985.

[5] Bryan, T., Personnal Communication, 1984.

[6] Goldstein, H., "Classical Mechanics," Addison-Wesley, Inc., 1965.

[7] Symon, K., "Mechanics," Addison-Wesley, Inc., 1965.

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

[9] Robinson, A.C., "On the Use of Quaternions in Simulation of Rigid Body Motion," Wright Air Development Center, 1958.

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

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

[12] McCracken, D., "Numerical Methods with FORTRAN IV," John Wiley & Sons, 1972.

[13] Smith, J., "Mathematical Modeling and Digital Simulation for Engineers and Scientists," John Wiley & Sons, 1977.

[14] Michael, J.D., "Software Specifications for Docking Simulations of the Orbital Maneuvering Vehicle (OMV)," NASAN.MSFC System Dynamics Lab, Memo ED15-83-64, 1984.

[15] Carnahan, B., Luther, H., and Wilkes, J., "Applied Numerical Methods," John Wiley & Sons, 1969.

[16] "GRAPHICS MASTER Reference Manual," Tecmar Corp., 1983.

[17] Tietz, J.C. and Richardson, T.E., "Development of an Autonomous Video Randezvous and Docking System," Martin Marrieta Aerospace Phase 2 Final Report, 1983, NASA Contract # NASA8-34679.

[18] Saenger, E.L. and Woltosz, W.S., "Design of a Terminal Pointer Hand Controller for Teleoperator Applications," URS/Matrix Company Final Report, NASA Contract # NAS8-28760, 1973.

[19] Hendley, A.C., "Quaternions for Control of Space Vehicles," Sperry Rand Corp., NASA Contract #NAS8-20055, 1973.

[20] "Three Axis Low-G Accelerometer Package." NASA TM-78211, 1978.

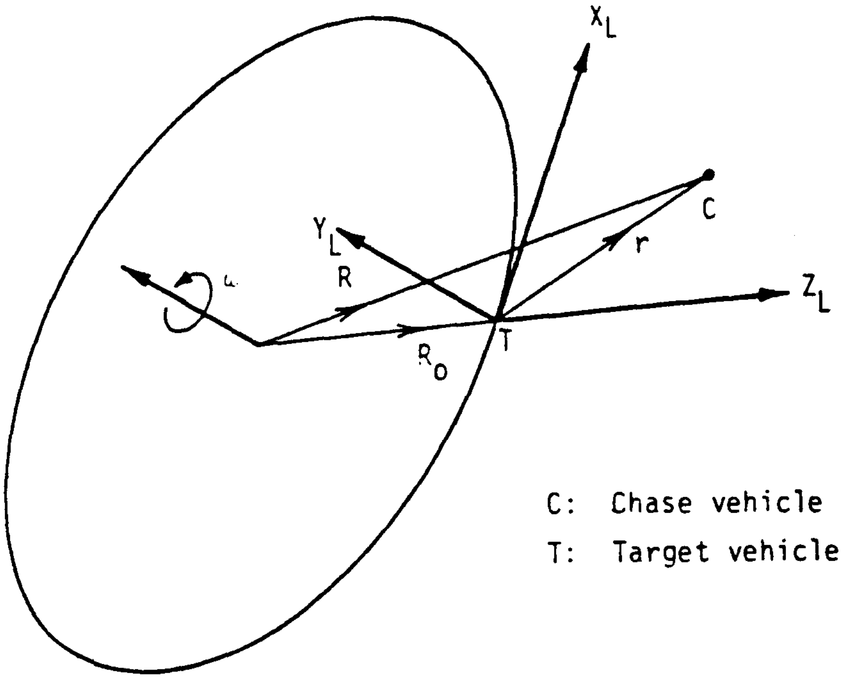
APPENDIX 1

OMV Translational Equations of Motion

Appendix 1.

OMV Translational Equations Of Motion

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



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

The equation of motion of the chase vehicle is easily deduced from Newton's second law, namely,

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

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

From the above figure, it is obvious that

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

it follows that

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

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

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

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

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

and

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

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

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

Furthermore, for a circular orbit,

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

therefore,

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

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

$$
\begin{aligned}
\mathbf{R} &= \mathbf{R}_o + \mathbf{r} \\
R^2 &= (\mathbf{R}_o + \mathbf{r}) \cdot (\mathbf{R}_o + \mathbf{r}) \\
&= R_o^2 + r^2 + 2\mathbf{R}_o.\mathbf{r} \\
&= R_o^2 + 2\mathbf{R}_o.\mathbf{r} \\
&= R_o^2 \{1 + 2(\mathbf{R}_o.\mathbf{r}) / R_o^2 \}
\end{aligned}
$$

so that

$$
\begin{aligned}
R^{-3} &= R_o^{-3}\{1 + 2(\mathbf{R}_o.\mathbf{r}) / R_o^2 \}^{-3/2} \\
&\cong R_o^{-3}\{1 - 3(\mathbf{R}_o.\mathbf{r}) / R_o^2 \}
\end{aligned}
$$

Thus,

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

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

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

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

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

which, after re-arranging, gives :

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

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

$$\mathbf{r} = [X,\ Y,\ Z]^T$$

$$\mathbf{R}_o = [0,\ 0,\ R_o]^T$$

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

$$\mathbf{A} = [A_x,\ A_y,\ A_z]^T$$

and it can easily be shown that :

$$2\mathbf{w} \times \dot{\mathbf{r}} = [\ 2w\dot{Z},\ \ 0,\ -2w\dot{X}\,]^T$$

$$\mathbf{w} \times (\mathbf{w} \times \mathbf{r}) = [-w^2X,\ \ 0,\ -w^2Z]^T$$

$$3\mathbf{w}(\mathbf{R}_o.\mathbf{r}/R_o^2)\mathbf{R}_o = [\ \ 0,\ \ 0,\ 3w^2Z]^T \quad \text{and}$$

$$w^2\mathbf{r} = [\ w^2X,\ w^2Y,\ \ w^2Z]^T$$

and substituting into equation (7) yields

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

or

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

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

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

```
C        system
C
C
C-----------------------------------------------------------------------
C
C
         REAL * 8    H, X, P, C, S
         REAL        XFORM(3,3), SDFORM(3,3), VO(3,10), V(3,10)
         REAL        ROT(3,3), FLOOR(3,3), V1(3,10)
         REAL        CC, DD, LL, RR, WW, TT
         INTEGER     FLAG, N, CLR, EF,EEF, PRTFG
C
         COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
         COMMON /MF/ XFORM, SDFORM, VO, V1
         COMMON /ME/ EF, EEF, PRTFG
C
C
         N  = 10
         AA = 1.0
C
C        ***  define mock-up module shape at origin
C
         DO 100 K = 1, N
1           V(3, K) = 1.0
1   100  CONTINUE
C                                        << point  A >>
         V(1,1)  =  TT
         V(2,1)  = -DD
C                                        << point  B >>
         V(1,2)  = -TT
         V(2,2)  = -DD
C                                        << point  C >>
         V(1,3)  = -TT
         V(2,3)  =  DD
C                                        << point  D >>
         V(1,4)  =  TT
         V(2,4)  =  DD
C
C        ***  rotate it by P radians
C
         CALL  SINCOS (P, S, C)
         CALL  NOTHNG (ROT, 3)
         ROT(1,1) = C
         ROT(1,2) = -S
         ROT(2,1) =  S
         ROT(2,2) =  C
         CALL  XMUL (ROT, V, 4)
C
C        ***  calculate translation
```

```
C
      PX = CC + LL * C
      PY = H +      LL * S
C
C     ***  move the rotated module out there
C
      CALL  NOTHNG (ROT, 3)
      ROT(1,3) = PX
      ROT(2,3) = PY
      CALL  XMUL (ROT, V, 4)
C
C     ***  now calculate the shape of the base
C
C     XX = X + CC
C                                 << point  E >>
      V(1,5) = CC
      V(2,5) = H
C                                 << point  F >>
      V(1,6) = CC
      V(2,6) = AA
C                                 << point  G >>
      V(1,7) = CC
      V(2,7) = 0.
C                                 << point  H >>
      V(1,8) = -RR
      V(2,8)  = 0.
C                                 << point  I >>
      V(1,9) = -RR
      V(2,9) = AA
C
      V(1,10) = PX
      V(2,10) = PY
C
C     ***  Transform to floor coordinates
C
      CALL  NOTHING (FLOOR, 3)
      FLOOR(1,3) = X
      CALL  XMUL (FLOOR, V, N)
C
C     ***  transform to screen coordinates
C
      CALL  XMUL (SDFORM, V, N)
C
C     ***  erase old picture
C
      CALL  DRWFLR (VO)
      IF ((EF .EQ. 0) .AND. (EEF .NE. 0)) THEN
         CLR = 0
         CALL  SDRAW (V1, N, CLR)
```

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

Name    Type          Offset P Class

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


    156 $PAGE
```

```
.ine# 1     7                              Microsoft FORTRAN77 V3.13 8/05/83
C
C
      SUBROUTINE  SDRAW (V, N, CLR)
C
C
C-----------------------------------------------------------------------
C
C
      This procedure draws the side view of TOM_B
C
C
C-----------------------------------------------------------------------
C
C
      REAL        V(3,10)
      INTEGER     N, CLR, X1, X2, Y1, Y2
C
      ***  draw mobile base
C
      CALL  RCT (V, 5, CLR)
C
      ***  draw linkage
C
      X1  =  V(1,6)
      Y1  =  V(2,6)
      X2  =  V(1,5)
      Y2  =  V(2,5)
      CALL  LINE (X1, Y1, X2, Y2, CLR)
      X1  =  V(1,10)
      Y1  =  V(2,10)
      CALL  LINE (X2, Y2, X1, Y1, CLR)
C
      ***  draw mock-up module
C
      CALL  RCT (V, 0, CLR)
      CALL  PURGE
      CALL  GRFRDY
C
      CALL  HOME
C
      RETURN
      END

ame    Type          Offset P Class

.R     INTEGER*4          8 *
       INTEGER*4          4 *
       REAL               0 *
1      INTEGER*4        238
```

```
D Line# 1      7
X2       INTEGER*4        246
Y1       INTEGER*4        242
Y2       INTEGER*4        250

    199 SPACE
```

```
Line# 1      7                                Microsoft FORTRAN77 V3.13 8/05/83
  200 C
  201 C
  202             SUBROUTINE  RCT  (V, OFF, CLR)
  203 C
  204 C
  205 C-----------------------------------------------------------------------
  206 C
  207 C           This procedure draws a rectangle
  208 C
  209 C
  210 C-----------------------------------------------------------------------
  211 C
  212 C
  213             REAL         V(3,10)
  214             INTEGER      OFF, CLR, X(10), Y(10)
  215 C
  216             DO 100 K = 1, 4
  217                J = K + OFF
  218                X(K) = V(1,J)
  219                Y(K) = V(2,J)
  220 100         CONTINUE
  221             CALL  POLYGN(4, X, Y, CLR)
  222             RETURN
  223             END

me    Type           Offset P Class

R     INTEGER*4          8 *
      INTEGER*4        338
      INTEGER*4        334
F     INTEGER*4          4 *
      REAL               0 *
      INTEGER*4        254
      INTEGER*4        294

  224 $PAGE
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
      SUBROUTINE  PLOT (CMD)
C
C
C-------------------------------------------------------------------
C
C
C         This is the plot part of the graphical package, and can
C         be directly callable from OMV or SVX. The value of FLAG
C         obtained from the disk file named SIZE.DAT dictates one
C         of top or side view to be displayed.
C
C
C-------------------------------------------------------------------
C
C
          INTEGER     CMD(7), FLAG
          REAL * 8    X, Y, T, UL, UA, H
          REAL        XFORM(3,3),SDFORM(3,3),CC,LL,DD,RR,WW,TT
          REAL        V0(3,10), V1(3,10)
C
          COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
          COMMON /MF/ XFORM, SDFORM, V0, V1
C
          UL = 10000.0
          UA = UL
C
          IF (FLAG .EQ. 0) THEN
             T  = CMD(1) / UA
             X  = CMD(2) / UL
             Y  = CMD(3) / UL
             CALL  TOPVEW (X, Y, T)
          ELSE
             H = CMD(4) / UL
             X = CMD(2) / UL
             T = CMD(5) / UA
             CALL  SIDEVEW (H, X, T)
          END IF
C
          RETURN
          END
```

```
ame    Type        Offset P Class

C      REAL             4   /MG    /
MD     INTEGER*4        0 *
D      REAL             8   /MG    /
LAG    INTEGER*4        0   /MG    /
I      REAL*8         382
L      REAL            12   /MG    /
```

```
ine# 1      7
    REAL             16    /MG     /
ORM 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
```

```
C
C
      SUBROUTINE  TOPVEW (PX, PY, THETA)
C
C-----------------------------------------------------------------------
C
C
C     This procedure constructs the top view of TOM_B. No
C     correction to perspective distortion is implemented
C
C
C-----------------------------------------------------------------------
C
      REAL * 8    PX, PY, THETA, S, C
      REAL        V(3,10), VO(3,10), SDFORM(3,3)
      REAL        ROT(3,3), FLOOR(3,3), XFORM(3,3)
      REAL        CC, DD, LL, RR, WW, TT, V1(3,10)
      INTEGER     FLAG, N, CLR, EF, EEF, PRTFG
C
      COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
      COMMON /MF/ XFORM, SDFORM, VO, V1
      COMMON /ME/ EF, EEF, PRTFG
C
      N = 10
C
C     ***  get TOM_B shape at the origin
C
      CALL  ORGPOS (V, N)
C
C     ***  rotate by THETA if needed
C
C     IF  (THETA .NE. 0.0) THEN
C         ***  construct rotation matrix
          CALL  NOTHNG (ROT, 3)
          CALL  SINCOS (THETA, S, C)
          ROT(1,1)  =  C
          ROT(1,2)  = -S
          ROT(2,1)  =  S
          ROT(2,2)  =  C
C         ***  rotate it
          CALL  XMUL (ROT, V, N)
C     END IF
C
C     ***  transform to floor coordinates
C
      CALL  NOTHNG (FLOOR, 3)
      FLOOR(1,3) = PX
      FLOOR(2,3) = PY
      CALL  XMUL (FLOOR, V, N)
```

```
ine# 1      7
 315 C
 316 C         ***  transform to screen coordinates
 317 C
 318           CALL  XMUL (XFORM, V, N)
 319 C
 320 C         ***  get ready to draw, but first erase old picture
 321 C
 322           CALL  DRWFLR (V1)
 323           IF ((EF .EQ. 0) .AND. (EEF .NE. 0)) THEN
 324               CLR = 0
 325               CALL  DRAW (VO, N, CLR)
 326           END IF
 327 C
 328           CLR = 1
 329           CALL  DRAW (V, N, CLR)
 330           CALL  MOVE (V, VO, N)
 331           EEF = 1
 332 C
 333           RETURN
 334           END

me    Type         Offset P Class

      REAL*8          594
      REAL              4   /MG     /
 R    INTEGER*4       602
      REAL              8   /MG     /
EF    INTEGER*4         4   /ME     /
      INTEGER*4         0   /ME     /
AG    INTEGER*4         0   /MG     /
LOOR  REAL            546
      REAL             12   /MG     /
      INTEGER*4       582
RTFG  INTEGER*4         8   /ME     /
X     REAL*8            0 *
      REAL*8            4 *
)T    REAL            510
R     REAL             16   /MG     /
      REAL*8          586
)FORM REAL             36   /MF     /
HETA  REAL*8            8 *
T     REAL             24   /MG     /
      REAL            390
O     REAL             72   /MF     /
1     REAL            192   /MF     /
V     REAL             20   /MG     /
FORM  REAL              0   /MF     /
  335 $PAGE
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
D Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
    336 C
    337 C
    338           SUBROUTINE  MOVE (V, VO, N)
    339 C
    340 C
    341 C-----------------------------------------------------------------
    342 C
    343 C
    344 C         This procedure saves the shape vector V
    345 C
    346 C
    347 C-----------------------------------------------------------------
    348 C
    349 C
    350           REAL        V(3,10), VO(3,10)
    351 C
    352           DO 100 K = 1, N
1   353              DO 100 J = 1, 3
2   354                 VO(J,K) = V(J,K)
2   355 100       CONTINUE
    356 C
    357           RETURN
    358           END

Name    Type         Offset P Class

J       INTEGER*4      614
K       INTEGER*4      606
N       INTEGER*4        8 *
V       REAL             0 *
VO      REAL             4 *

   359 $PAGE
```

```
C
C
          SUBROUTINE  NOTHNG (A, N)
C
C
C---------------------------------------------------------------------
C
C
          This procedure initializes an N x N matrix A to a
C         unit matrix
C
C
C---------------------------------------------------------------------
C
C
          REAL          A(N,N)
C
          DO 100 K = 1, N
             DO 200 J = 1, N
                A(K,J) = 0.0
200          CONTINUE
             A(K,K) = 1.0
100       CONTINUE
C
          RETURN
          END

me   Type          Offset P Class

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


$PAGE
```

```
D Line# 1      7                                Microsoft FORTRAN77 V3.13 8/05/83
    387 C
    388 C
    389          SUBROUTINE  XMUL (R, V, N)
    390 C
    391 C
    392 C------------------------------------------------------------------
    393 C
    394 C
    395 C        This procedure uses a transformation matrix R and
    396 C        transforms the shape vector V having N columns
    397 C
    398 C
    399 C------------------------------------------------------------------
    400 C
    401 C
    402          REAL        R(3,3), V(3,10), T(3), S
    403          INTEGER     ROW, COL
    404 C
    405          DO 100 COL = 1, N
1   406             DO 200 ROW = 1, 3
2   407                S = 0.0
2   408                DO 300 J = 1, 3
3   409                   S = S + R(ROW,J) * V(J, COL)
3   410 300            CONTINUE
2   411                T(ROW) = S
2   412 200         CONTINUE
1   413             DO 400 L = 1, 3
2   414                V(L,COL) = T(L)
2   415 400         CONTINUE
1   416 100      CONTINUE
    417 C
    418          RETURN
    419          END

Name    Type          Offset P Class

COL     INTEGER*4       646
J       INTEGER*4       662
        INTEGER*4       666
        INTEGER*4         8 *
R       REAL              0 *
ROW     INTEGER*4       654
        REAL            658
        REAL            634
        REAL              4 *


  420 $PAGE
```

ORIGINAL PAGE IS
OF POOR QUALITY

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

```
D Line# 1      7
    470             V(2, 6) =   0
    471 C                                   Corner  << F >>
    472             V(1, 7) =  CL + TT
    473             V(2, 7) = -DD
    474 C                                   Corner  << G >>
    475             V(1, 8) =  CL - TT
    476             V(2, 8) = -DD
    477 C                                   Corner  << H >>
    478             V(1, 9) =  CL - TT
    479             V(2, 9) =  DD
    480 C                                   Corner  << I >>
    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
CORNR   INTEGER*4        678
DD      REAL               8   /MG     /
EEF     INTEGER*4          4   /ME     /
EF      INTEGER*4          0   /ME     /
FLAG    INTEGER*4          0   /MG     /
K       INTEGER*4        694
LL      REAL              12   /MG     /
N       INTEGER*4          4 *
PRTFG   INTEGER*4          8   /ME     /
RR      REAL              16   /MG     /
SDFORM  REAL              36   /MF     /
TT      REAL              24   /MG     /
V       REAL               0 *
VO      REAL              72   /MF     /
V1      REAL             192   /MF     /
W       REAL             670
WW      REAL              20   /MG     /
XFORM   REAL               0   /MF     /

   486 $PAGE
```

```
C
C
      SUBROUTINE  INITPL
C
C
C------------------------------------------------------------------
C
C
C     This procedure initializes the system and calculates
C     all the necessary transformation matrices based on
C     the data obtained from SIZE.DAT
C
C
C------------------------------------------------------------------
C
C
      REAL     VO(3,10),XFORM(3,3),SDFORM(3,3), W(2)
      REAL     CC, DD, LL, RR, WW, TT, V1(3,10)
C     REAL      CORNR(2,2), W(2)
      INTEGER  FLAG, EF, CORNR(2,2), EEF, CORNS(2,2), PRTFG
C
      COMMON /MG/ FLAG, CC, DD, LL, RR, WW, TT
      COMMON /MF/ XFORM, SDFORM, VO, V1
      COMMON /ME/ EF, EEF, PRTFG
C
      EEF = 0
      OPEN (7, FILE = 'SIZE.DAT')
      READ (7, 10) CC, DD, LL, RR, WW, TT
      DO 200 K = 1, 2
         READ (7, 20) (CORNR(K,J), J=1, 2)
200   CONTINUE
      W(1) = 12.2
      W(2) = 24.4
      CALL  CORDX (CORNR, XFORM, W)
C
      DO 300 K = 1, 2
         READ (7, 20) (CORNS(K,J), J=1,2)
300   CONTINUE
      W(1) = 12.2
      W(2) = 6.096
      CALL  CORDX (CORNS, SDFORM, W)
C
      READ (7,20) EF
      READ (7, 20) FLAG
      READ (7, 20) PRTFG
      CLOSE (7)
      FLG  = 1
C
C     ***  calculate floor shape
```

```
C
      JW = 30
      JL = 44
C
      IF (FLAG .EQ. 0) THEN
          J1       = CORNR(1,1)
          L1       = CORNR(1,2)
          J2       = CORNR(2,1)
          L2       = CORNR(2,2)
          JJ       = (L2 - L1 + 1) / 2
          V1(1,1) = J1
          V1(2,1) = L1
          V1(1,2) = J2
          V1(2,2) = L1
          V1(1,3) = J2
          V1(2,3) = L2
          V1(1,4) = J1
          V1(2,4) = L2
          V1(1,5) = J1
          V1(2,5) = L2 + JW - JJ
          V1(1,6) = J1 - JL
          V1(2,6) = L2 + JW - JJ
          V1(1,7) = J1 - JL
          V1(2,7) = L2 - JL - JJ
          V1(1,8) = J1
          V1(2,8) = L2 - JL - JJ
          V1(1,9) = -1000.0
          V1(2,9) = -1000.0
      ELSE
          J1       = CORNS(1,1)
          L1       = CORNS(1,2)
          J2       = CORNS(2,1)
          L2       = CORNS(2,2)
          VO(1,1) = J1 - JL
          VO(2,1) = L2 + 1
          VO(1,2) = J2 + JL
          VO(2,2) = L2 + 1
          VO(1,3) = -1000.0
          VO(2,3) = -1000.0
      END IF
C
      CALL  GRAFICS
      RETURN
10    FORMAT (F15.8)
20    FORMAT (I3)
      END

ame   Type        Offset P Class
```

```
Line# 1      7                                  Microsoft FORTRAN77 V3.13 8/05/83
CC        REAL           4    /MG     /
 ORNR     INTEGER*4    714
 ORNS     INTEGER*4    730
DD        REAL           8    /MG     /
EF        INTEGER*4      4    /ME     /
 F        INTEGER*4      0    /ME     /
FLAG      INTEGER*4      0    /MG     /
FLG       REAL         758
          INTEGER*4    750
 1        INTEGER*4    770
J2        INTEGER*4    778
 J        INTEGER*4    786
 L        INTEGER*4    766
JW        INTEGER*4    762
          INTEGER*4    746
 1        INTEGER*4    774
 2        INTEGER*4    782
 L        REAL          12    /MG     /
 RTFG     INTEGER*4      8    /ME     /
 R        REAL          16    /MG     /
SDFORM    REAL          36    /MF     /
 T        REAL          24    /MG     /
 O        REAL          72    /MF     /
 1        REAL         192    /MF     /
          REAL         706
 W        REAL          20    /MG     /
 FORM     REAL           0    /MF     /

  582 $PAGE
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
ine# 1     7
 583 C
 584 C
 585           SUBROUTINE  DRWFLR (V)
 586 C
 587 C
 588 C-----------------------------------------------------------------
 589 C
 590 C         This subroutine draws the floor portion of graphics
 591 C
 592 C
 593 C-----------------------------------------------------------------
 594 C
 595 C
 596           REAL          V(3,10)
 597           INTEGER       CT, X(10), Y(10)
 598 C
 599           CT = 1
 600 C
 601 C         REPEAT
 602 100          K = CT
 603               X(K) = V(1,K)
 604               Y(K) = V(2,K)
 605               CT   = CT + 1
 606               IF (V(1,CT) .GE. -100.0) GO TO 100
 607 C         UNTIL  V(1,CT)  <   -100.0
 608 C
 609           CALL  POLYGN (K, X, Y, 1)
 610           RETURN
 611           END

ame   Type          Offset P Class

      INTEGER*4       884
      INTEGER*4       888
      REAL              0 *
      INTEGER*4       804
      INTEGER*4       844


 612 $PAGE
```

```
 Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
   613 C
   614 C
   615          SUBROUTINE  DRAW (V, N, CLR)
   616 C
   617 C
   618 C-------------------------------------------------------------------
   619 C
   620 C
   621 C        This procedure actually draws the top view of TOM_B.
   622 C
   623 C        This procedure must be modified if different hardware
   624 C        is used for the graphics display
   625 C
   626 C
   627 C-------------------------------------------------------------------
   628 C
   629 C
   630          REAL        V(3, 10)
   631          INTEGER     X1, X2, Y1, Y2
   632          INTEGER     CLR
   633 C
   634 C        ***  draw mobile base
   635 C
   636          CALL  RCT (V, 1, CLR)
   637 C
   638 C        ***  draw connecting line
   639 C
   640          X1  = V(1,1)
   641          Y1  = V(2,1)
   642          X2  = V(1,6)
   643          Y2  = V(2,6)
   644          CALL  LINE (X1, Y1, X2, Y2, CLR)
   645 C
   646 C        ***  draw mocked-up
   647 C
   648          CALL  RCT (V, 6, CLR)
   649 C
   650          CALL  PURGE
   651          CALL  GRFRDY
   652          CALL  HOME
   653 C
   654          RETURN
   655          END

me    Type           Offset P Class

R     INTEGER*4          8 *
      INTEGER*4          4 *
      REAL               0 *
```

```
      INTEGER*4        892
      INTEGER*4        900
      INTEGER*4        896
      INTEGER*4        904

  656 $PAGE
```

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

me    Type          Offset P Class

      INTEGER*4          0 *
      REAL               4 *
      REAL               8 *


 $PAGE
```

```
ine# 1      7                                      Microsoft FORTRAN77 V3.13 8/05/83
 692 C
 693 C       This is a graphics package for the TECMAR GRAPHICS MASTER board
 694 C       written under Microsoft's FORTRAN 77.  To use this package, one
 695 C       must include this package in the source file. A graphics master
 696 C       must already be installed, or the software will hang.
 697 C
 698 C
 699         SUBROUTINE       PURGE
 700 C
 701 C       This procedure purges the graphics buffer and forces the board
 702 C            to complete the drawing by closing the graphics channel.
 703 C
 704         INTEGER          GRF
 705         CHARACTER        ESC
 706         COMMON    /GMBD/ GRF, ESC
 707         CLOSE (GRF)
 708         RETURN
 709         END

e    Type          Offset P Class

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


 710 $PAGE
```

```
D Line# 1      7                                  Microsoft FORTRAN77 V3.13 8/05/83
   711 C
   712 C
   713          SUBROUTINE       GRFRDY
   714 C
   715 C        This procedure opens the graphics channel and sets it ready for
   716 C             communication
   717 C
   718 C
   719          INTEGER          GRF
   720          CHARACTER        ESC
   721          COMMON    /GMBD/ GRF, ESC
   722          OPEN (GRF, FILE = 'gm')
   723          RETURN
   724          END

Name    Type         Offset P Class

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


   725 $PAGE
```

```
ine# 1      7
  726 C
  727 C
  728         SUBROUTINE      SETFB  (FG, BG)
  729 C
  730 C
  731 C       This procedure sets the foreground color to FG and the backgroun
  732 C            color to BG. Both arguments must be of INTEGER type.
  733 C
  734 C
  735         INTEGER          GRF, FG, BG
  736         CHARACTER        ESC
  737         COMMON    /GMBD/ GRF, ESC
  738         WRITE (GRF, 10)  ESC, FG, BG
  739         RETURN
  740 10      FORMAT (' ', A1, '[!', I2, ';', I2, 'c'\)
  741         END

ame   Type         Offset P Class

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

  742 $PAGE
```

```
D Line# 1      7
   743 C
   744 C
   745         SUBROUTINE      GRAFICS
   746 C
   747 C
   748 C       This procedure enters the GM graphics mode with a four-line text
   749 C            window at the bottom
   750 C
   751 C
   752         INTEGER             GRF
   753         CHARACTER           ESC
   754         COMMON     /GMBD/ GRF, ESC
   755 C
   756         GRF = 9
   757         ESC = CHAR(27)
   758         CALL       GRFRDY
   759         WRITE (GRF, 10)  ESC
   760         WRITE (GRF, 20)  ESC
   761 C       WRITE (GRF, 30)  ESC
   762         CALL       SETFB  (1, 0)
   763         CALL  HOME
   764         RETURN
   765 C
   766 10      FORMAT (' ', A1, '[!0m'\)
   767 20      FORMAT (' ', A1, '[!640;352;2g'\)
   768 30      FORMAT (' ', A1, '[21;24r'\)
   769         END

Name    Type         Offset P Class

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


   770 $PAGE
```

```
inc# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
 771 C
 772 C
 773          SUBROUTINE  QUITGM
 774 C
 775 C
 776 C        This procedure gets one out of graphics mode and returns
 777 C        to text mode
 778 C
 779 C
 780          CHARACTER     CH, ESC
 781          INTEGER       GRF
 782          COMMON /GMBD/ GRF, ESC
 783 C
 784          CALL  HOME
 785          WRITE (GRF, 30)
 786          CALL  PURGE
 787          READ  (*, 10) CH
 788          CALL  GRFRDY
 789          CALL  TEXT
 790          RETURN
 791 10       FORMAT (A1)
 792 30       FORMAT ('Press <CR> to continue ... '\)
 793          END

e   Type         Offset P Class

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

 794 $PAGE
```

```
D Line# 1      7                                    Microsoft FORTRAN77 V3.13 8/05/83
    795 C
    796 C
    797          SUBROUTINE      TEXT
    798 C
    799 C
    800 C        This procedure returns the system to text mode
    801 C
    802 C
    803          INTEGER          GRF
    804          CHARACTER        ESC
    805          COMMON    /GMBD/ GRF, ESC
    806          WRITE (GRF, 10) ESC
    807          RETURN
    808 C
    809 10       FORMAT (' ', A1, '[!80;25;1a'\)
    810          END

Name    Type        Offset P Class

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

    811 $PAGE
```

```
ine# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
 812 C
 813 C
 814        SUBROUTINE       LINE  (X1, Y1, X2, Y2, COLOR)
 815 C
 816 C
 817 C      This procedure draws a line from (X1,Y1) to (X2,Y2) in COLOR
 818 C
 819 C
 820        INTEGER          GRF, X1, Y1, X2, Y2, COLOR
 821        CHARACTER        ESC
 822        COMMON    /GMBD/ GRF, ESC
 823        WRITE (GRF, 10)  ESC, X1, Y1, X2, Y2, COLOR
 824 10     FORMAT (' ', A1, '[!', 4(I3,';'), I3, '1'\)
 825        END

ie    Type         Offset P Class

LOR   INTEGER*4        16 *
C     CHAR*1            4   /GMBD  /
F     INTEGER*4         0   /GMBD  /
      INTEGER*4         0 *
      INTEGER*4         8 *
      INTEGER*4         4 *
      INTEGER*4        12 *


 826 SPAGE
```

```
   827 C
   828 C
   829          SUBROUTINE       HIDELN (X1, Y1, X2, Y2, COLOR)
   830 C
   831 C
   832 C        This procedure draws the line (X1,Y1) - (X2,Y2) but aborts drawi
   833 C             before reaching target if a dot in a color other that BG is
   834 C             encountered
   835 C
   836 C
   837          INTEGER          GRF, X1, Y1, X2, Y2, COLOR
   838          CHARACTER        ESC
   839          COMMON    /GMBD/ GRF, ESC
   840          WRITE (GRF, 10)  ESC, X1, Y1, X2, Y2, COLOR
   841          RETURN
   842 10       FORMAT (' ', A1, '[!', 4(I3, ';'), I3, 'S'\)
   843          END

ame   Type          Offset P Class

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

   844 SPAGE
```

```
ine# 1      7
     C
     C
           SUBROUTINE        POLYGN (N, X, Y, COLOR)
     C
     C
     C     This procedure draws a closed polygon whose N vertices are store
     C         in the arrays X and Y. The color to be used is COLOR
     C
     C
           INTEGER           GRF, X(N), Y(N), COLOR
           CHARACTER         ESC
           COMMON    /GMBD/  GRF, ESC
           WRITE (GRF, 10) ESC
           DO 100 K = 1, N
              WRITE (GRF, 20) X(K), Y(K)
      100  CONTINUE
           WRITE (GRF, 30) COLOR
           RETURN
      10   FORMAT (' ', A1, '[!'\)
      20   FORMAT (      2(I3, ';')\)
      30   FORMAT (      I3, 'p'\)
           END

me    Type        Offset P Class

LOR   INTEGER*4       12 *
C     CHAR*1           4   /GMBD   /
F     INTEGER*4        0   /GMBD   /
      INTEGER*4     1160
      INTEGER*4        0 *
      INTEGER*4        4 *
      INTEGER*4        8 *

      $PAGE
```

```
D Line# 1      7
    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        COMMON /GMBD/ GRF, ESC
    880 C
    881        WRITE (GRF, 10) ESC
    882        RETURN
    883 10     FORMAT (' ', A1, '[ 1;1 f'\)
    884        END

Name    Type          Offset P Class

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

    885 $PAGE
```

```
<   O M V   P L O T   >>>                                                    Page  34
                                                                             07-14-84
                                                                             14:03:20
Line# 1      7                                  Microsoft FORTRAN77 V3.13 8/05/83

 e    Type              Size    Class

)RDX                            SUBROUTINE
  W                             SUBROUTINE
  FLR                           SUBROUTINE
IBD                        5    COMMON
'AFIC                           SUBROUTINE
  RDY                           SUBROUTINE
.DELN                           SUBROUTINE
)ME                             SUBROUTINE
  TPL                           SUBROUTINE
  E                             SUBROUTINE
                          12    COMMON
                         312    COMMON
                          28    COMMON
VE                              SUBROUTINE
THNG                            SUBROUTINE
 POS                            SUBROUTINE
.JT                             SUBROUTINE
)LYGN                           SUBROUTINE
RGE                             SUBROUTINE
 TGM                            SUBROUTINE
 T                              SUBROUTINE
RAW                             SUBROUTINE
 FB                             SUBROUTINE
DEVE                            SUBROUTINE
NCOS                            SUBROUTINE
XT                              SUBROUTINE
PVEW                            SUBROUTINE
IUL                             SUBROUTINE

'ass One      No Errors Detected
           885 Source Lines
```

ORIGINAL PAGE IS
OF POOR QUALITY

APPENDIX 3

OMV Data Files Used During Development

```
                           File :  INITCON.DAT

            This file contains all the needed initial conditions

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


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

File : MDLPRM.DAT

This file contains all the model parameters needed by OMV

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

File : SVXINT.DAT

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

```
-------------------------------------------------------------------------------

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

File : HNDSGL.DAT

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

```
-----------------------------------------------------------------------------


100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
100000000000
```

File : SIZE.DAT

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

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

APPENDIX 4

OMV Mathematical Model (OMM) Source Listing

```
$LINESIZE:79
$PAGESIZE: 56
$TITLE: '<<<   O M V   >>>'
C
C                        OMV SIMULATION MODEL
C
C
C                                 by
C
C
C                            Dr. W. Teoh
C
C                        U A H    Huntsville
C                                1984
C
C-----------------------------------------------------------------------
C
C           This is a simplified version of a mathematical simulation
C       model of the OMV. In this model, the following simplfications
C       and assumptions are made :
C
C       1. The hand controllers provide  signals that are interpreted
C          as a force at the center of mass and/or a torque about the
C          center of mass to provide a rotation of constant angular
C          velocity.
C       2. The target vehicle is in a circular orbit; the altitude of
C          this orbit is inputted from the MDLPRM.DAT file.
C       3. Orbital mechanics is implemented, but smaller perturbation
C          effects are totally ignored.
C       4. Detailed placement of thrusters is not considered  (Please
C          see assumption 1. above)
C       5. Roll, pitch and yaw denote the  instantaneous  orientation
C          of the OMV.
C
C       A  14  component state vector is generated by this model, and
C       this state vector serves as input to the SVX module.
C
C
C-----------------------------------------------------------------------
C
        REAL * 8     X(3), V(3), E(3), A(3), W(3), Q(4)
        REAL * 8     POS(3), VEL(3), EUL(3), OMEGA
        REAL * 8     III(3), S(14), MASS, CYCLE
        INTEGER      CMD(7), IN, FLAG, MODE, STEP
        INTEGER * 4  TIME
C
        COMMON  /MC/ III, MASS, CYCLE, MODE, STEP
        COMMON  /PC/ POS, VEL, EUL,  OMEGA
C
```

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

```
Line# 1     7                                  Microsoft FORTRAN77 V3.13 8/05/83
   99 900      CONTINUE
  100          CLOSE (IN)
  101 C
  102 C        *** ***  This is also a call to the graphics package
  103 C
  104          CALL  QUITGM
  105 C
  106 C        ***  Grand exit, stage left
  107 C
  108          STOP
  109          END

.me   Type         Offset P Class

      REAL*8          242
)     INTEGER*4       266
CLE   REAL*8           32    /MC     /
      REAL*8           74
.     REAL*8           48    /PC     /
.AG   INTEGER*4       302
'I    REAL*8            0    /MC     /
      INTEGER*4       294
.SS   REAL*8           24    /MC     /
)DE   INTEGER*4        40    /MC     /
EGA   REAL*8           72    /PC     /
S     REAL*8            0    /PC     /
      REAL*8           98
      REAL*8          130
'EP   INTEGER*4        44    /MC     /
.ME   INTEGER*4       298
      REAL*8           26
:L    REAL*8           24    /PC     /
      REAL*8           50
      REAL*8            2


  110 $PAGE
```

```
C
C
      SUBROUTINE  OMVMDL (IN)
C
C-----------------------------------------------------------------------
C
C     This procedure obtains the necessary parameters of the OMV
C     by reading them from a disk file called  MDLPRM.DAT  after
C     getting the initial state of the  OMV  (from a file called
C     INITCON.DAT
C
C-----------------------------------------------------------------------
C
      REAL * 8       POS(3), VEL(3), EUL(3), OMEGA
      REAL * 8       ACC(3), III(3), WWB(3), INV(3)
      REAL * 8       MASS,   CYCLE,  ORBIT
      INTEGER        IN,     MODE, STEP
C
      COMMON  /DC/  ACC,     WWB
      COMMON  /MC/  III,     MASS, CYCLE, MODE, STEP
      COMMON  /PC/  POS,     VEL, EUL,  OMEGA
C
C     ***  get initial conditions of the OMV
C
      OPEN  (IN, FILE = 'INITCON.DAT')
      CALL  VECTOR (IN, POS, 3)
      CALL  VECTOR (IN, VEL, 3)
      CALL  VECTOR (IN, EUL, 3)
      CLOSE (IN)
C
C     ***  read acceleration, angular rates and
C     ***  principal moments of inertia in body frame
C
      OPEN  (IN, FILE = 'MDLPRM.DAT')
      CALL  VECTOR (IN, ACC, 3)
      CALL  VECTOR (IN, WWB, 3)
      CALL  VECTOR (IN, III, 3)
C
C     ***  read mass characteristics & other parameters
C
      READ  (IN, 10)  MASS
      READ  (IN, 10)  CYCLE
      READ  (IN, 20)  MODE
      READ  (IN, 30)  STEP
      READ  (IN, 10)  ORBIT
      CLOSE (IN)
C
C     ***  calculate orbital frequency
C
```

```
 Line# 1      7                                  Microsoft FORTRAN77 V3.13 8/05/83
   160          CALL  ANGFRE (ORBIT, OMEGA)
   161 C
   162 C
   163          RETURN
   164 10       FORMAT (F15.8)
   165 20       FORMAT (I1)
   166 30       FORMAT (I2)
   167          END

ame   Type        Offset P Class

 C    REAL*8           0   /DC     /
.CLE  REAL*8          32   /MC     /
UL    REAL*8          48   /PC     /
 I    REAL*8           0   /MC     /
      INTEGER*4        0 *
NV    REAL*8         306
SS    REAL*8          24   /MC     /
 DE   INTEGER*4       40   /MC     /
MEGA  REAL*8          72   /PC     /
RBIT  REAL*8         330
 S    REAL*8           0   /PC     /
.EP   INTEGER*4       44   /MC     /
EL    REAL*8          24   /PC     /
 B    REAL*8          24   /DC     /

  168 $PAGE
```

```
D Line# 1      7                                 Microsoft FORTRAN77 V3.13 8/05/83
    169 C
    170 C
    171         SUBROUTINE  ANGFRE(ORB, W)
    172 C
    173 C
    174 C-----------------------------------------------------------------------
    175 C
    176 C       This  procedure calculates the orbital angular frequency
    177 C       at a given altitude.  In this calculation,  the altitude
    178 C       must be given in kilo-meters. This is necessary in order
    179 C       for the calculations  to be carried out  without lossing
    180 C       precision. The angular frequency W is in rad/second
    181 C
    182 C-----------------------------------------------------------------------
    183 C
    184         REAL * 8  ORB
    185         REAL * 8  ALT, R3, W
    186 C
    187         ALT = ORB * 0.001
    188         R3  = (6.370 + ALT) ** 3
    189         W   = DSQRT (398.86 / R3) * 0.001
    190         RETURN
    191         END

Name    Type         Offset P Class

ALT     REAL*8          358
DSQRT                          INTRINSIC
ORB     REAL*8            0 *
R3      REAL*8          366
W       REAL*8            4 *

    192 $PAGE
```

```
Line# 1      7
  193 C
  194 C
  195          SUBROUTINE  VECTOR (M, A, N)
  196 C
  197 C
  198 C--------------------------------------------------------------------
  199 C
  200 C
  201 C        This procedure reads a vector A of N elements from input
  202 C        unit M
  203 C
  204 C--------------------------------------------------------------------
  205 C
  206          INTEGER     M, N
  207          REAL * 8    A(N)
  208 C
  209          DO 100  K = 1, N
  210              READ (M, 10) A(K)
  211 100      CONTINUE
  212          RETURN
  213 10       FORMAT (F15.8)
  214          END

ıe  Type         Offset P Class

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

  215 $PAGE
```

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

```
 e   Type         Offset P Class

     REAL*8            8 *
     REAL*8            0   /DC    /
AG   INTEGER*4         4 *
     INTEGER*4         0 *
     INTEGER*4       414
     INTEGER*4       390
     REAL*8           12 *
     REAL*8           24   /DC    /

  264 $PAGE
```

```
D Line# 1      7                                  Microsoft FORTRAN77 V3.13 8/05/83
    265 C
    266 C
    267           SUBROUTINE  FUDGE  (A, ACC, SL)
    268 C
    269 C---------------------------------------------------------------------
    270 C
    271 C         ***  Sets appropriate components based on SL
    272 C
    273 C---------------------------------------------------------------------
    274 C
    275           INTEGER     SL(6),  T, K, J
    276           REAL * 8    ACC(3), A(3), X
    277           DO 100 K = 1, 6, 2
1   278               J  =  (K + 1) / 2
1   279               X  =  0.0
1   280               T  =  SL(K) + SL(K+1)
1   281               IF (T .EQ. 1) THEN
1   282                   X  =  ACC(J)
1   283                   IF (SL(K) .EQ. 0) X = -X
1   284               END IF
1   285               A(J) = X
1   286 100       CONTINUE
    287           RETURN
    288           END

Name    Type         Offset P Class

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

    289 $PAGE
```

```
D Line# 1      7                                       Microsoft FORTRAN77 V3.13 8/05/83
    339           S(14)  =  MASS
    340 C
    341           RETURN
    342           END

ame     Type          Offset P Class

        REAL*8          642
        REAL*8          570
CYCLE   REAL*8           32    /MC     /
UL      REAL*8           48    /PC     /
 II     REAL*8            0    /MC     /
L       REAL*8          546
 B      REAL*8          498
 L      REAL*8          522
ASS     REAL*8           24    /MC     /
MODE    INTEGER*4        40    /MC     /
        INTEGER*4       714
 MEGA   REAL*8           72    /PC     /
POS     REAL*8            0    /PC     /
        REAL*8            0 *
 Q      REAL*8          466
        REAL*8            4 *
TEP     INTEGER*4        44    /MC     /
 EL     REAL*8           24    /PC     /
        REAL*8            8 *


   343 $PAGE
```

```
Line# 1     7                                     Microsoft FORTRAN77 V3.13 8/05/83
  344 C
  345 C
  346           SUBROUTINE  PUT (N, S, A, M)
  347 C
  348 C------------------------------------------------------------------------
  349 C
  350 C         ***  The procedure copies a vector A into a larger one S
  351 C              starting at the N-th element of S
  352 C
  353 C------------------------------------------------------------------------
  354 C
  355           REAL * 8    S(14)
  356           REAL * 8    A(M)
  357 C
  358           DO 100 K = 1, M
  359                N    = N + 1
  360                S(N)= A(K)
  361 100       CONTINUE
  362           RETURN
  363           END

ne   Type         Offset P Class

     REAL*8           8 *
     INTEGER*4      718
     INTEGER*4       12 *
     INTEGER*4        0 *
     REAL*8           4 *


  364 $PAGE
```

```
D Line# 1      7                            Microsoft FORTRAN77 V3.13 8/05/83
    365 C
    366 C
    367            SUBROUTINE  DOTPRD (A, B, C, N)
    368 C
    369 C-----------------------------------------------------------------------
    370 C
    371 C          ***  This procedure calculates a vector C from two other
    372 C               vectors A and B such that
    373 C                    C(I)  =  A(I) * B(I)
    374 C               for all i = 1 to N
    375 C
    376 C-----------------------------------------------------------------------
    377 C
    378            REAL * 8    A(N), B(N), C(N)
    379            DO 100 K = 1, N
1   380               C(K) = A(K) * B(K)
1   381 100        CONTINUE
    382            RETURN
    383            END

Name    Type        Offset P Class

A       REAL*8           0 *
B       REAL*8           4 *
C       REAL*8           8 *
K       INTEGER*4      726
N       INTEGER*4       12 *

    384 $PAGE
```

```
  385 C
  386 C
  387       SUBROUTINE  DETQ (E, Q)
  388 C
  389 C-------------------------------------------------------------
  390 C
  391 C     ***  calculates quaternions from the Euler angles
  392 C          using expression given by Zack.
  393 C
  394 C-------------------------------------------------------------
  395 C
  396       REAL * 8    E(3), Q(4)
  397       REAL * 8    C1, S1, C2, S2, C3, S3, THETA
  398 C
  399       THETA = E(1) / 2.0
  400       CALL SINCOS (THETA, S1, C1)
  401       THETA = E(2) / 2.0
  402       CALL SINCOS (THETA, S2, C2)
  403       THETA = E(3) / 2.0
  404       CALL SINCOS (THETA, S3, C3)
  405 C
  406       Q(1) = S1 * C3 * C2  +  C1 * S3 * S2
  407       Q(2) = S1 * S3 * C2  +  C1 * C3 * S2
  408       Q(3) = C1 * S3 * C2  -  S1 * C3 * S2
  409       Q(4) = C1 * C3 * C2  -  S1 * S3 * S2
  410 C
  411       RETURN
  412       END

ame    Type        Offset P Class

       REAL*8        750
2      REAL*8        766
3      REAL*8        782
       REAL*8          0 *
       REAL*8          4 *
1      REAL*8        742
       REAL*8        758
       REAL*8        774
HETA   REAL*8        734


  413 $PAGE
```

```
    414 C
    415 C
    416          SUBROUTINE SINCOS (THETA, S, C)
    417 C
    418 C
    419 C-----------------------------------------------------------------------
    420 C
    421 C        ***  this procedure returns the sine and cosine of an
    422 C             angle THETA.
    423 C
    424 C-----------------------------------------------------------------------
    425 C
    426          REAL * 8    THETA, S, C, A
    427 C
    428          C = DCOS(THETA)
    429          S = DSIN(THETA)
    430          RETURN
    431          END

Name    Type        Offset P Class

A       REAL*8       *****
C       REAL*8           8 *
DCOS                         INTRINSIC
DSIN                         INTRINSIC
S       REAL*8           4 *
THETA   REAL*8           0 *

    432 $PAGE
```

```
C
C
      SUBROUTINE  MOTION (X, V, E, A, W, Q)
C
C---------------------------------------------------------------------
C
C     ***  This procedure solves the equation of motion
C
C---------------------------------------------------------------------
C
      REAL * 8         POS(3), VEL(3), EUL(3), OMEGA
      REAL * 8         X(3), V(3), E(3), A(3), W(3), Q(4)
      REAL * 8         CIN(3,3), C(3,3), AA(3,10), B(3), QQ(4)
      REAL * 8         WW(3), PI, TWO
      REAL * 8         III(3), MASS, CYCLE
      INTEGER          MODE, STEP
C
      COMMON   /MC/    III, MASS, CYCLE, MODE, STEP
      COMMON   /PC/    POS, VEL, EUL, OMEGA
C
      H = CYCLE / FLOAT(STEP)
      N = STEP
      PI = 355.0 / 113.0
      TWO= PI * 2.0
C
C     ***  Divide 1 major cycle into N equal subintervals and
C     ***  determine the OMV state for each interval
C
      DO 100 KK = 1, N
C
C        ***   Update orientation
C
         DO 200 J = 1, 3
            WW(J) = W(J) * H
            E(J) = E(J) + WW(J)
            IF (E(J)  .GT. TWO) E(J) = E(J) - TWO
  200    CONTINUE
C
C     ***  Calculate quaternion for this rotation, and transform
C     ***  it to local vertical frame with respect to initial frame
C
         CALL  DETQ( WW, QQ)
         CALL  UPDQ (Q, QQ)
C
C        ***  from the direction cosine matrix, calculate the
C        ***  acceleration vector in LVF and store it in the
C        ***  acceleration matrix AA
C
         CALL  DCSINV (Q, CIN)
```

ORIGINAL PAGE IS
OF POOR QUALITY

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

Name     Type          Offset P Class

A        REAL*8            12 *
AA       REAL*8           990
B        REAL*8          1230
C        REAL*8           918
CIN      REAL*8           846
CYCLE    REAL*8            32   /MC     /
E        REAL*8             8 *
EUL      REAL*8            48   /PC     /
FLOAT                           INTRINSIC
H        REAL            1254
III      REAL*8             0   /MC     /
J        INTEGER*4       1286
KK       INTEGER*4       1278
MASS     REAL*8            24   /MC     /
MODE     INTEGER*4         40   /MC     /
N        INTEGER*4       1258
OMEGA    REAL*8            72   /PC     /
PI       REAL*8          1262
POS      REAL*8             0   /PC     /
Q        REAL*8            20 *
QQ       REAL*8           814
STEP     INTEGER*4         44   /MC     /
TWO      REAL*8          1270
V        REAL*8             4 *
VEL      REAL*8            24   /PC     /
W        REAL*8            16 *
WW       REAL*8           790
X        REAL*8             0 *


    493 $PAGE
```

```
Line# 1      7                                   Microsoft FORTRAN77 V3.13 8/05/83
  494 C
  495 C
  496         SUBROUTINE  MATCH (A, B, C, P, Q, R, N)
  497 C
  498 C
  499 C-----------------------------------------------------------------------
  500 C
  501 C       ***  This procedure makes an exact duplicate B of a
  502 C            vector A of N elements
  503 C
  504 C-----------------------------------------------------------------------
  505 C
  506         REAL * 8    A(N), B(N), C(N), P(N), Q(N), R(N)
  507         DO 100 K = 1, 3
  508             P(K) = A(K)
  509             Q(K) = B(K)
  510             R(K) = C(K)
  511 100     CONTINUE
  512         RETURN
  513         END

ame   Type         Offset P Class

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


  514 $PAGE
```

```
D Line# 1      7
    515 C
    516 C
    517          SUBROUTINE  STORE (AAA, AA, K)
    518 C
    519 C
    520 C-----------------------------------------------------------------
    521 C
    522 C            This procedure takes an  instantaneous  acceleration vector
    523 C        AAA and stores it in the acceleration matrix AA which is needed
    524 C        by the numerical integration process
    525 C
    526 C-----------------------------------------------------------------
    527 C
    528          REAL * 8    AA(3, 10)
    529          REAL * 8    AAA(3)
    530          DO 100 J = 1, 3
1   531              AA(J,K) = AAA(J)
1   532  100     CONTINUE
    533          RETURN
    534          END

Name    Type        Offset P Class

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

    535 $PAGE
```

```
C
C
         SUBROUTINE SOLVE(X,V,A,N,H,W)
C
C-------------------------------------------------------------------
C
C            This subroutine  produces the numerical solution to the
C        system of equations of motion using a 3 step Adam-Brashford
C        method.
C
C-------------------------------------------------------------------
C
         LOGICAL FLAG
         REAL*8  X(3), V(3), A(3,10), AA(3,13), U(6,13)
         REAL*8  WX2,  WXW,  WXWX3,   HD12,     F,       W
         COMMON /BLOCK/ AA,  U,  WX2, WXW, WXWX3,        HD12
         DATA FLAG /.TRUE./
C
C        ***  pack user supplied nonhomomgenous part of DE
C        ***  into the higher part of AA
C
         DO 10 I = 1,10
            DO 10 K = 1,3
               AA(K,I+3) = A(K,I)
10       CONTINUE
C
C        ***  if this is the first call to solve (FLAG = T), then
C        ***  it is necessary to initialize some parameters
C
         IF (FLAG) THEN
            CALL INNIT(X,V,W,H)
            FLAG = .FALSE.
         END IF
C
C        ***  use the Adams-Brashford 3-step method to advance the
C        ***  solution H time units.  Place the solution back into
C        ***  X and V.
C
         DO 100 I = 4,N+3
            DO 100 J  = 1,6
               U(J,I) = U(J,I-1) +
     +                  HD12*(23*F(J,I-1)-16*F(J,I-2)+5*F(J,I-3))
100      CONTINUE
         X(1) = U(1,N+3)
         V(1) = U(2,N+3)
         X(2) = U(3,N+3)
         V(2) = U(4,N+3)
         X(3) = U(5,N+3)
         V(3) = U(6,N+3)
```

```
    585 C
    586 C        ***  reset U and AA for the next call to SOLVE
    587 C
    588          DO 200 J = 1,6
1   589             DO 200 I = 1,3
2   590                U(J,I) = U(J,N+I)
2   591                IF (J .LE. 3) AA(I,J) = AA(I,N+J)
2   592  200     CONTINUE
    593          RETURN
    594          END

Name     Type         Offset P Class

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

    595 $PAGE
```

```
C
C
      SUBROUTINE INNIT(X,V,W,H)
C
C
C-----------------------------------------------
C
C         This procedure initializes all the necessary  parameters
C     before solving the system of ordinary differential equations.
C     This procedure is invoked only once.
C
C-----------------------------------------------
C
      REAL * 8   X(3), V(3), AA(3,13), U(6,13), WX2, WXW, WXWX3
      REAL * 8   CWT,  SWT,  T,        W,       HD12
      COMMON /BLOCK/ AA, U, WX2, WXW, WXWX3, HD12
      WXW    = W*W
      WXWX3 = 3*WXW
      WX2    = 2*W
      HD12   = DBLE(H)/12.0
C
      DO 100 K = 1,3
         U(2*K-1,3) = X(K)
         U(2*K  ,3) = V(K)
         DO 100 J = 1,6
            AA(J,K) = 0.0
C        CONTINUE
100   CONTINUE
C
      DO 300 I  = 1,2
         T      = H*(I-3)
         CWT    = DCOS(W*T)
         SWT    = DSIN(W*T)
         U(1,I) = X(1) + V(1)*(4*SWT-3*W*T)/W +
     +            6*X(3)*(SWT-W*T) + 2*V(3)*(CWT-1.0)/W
         U(2,I) = V(1)*(4*CWT-3.0) + 6*W*X(3)*(CWT-1.0) -
     +            2*V(3)*SWT
         U(3,I) = X(2)*CWT + V(2)*SWT/W
         U(4,I) = -X(2)*W*SWT + V(2)*CWT
         U(5,I) = 2*V(1)*(1.0-CWT)/W + X(3)*(4.0-3*CWT) +
     +            V(3)*SWT/W
         U(6,I) = 2*V(1)*SWT + 3*X(3)*W*SWT + V(3)*CWT
300   CONTINUE
      RETURN
      END
```

```
ame   Type       Offset P Class

A     REAL*8          0   /BLOCK /
```

```
D Line# 1       7                                     Microsoft FORTRAN77 V3.13 8/05/83
CWT     REAL*8           1362
)BLE                                INTRINSIC
DCOS                                INTRINSIC
DSIN                                INTRINSIC
I       REAL               12 *
ID12    REAL*8            960       /BLOCK /
I       INTEGER*4        1350
J       INTEGER*4        1346
(       INTEGER*4        1342
SWT     REAL*8           1370
T       REAL*8           1354
J       REAL*8            312       /BLOCK /
V       REAL*8              4 *
W       REAL*8              8 *
VX2     REAL*8            936       /BLOCK /
VXW     REAL*8            944       /BLOCK /
WXWX3   REAL*8            952       /BLOCK /
X       REAL*8              0 *

    641 $PAGE
```

```
D Line# 1       7
      C
      C
            FUNCTION F(J,I)
      C
      C
      C-----------------------------------------------------------------------
      C
      C
            REAL*8 AA(3,13),U(6,13),WX2,WXW,WXWX3,HD12,F
            COMMON /BLOCK/ AA,U,WX2,WXW,WXWX3,HD12
      C
            GO TO (10,20,30,40,50,60), J
       10   CONTINUE
               F = U(2,I)
               RETURN
       20   CONTINUE
               F = -WX2*U(6,I) + AA(1,I)
               RETURN
       30   CONTINUE
               F = U(4,I)
               RETURN
       40   CONTINUE
               F = -WXW*U(3,I) + AA(2,I)
               RETURN
       50   CONTINUE
               F = U(6,I)
               RETURN
       60   CONTINUE
               F = WX2*U(2,I) + WXWX3*U(5,I) + AA(3,I)
               RETURN
            END

ame     Type          Offset P Class

A       REAL*8            0    /BLOCK /
D12     REAL*8          960    /BLOCK /
        INTEGER*4         4 *
J       INTEGER*4         0 *
        REAL*8          312    /BLOCK /
X2      REAL*8          936    /BLOCK /
XW      REAL*8          944    /BLOCK /
XWX3    REAL*8          952    /BLOCK /

  673 $PAGE
```

```
C
C
         SUBROUTINE  OUTPUT (A, W, X, V, E, Q, S, CMD, TIME)
C
C
C-------------------------------------------------------------------
C
C             This is the output section of the system. Any further
C        modification of the output requirements of this model must
C        be done in this procedure.  In particular, if no output to
C        the CRT or printer is needed,  it is recommended that  C's
C        be inserted into column 1 of all the WRITE statments.  The
C        simulation clock is updated in this procedure.
C
C-------------------------------------------------------------------
C
         REAL * 8    A(3), W(3), X(3), V(3), E(3), Q(4), S(14)
         INTEGER     CMD(7), EF, EEF, PRTFG
         INTEGER * 4 TIME, T
C
         COMMON /ME/ EF, EEF, PRTFG
C
         TIME = TIME + 1
         T    = (TIME / 10) * 10 - TIME
         IF ( (T .NE. 0) .OR. (PRTFG .EQ. 0)) RETURN
         IF (PRTFG .EQ. 1) GO TO 100
            OPEN (4, FILE = 'LPT1:')
            WRITE (4, 15) TIME / 10
C           WRITE (4, 10) A, W
            WRITE (4, 20) X, V
            WRITE (4, 30) E, W
            WRITE (4, 40) S
            WRITE (4, 50) CMD
            WRITE (4, 90)
            CLOSE (4)
100      IF (PRTFG .NE. 2) CALL  PLOT (CMD)
C
         RETURN
10       FORMAT (' A, W =', 3F10.6, 3X, 3F10.6)
12       FORMAT (' ', 7I10)
15       FORMAT (' TIME =', I6, '  Seconds')
20       FORMAT (' X, V =', 3F10.6, 3X, 3F10.6)
30       FORMAT (' E, W =', 3F10.6, 3X, 3F10.6/)
40       FORMAT ('    S =', 3F10.6, 3X, 3F10.6/
     1           '       ', 3F10.3/
     2           '       ', 4F10.6, 3X,F10.3/)
50       FORMAT ('  CMD =', 7I10)
90       FORMAT (1H0)
         END
```

```
Name    Type          Offset P Class

A       REAL*8             0 *
CMD     INTEGER*4         28 *
E       REAL*8            16 *
EEF     INTEGER*4          4   /ME     /
EF      INTEGER*4          0   /ME     /
PRTFG   INTEGER*4          8   /ME     /
Q       REAL*8            20 *
S       REAL*8            24 *
T       INTEGER*4       1378
TIME    INTEGER*4         32 *
V       REAL*8            12 *
W       REAL*8             4 *
X       REAL*8             8 *

    723 $PAGE
```

```
Line# 1     7
C
C
            SUBROUTINE  DMUL (A, B, C, N)
C
C-------------------------------------------------------------------
C
              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
C
C-------------------------------------------------------------------
C
            REAL * 8   A(N,N), B(N), C(N), S
C
            DO 100 I = 1, N
               S = 0.0
               DO 200 J = 1, N
                  S = S + A(I,J) * B(J)
  200          CONTINUE
               C(I) = S
  100       CONTINUE
            RETURN
            END

ie   Type         Offset P Class

     REAL*8           0 *
     REAL*8           4 *
     REAL*8           8 *
     INTEGER*4     1714
     INTEGER*4     1730
     INTEGER*4       12 *
     REAL*8        1722

 $PAGE
```

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

```
Name    Type        Offset P Class

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


    777 $PAGE
```

```
C
C
      SUBROUTINE  DCSINV (Q, C)
C
C
C--------------------------------------------------------------------
C
C     This subroutine takes the attitude quaternion Q and returns
C     the transpose of the direction cosine matrix
C
C--------------------------------------------------------------------
C
C
      REAL * 8 Q(4), C(3,3)
      REAL * 8 Q1,  Q2,  Q3,  Q4
      REAL * 8 Q11, Q22, Q33, Q44
      REAL * 8 Q12, Q13, Q23
      REAL * 8 Q14, Q24, Q34
C
      Q1  = Q(1)
      Q2  = Q(2)
      Q3  = Q(3)
      Q4  = Q(4)
C
      Q11 = Q1 * Q1
      Q22 = Q2 * Q2
      Q33 = Q3 * Q3
      Q44 = Q4 * Q4
C
      Q12 = 2.0 * Q1 * Q2
      Q13 = 2.0 * Q1 * Q3
      Q23 = 2.0 * Q2 * Q3
      Q14 = 2.0 * Q1 * Q4
      Q24 = 2.0 * Q2 * Q4
      Q34 = 2.0 * Q3 * Q4
C
      C(1,1) =  Q11 - Q22 - Q33 + Q44
      C(2,2) = -Q11 + Q22 - Q33 + Q44
      C(3,3) = -Q11 - Q22 + Q33 + Q44
C
      C(1,2) = Q12 - Q34
      C(2,1) = Q12 + Q34
      C(1,3) = Q13 + Q24
      C(3,1) = Q13 - Q24
      C(2,3) = Q23 - Q14
      C(3,2) = Q23 + Q14
      RETURN
      END
```

```
Name     Type          Offset P Class

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

    826 $PAGE
```

```
   O M V   >>>                                          Page  32
                                                        07-14-84
                                                        12:51:14
Line# 1      7                     Microsoft FORTRAN77 V3.13 8/05/83

e   Type          Size   Class

GFRE                     SUBROUTINE
)CK                968   COMMON
                    48   COMMON
SINV                     SUBROUTINE
Q                        SUBROUTINE
JL                       SUBROUTINE
)TPRD                    SUBROUTINE
    REAL*8               FUNCTION
)GE                      SUBROUTINE
DCTL                     SUBROUTINE
ITPL                     SUBROUTINE
IIT                      SUBROUTINE
IN                       PROGRAM
TCH                      SUBROUTINE
                    48   COMMON
                    12   COMMON
)TION                    SUBROUTINE
VMDL                     SUBROUTINE
TPUT                     SUBROUTINE
                    80   COMMON
OT                       SUBROUTINE
T                        SUBROUTINE
ITGM                     SUBROUTINE
NCOS                     SUBROUTINE
LVE                      SUBROUTINE
ATE                      SUBROUTINE
ORE                      SUBROUTINE
X                        SUBROUTINE
DQ                       SUBROUTINE
CTOR                     SUBROUTINE

ass One     No Errors Detected
           826 Source Lines
```

ORIGINAL PAGE IS
OF POOR QUALITY

APPENDIX 5

ADAM Source Listing

```
$PAGESIZE : 56
$TITLE: '        << A D A M >>'
C
C
C
C                    Program  :  A D A M
C
C
C                              by
C
C                         Dr. W. Teoh
C
C
C----------------------------------------------------------------------
C
C            This program uses the Adam Brashforth method to solve
C            the equation of motion (homogeneous case) numerically
C            and compares the solution with the analytical results
C            such that both outputs are printed.
C
C
C----------------------------------------------------------------------
C
C
      REAL*8 XE(3),VE(3),X(3),V(3),A(3,10),W
      REAL *8  XO(3), VO(3)
      DATA A/30*0.0/
      DATA N,H /10, 0.01/
C
C
C
      WRITE (*, 30)
      READ  (*,32)  W
C
C     get initial conditions
C
      CALL  GETINT (XO, VO, 3)
C
      DO 100 K = 1, 3
         X(K)  = XO(K)
         V(K)  = VO(K)
100   CONTINUE
C
      DO 10 I = 1,36000
         T = 0.1*I
C
C        ***  calculate the analytical solution
C
         CALL EXACT(T,XE,VE,W,XO,VO)
```

```
1     50 C
1     51 C          ***  now get the numerical solution
1     52 C
1     53            CALL SOLVE(X,V,A,N,H,W)
1     54 C
1     55 C          ***  output every 60 seconds
1     56 C
1     57            JJ = (I / 600) * 600
1     58            IF (JJ .EQ. I) THEN
1     59               WRITE(*,20) T,XE,VE
1     60               WRITE(*,20) T,X,V
1     61               WRITE (*, 22)
1     62            END IF
1     63 10      CONTINUE
      64 C
      65 20      FORMAT (F7.1, 6F12.6)
      66 30      FORMAT (' ORBITAL RATE  '\)
      67 22      FORMAT (1H )
      68 32      FORMAT (F15.8)
      69         STOP
      70         END

Name    Type          Offset P Class

A       REAL*8           146
H       REAL             390
I       INTEGER*4        406
JJ      INTEGER*4        414
K       INTEGER*4        402
N       INTEGER*4        386
T       REAL             410
V       REAL*8            98
VO      REAL*8           122
VE      REAL*8            26
W       REAL*8           394
X       REAL*8            50
XO      REAL*8            74
XE      REAL*8             2


     71 $PAGE
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
     << A D A M >>                                                  Page    3
                                                                    07-05-84
                                                                    21:33:40
Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
   72 C
   73 C
   74          SUBROUTINE EXACT(T,XE,VE,W,X,V)
   75 C
   76 C
   77 C-------------------------------------------------------------------
   78 C
   79 C        **  This subroutine calculates the exact solution
   80 C            of the homogeneous ODEs
   81 C
   82 C-------------------------------------------------------------------
   83 C
   84 C
   85 C
   86          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)
   91 C
   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

me    Type         Offset P Class

T     REAL*8          488
OS                          INTRINSIC
IN                          INTRINSIC
T     REAL*8          480
      REAL              0 *
      REAL*8           20 *
      REAL*8            8 *
      REAL*8           12 *
      REAL*8          472
      REAL*8           16 *
      REAL*8            4 *


  103 $PAGE
```

ORIGINAL PAGE IS
OF POOR QUALITY

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

```
.ine# 1      7                                  Microsoft FORTRAN77 V3.13 8/05/83
  153           V(1) = U(2,N+3)
  154           X(2) = U(3,N+3)
  155           V(2) = U(4,N+3)
  156           X(3) = U(5,N+3)
  157           V(3) = U(6,N+3)
  158 C
  159 C            reset U and AA for the next call to SOLVE
  160 C
  161           DO 200 J = 1,6
  162              DO 200 I = 1,3
  163                 U(J,I) = U(J,N+I)
  164                 IF (J .LE. 3) AA(I,J) = AA(I,N+J)
  165  200       CONTINUE
  166 C         DO 300 I = 1,3
  167 C            DO 300 K = 1,3
  168 C               AA(K,I) = AA(K,N+I)
  169 C300        CONTINUE
  170           RETURN
  171           END

1e   Type          Offset P Class

     REAL*8             8 *
     REAL*8             0   /BLOCK /
     REAL*8                 FUNCTION
.G   LOGICAL*4        496
     REAL              16 *
.2   REAL*8           960   /BLOCK /
     INTEGER*4        500
     INTEGER*4        512
     INTEGER*4        504
     INTEGER*4         12 *
     REAL*8           312   /BLOCK /
     REAL*8             4 *
     REAL*8            20 *
2    REAL*8           936   /BLOCK /
√    REAL*8           944   /BLOCK /
√X3  REAL*8           952   /BLOCK /
     REAL*8             0 *

 172 $PAGE
```

```
    173 C
    174 C
    175           SUBROUTINE INNIT(X,V,W,H)
    176 C
    177 C
    178 C-----------------------------------------------------------------
    179 C
    180 C
    181 C        This is the initialization routine which is called only once
    182 C
    183 C
    184 C-----------------------------------------------------------------
    185 C
    186 C
    187           REAL * 8   X(3), V(3), AA(3,13), U(6,13), WX2, WXW, WXWX3
    188           REAL * 8   CWT,  SWT,  T,        W,       HD12
    189           COMMON /BLOCK/ AA, U, WX2, WXW, WXWX3, HD12
    190           WXW   = W*W
    191           WXWX3 = 3*WXW
    192           WX2   = 2*W
    193           HD12  = DBLE(H)/12.0
    194 C
    195           DO 100 I = 1,3
1   196              DO 100 J = 1,6
2   197                 AA(J,I) = 0.0
2   198 100       CONTINUE
    199           DO 200 K = 1,3
1   200              U(2*K-1,3) = X(K)
1   201              U(2*K  ,3) = V(K)
1   202 200       CONTINUE
    203 C
    204           DO 300 I  = 1,2
1   205              T      = H*(I-3)
1   206              CWT    = DCOS(W*T)
1   207              SWT    = DSIN(W*T)
1   208              U(1,I) = X(1) + V(1)*(4*SWT-3*W*T)/W +
1   209      +                6*X(3)*(SWT-W*T) + 2*V(3)*(CWT-1.0)/W
1   210              U(2,I) = V(1)*(4*CWT-3.0) + 6*W*X(3)*(CWT-1.0) -
1   211      +                2*V(3)*SWT
1   212              U(3,I) = X(2)*CWT + V(2)*SWT/W
1   213              U(4,I) = -X(2)*W*SWT + V(2)*CWT
1   214              U(5,I) = 2*V(1)*(1.0-CWT)/W + X(3)*(4.0-3*CWT) +
1   215      +                V(3)*SWT/W
1   216              U(6,I) = 2*V(1)*SWT + 3*X(3)*W*SWT + V(3)*CWT
1   217 300       CONTINUE
    218           RETURN
    219           END
```

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

  e   Type          Offset P Class

A     REAL*8             0   /BLOCK /
      REAL*8           560
  E                          INTRINSIC
COS                          INTRINSIC
SIN                          INTRINSIC
      REAL              12 *
 2    REAL*8           960   /BLOCK /
      INTEGER*4        540
      INTEGER*4        544
      INTEGER*4        548
WT    REAL*8           568
      REAL*8           552
      REAL*8           312   /BLOCK /
      REAL*8             4 *
      REAL*8             8 *
 2    REAL*8           936   /BLOCK /
 W    REAL*8           944   /BLOCK /
WX3   REAL*8           952   /BLOCK /
      REAL*8             0 *

  220 $PAGE
```

```
C
C
      FUNCTION F(J,I)
C
C
C----------------------------------------------------------------------
C
C
C     User supplied function
C
C
C----------------------------------------------------------------------
C
C
      REAL*8 AA(3,13),U(6,13),WX2,WXW,WXWX3,HD12,F
      COMMON /BLOCK/ AA,U,WX2,WXW,WXWX3,HD12
C
      GO TO (10,20,30,40,50,60), J
10    CONTINUE
         F = U(2,I)
         RETURN
20    CONTINUE
         F = -WX2*U(6,I) + AA(1,I)
         RETURN
30    CONTINUE
         F = U(4,I)
         RETURN
40    CONTINUE
         F = -WXW*U(3,I) + AA(2,I)
         RETURN
50    CONTINUE
         F = U(6,I)
         RETURN
60    CONTINUE
         F = WX2*U(2,I) + WXWX3*U(5,I) + AA(3,I)
         RETURN
      END

  Type        Offset P Class

  REAL*8           0   /BLOCK /
  REAL*8         960   /BLOCK /
  INTEGER*4        4 *
  INTEGER*4        0 *
  REAL*8         312   /BLOCK /
  REAL*8         936   /BLOCK /
  REAL*8         944   /BLOCK /
3 REAL*8         952   /BLOCK /
 $PAGE
```

```
 Pass One     No Errors Detected
             285 Source Lines
```

PRECEDING PAGE BLANK NOT FILMED

APPENDIX 6

State Vector Transformation (SVX) Source Listing

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

```
C
C
C     Summary of command string components:
C
C        component    meaning          coord system
C            1          YAW          body frame
C            2           X           floor coordinate
C            3           Y           floor coordinate
C            4           Z           floor coordinate
C            5         PITCH         body frame
C            6          ROLL         body frame
C            7          MODE         integer
c
C
C     This module maintains a local counter to process initial
C     conditions at the start of the simulation.
C
C---------------------------------------------------------------------
C
C
         REAL * 8    S(14)
         REAL * 8    X(3), V(3), L(3), Q(4)
         REAL * 8    XO(3), XM(3), E(3), XHOLD(3)
         REAL * 8    IINV(3), LB(3), W(4)
         REAL * 8    RPY(3), QDOT(4), QW(4,4), A(3,3)
         REAL * 8    LL, UL, UA, CC, AA, HH, QQ, TX, TY, Z
         REAL * 8    ROLL, PITCH, YAW, ROLDOT, PITDOT, YAWDOT
         REAL * 8    Q1, Q2, SY, CY, VX, VY, VZ
C
         INTEGER  CMDRAW(7), COUNT, MODE
C
C        ***  load-time initialization
C
         DATA     COUNT /0/
C
C        ***  decompose state vector and process it
C
         CALL  DECOMP (S, X, V, L, Q)
         IF (COUNT .NE. 0) GOTO 300
C
C           ***  initialization before start
C
            CALL ZERO (XO, 3)
C
C           ***  read parameters
C
            OPEN (1, FILE = 'SVXINT.DAT', STATUS = 'OLD')
            READ (1, 20) CC, LL, AA, HH
            READ (1, 20) IINV
```

```
D Line# 1     7                                  Microsoft FORTRAN77 V3.13 8/05/83
    99 C
   100 C            ***  calculate inverse of moment of inertia tensor
   101 C
   102              DO 50 K = 1, 3
1  103                 IINV(K) = 1.0 / IINV(K)
1  104 50           CONTINUE
   105              CLOSE (1)
   106 C
   107 C            ***  set conversion factors
   108 c
   109              UL = 10000.0
   110              UA = UL
   111              COUNT = COUNT + 1
   112 C
   113 C            ***  set transformation matrix elements to floor coord.
   114 C
   115              E(1) = CC + LL - XO(1)
   116              E(2) = AA - XO(2)
   117              E(3) = HH - XO(3)
   118 C
   119 C            ***  initialize to home orientation
   120 C
   121              CALL  ZERO (RPY, 3)
   122              COUNT = COUNT + 1
   123 C
   124 300       IF (MODE .NE. 1) GO TO 400
   125 C
   126 C            ***  position commands
   127 C
   128 C            ***  update orientation and position
   129 C
   130              CALL QTRPY  (Q, ROLL, PITCH, YAW)
   131              CALL UPDPOS (XM, X, XHOLD, E, 3)
   132 C
   133 C            ***  set orientation part of the command string
   134 C
   135              CMDRAW(7) = 1
   136              CMDRAW(6) = JFIX(ROLL * UA)
   137              CMDRAW(5) = JFIX(PITCH * UA)
   138              CMDRAW(1) = JFIX(YAW * UA)
   139 C
   140 C            ***  transform to TOM_B position in floor coordinates
   141 C
   142              QQ = CC + LL * DCOS(PITCH)
   143 C
   144 C            ***  X-component
   145 C
   146              TX = XM(1) - QQ * DCOS(YAW)
   147              CMDRAW(2) = JFIX (TX * UL)
```

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

```
D Line# 1      7                                Microsoft FORTRAN77 V3.13 8/05/83
    197 C
    198              VX = V(1) + Q1 * CY + Q2 * SY
    199              CMDRAW(2) = JFIX (VX * UL)
    200 C
    201 C
                     ***  Y-component of velocity in floor coordinate
    202 C
    203              VY = V(2) + Q1 * SY - Q2 * CY
    204              CMDRAW(3) = JFIX (VY * UL)
    205 C
    206 C            ***  Z-component
    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            ***  We have an un-recognizable code, default to 1 for
    215 C            ***  position control
    216 C
    217              MODE = 1
    218              GO TO 300
    219 C
    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                         INTRINSIC
DSIN                         INTRINSIC
E       REAL*8        418
IH      REAL*8        566
INV     REAL*8        442
K       INTEGER*4     574
L       REAL*8        370
LB      REAL*8        394
LL      REAL*8        550
MODE    INTEGER*4       8 *
PITCH   REAL*8        602
PITDOT  REAL*8        658
Q       REAL*8        154
Q1      REAL*8        674
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
<   S V X   >>>                                                Page   6
                                                               07-14-84
                                                               13:01:57
 ine# 1      7                        Microsoft FORTRAN77 V3.13 8/05/83
:      REAL*8          682
'^T    REAL*8          210
       REAL*8          618
!      REAL*8          242
)I.DOT REAL*8          650
 L     REAL*8          594
,      REAL*8          186
       REAL*8            0 *
       REAL*8          690
       REAL*8          626
       REAL*8          634
       REAL*8          586
       REAL*8          578
       REAL*8           98
       REAL*8          706
       REAL*8          714
       REAL*8          722
       REAL*8          122
       REAL*8            2
       REAL*8           26
OLD    REAL*8           74
       REAL*8           50
N      REAL*8          610
WDOT   REAL*8          666
       REAL*8          642

  223 $PAGE
```

```
D Line# 1      7                                 Microsoft FORTRAN77 V3.13 8/05/83
    224 C
    225 C
    226         SUBROUTINE   DECOMP (S, X, V, L, Q)
    227 C
    228 C-------------------------------------------------------------------------
    229 C
    230 C       This procedure decomposes the State vector S into its components
    231 C       which are also vectors. They have the following meaning :
    232 C
    233 C          Vector    Dimension          Meaning
    234 C            X           3         Position vector in LVF
    235 C            V           3         Velocity vector in LVF
    236 C            L           3         Angular momentum in LVF
    237 C            Q           4         Unit quaternion in body frame
    238 C
    239 C-------------------------------------------------------------------------
    240 C
    241            REAL * 8     S(14), X(3), V(3), L(3), Q(4)
    242 C
    243            CALL   LD (S, X,  1, 3)
    244            CALL   LD (S, V,  4, 3)
    245            CALL   LD (S, L,  7, 3)
    246            CALL   LD (S, Q, 10, 4)
    247 C
    248            RETURN
    249            END

Name    Type         Offset P Class

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

   250 $PAGE
```

```
  251 C
  252 C
  253           SUBROUTINE   LD (A, B, M, N)
  254 C
  255 C------------------------------------------------------------------
  256 C
  257 C     This procedure copies N elements of vector A to vector B,
  258 C     starting at the M-th element
  259 C
  260 C------------------------------------------------------------------
  261 C
  262           REAL * 8  A(14), B(N)
  263           DO 100 K = 1, N
  264              B(K) = A(M + K - 1)
  265 100       CONTINUE
  266           RETURN
  267           END

ne   Type         Offset P Class

     REAL*8            0 *
     REAL*8            4 *
     INTEGER*4       750
     INTEGER*4         8 *
     INTEGER*4        12 *


  268 SPAGE
```

```
D Line# 1      7
    269 C
    270 C
    271             SUBROUTINE  MMUL (A, B, C, N)
    272 C
    273 C-----------------------------------------------------------------
    274 C
    275 C     This procedure performs a matrix multiplication of an NxN
    276 C     matrix A to an  N-element column matrix  B to yield an N-
    277 C     element column matrix C
    278 C
    279 C-----------------------------------------------------------------
    280 C
    281             REAL * 8    A(N,N), B(N), C(N), S
    282 C
    283             DO 100 I = 1, N
1   284                S = 0.0
1   285                DO 200 J = 1, N
2   286                   S = S + A(I,J) * B(J)
2   287 200            CONTINUE
1   288                C(I) = S
1   289 100         CONTINUE
    290             RETURN
    291             END

Name    Type          Offset P Class

A       REAL*8            0 *
B       REAL*8            4 *
C       REAL*8            8 *
I       INTEGER*4       758
J       INTEGER*4       774
N       INTEGER*4        12 *
S       REAL*8          766

   292 $PAGE
```

```
ine# 1     7                                 Microsoft FORTRAN77 V3.13 8/05/83
 293 C
 294 C
 295          SUBROUTINE  ZERO (A, N)
 296 C
 297 C----------------------------------------------------------------------
 298 C
 299 C     This procedure initializes an N-element array A to zero at
 300 C     run time
 301 C
 302 C----------------------------------------------------------------------
 303 C
 304          REAL * 8  A(N)
 305          DO 100 K = 1, N
 306             A(K) = 0.0
 307 100      CONTINUE
 308          RETURN
 309          END

ne   Type          Offset P Class

     REAL*8           0 *
     INTEGER*4      782
     INTEGER*4        4 *


 310 $PAGE
```

```
D Line# 1      7                              Microsoft FORTRAN77 V3.13 8/05/83
    311 C
    312 C
    313           SUBROUTINE  UPDPOS (XM, X, XHOLD, E, N)
    314 C
    315 C-------------------------------------------------------------------
    316 C
    317 C     This procedure updates the position of the OMV in local vertical
    318 C     frame (XHOLD).
    319 C
    320 C     The new position of the module in floor coordinates is then com-
    321 C     puted (XM)
    322 C
    323 C-------------------------------------------------------------------
    324 C
    325 C
    326           REAL * 8   XM(N), X(N), XHOLD(N), E(N)
    327 C
    328           DO 100 K = 1, N
    329              XHOLD(K) = X(K)
    330              XM(K)    = XHOLD(K) + E(K)
1   331 100       CONTINUE
    332           RETURN
    333           END

Name    Type         Offset P Class

E       REAL*8          12 *
K       INTEGER*4      790
N       INTEGER*4       16 *
        REAL*8           4 *
XHOLD   REAL*8           8 *
XM      REAL*8           0 *

    334 $PAGE
```

```
ine# 1     7                                Microsoft FORTRAN77 V3.13 8/05/83
 335 C
 336 C
 337          INTEGER FUNCTION JFIX (RR)
 338 C
 339 C-----------------------------------------------------------------------
 340 C
 341 C     This procedure properly rounds a real number R to the nearest
 342 C     integer.
 343 C
 344 C-----------------------------------------------------------------------
 345 C
 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

me   Type        Offset P Class

-IX                       INTRINSIC
     REAL           798
     REAL*8           0 *


 356 $PAGE
```

```
D Line# 1      7                               Microsoft FORTRAN77 V3.13 8/05/83
    357 C
    358 C
    359         SUBROUTINE   SETQ (QW, Q)
    360 C
    361 C-------------------------------------------------------------------
    362 C
    363 C      This procedure constructs a 4x4 transformation matrix QW  from
    364 C      the attitude quaternions Q
    365 C
    366 C      For reference, please see "Software Specifications For Docking
    367 C      Simulation Of The OMV" by J. Micheals, January, 1984.
    368 C
    369 C-------------------------------------------------------------------
    370 C
    371         REAL * 8    QW(4,4), Q(4)
    372 C
    373         DO 100 I = 1, 3
 1  374             DO 110  J = I+1, 4
 2  375                KK = I + J
 2  376                K = KK - (KK/4) * 4
 2  377                IF (K .EQ. 0) K = 2
 2  378                ISGNN = 1
 2  379                IF ((J .EQ. I+1) .AND. (J.NE. 4)) ISGNN = -1
 2  380                QW(I,J) = ISGNN * Q(K)
 2  381 110         CONTINUE
 1  382             QW(I,I) = Q(4)
 1  383 100     CONTINUE
    384         QW(4,4) = Q(4)
    385 C
    386         DO 200 I = 2, 4
 1  387            KK = I - 1
 1  388            DO 200 J = 1, KK
 2  389               QW(I,J) = -QW(J,I)
 2  390 200     CONTINUE
    391         RETURN
    392         END

Name    Type         Offset P Class

[       INTEGER*4      802
ISGNN   INTEGER*4      818
J       INTEGER*4      806
:       INTEGER*4      814
:K      INTEGER*4      810
?       REAL*8           4 *
)W      REAL*8           0 *

   393 $PAGE
```

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

```
ame    Type        Offset P Class

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

ORIGINAL PAGE IS
OF POOR QUALITY

```
J       INTEGER*4         830
K       INTEGER*4         826
Q       REAL*8              4 *
QKJ     REAL              854
QKS     REAL*8          *****
QRS     REAL*8            858
S1      REAL*8            842

    438 $PAGE
```

```
 439 C
 440 C
 441          REAL    FUNCTION  DLTKRK (K,J)
 442 C
 443 C
 444 C-----------------------------------------------------------------------
 445 C
 446 C
 447          REAL                S
 448          INTEGER          K, J
 449          S = 1.0
 450          IF (K .NE. J) S = -1.0
 451          DLTKRK = S
 452          RETURN
 453          END

ne   Type          Offset P Class

     INTEGER*4          4 *
     INTEGER*4          0 *
     REAL             866


 454 $PAGE
```

```
    455 C
    456 C
    457        REAL    FUNCTION QSIGN(K,J)
    458 C
    459 C
    460 C----------------------------------------------------------------------
    461 C
    462 C
    463        S = 1.0
    464        L = J + K
    465        IF (MOD(L,2) .EQ. 0) S = -1.0
    466        QSIGN = S
    467        RETURN
    468        END

Name    Type         Offset P Class

J       INTEGER*4         4 *
K       INTEGER*4         0 *
L       INTEGER*4       874
MOD                           INTRINSIC
S       REAL            870

    469 $PAGE
```

```
ine# 1     7
 470 C
 471 C
 472          SUBROUTINE  QTRPY (Q, R, P, Y)
 473 C
 474 C
 475 C        This subroutine calculates a reasonable set of roll,
 476 C        pitch and yaw from the quaternion Q
 477 C
 478 C
 479          REAL * 8    Q(4), R, P, Y, M, THETA, CA, CB, CG
 480 C
 481          M  = DSQRT (Q(1)**2 + Q(2)**2 + Q(3)**2)
 482 C
 483 C        calculate direction cosines CA, CB, CG
 484 C
 485          IF (DABS(M) .LE. 1.0D-20) THEN
 486             CA = 0.0
 487             CB = 0.0
 488             CG = 0.0
 489          ELSE
 490             CA = Q(1) / M
 491             CB = Q(2) / M
 492             CG = Q(3) / M
 493          END IF
 494 C
 495 C        calculate angle of rotation about Euler axis
 496 C
 497          THETA = 2.0 * DACOS(Q(4))
 498 C
 499 C        now determine the roll, pitch and yaw
 500 C
 501          R = CA * THETA
 502          P = CB * THETA
 503          Y = CG * THETA
 504          RETURN
 505          END

ame    Type        Offset P Class

\      REAL*8        886
3      REAL*8        894
;      REAL*8        902
\BS                          INTRINSIC
\COS                         INTRINSIC
SQRT                         INTRINSIC
       REAL*8        878
       REAL*8          8 *
       REAL*8          0 *
       REAL*8          4 *
```

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

    506 $PAGE
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
<<   S V X   >>>                                     Page  20
                                                     07-14-84
                                                     13:01:57
ine# 1     7                    Microsoft FORTRAN77 V3.13 8/05/83

ne    Type          Size    Class

ECOMP                       SUBROUTINE
IRCOS                       SUBROUTINE
 KRK  REAL                  FUNCTION
 IX   INTEGER*4             FUNCTION
D                           SUBROUTINE
 JL                         SUBROUTINE
 IGN  REAL                  FUNCTION
TRPY                        SUBROUTINE
TQ                          SUBROUTINE
 (                          SUBROUTINE
DPOS                        SUBROUTINE
ERO                         SUBROUTINE

Pass One     No Errors Detected
           506 Source Lines
```

ORIGINAL PAGE IS
OF POOR QUALITY

APPENDIX 7

Mobility-base Control Logic (TOM_C) Source Listing

```
C
C
C
C                          TOM-B EXECUTIVE
C
C
C
C
C
C--------------------------- Version  11.2 --------------------------
C
C     This is the main program for the OMV on-board processing
C         logic.
C     The following steps are carried out:
C         1.  Performs a system initialization
C         2.  Set up an infinite loop to process each major cycle
C             until both CMDRAW(1) & CMDRAW(2) are <= -99.
C             During each major cycle, a subroutine PMAJOR
C             performs all the necessary functions.  It then waits
C             for the next cycle.  The start of the next cycle is
C             indicated when FLAG is cleared.
C
C     Each major cycle has a period of 0.1 sec; this value is in-
C             put from disk during system initialization.  Since
C             cycle execution is tracked by using this variable,
C             the period may be altered by changing its
C             value on disk.
C
C     Absolute commands will be used throughout.
C
C     It is assumed that a routine SETUP sets an interrupt schecule
C             and performs all the necessary services.
C     ------------------------------------------------------------
C
      INTEGER * 4   FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
      INTEGER * 4   CYCLE
      COMMON /CMMD/ CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
      COMMON /CYCL/ CYCLE, JSTF1
C
C     ***  system initialization  ***
C
      CALL  INITOM
C
C     CALL GOHOME
C
C
C
C     ***  MONITOR CYCLE PROCESS  ***
C
C
C     WHILE ((CMDRAW(3).GT.-99) .AND. (CMDRAW(2).GT.-99)) DO
 100       IF((CMDRAW(3).LT.-99).AND.(CMDRAW(2).LT.-99))GOTO 900
C
C          ***  PROCESS A MAJOR CYCLE  ***
C
           CALL PMAJOR
C
C          ***  WAIT UNTIL NEXT CYCLE & CONTINUE  ***
C
           CALL WAIT
```

```
          GO TO 100
C     END WHILE
 900  CONTINUE
C
C     ***  perform house cleaning before quitting  ***
C
C     CALL GOHOME
      STOP
      END
C
C
      SUBROUTINE INITOM
C
C     ---------------------------------------------------------
C
C     This procedure performs a system initialization.
C          1.  A disk data file called INITOM is accessed for the
C              pertinent information.
C          2.  power to disk drive may then be disconnected
C          3.  press <CR> to continue.
C          4.  INIT calls SETUP to establish an interrupt schedule.
C
C      ---------------------------------------------------------
C
       INTEGER * 4      FLAG,   CMDMOD, CMDRAW(9),   CMDRET(9)
       INTEGER          DOF
       INTEGER * 2      FRTBLX(20,2),   FRTBLY(20,2),JETBUF(40)
       REAL             LX,     LY,     MASS,        MAJOR,  JZZ
       REAL             MTRVRD(6),      MTRVCL(6),   MTRVOF(6)
       REAL             NAVVAL(3),      NAVCAL(3),   NAVOFF(6)
       REAL             MTRPRD(6),      MTRPCL(6),   MTRPOF(6)
       COMMON /CMMD/    CMDRET, CMDRAW, CMDVAL(9),   CMDMOD, FLAG
       COMMON /DACO/    DACRDG(6),      DACCAL(6),   DACCOF(6)
       COMMON /DYNA/    THRUST,         ACC(2),      LX, LY, DOF
       COMMON /JETS/    NTHRX, NTHRY, FRTBLX, FRTBLY, JETBUF,SCLX,SCL
       COMMON /MOTR/    MTRPRD,         MTRPCL,      MTRPOF
       COMMON /MOTV/    MTRVRD,         MTRVCL,      MTRVOF
       COMMON /NAVG/    NAVVAL,         NAVCAL,      NAVOFF
       COMMON /PHYS/    MASS,           MAJOR,       JZZ,    PIRAD
       COMMON /POSN/    POSTN(9),       OPOSTN(9)
       COMMON /RATE/    VLCTY(9),       OLDVEL(9)
       COMMON /SNSR/    SNRR(3),        SNRC(3),     SNRB(3)
       COMMON /PRCN/    EPSL, EPSA, UL, UA
       COMMON /DELV/    DV(3)
C      Whew
C
C      WRITE (*,39)
39     FORMAT (' in  INITOM')
C
C
C      Implementation notes :
C
C      PHY. QTY             STANDARD                   MKS
C      ========       ===================      ===================
C
C      MASS           77.64 SLUG (2500 LB)     1132.77       KG
C      MAJOR          0.1         SEC          0.1           SEC
C      JZZ            334.17 SLUG-FT-FT        452.95      KG-M-M
C      THRUST         3       LB               13.345        NT
C      LX             32      IN               0.787         M
C      LY             31      IN               0.762         M
C
C      ACC            0.0773 FT/SEC/SEC        0.02356 M/SEC/SEC
C      WZ             0.04788   RAD/SEC        0.04788   RAD/SEC
C
C      o     Both ACC and WZ are used in the model OMV
```

```
C
C
      LG = 4
      OPEN (LG, FILE = 'INITOM.DAT',STATUS='OLD')
C
      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
C
      READ (LG, 20)  NTHRX
      READ (LG, 20)  NTHRY
      READ (LG, 20)  DOF
C
      READ (LG, 10)  SCLX
      READ (LG, 10)  SCLY
C
      DO 100 K = 1, 3
         READ (LG, 10) SNRC(K), SNRB(K)
100   CONTINUE
C
      DO 110 K = 1, 3
         READ (LG, 10) NAVCAL(K), NAVOFF(K)
110   CONTINUE
C
      DO 120 K = 1, 3
         READ (LG, 10) MTRPCL(K), MTRPOF(K)
120   CONTINUE
C
      DO 130 K = 1, 3
         READ (LG, 30) MTRVCL(K), MTRVOF(K)
130   CONTINUE
C
      DO 140 K = 1, 3
         READ (LG, 30) DACCAL(K), DACCOF(K)
140   CONTINUE
C
      DO 200 K = 1, DOF
         READ (LG, 10) POSTN(K)
         OPOSTN(K) = POSTN(K)
         VLCTY(K)  = 0.0
         OLDVEL(K) = 0.0
         IF (K .LE. 3) DV(K) = 0.0
200   CONTINUE
C
      NN = NTHRX * 4
      DO 300 K = 1, NN
         READ (LG, 20) FRTBLX(K,1)

300   CONTINUE
C
      NN = NTHRY * 4
      DO 350 K = 1, NN
         READ (LG, 20) FRTBLY(K,1)
350   CONTINUE
C
C     Compute other quantities
C
      PI    = 355.0 / 113.0
      PIRAD = 180.0 / PI
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        A      = THRUST / MASS
        ACC(1)= 2 * NTHRX * A
        ACC(2)= 2 * NTHRY * A
        CALL  SETUP
        RETURN
10           FORMAT (F15.8)
20           FORMAT (I2)
30           FORMAT (F15.8)
        END
C
C
      SUBROUTINE   WAIT
C
C     --------------------------------------------------------------
C
C     This procedure synchronizes TOM_B EXECUTIVE to the interrupt
C     service routine.
C
C     This procedure
C          A.  Transmits the current position & orientation  to the
C              main-framd computer &
C          B.  waits until interrupt service routine is completed
C              when FLAG is cleared.
C              Note that
C               FLAG = 0      means system is OK. TOM_B EXECUTIVE sh-
C                             ould proceed in the normal manner.
C               FLAG = -1     means there is a hardware failure of
C                             some sort.  In this case, the main frame
C                             is notified and the mission aborted.
C               FLAG = 1      means not ready.  Wait some more.
C              There is no provision to halt and power down TOM_B in
C              case of hardware failure from software at this time.
C
C     --------------------------------------------------------------
C
      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
C
C     ***  Report position  ***
C
      CALL XMIT
C
C     ***  Wait until ready  ***
C
C     WHILE (FLAG .GT. 0) DO
 100       IF (FLAG .LE. 0) GO TO 200
C          GO TO 100
C     END WHILE
C
C     ***  See if there is any hardware failures  ***
 200  IF (FLAG .GE. 0) RETURN

C
C        ***  We have hardware failure  ***
C
         DO 300 K=1,DOF
            CMDRET(K) = -99
 300     CONTINUE
C
C        ***  Tell mainframe & abort mission  ***
C
         CALL SENDIT
         STOP
```

```
C     ENDIF
C
C
  900 RETURN
      END
C
C
      SUBROUTINE XMIT
C
C     ----------------------------------------------------------
C
C     This procedure takes the current TOM_B position & places it
C     in a buffer.  An I / O driver SENDIT is called to transmit
C     this information to the main frame.
C     All lengths are expressed in meters, while all angular quantiti
C     are expressed in radians.  All must be scaled before sending.
C
C     ----------------------------------------------------------
C
      REAL            LX, LY
      INTEGER         DOF
      INTEGER * 4     CMDRET(9), FLAG, CMDMOD, CMDRAW(9)
      COMMON /PHYS/   MASS, MAJOR, JZZ, PIRAD
      COMMON /DYNA/   THRUST, ACC(2), LX, LY, DOF
      COMMON /PRCN/   EPSL, EPSA, UL, UA
      COMMON /POSN/   POSTN(9), OPOSTN(9)
      COMMON /CMMD/   CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
C************************************************************
      COMMON /RATE/   VLCTY(9), OLDVEL(9)
C************************************************************
C
      DO 100 K=1,DOF
          FACTOR = UA
          IF ((K .GT. 1) .AND. (K .LT. 5)) FACTOR = UL
          TMP = POSTN(K) * FACTOR
          CMDRET(K) = IFIX(TMP + 0.5)
  100 CONTINUE
C
      CALL SENDIT
C
      RETURN
      END
C
C
      SUBROUTINE PMAJOR
C
C     ----------------------------------------------------------
C
C     This procedure processes a major cycle by:
C          A.  determine its current position.
C          B.  determine its current velocity.
C          C.  decode the command sequence.
C          D.  decide if it needs to adjust its position/velcity
C              based on the value of FIRFLG :

C              1 :  FIRFLG = 0  ; no adjustment needed.
C              2 :  FIRFLG = 10 ; use thrusters
C              3 :  FIRFLG = 1  ; use motors
C              4 :  FIRGLG = 11 ; use both thrusters & motors
C          E.  In case when both thrusters  &  motors need to be
C              used, the thrusters are fired first.
C
C     ----------------------------------------------------------
C
      INTEGER        FIRFLG, JSTF1
      INTEGER * 4    CYCLE
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
      COMMON /CYCL/ CYCLE, JSTF1
C
C     ***  interpret command sequence & place them in CMDVAL(1..6)
C
      CALL CMDFIX
C
C     ***  determine present position & rate   ***
C
      CALL UPDATE
C
C     ***  check to see if it is necessary to move anything  ***
C
      FIRFLG = 0
      CALL DECISN (FIRFLG)
      IF (FIRFLG .GE. 10) CALL  THRSTR
      JSTF1 = 0
C
C     ***  see if it is necessary to move any motors as well  ***
C
      FIRFLG = FIRFLG - 10
C**** IF (FIRFLG .GT. 0) CALL MOTORS
C
C     ***  Grand exit stage left  ***
C
      RETURN
      END
C
C
      SUBROUTINE UPDATE
C
C     ------------------------------------------------------------
C
C     This procedure updates the position and velocities of all
C     the six axis of the mobile base, after having saved its
C     current state
C     The axes assignment is as follows :
C
C             Axis        Dynamic quantity
C             ====        ================
C               1         yaw of mobile base
C               2          X
C               3          Y
C               4          Z
C               5         pitch
C               6         roll
C
C
C     Release notes :
C
C     o   Triangulation navigation system is not ready.  Position
C         X and Y are calculated in NAVGN instead of measured.
C
C     o   Motor rate feedback is unreliable, but position feedback
C         is.  Thus, motor rates are derived from the position feed-

C         vack data by differentiation, until hardware is rectified.
C
C
C     ------------------------------------------------------------
C
      INTEGER * 2   MTRBUF(6), MTVBUF(6)
      INTEGER * 2   SNRBUF(3), NAVBUF(3), GYRBUF(18), DACBUF(6)
      INTEGER       DOF
      REAL          MASS, MAJOR, JZZ, LX, LY
      REAL          MTRPRD(6), MTRPCL(6), MTRPOF(6)
```

```
      REAL          THETA, V(3), JG, W(2), VV(3)
C
      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, MTVBUF, SNRBUF, DACBUF
      COMMON /SNSR/ SNRR(3), SNRC(3), SNRB(3)
C
C
      DO 100 K = 1, DOF
          OPOSTN(K) = POSTN(K)
          OLDVEL(K) = VLCTY(K)
100   CONTINUE
      THETA = POSTN(1)
C
      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)
220      CONTINUE
         SNRBUF(K) = JG / 100000.0
         V(K)      = V(K) + JG/100000.0
200   CONTINUE
C
C     transform to floor coordinates
C
      VLCTY(1) = V(1)
      V(1)  = V(2)
      V(2)  = V(3)
      CALL  BTF (V, THETA, W)
      VLCTY(2) = W(1)
      VLCTY(3) = W(2)
C
      CALL  NAVGN (MAJOR, GYRBUF, 18)
C
C
C     ***  Find position & velocity of motors (axes 4..6)
C          rates are obtained by differentiation
C
      KK = DOF - 3
C     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
400       CONTINUE
900   CONTINUE
C
      RETURN
      END
C
C
       SUBROUTINE  FTB (F, THETA, B)
```

```
C
C--------------------------------------------------------------------
C
C
C      This subroutine takes a vector F(2) as expressed in flat floor
C      coordinates and transforms it to body coordinates through a
C      rotation of THETA radians.  The transformed vector is placed
C      in the array B.
C
C
C--------------------------------------------------------------------
C
C
      REAL    F(2), B(2)
C
      C    =  COS (THETA)
      S    =  SIN (THETA)
      B(1) =  F(1) * C  +  F(2) * S
      B(2) = -F(1) * S  +  F(2) * C
      RETURN
      END
C
C
      SUBROUTINE  BTF (B, THETA, F)
C
C
C--------------------------------------------------------------------
C
C
C      This subroutine takes a body vector and transforms it to
C      flat floor coordinates via a pure rotation by THETA radians.
C
C--------------------------------------------------------------------
C
C
      REAL    B(2), F(2)
C
      C    =  COS(THETA)
      S    =  SIN (THETA)
      F(1) =  B(1) * C  -  B(2) * S
      F(2) =  B(1) * S  +  B(2) * C
      RETURN
      END
C
C
      SUBROUTINE NAVGN (PERIOD, JBUF, N)
C
C     ---------------------------------------------------------------
C
C     This is a temporary procedure to determine absolute position  &
C     orientation of TOM_B by using the rate information to allow for
C     system checkout.
C
C     This effectively by-passes the triangulation navigation system.
C
C
C     This procedure must be replaced ultimately by an appropriate on
C
C     ---------------------------------------------------------------
C
      INTEGER * 2       JBUF(N)
      REAL * 8          THETA, BODE6
      COMMON /POSN/     POSTN(9), OPOSTN(9)
      COMMON /RATE/     VLCTY(9), OLDVEL(9)
C
C     EPS = 0.0001
```

```
C
C     THETA = BODE6(JBUF, 6, 0.0, 0.1)
C     POSTN(1) = THETA
C
      DO 100 K = 1, 3
         DELTA = (VLCTY(K) + OLDVEL(K)) * PERIOD / 2.0
         POSTN(K) = OPOSTN(K) + DELTA
100   CONTINUE
C
      RETURN
      END
C
C
      REAL * 8  FUNCTION BODE6 (F, N, A, B)
C
C
C     -------------------------------------------------------------
C
C     This subroutine uses simpson's rule to perform a simple
C     integration to obtain THETA
C
C     -------------------------------------------------------------
C
C
      INTEGER   F(N)
      REAL * 8  H, SUM
C
C     WRITE (*,39)
39    FORMAT (' in  BODE')
C
      H = (B - A) / FLOAT(N - 1)
      SUM =     19.0 * (FLOAT(F(1)) + FLOAT(F(6)))
     1        + 75.0 * (FLOAT(F(2)) + FLOAT(F(5)))
     2        + 50.0 * (FLOAT(F(3)) + FLOAT(F(4)))
      BODE6 = 5.0 * H * SUM / 288.0
      RETURN
      END
C
C
      SUBROUTINE CMDFIX
C
C     -------------------------------------------------------------
C
C     This procedure processes transmitted commands in CMDRAW and
C     calculate their actual values and places them in CMDVAL.
C
C     It is assumed that absolute ( and not delta ) commands will
C     be used.  Depending on the value of CMDMOD, rate or position
C     commands are implemented:
C                   CMDMOD = 0   means rate control
C                   CMDMOD = 1   means positional control
C
C     System of units used in TOM_B EXECUTIVE is MKS.
C

C
C
C     According to TOM BRYAN, delta commands will never be used,
C     but this procedure can be modified if & when delta commands
C     are desired.
C
C     Command index assignment:
C
C     INDEX       AXIS        TYPE        MODE=0       MODE=1
C     -----       ----        ----        ------       ------
C       1         YAW         angular     THETA.       THETA (YAW)
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
C        2          X          length        X*            X
C        3          Y          length        Y*            Y
C        4          Z          length        Z*            A
C        5        PITCH       angular        P*            P
C        6         ROLL       angular        R*            R
C
C
C       ---------------------------------------------------------
C
      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 /CMMD/  CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
      COMMON /PRCN/  EPSL, EPSA, UL, UA
C
C     ***  CONVERT AHOY! ***
C
      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
 100  CONTINUE
C
      CMDMOD = CMDRAW(7)
C
      RETURN
      END
C
C
      SUBROUTINE DECISN(FIRFLG)
C
C      ---------------------------------------------------------
C
C     This procedure decides whether or  not corrective action
C     needs to be taken by setting and returning a flag FIRFLG :
C           A.   FIRFLG = 0        ; No action needed
C           B.   0<FIRFLG<10       ; Need to move DC motors
C           C.   FIRFLG >=10       ; Need to fire thrusters
C           D.   FIRFLG = 11       ; Need to do both
C     Decision is made based on the comparison between  the com-
C     mand sequence & current TOM_B dynamic  quantities,  remem-
C     bering that the system at this instance  is  under  either
C     position or rate control, and that the commands are absol-
C     ute commands.
C
C     ----------------------------------------------------------
C
      INTEGER        DOF, FIRFLG, FG
      INTEGER * 4    FLAG, CMDMOD, CMDRAW(9),CMDRET(9)
      REAL           LX, LY
      COMMON /DYNA/ THRUST, ACC(2), LX, LY, DOF

      COMMON /PRCN/ EPSL, EPSA, UL, UA
      COMMON /CMMD/ CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
C
C     ***  Check motor section  ***
C
      CALL CHKCMD(4,DOF,EPSL,EPSA,FG)
      FIRFLG = FG
C
C     ***  Check thrusters section   ***
C
      IF (CMDMOD .NE. 0) GOTO 100
```

```
            EPSL = 1.0E-6
            EPSA = EPSL
C     END IF
100   CONTINUE
      CALL CHKCMD(1,3,EPSA,EPSL,FG)
      FIRFLG = FIRFLG + FG * 10
C
      RETURN
      END
C
C
      SUBROUTINE CHKCMD(FIRST,LAST,EP1,EP2,FG)
C
C     ---------------------------------------------------------
C
C     This procedure checks the absolute command against the ve-
C     hicle's position or velocity to determine if any corrective
C     action needs to be taken.  If it does, the flag FG sill be
C     set. FG is either 0 or 1 on return from this subroutine.
C
C     ---------------------------------------------------------
C
      INTEGER        FIRST, LAST, FG
      INTEGER * 4    FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
      COMMON /POSN/  POSTN(9), OPOSTN(9)
      COMMON /RATE/  VLCTY(9), OLDVEL(9)
      COMMON /CMMD/  CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
C
C     ***  initialize loop parameters  ***
C
      FG = 0
      K = FIRST
      EPSLN = EP1
C
C     ***  check between FIRST & LAST inclusive  ***
C
C     REPEAT
 100        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
C     UNTIL K > LAST OR FG = 1
C
 200  RETURN
      END
C
C
      REAL FUNCTION FSIGN(X)
C
C     ---------------------------------------------------------
C

C     This procedure returns the sign of a REAL variable as  +1.0
C     or -1.0.
C
C     ---------------------------------------------------------
C
      REAL   X
C
      IF (X - 0.0) 100, 200, 200
100   FSIGN = -1.0
      RETURN
200   FSIGN = 1.0
```

```
C
      RETURN
      END
C
C
      SUBROUTINE ITABLE
C
C     ------------------------------------------------------------
C
C     This procedure initializes all entries of both firing tables
C     to zero.
C
C     ------------------------------------------------------------
C
      INTEGER * 2   FRTBLX(20,2), FRTBLY(20,2), JETBUF(40)
      COMMON /JETS/ NTHRX, NTHRY, FRTBLX, FRTBLY, JETBUF, SCLX,SCLY
C
C     ***  initialize   X-  firing table  ***
C
      NX = NTHRX * 4
      NY = NTHRY * 4
C
      DO 100 K=1,NX
          FRTBLX(K,2) = 0
  100 CONTINUE
C
C     ***  Now take care of  Y-  firing table  ***
C
      DO 200 K=1,NY
          FRTBLY(K,2) = 0
  200 CONTINUE
      DO 300 K=1,40
          JETBUF(K) = 0
  300 CONTINUE
C
      RETURN
      END
C
C
      SUBROUTINE TABLE(F1,F2,NT,TBL,SCALE,NDIR)
C
C     ------------------------------------------------------------
C
C     This procedure sets up the appropriate firing table by :
C          A.  determining the appropriate # of thrusters to be used
C          B.  calculate the corresponding firing times. &
C          C.  load the information in the firing table buffer.
C
C     To ensure stability of the vehicle, F1 & F2 must be symmetrized
C     (if such a word exists at all).
C
C     -------------------------------------------------------------
C
      REAL          T(2), TIME(2), LX, LY, MASS, MAJOR, JZZ

      INTEGER       BASE(2),N(2),DOF
      INTEGER * 2   TBL(20,2)
      COMMON /DYNA/ THRUST, ACC(2), LX, LY, DOF
      COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD
C
C     ***  Calculate firing times & make them symmetric when possible
C          Firing times are in seconds
C
      T(1) = F1 / THRUST
      T(2) = F2 / THRUST
```

```
C     Same  EPS  as in TSTFIR
C
      EPS  = 0.001 * MAJOR
      CALL SYMM(T,EPS)
C
C     ***  set base indes & actual firing times  ***
C
      DO 100 K=1,2
          BASE(K) = (K-1) * NT + 1
          TM = T(K)
          IF (TM .LT. 0) BASE(K) = BASE(K)  +  2 * NT
C         ***  calculate # of thrusters to be used  ***
C         TM = ABS(TM)
          CALL NMTHR(TM,NN,NT)
          N(K) = NN
C         ***  NOTE:  NN is the # of thrusters to be used  ***
          TIME(K) = (TM / FLOAT(NN)) * SCALE / MAJOR
 100  CONTINUE
C
C     ***  Symmetrize TIME(1) & TIME(2)
C
      CALL SYMM(TIME,EPS)
C
C     ***  fill up the firing table buffer  ***
C
      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
C
      RETURN
      END
C
C
      SUBROUTINE SYMM(T,EPSLN)
C
C     ------------------------------------------------------------
C
C     This procedure symmetrizes two forces T(1), T(2) acting along
C     the same line, but can be in opposite directions.
C
C     When the magnitudes of the two forces has an absolute dif-
C     ference less than the required precision EPSLN the two magni-
C     titudes are made to be identical.
C
C     This procedure is implemented hopefully to take care of minor
C     truncation errors since all computations are carried  out  in
C     single precision.
C
C     ------------------------------------------------------------
C
      REAL  T(2)
C
C     ***  Calculate magnitudes & signs of each force
C
      T1  = T(1)
      AT1 = ABS(T1)
      S1  = FSIGN(T1)
C
      T2  = T(2)
      AT2 = ABS(T2)
      S2  = FSIGN(T2)
```

```
      TT  = AMIN1(AT1,AT2)
C
C     ***  Now symmetrize them  ***
C
C     IF (ABS(AT1 - AT2) .LE. EPSLN) THEN
          IF (ABS(AT1-AT2) .GT. EPSLN) GO TO 100
          T(1) = S1 * TT
          T(2) = S2 * TT
C     ENDIF
C
 100  RETURN
      END
C
C
      SUBROUTINE NMTHR(T,NN,NT)
C
C     ------------------------------------------------------------
C
C     This procedure calculates the optimal number of thrusters to
C     be used on each side.
C
C     T :  Firing time in major cycles
C     NN:  # of thrusters to be used
C     NT:  Total # of thrusters available on 1 side.
C
C     At present, it is decided that an ad hoc limit of 5 major cy-
C     cles will be used.
C
C     E.G.     If it takes 1 thruster  for 6 seconds,
C              we will use 2 thrusters for 3 seconds.
C
C     Thus, the # of thrusters on each side that is needed is:
C
C                          NN = FIRING TIME/5
C
C     Once NN is decided, the new firing times must be readjusted to
C     reflect the change.  This is done in the calling procedure TA-
C     BLE.
C
C     It is necessary that 1 <= NN <= NT
C
C     --------------------------------------------------------------
C
      REAL           MASS,  MAJOR,  JZZ
      COMMON /PHYS/  MASS,  MAJOR,  JZZ,  PIRAD
C
      TX = ABS(T)
      NN = IFIX (TX / MAJOR + 0.5)
      IF (NN .EQ. 0) NN = 1
      IF (NN .GT. NT) NN = NT
C
      RETURN
      END
C
C
      SUBROUTINE LOADIT(K,TAB,JB)
C     ------------------------------------------------------------
C
C     This procedure takes the contents of a firing table & loads
C     them into the JET buffer.  This feature is implemented  for
C     easy future expansion when more thrusters will be added.
C
C     Here,  JB(40)    is the jet buffer
C            TAB(K,2)  is the appropriate firing table
C            K         is the TAB index
```

```
C
C      ------------------------------------------------------------
C
       INTEGER * 2  TAB(K,2), JB(1)
       INTEGER      TIME
C
       TIME = TAB(K,2)
       INDEX = TAB(K,1)
       JB(INDEX) = TIME
C
       RETURN
       END
C
C
       SUBROUTINE FIRE
C
C      ------------------------------------------------------------
C
C      This procedure loads firing times from firing tables into JETBU
C      and then invokes the I/O driver LDCTR to fire  the  appropriate
C      thrusters.  NOTE:  LDCTR will only load the non-zero table ent-
C      ries.
C
C      ------------------------------------------------------------
C
       INTEGER * 2   FRTBLX(20,2), FRTBLY(20,2), JETBUF(40), IT, II
       COMMON /JETS/ NTHRX, NTHRY, FRTBLX, FRTBLY, JETBUF, SCLX, SCLY
C
C      ***  Find the larger of the two  ***
C
       NX = NTHRX * 4
       NY = NTHRY * 4
C
       CALL  LDBUF (NX, FRTBLX, JETBUF)
       CALL  LDBUF (NY, FRTBLY, JETBUF)
C
       CALL LDCTR(JETBUF,40)
C
       RETURN
       END
C
C
        SUBROUTINE  LDBUF (N, T, J)
C
C
C      ------------------------------------------------------------
C
C      This subroutine takes the contents of a firing table and per-
C      forms a "this is a good place for a stick up" and places the
C      corresponding firing times into JETBUF
C
C      ------------------------------------------------------------
C
C
       INTEGER * 2   T(20,2), J(40)
C
       DO 100 K = 1, N
           IT    = T(K,2)
           II    = T(K,1)
           J(II)= IT
100    CONTINUE
       RETURN
       END
C
C
```

```
      SUBROUTINE MOTORS
C
C     ------------------------------------------------------------
C
C     This procedure calculates the required DC motor rates, converts
C     them into DAC values & sends them out to the corresponding DAC.
C     An I/O driver is then called on to move the motors.
C
C     The logic depends on the command mode    ( Rate  or  positional
C     control).
C
C     It is explicitly assumed that:
C         A.   The DC motors are rate driven.  Therefore, the DAC out-
C              puts dictate the rate.
C         B.   Each DAC is 12 bit and is wired for bi-polar output.
C         C.   When position commands are used, a DC motor rate based
C              on a three-cycle period is used.  The choice of 3 is
C              arbitrary, and can be adjusted in the final testing.
C
C     -----------------------------------------------------------
C
      REAL            MASS, MAJOR, JZZ, LX, LY
      INTEGER         DOF, F
      INTEGER * 4      FLAG, CMDMOD, CMDRAW(9), CMDRET(9)
      INTEGER * 2     GYRBUF(18)
      INTEGER * 2     NAVBUF(3),MTRBUF(6),MTVBUF(6),SNRBUF(3),DACBUF(6
      COMMON /PHYS/   MASS, MAJOR, JZZ, PIRAD
      COMMON /DYNA/   THRUST, ACC(2), LX, LY, DOF
      COMMON /CMMD/   CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
      COMMON /POSN/   POSTN(9), OPOSTN(9)
      COMMON /RATE/   VLCTY(9), OLDVEL(9)
      COMMON /DACO/   DACRDG(6), DACCAL(6), DACCOF(6)
      COMMON /BUFF/   GYRBUF, NAVBUF, MTRBUF, MTVBUF, SNRBUF, DACBUF
C
C     ***  Whew !  ***
C
      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
C        ***  Convert to DAC count  ***
C
         R = Q * DACCAL(M) + DACCOF(M)
         IR = IFIX(R + 0.5)
         SR = FSIGN(R)
C
C        ***  Make sure there is no sudden change in direction  ***
C
         X = VLCTY(M3)

         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
C             IF (FSIGN(X) * SR .LT. 0) THEN
 200               IF (FSIGN(X) * SR .GE. 0) GOTO 300
C                  ***  There is sign reversal.  Better stop motor no
                   IR = 0
                   SR = 1.0
```

```
C             ENDIF
C        ENDIF
C
C        ***  Make sure DAC count is within limits  ***
C
 300     JR = IABS(IR)
         IF (JR .GT. 2047) JR = 2047
         RR = JR * SR
         IR = IFIX(RR + 0.5)
C
C        ***  This is a good place to stick up  ***
C
         DACRDG(M) = RR
         DACBUF(M) = IR
 100  CONTINUE
C
C     ***  Move the motors  ***
C
      CALL MTRDRV(DACBUF,KK)
C
      RETURN
      END
C
C
      SUBROUTINE THRSTR
C
C     --------------------------------------------------------------
C
C     This procedure handles thruster logic.
C
C     --------------------------------------------------------------
C
      REAL          FF(2), F(2), A(2), T(3)
      REAL          MASS,  MAJOR,  JZZ,  LX,  LY
C
      INTEGER       DOF
      INTEGER * 2   FRTBLX(20,2),  FRTBLY(20,2), JETBUF(40)
      INTEGER * 4   FLAG,  CMDMOD,  CMDRAW(9), CMDRET(9)
      COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD
      COMMON /DYNA/ THRUST, ACC(2), LX, LY, DOF
      COMMON /RATE/ VLCTY(9), OLDVEL(9)
      COMMON /POSN/ POSTN(9), OPOSTN(9)
      COMMON /JETS/ NTHRX, NTHRY, FRTBLX, FRTBLY, JETBUF, SCLX, SCLY
      COMMON /CMMD/ CMDRET, CMDRAW, CMDVAL(9), CMDMOD, FLAG
C
C     transform acceleration vector ACC to floor coordinates
C
      THETA = POSTN(1)
      CALL  BTF (ACC, THETA, A)
C
C
C     ***  calculate required impulses.  This is mode dependent  ***
C
C     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
          TORQ  = JZZ  * (CMDVAL(1) - VLCTY(1)) / 2
          GO TO 120
C     ELSE
100       CONTINUE
          DO 150 K = 1, 3
             V  = VLCTY(K)
             P  = POSTN(K)
             C  = CMDVAL(K)
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
             IF (K .GT. 1) GOTO 130
                AX = 2 * THRUST * LX / JZZ
                AA = AX
C               IF ((C-P) .LT. 0.0) AA = -AX
                GO TO 135
C            ELSE
130             AA = A(K-1)
C            END IF
135          CONTINUE
C            WRITE (*,10) V, P, C, AA
10           FORMAT (' ', 4E15.8)
             T(K) = G(V, P, C, AA)
150       CONTINUE
          T1    = T(2)
          T2    = T(3)
          TQ    = 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
120   CONTINUE
C
C     ***  Transform force from floor coordinates to TOM_B coords  ***
C
      CALL  FTB (FF, THETA, F)
      FX = F(1)
      FY = F(2)
C
C     ***  Use control laws to calculate force along X & Y directions
C          of TOM_B  ***
C
      CALL CTRLLW(TORQ, FX,FY,FX1,FX2,FY1,FY2)
C
C     ***  Convert to firing times and put into firing tables  ***
C
      CALL  ITABLE
      CALL  TABLE(FX1,FX2,NTHRX,FRTBLX,SCLX, 2)
      CALL  TABLE(FY1,FY2,NTHRY,FRTBLY,SCLY, 3)
C
C     ***  Fire them thrusters  ***
C
      CALL FIRE
C
      RETURN
      END
C
C
      SUBROUTINE CTRLLW(TORQ,FX,FY,FX1,FX2,FY1,FY2)
C
C     -------------------------------------------------------------
C
C     This procedure calculates FX1, FX2 from FX & FY1, FY2 from FY

C     & TORQ.
C     It also checks that each FX1, FX2, FY1, FY2  does not exceed
C     the maximum developed thrust on TOM_B.
C
C     -------------------------------------------------------------
C
      REAL          LX, LY
      INTEGER       DOF
      INTEGER * 2   FRTBLX(20,2), FRTBLY(20,2), JETBUF(40)
      COMMON /DYNA/ THRUST, ACC(2), LX, LY, DOF
      COMMON /JETS/ NTHRX, NTHRY, FRTBLX, FRTBLY, JETBUF, SCLX, SCLY
```

```
C
      S  =  1.0
C     IF (FX .LE. FY)  THEN
          IF (FX .GT. FY) GOTO 100
          FY1 = FY / 2.0 + TORQ / (2.0 * LY)
          FY2 = FY - FY1
          CALL CHECK(FY1,FY2,NTHRY,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,NTHRX,THRUST, TORQ)
          IF ((FX1.LT.0.0).AND.(FX2.LT.0.0)) CALL SWAP(FX1,FX2,S)
          GOTO 900
C     ELSE
 100      FX1 = FX / 2.0 + TORQ / (2 * LX)
          FX2 = FX - FX1
          CALL CHECK(FX1,FX2,NTHRX,THRUST, TORQ)
          IF ((FX1.LT.0.0).AND.(FX2.LT.0.0)) CALL SWAP(FX1,FX2,S)
          DTQ = TORQ + S*(FX2 - FX1) * LX
          FY1 = FY / 2.0 + DTQ / (2.0 * LY)
          FY2 = FY - FY1
          CALL CHECK(FY1,FY2,NTHRY,THRUST, TORQ)
          IF ((FY1.LT.0.0) .AND.(FY2.LT.0.0)) CALL SWAP(FY1,FY2,S)
C     ENDIF
C
 900  RETURN
      END
C
C
      SUBROUTINE  SWAP (X,Y,S)
C
C     -------------------------------------------------------------
C
C     This subroutine exchanges X and Y
C
C     -------------------------------------------------------------
C
C
      REAL   T
C
      S = -1.0
      T = X
      X = Y
      Y = T
      RETURN
      END
C
C
      SUBROUTINE  CHECK (F1, F2, NTHR, THRUST, TORQ)
C
C     ------------------------------------------------------------
C
C     This procedure ensures that the thrust required does not ex-

C     ceed the maximum thrust that TOM_B can deliver.
C
C     ------------------------------------------------------------
C
C
      REAL           LIMIT
      REAL           MASS, MAJOR, JZZ, PIRAD
      INTEGER        FG
      COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD
C
      FM = NTHR * THRUST * MAJOR
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
      S1 = FSIGN(F1)
      S2 = FSIGN(F2)
      SQ = FSIGN(TORQ)
      F1 = ABS  (F1)
      F2 = ABS  (F2)
      FQ = ABS(TORQ)
      IF (FQ .GT. 0.0001) GOTO 92
          FQ = 0.0
          SQ = 1
C     END IF
92    CONTINUE
C
      FG = 1
      IF (F2 .GT. F1) FG = 2
C
      IF (S1 * S2) 100, 200, 200
100      CONTINUE
C        F1 & F1 are antiparallel
         IF (F1 .GT. FM) F1 = FM
         IF (F2 .GT. FM) F2 = FM
         GO TO 800
C     ELSE
C        F1 & F2 are parallel
200      CONTINUE
         DF = ABS (F1 - F2)
         IF ((DF.GT.0.0001) .OR. (F1 .GT. 0.0001)) GOTO 207
            F1 = 0.0
            F2 = 0.0
            GOTO 800
207      CONTINUE
         BG = AMAX1 (F1,F2)
         IF (BG .GT. FM) BG = FM
         IF (DF .GT. FM) DF = FM
         CR = BG - DF
         IF (CR .LT. 0.0) CR = 0.0
         IF (FG - 1) 210, 210, 220
C           F1 >= F2
210         CONTINUE
            F1 = BG
            F2 = CR
            GO TO 700
C        ELSE
C           F1 < F2
220         CONTINUE
            F1 = CR
            F2 = BG
C        END IF
C     END IF
700   CONTINUE
800   CONTINUE
      F1 = S1 * F1
      F2 = S2 * F2
      RETURN
      END

C
C
          REAL FUNCTION G (V0, X0, CMDX, AC)
C
C     ------------------------------------------------------------
C
C     This procedure calculates the optimum firing time for
C     thrusters in a direction when position control is used.
C
C     A distinction is made between a firing time <= 1 major cycle
C     and that  > 1 major cycle.
```

```
C
C      If a firing time < 1/20 of a major cycle, (5 MS)  it is set
C      to zero.
C
C      NOTE :  all dynamic variables are in floor coordinates !!!
C              and time is expressed in seconds.
C
C      ----------------------------------------------------------
C
          REAL          MASS, MAJOR, JZZ, PIRAD
          COMMON /PHYS/ MASS, MAJOR, JZZ, PIRAD
C
C
      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
C     ELSE
31       XX0 = X0
         CMD = CMDX
         GOTO 38
32    CONTINUE
          IF (DX .GE. 0.0) GOTO 33
          XX0 = X0
          CMD = CMDX
          GOTO 38
33    CONTINUE
          XX0 = CMDX
          CMD = X0
C     END IF
38    CONTINUE
C
      D  = ABS (DX)
      V  = ABS (V0)
      A  = ABS (AC)
C
      IF (SD * SV .GT. 0.0) GO TO 50
C
C         DX and V0 are anti-parallel
C
          T1 = V / A
          RA = T1 * T1 + 2 * D / A
          T2 = SQRT (RA)
          G  = SD * (T1 + T2)
          RETURN
C
C         DX and V0 are parallel
C
50        CONTINUE
          T  = MAJOR

          X  = ABS (XX0)
          X1 = ABS (XX0) + V * T
          X2 = X1 + A * T * T / 2.0
          XC = ABS (CMD)
C
C         DO  CASE
C         1:  XC <= X1
              IF (XC .GT. X1) GO TO 200
              RA = T * T - 2.0 * (X1 - XC) / A
              IF (RA .LT. 0.0) GO TO 250
                 G = -SD * (T - SQRT(RA))
```

```
                     RETURN
C                ELSE
250                  RA = V * V - 2 * A * (XC - X)
                     G  = -SD * (V + SQRT(RA))
                     RETURN
C                END IF
C
C           2:   X2 >= XC > X1
200              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
C                END IF
C
C           3:   XC > X2
300              CONTINUE
                 TF = (SQRT(V * V + 2.0 * A * D) - V) / A
                 G  = SD * TF
                 RETURN
C
C           END CASE
C
            END
C
$
```