REPRO COPY

RF Project 762012/712655
Technical Report

the
ohio
state
university

research foundation
1314 kinnear road
columbus, ohio
43212

COMPUTER COORDINATION OF LIMB MOTION FOR A THREE-LEGGED WALKING ROBOT

Charles A. Klein and Mark R. Patterson
Department of Electrical Engineering

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Langley Research Center
Hampton, Virginia 23665

Grant No. NAG1-30

September, 1980

ACKNOWLEDGMENTS

The authors would like to acknowledge the help provided by Charles A. Csuri and other members of the Computer Graphics Research Group, the Ohio State University, Columbus, Ohio; in providing computer time and facilities that made this work possible.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

TABLE OF CONTENTS (Cont'd.)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (cont'd.)

Chapter 1
INTRODUCTION

1.1. General Background.

It is obvious that in nature animals use quite a different method of locomotion over land than do vehicles designed by man. Animals, including man, use independent limbs to walk over solid surfaces, but man-made vehicles almost exclusively use wheels. Each of these approaches has its advantages. Wheeled vehicles on hard surfaces allow very fast and efficient movement of heavy loads. Legs, on the other hand, usually move more slowly but do not require hard, smooth surfaces.

Although wheeled vehicles have been in use for centuries, only relatively recently has much research been conducted on the use of legs for man-made vehicles. This is due to the fact that the coordination of limb motions to produce locomotion of a legged vehicle is a very complex process. Advances in the speed and capabilities of digital computers have finally allowed this burden to be removed from a human controller, making artificial legged locomotion a practicable goal. Vehicles such as the Mars Rover [1] which use legged vehicle concepts are now being considered for use in terrain too rough for wheeled vehicles.

1.2. Objectives of this Work.

It has also been observed that multi-armed vehicles with grasping hands could walk over structures even in the absence of gravity [2]. Since serious consideration is being given to the construction of large structures in space [3], the problem of coordinating the limb motion of a vehicle which could perform assembly and maintenance operations on those structures is of practical as well as theoretical interest. That problem is the subject of this work.

1.3. Organization.

The remainder of this work is divided into five parts, Chapters 2 through 6. Chapter 2 reviews earlier research which is relevant to this work. This includes the previous work on manipulator kinematics and that on walking robots.

Chapter 3 is a statement of the problem which this work addresses. The robot itself is described and some introductory material is presented on the coordinate systems used. The basic control scheme which is implemented is also discussed in this chapter.

Chapter 4 addresses the problem of the control of the individual arms. Arm velocities are generally described in Cartesian coordinates, so the first section of the chapter discusses the conversion from Cartesian velocities to joint velocities using the Jacobian matrix. The second section of the chapter describes the calculation of a trajectory for an arm given a sequence of points through which it is to pass.

Chapter 5 is a discussion of the free gait algorithm which controls the lifting and placing of legs for the robot. The first section describes the generation of commanded velocities for the robot, and the second section discusses the implementation of those velocities by the algorithm.

Chapter 6 summarizes the results of the simulation. It also gives suggestions for further work in the area of robot legged locomotion.

Chapter 2
SURVEY OF PREVIOUS WORK

2.1. Introduction.

The coordination of the limbs of a walking robot which uses manipulators as its legs (arms) can be decomposed into two tasks. The first task is the control of the individual manipulators' movements, and the second is the coordination of the manipulators to produce walking. Previous work in both these areas is reviewed in this chapter.

2.2. Previous Work on Manipulator Kinematics.

2.2.1. Cartesian-Coordinate-to-Joint-Coordinate Conversion

In the control of a manipulator, it is desirable to be able to give commands to the manipulator in Cartesian coordinates, since that is the system in which humans are most accustomed to working. The manipulator's motors operate in joint coordinates, though, so a method must be found for converting from one system to the other. There are two basic approaches to this problem: physical and mathematical.

The physical approach uses the manipulator itself or a replica of it to record motion directly in joint coordinates.

A copy of the manipulator is used in master-slave control, in which the replic[illegible] often smaller, of the arm is moved by a human operator. The replica has potentiometers on the joints whose values are used to obtain commands for the manipulator's motors. Another physical approach, sometimes referred to as "training," involves moving the manipulator itself through its desired trajectory, measuring and recording intermediate positions along the way in joint coordinates. These data, along with timing information, constitute a command sequence. This method is often applied when a manipulator is used in a situation such as an assembly line in which the same sequence of motions is repeated over and over [4].

Neither of the above physical approaches is satisfactory for legged locomotion. Master-slave control does not work well because a human being cannot effectively control the number of legs required for locomotion except at very slow speeds [5]. Training is not applicable since the trajectory which a leg must follow will vary as the terrain and/or the operator's commands change.

Thus for legged locomotion a mathematical approach is needed to convert the commands from Cartesian coordinates to joint coordinates. Although this could be done for commands given as positions and orientations, most manipulators have several sets of joint coordinates which will result in the same end effector position and orientation [6]. A better approach is that of Resolved Motion Rate Control [7], in which velocity commands are used. For instance, with a six-degree-of freedom arm, if $\underline{\dot{\theta}}$ is a vector consisting of all six joint velocities and $\underline{\dot{x}}$ is a vector consisting of the translational velocities along and the rotational velocities about the x, y, and z axes,

$$\underline{\dot{x}} = \underline{J}\,\underline{\dot{\theta}} \quad , \tag{2-1}$$

where $\underline{J}$ is the Jacobian matrix of derivatives of the Cartesian coordinate values ($\underline{x}$) with respect to the joint angles ($\underline{\theta}$):

$$J_{ij} = \frac{\partial x_i}{\partial \theta_j} \quad . \tag{2-2}$$

Joint velocities can then be found from the rectilinear velocities using the inverse Jacobian matrix:

$$\underline{\dot{\theta}} = \underline{J}^{-1}\,\underline{\dot{x}} \quad . \tag{2-3}$$

The advantage of using velocities rather than positions is that for a six-degree-of-freedom arm, there is one and only one set of joint velocities that will produce a given translational and rotational velocity of the end effector, except in those configurations in which the Jacobian matrix is singular. At a singularity of the Jacobian, velocity in at least one direction is impossible, and velocities in other directions may be obtainable with an infinite number of combinations of joint velocities. Some work has been done on the general problem of finding which configurations will result in Jacobian singularities for a particular arm [8]. For arms with more than six degrees of freedom, there are generally an infinite number of ways to attain a given velocity, and the extra degrees of freedom can be used to optimize some criterion (e.g., to minimize the sum of the joint velocities) [9, 10].

Another possible approach to the problem of determining the joint velocities is that of full or partial table look-up. Although in this work only kinematics are considered, in an actual implementation the torques required to produce a set of joint velocities would also have to be determined, and it is possible that the two problems could be combined. If that could be done, one of the published methods for torque solution by table look-up could be used. Possible approaches would be the full table look-up of Albus's "Cerebellar Model Articulation Controller," or CMAC [11], or the partial look-ups suggested by Horn and Raibert [12, 13].

2.2.2. Homogeneous Transformation Matrices.

It will be seen later that it is desirable to be able to express the position and orientation of one part of the robot with respect to another. A compact way of doing this which will be used in this work is the homogeneous transformation matrix, which was introduced by Denavit and Hartenberg [14] and has been applied more specifically to manipulators by Pieper [15] and Paul [16].

A homogeneous transformation matrix is a 4x4 matrix of the form shown in Equation (2-4).

$$\underline{A} = \begin{bmatrix} & [\underline{R}] & & [\underline{p}] \\ 0 & 0 & 0 & 1 \end{bmatrix} . \qquad (2\text{-}4)$$

It describes the position and orientation of one coordinate system (say ($\underline{\hat{x}}'$, $\underline{\hat{y}}'$, $\underline{\hat{z}}'$)) with respect to another ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$). Orientation information is carried in the upper left 3x3 submatrix, $\underline{R}$, which is a rotation matrix whose columns are the unit vectors $\underline{\hat{x}}'$, $\underline{\hat{y}}'$, and $\underline{\hat{z}}'$ expressed in ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$) coordinates. Position information is in the upper right 3x1 submatrix, $\underline{p}$, which is a vector that gives the position of the origin of ($\underline{\hat{x}}'$, $\underline{\hat{y}}'$, $\underline{\hat{z}}'$) in ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$) coordinates. The last row consists of three zeros and a one.

As mentioned above, the 3x3 rotation submatrix of the homogeneous transformation matrix contains relative orientation information for coordinate systems ($\underline{\hat{x}}'$, $\underline{\hat{y}}'$, $\underline{\hat{z}}'$) and ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$). Another method of expressing the orientation relationship between two coordinate systems which will also be used in this work is that of the three Euler angles α, β, and γ shown in Figure 1. The transformation of system ($\underline{\hat{x}}'$, $\underline{\hat{y}}'$, $\underline{\hat{z}}'$) into system ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$) is accomplished by a rotation of $-\gamma$ about $\underline{\hat{z}}'$,

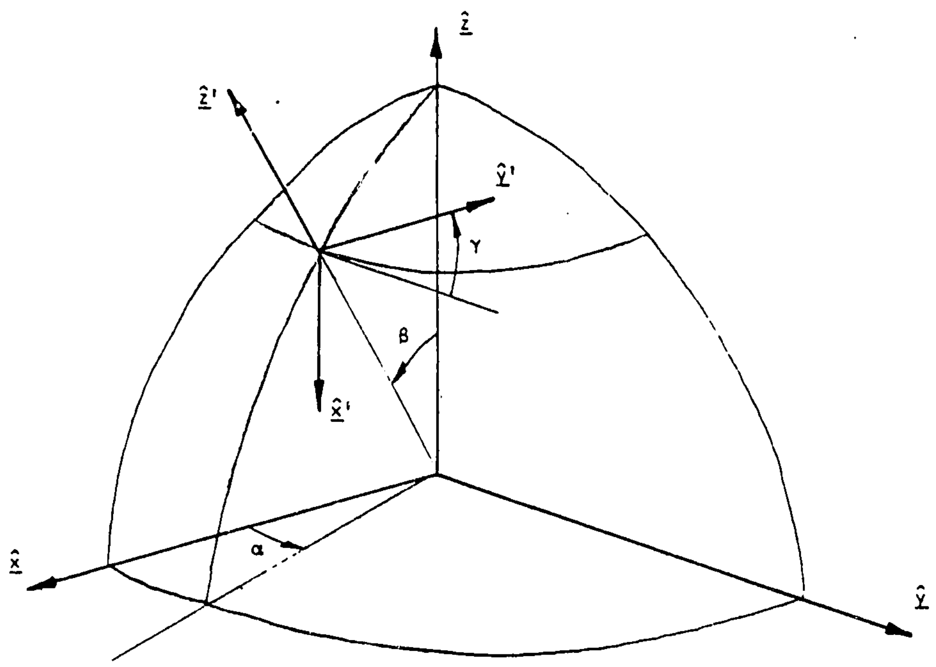

Figure 1. Definition of Euler Angles α, β, γ Which Specify Orientation of Coordinate System ($\underline{\hat{x}}'$, $\underline{\hat{y}}'$, $\underline{\hat{z}}'$) with respect to Coordinate System ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$)

a rotation of $-\beta$ about $\underline{\hat{y}}'$, and a rotation of $-\alpha$ about $\underline{\hat{z}}'$, in that order. The rotation matrix $\underline{R}$ can then be expressed in terms of these Euler angles as [6]

$$\underline{R} = \begin{bmatrix} \cos\alpha\cos\beta\cos\gamma - \sin\alpha\sin\gamma & -\cos\alpha\cos\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta \\ \sin\alpha\cos\beta\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\cos\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta \\ -\sin\beta\cos\gamma & \sin\beta\sin\gamma & \cos\beta \end{bmatrix} \quad (2\text{-}5)$$

Then, if $\underline{r}'$ is a direction vector in system ($\underline{\hat{x}}'$, $\underline{\hat{y}}'$, $\underline{\hat{z}}'$), the same direction vector $\underline{r}$ in system ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$) is

$$\underline{r} = \underline{R}\,\underline{r}' \quad . \qquad (2\text{-}6)$$

If the position of the origin of $(\underline{\hat{x}}', \underline{\hat{y}}', \underline{\hat{z}}')$ in coordinate system $(\underline{\hat{x}}, \underline{\hat{y}}, \underline{\hat{z}})$ is $\underline{p}'$, the homogeneous transformation matrix $\underline{A}$ describing the position and orientation of system $(\underline{\hat{x}}', \underline{\hat{y}}', \underline{\hat{z}}')$ with respect to system $(\underline{\hat{x}}, \underline{\hat{y}}, \underline{\hat{z}})$ is then

$$\underline{A} = \begin{bmatrix} & & & p'_x \\ & [\underline{R}] & & p'_y \\ & & & p'_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \qquad (2\text{-}7)$$

Then if (r'_x, r'_y, r'_z) are the coordinates of a point in system $(\underline{\hat{x}}', \underline{\hat{y}}', \underline{\hat{z}}')$ and (r_x, r_y, r_z) are the coordinates of that point in system $(\underline{\hat{x}}, \underline{\hat{y}}, \underline{\hat{z}})$

$$\begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = \underline{A} \begin{bmatrix} r'_x \\ r'_y \\ r'_z \\ 1 \end{bmatrix} . \qquad (2\text{-}8)$$

2.3. Previous Work on Walking Robots.

In the past two decades, research has begun on walking robots in hope that their advantages can eventually be used for efficient transportation over terrain unsuitable for wheeled vehicles. A summary of this research has been presented by McGhee [17], and the next section presents those portions of that summary which are relevant to the present work. The final section of this chapter discusses the problems of gait selection and foot placement for a walking robot.

2.3.1. Legged Vehicle Control.

Three different approaches have been considered for the control of legged locomotion systems. These approaches are finite-state control, model reference control, and algorithmic control, each of which is discussed in the following paragraphs.

In 1961, Tomovic [18] noted that some modes of locomotion could be characterized by finite-state models. He observed that locomotion could be characterized by representing each leg as a two-state device (on the ground or in the air). This concept was extended when Tomovic and McGhee [19] pointed out that at least four actuator states are needed to obtain a more or less natural gait. The validity of the finite-state control concept was confirmed in 1968 when a small quadruped controlled by 16 flip-flops attained stable locomotion at the University of Southern California [20].

A second approach to control of legged systems is "model reference control [21]." This approach uses a real-time simulation of vehicle kinematics for the generation of the joint angles and rates needed for ideal locomotion. These ideal values are then used as commands to the actuators of the vehicle. This method has the advantage of being much more flexible than finite-state control, since the actuator control signals are infinitely variable instead of having a finite number of states, but the problem of deciding when the vehicle should lift its legs and when and where it should place them must be solved before it can be used. This problem is discussed in the next section.

A variant of model reference control has been suggested by Vukobratovic [22]. With this approach, called algorithmic control, a nominal trajectory is obtained from a kinematic model as with model reference control. With algorithmic control, though, not all joint angles and rates are derived from the model. Instead, the model imposes certain overall constraints on the system to lower the order of its equations. These lower-order equations are then used with artificial sensor feedback to produce stable motion. To date, this technique has been applied only to biped locomotion.

2.3.2. Gait Selection and Foot Placement.

A considerable amount of past work has been devoted to the leg sequencing problem for the special case of straight-line, constant-speed locomotion over level terrain. A periodic solution to this problem has usually been referred to as a "gait." Combinatorial studies show that there are a large number of gaits possible (40,000,000 for a six-legged vehicle [23]), so optimization is required. In all but one study [24], the optimization criterion used has been one related to the degree of stability of the system under consideration. In most of the studies the stability criterion used has been the longitudinal stability margin, the distance the vehicle's center of gravity would have to be moved forward or backward before the vehicle toppled over. For vehicles whose legs are evenly spaced in right-left pairs along a longitudinal motion axis, the gaits which maximize the longitudinal stability margin are known [25]. Those gaits are known as wave gaits, all of which are characterized by a forward wave of stepping actions on each side of the body with a half-cycle phase shift between the members of each right-left pair of legs.

More recently, Kugushev and Jaroshevskij [24] have suggested that the approach used in the study of gaits for straight-line locomotion can be extended to include a more general case in which nonperiodic leg sequences known as "free gaits" may be expected. Specifically, they present a partial problem formulation in which a trajectory is specified in advance for the center of gravity of a legged system over a given terrain containing certain regions which are unsuitable for support. The remainder of the paper is devoted to a completion of the formalization of this problem and a description of one heuristic algorithm for its solution.

McGhee and Iswandhi [26] have presented a more complete formalization of the free gait problem. Central to their approach, which is basically similar to that of Kugushev and Jaroshevskij, is the concept of kinematic margin. The kinematic margin for a particular leg and foothold is the distance for which the vehicle can continue along its present path before that leg must be lifted from that foothold. McGhee and Iswandhi's algorithm proceeds by lifting and placing legs so as to maximize the minimum value of kinematic margin over all supporting legs. This approach maximizes the distance the vehicle can proceed forward in its current configuration, thereby increasing the likelihood that it will be able to find a new configuration with which to continue.

2.4. Summary.

This chapter has summarized some of the research in the general areas of manipulator kinematics and walking robots. Chapter 3 will present the specific problem to be addressed in this work, and Chapters 4 and 5 will discuss the solution of that problem.

Chapter 3
PROBLEM FORMULATION

3.1. Introduction.

As was mentioned in the first chapter, there are situations in which a mobile robot with six-degree-of-freedom arms capable of grasping handholds would be very useful. It is the purpose of this work to describe an algorithm for controlling the motion of such a vehicle. In this chapter, the design of the robot which was used in the simulation is presented, and the overall control philosophy is explained.

The first section of the chapter is divided into two parts: The first describes the robot's body and the second its arms. The second section of the chapter presents the control scheme that is to be implemented. The basic approach chosen is that of supervisory control [27], in which a human operator provides only high-level commands, such as robot velocity or destination. A control computer then automatically solves the limb coordination problem to move the vehicle in response to the operator's commands.

3.2. Robot Description.

The robot investigated here consists of a body and three six-degree-of-freedom arms [28]. Descriptions of the body and arms are given in the following paragraphs.

3.2.1. Body Model.

Figure 2 is a drawing of the three-armed vehicle. The body is modelled as an equilateral triangle with an arm attached to each corner. The next few paragraphs discuss this model and the coordinate systems used to describe it.

For calculations of "absolute" positions and velocities, both of the robot and of its arms, a coordinate system which is fixed relative to the surface on which the robot walks is required. For this purpose a right-handed coordinate system ($\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$) (circumflexes will indicate unit vectors) is defined. Since this system is fixed relative to the surface on which the robot is walking, its definition depends on the robot's intended use. If the robot is to be used on the earth, the system would be an earth-fixed coordinate system; if it is to be used in space, it would be a system fixed to the structure on which it is walking. In this work, the vehicle is assumed to be operating on a cylinder in space, so the coordinate system is chosen to be fixed relative to that cylinder (see Figure 2).

Also shown in Figure 2 is a coordinate system, ($\underline{\hat{x}}_v$, $\underline{\hat{y}}_v$, $\underline{\hat{z}}_v$), which is fixed to the moving vehicle. Throughout

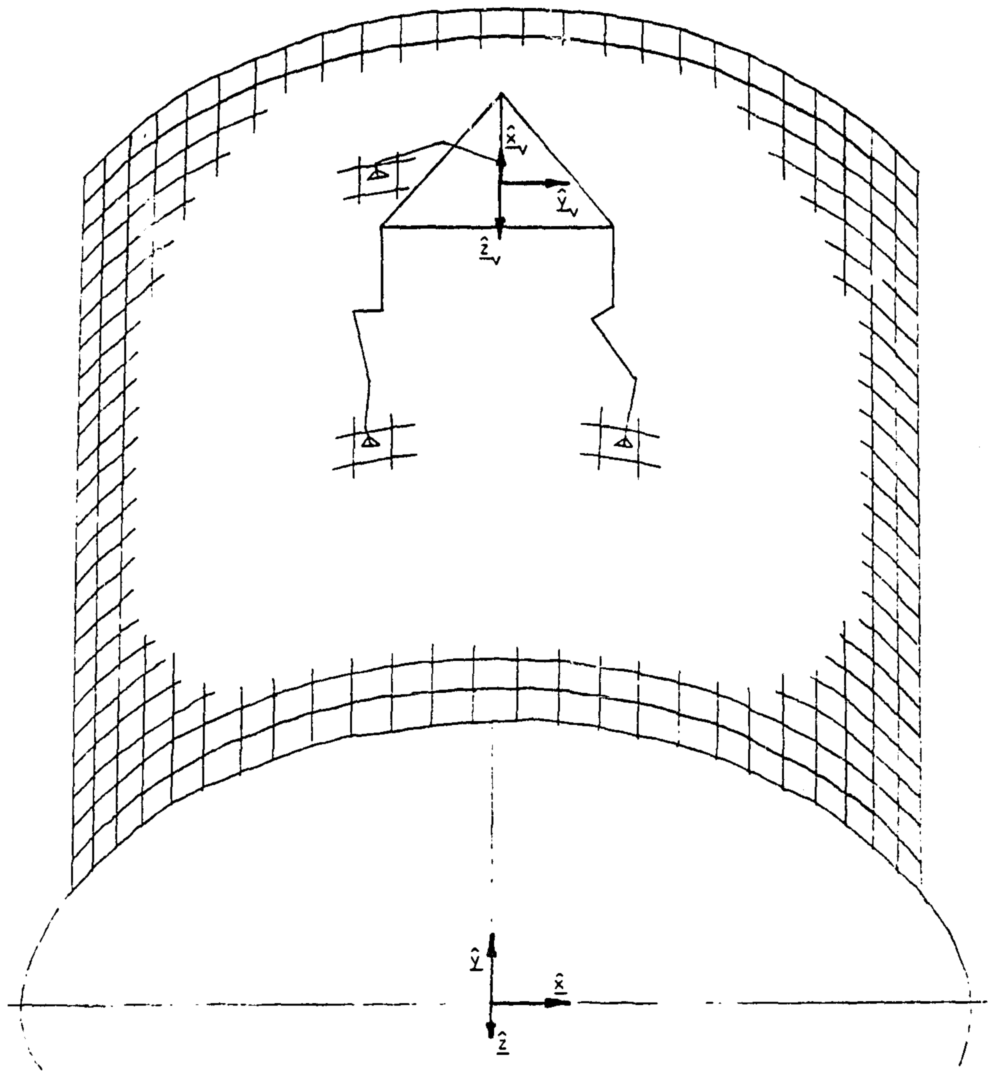

Figure 2. Definitions of Terrain-Fixed and Vehicle-Fixed Coordinate Systems

this work, subscripts will be used to indicate the coordinate system to which a unit vector belongs; the absence of a subscript will indicate the fixed coordinate system. Thus $\underline{\hat{x}}_v$ is the unit vector in the x direction for the vehicle coordinate system, and $\underline{\hat{x}}$ belongs to the fixed system.

Vectors will be used to describe the position or velocity of the robot's body or one of its arms with respect to the cylinder, the body, or another arm. There will be a coordinate system fixed to each part of the robot (including the body and each link of each arm) as well as the cylinder, so these vectors can more conveniently be thought of as describing the position or velocity of one coordinate system with respect to another. Subscripts will be used to indicate the system whose location or motion is being referred to, and superscripts will denote the coordinate system to which that location or motion is referred (again, absence of a superscript will indicate the fixed system). Thus the position and linear velocity of the origin of the vehicle coordinate system with respect to the fixed system will be denoted by $\underline{p}_v$ and $\underline{v}_v$, respectively. The angular velocity of the vehicle system with respect to the fixed system will be denoted by a three-element vector $\underline{\omega}_v$, whose elements are the angular rotation rates about $\underline{\hat{x}}$, $\underline{\hat{y}}$, and $\underline{\hat{z}}$. The orientation of the vehicle system with respect to the fixed system can most concisely be described by three Euler angles, α_v, β_v, and γ_v. There is no standard convention for choosing these angles; the ones used in this work are those described by Horn and Inoue [6] and shown in Figure 1 (in Chapter 2).

A vector in vehicle coordinates can be transformed into fixed coordinates by multiplying by a 3x3 rotation matrix, $\underline{R}_v$ (in this work, matrices will be denoted by underlined, upper-case letters); that is,

$$\underline{r} = \underline{R}_v \underline{r}^v \tag{3-1}$$

where $\underline{R}_v$ is of the form given in Equation (2-4). The coordinates of a point in vehicle coordinates can also be transformed to fixed coordinates by multiplying them by a matrix, in this case a 4x4 homogeneous transformation matrix, $\underline{A}_v$; thus

$$\begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = \begin{bmatrix} & [\underline{R}_v] & & p_{v_x} \\ & & & p_{v_y} \\ & & & p_{v_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_x^v \\ r_y^v \\ r_z^v \\ 1 \end{bmatrix} . \tag{3-2}$$

To simplify the equations in this thesis, the above equation will be written

$$\underline{r} = \underline{A}_v \underline{r}^v \quad . \tag{3-3}$$

If the matrix in the equation is a homogeneous transformation matrix, the vector $\underline{r}$ is to be understood as a four-element vector $(r_x, r_y, r_z, 1)^T$; if the matrix is a rotation matrix, as in Equation (3-1), $\underline{r}$ is the three-element vector $(r_x, r_y, r_z)^T$.

3.2.2. Arm Model.

A drawing of one of the robot's arms is shown in Figure 3. The configuration and dimensions of the arm are those of a six-degree-of-freedom industrial manipulator, the PUMA 600, made by Unimation, Inc., Danbury, Connecticut. The arm has six rotational joints, one about each of the z-axes except $\underline{\hat{z}}_6$ (numerical subscripts indicate references to link coordinate systems). The angular displacement of joint i will be denoted by θ_i; θ_i is zero when joint i is at the center of its travel and positive when the displacement is in a positive sense with respect to $\underline{\hat{z}}_{i-1}$. In Figure 3 each of the joints is pictured at the center of its travel; the rotational limits of the joints are given in Table I.

TABLE I
PUMA 600 Manipulator Joint Limits

Joint	Limits
1	±160°
2	±165°
3	±135°
4	±135°
5	±105°
6	±270°

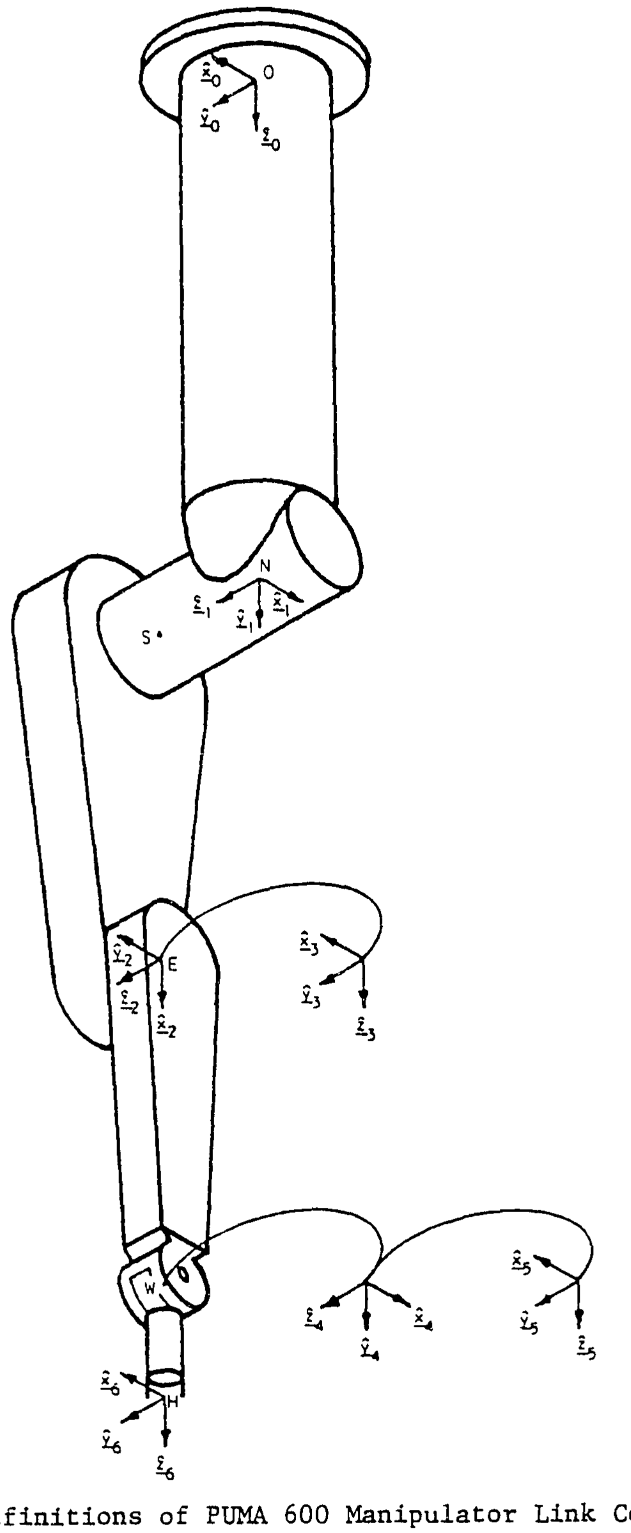

Figure 3. Definitions of PUMA 600 Manipulator Link Coordinate Systems and Designations of Points on Manipulator: O = base, N = "neck," S = "shoulder," E = "elbow," W = "wrist," H = "hand"

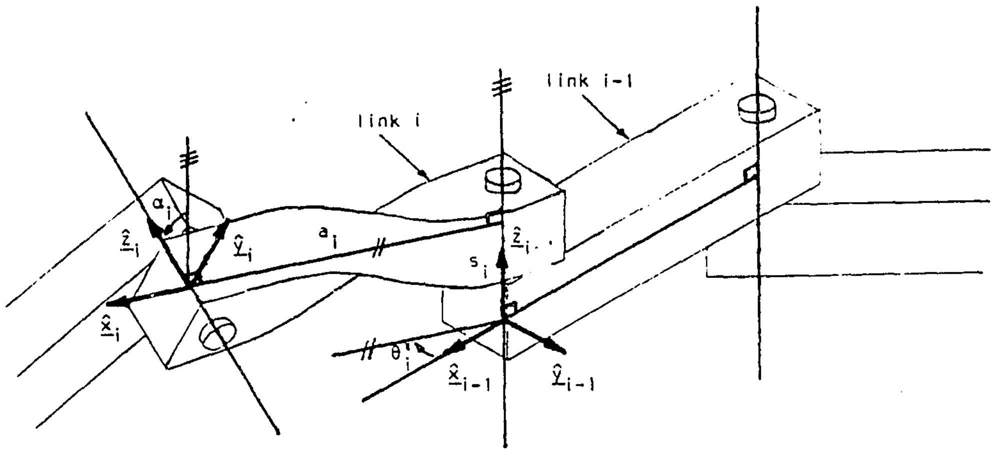

Figure 4. Definitions of General Link Coordinate Systems and Parameters Describing Relationships between Coordinate Systems

The arm is made up of seven links (including the base), with each two successive links connected by a rotary joint. Associated with each link is a coordinate system which is fixed with respect to that link (see Figure 4). For link coordinate system i, $\underline{\hat{z}}_i$ is directed along the axis of the joint between link i and link i+1, $\underline{\hat{x}}_i$ is along the common normal between the two joint axes associated with link i in the direction from $\underline{\hat{z}}_{i-1}$ toward $\underline{\hat{z}}_i$, and $\underline{\hat{y}}_i$ is chosen to complete the right-handed set [15].

Link coordinate system i can be transformed into link system i-1 by performing a rotation, two translations, and a final rotation as follows (refer to Figure 4):

1. A rotation of $-\alpha_i$ about $\underline{\hat{x}}_i$ to make $\underline{\hat{z}}_i$ parallel to $\underline{\hat{z}}_{i-1}$.

2. A translation of $-a_i$ along $\underline{\hat{x}}_i$ to locate the origin at the point where the common normal between $\underline{\hat{z}}_{i-1}$ and $\underline{\hat{z}}_i$ intersects $\underline{z}_{i-1}$.

3. A translation of $-s_i$ along $\underline{\hat{z}}_i$ to bring the origins into coincidence.

4. A rotation of $-\theta_i'$ about $\underline{\hat{z}}_i$ to bring $\underline{\hat{x}}_i$ into coincidence with $\underline{\hat{x}}_{i-1}$. ($\theta_i'$ may differ from θ_i since this method of describing θ_i' does not necessarily result in its being equal to zero at the center of joint i's travel, as θ_i was defined to be.)

Each of the above four operations can be described by a 4x4 transformation matrix, and the product of those four matrices is the homogeneous transformation matrix which describes the overall transformation from coordinate system i to system i-1; that is,

$$\underline{A}_i^{i-1} = \left[\text{rot}_{\underline{\hat{z}}_i,-\theta_i'}\right]\left[\text{trans}_{\underline{\hat{z}}_i,-s_i}\right]\left[\text{trans}_{\underline{\hat{x}}_i,-a_i}\right]\left[\text{rot}_{\underline{\hat{x}}_i,-\alpha_i}\right] \tag{3-4}$$

where $\text{rot}_{\underline{r},\theta}$ is a rotation of θ about $\underline{r}$

$\text{trans}_{\underline{r},s}$ is a translation of s along $\underline{r}$.

If the above multiplications are carried out, $\underline{A}_i^{i-1}$ can also be expressed in a more useful form as a function of the four joint parameters as [14]

$$\underline{A}_i^{i-1} = \begin{bmatrix} \cos\theta_i' & -\cos\alpha_i\sin\theta_i' & \sin\alpha_i\sin\theta_i' & a_i\cos\theta_i' \\ \sin\theta_i' & \cos\alpha_i\cos\theta_i' & -\sin\alpha_i\cos\theta_i' & a_i\sin\theta_i' \\ 0 & \sin\alpha_i & \cos\alpha_i & s_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}5)$$

TABLE II

Homogeneous Transformation Matrix Parameters for PUMA 600 Manipulator Joints

Joint, i	α_i	a_i	s_i	θ_i'
1	90°	0	NO	$\theta_i + 180°$
2	0°	ES	SN	$\theta_2 + 90°$
3	90°	0	0	$\theta_3 + 90°$
4	90°	0	WE	$\theta_4 + 180°$
5	90°	0	0	$\theta_5 + 180°$
6	0°	0	HW	θ_6

NO = neck-to-base distance = 26 in.

SN = shoulder-to-neck distance = 6 in.

ES = elbow-shoulder distance = 17 in.

WE = wrist-to-elbow distance = 17 in.

HW = hand-to-wrist distance = 6 in.

The four parameters for each joint which determine the transformation matrices between link systems are given in Table II. When those parameters are introduced into Equation

(3-5), the following six transformation matrices result:

$$\underline{A}_1^0 = \begin{bmatrix} -\cos\theta_1 & 0 & -\sin\theta_1 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 & 0 \\ 0 & 1 & 0 & NO \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}6)$$

$$\underline{A}_2^1 = \begin{bmatrix} -\sin\theta_2 & -\cos\theta_2 & 0 & -ES\sin\theta_2 \\ \cos\theta_2 & -\sin\theta_2 & 0 & ES\cos\theta_2 \\ 0 & 0 & 1 & SN \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}7)$$

$$\underline{A}_3^2 = \begin{bmatrix} -\sin\theta_3 & 0 & \cos\theta_3 & 0 \\ \cos\theta_3 & 0 & \sin\theta_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}8)$$

$$\underline{A}_4^3 = \begin{bmatrix} -\cos\theta_4 & 0 & -\sin\theta_4 & 0 \\ -\sin\theta_4 & 0 & \cos\theta_4 & 0 \\ 0 & 1 & 0 & WE \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}9)$$

$$\underline{A}_5^4 = \begin{bmatrix} -\cos\theta_5 & 0 & -\sin\theta_5 & 0 \\ -\sin\theta_5 & 0 & \cos\theta_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}10)$$

$$\underline{A}_6^5 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ \sin\theta_6 & \cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & HW \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}11)$$

The six transformation matrices in Equations (3-6) through (3-11) can be multiplied together to obtain one overall transformation matrix, $\underline{A}_6^0$, for the whole arm:

$$\underline{A}_6^0 = \underline{A}_1^0\underline{A}_2^1\underline{A}_3^2\underline{A}_4^3\underline{A}_5^4\underline{A}_6^5 \quad . \qquad (3\text{-}12)$$

For simplicity, in the rest of this thesis a vector in the arm base coordinates will be denoted by the superscript a, and a vector in the end effector coordinates will be denoted by the superscript ee. Then

$$\underline{r}^a = \underline{r}^0 \quad , \qquad (3\text{-}13)$$

$$\underline{r}^{ee} = \underline{r}^6 \quad , \qquad (3\text{-}14)$$

and

$$\underline{r}^a = \underline{A}_{ee}^a \underline{r}^{ee} \quad . \qquad (3\text{-}15)$$

Vectors describing the position and orientation of the end effector will be similar to those used for the vehicle body. The position and velocity of the origin of the end effector coordinate system will be denoted by $\underline{p}_{ee}$ and $\underline{v}_{ee}$, respectively. The orientation of the end effector system can be specified by the three Euler angles α_{ee}, β_{ee}, γ_{ee} (where these are defined as in Figure 1), and the angular velocity of the system by $\underline{\omega}_{ee}$.

Then the orientation of the end effector system with respect to the arm base system can be described by the 3x3 rotation matrix $\underline{R}^a_{ee}$ which is, again, of the form given in Equation (2-4). As with the robot system, both position and orientation information are contained in the homogeneous transformation matrix, in this case $\underline{A}^a_{ee}$, where

$$\underline{A}^a_{ee} = \begin{bmatrix} & & & p^a_{ee_x} \\ & \left[\underline{R}^a_{ee}\right] & & p^a_{ee_y} \\ & & & p^a_{ee_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \qquad (3\text{-}16)$$

3.3. Control System.

As was mentioned earlier in this chapter, the control system described in this work is a supervisory control system [27] in which the higher-level commands of steering, speed, etc., are supplied to the control computer by a human operator. The computer then calculates the arm motions which are required to implement those commands.

Another method of dividing the control problem into levels applicable to walking robots has been suggested by

Narinyani, et al. [29]. This model divides the problem into three levels. The upper level is the overall planning of the route based on the robot's basic capabilities and limitations (for instance, the size of obstacle it can overcome). The middle level involves looking ahead for several steps in the direction of the route fixed by the upper level. This level then arrives at a corridor of possible paths based on robot parameters (speed, balance, body height, etc.) and local terrain features. The lower level involves placing the feet to move the robot along one of the paths in the corridor determined by the middle level.

In many cases, it would be useful to consider the lower level as consisting of two levels: One which determines the location at which a foot is to be placed, and another which calculates the trajectory which the foot should follow in reaching that location. This division seems justified since in difficult terrain human beings often consider where to place their feet but they rarely think about the actual path their feet follow in the air. This fact can be applied to walking robots if access is allowed to the program between the above two levels. Then in difficult terrain a human operator could determine where the robot's feet should be placed and let the computer determine how to get them there. This mode of operation would also be useful if one of the robot's legs is to be as an arm (to manipulate tools, for example), a circumstance which might well occur with the robot in this work, since its legs are industrial manipulators.

If Narinyani's lower level is divided as suggested above, the functional division of robot control into levels becomes

Level I	- overall planning of route based on robot's basic capabilities and limitations
Level II	- determining corridor of paths based on robot parameters and local terrain features
Level III	- choosing locations at which feet are placed
Level IV	- calculating foot trajectories

Although in the general robot walking problem Level II is very important, in this work the terrain is structured enough that Levels I and II can be combined. There are generally enough footholds that the robot's flexibility allows it to move along any path specified for its center of gravity.

The above four levels constitute a model of the general robot walking problem. For the specific problem addressed in this work, three modes of operation were chosen:

1. Robot Destination Control.
2. Robot Velocity Control.
3. Arm Control.

In the following paragraphs these three modes are discussed in terms of the four levels described above.

With Robot Destination Control, the operator enters a sequence of points (in coordinates based on the grid on the cylinder), and the computer calculates the trajectory required for the robot to pass through those points. In this mode, the computer performs the operations required for part of Levels I and II and all of Levels III and IV.

With Robot Velocity Control, a three-axis joystick is used. Two of the axes control the linear velocity of the robot on the grid, and the other axis controls the rotation rate of the robot body. In this mode the operator handles Levels I and II and the computer performs the calculations for Levels III and IV.

Arm Control can be used in three different ways. In all three of those, the robot is stationary, so Levels I and II are not operating. With Arm Handhold Control, the operator enters a handhold location (in grid coordinates) and the arm moves to and grasps that handhold. With Arm Destination Control, a sequence of desired coordinates (positions and orientations) for the arm is entered and the computer calculates the trajectory required to pass through those points. In these two modes the computer performs only the operations for Level IV. Finally, with Arm Velocity Control, the translational velocity of the end effector is controlled by the three-axis joystick, and its rotational velocity is controlled by a set of three dials. In this mode all of the levels listed above are handled by the operator, with the computer only converting commands from Cartesian coordinates to joint coordinates.

3.4. Summary.

In this chapter, some of the necessary background information has been given for the problem of control of the three-armed robot discussed in this work. Models for the body of the robot and for its arms have been given, and the control system employed has been outlined. Chapters 4 and 5 will discuss the solution of the problem formulated in this chapter.

Chapter 4
ARM KINEMATICS ALGORITHMS

4.1. Introduction.

The problem of manipulator kinematics can conveniently be divided into two parts. The first is the conversion of Cartesian velocity commands to joint velocity commands. As discussed in Chapter 2, it is usually convenient to derive manipulator commands in Cartesian coordinates since that is the system in which human beings are most accustomed to working. Those commands must then be converted to joint coordinates for the actuator signals. This problem is the topic of Section 4.2.

Although the Cartesian velocity commands can be provided at every instant in time by a joystick or similar device, in many cases the desired path of the manipulator is specified in the form of a sequence of points through which it is to pass. Deriving a trajectory (Cartesian velocities as functions of time) from that sequence of points is the second part of the manipulator kinematics problem and is discussed in Section 4.3.

4.2. Jacobian Solution.

The most straightforward way of deriving joint velocities from Cartesian velocities is to use basic Resolved Motion Rate Control [7]. For a six-degree-of-freedom manipulator this involves first calculating the 6x6 Jacobian matrix $\underline{J}$, whose elements are the partial derivatives with respect to the joint angles of the components of $\underline{v}_{ee}^{a}$ and $\underline{\omega}_{ee}^{a}$ (the translational and rotational velocities of the end effector with respect to the arm base). That is,

$$\begin{bmatrix} v_{ee_x}^{a} \\ v_{ee_y}^{a} \\ v_{ee_z}^{a} \\ \omega_{ee_x}^{a} \\ \omega_{ee_y}^{a} \\ \omega_{ee_z}^{a} \end{bmatrix} = \underline{J} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix} . \qquad (4\text{-}1)$$

Then to derive joint velocities from Cartesian velocities ($\underline{v}_{ee}^{a}$ and $\underline{\omega}_{ee}^{a}$) the Jacobian must be inverted and multiplied by those Cartesian velocities:

$$\underline{\dot{\theta}} = \underline{J}^{-1} \left[\begin{array}{c} \underline{v}_{ee}^{a} \\ \hline \underline{\omega}_{ee}^{a} \end{array} \right] . \qquad (4\text{-}2)$$

Using this method for a six-degree-of-freedom arm involves inverting a 6x6 matrix. This can be done symbolically, which is very difficult, or numerically every time the velocities are calculated, which is computationally expensive. Since neither of the above alternatives is desirable, an alternative solution has been used in this work. For some arm designs, including the one used here, the joint velocity problem can be divided into two parts, each of which requires the inversion of a 3x3 matrix. These two 3x3 matrices can be inverted symbolically, eliminating the need for numerical inversions at run time. This approach is discussed more fully in Section 4.2.1.

No matter how the joint velocity problem is solved, there are some arm configurations (those in which the Jacobian is singular) in which Cartesian velocities in one or more directions are impossible. Although mathematically these singularities are all similar, the physical limitations they place on the arm may be very different so they all must be considered individually. The three singularities of the PUMA 600 arm are considered in Section 4.2.2.

4.2.1. Arm Decomposition Approach.

In the PUMA 600 arm the last three joint axes intersect in a point which can be called the "wrist." This allows the first three and last three joint velocities to be calculated independently, given the desired translational and rotational velocity of the end effector. The procedure which is used to calculate those joint velocities is described in the following paragraphs.

The first step is to calculate the translational velocity of the wrist, which is located at the origin of coordinate system $(\hat{\underline{x}}_4, \hat{\underline{y}}_4, \hat{\underline{z}}_4)$. That velocity depends only on the first three joint velocities, so if it is known, those joint velocities can be determined. Since the end effector and wrist are connected by a rigid link, the wrist's translational velocity $(\underline{v}_4^0)$ can be shown to be the sum of the end effector translational velocity $(\underline{v}_6^0)$ and the cross product of the end effector rotational velocity $(\underline{\omega}_6^0)$ and a vector from the end effector to the wrist $(\underline{p}_4^0 - \underline{p}_6^0)$; that is,

$$\underline{v}_4^0 = \underline{v}_6^0 + \underline{\omega}_6^0 \times (\underline{p}_4^0 - \underline{p}_6^0) \quad . \tag{4-3}$$

Then to calculate the first three joint velocities, the Jacobian matrix must be found whose elements are the partial derivatives of the components of the wrist position, $\underline{p}_4^0$, with respect to the first three joint angles. The easiest way to calculate that Jacobian is to calculate $\underline{p}_4^0$ and then make the required partial differentiations; thus

$$\underline{p}_4^0 = \underline{A}_1^0 \underline{A}_2^1 \underline{A}_3^2 \;\; \underline{p}_4^3 \quad . \tag{4-4}$$

$$\begin{bmatrix} p^0_{4_x} \\ p^0_{4_y} \\ p^0_{4_z} \\ 1 \end{bmatrix} = \underline{A}^0_1 \underline{A}^1_2 \underline{A}^2_3 \begin{bmatrix} 0 \\ 0 \\ WE \\ 1 \end{bmatrix} \tag{4-5}$$

$$= \begin{bmatrix} c1s23WE + c1s2ES - s1SN \\ s1s23WE + s1s2ES + c1SN \\ c23WE + c2ES + NO \\ 1 \end{bmatrix} \tag{4-6}$$

where $si = \sin\theta_i$ $ci = \cos\theta_i$

$s23 = \sin(\theta_2+\theta_3)$ $c23 = \cos(\theta_2+\theta_3)$.

Then the Jacobian is

$$\underline{J}_1 = \begin{bmatrix} \frac{\partial p^0_{4_x}}{\partial\theta_1} & \frac{\partial p^0_{4_x}}{\partial\theta_2} & \frac{\partial p^0_{4_x}}{\partial\theta_3} \\ \frac{\partial p^0_{4_y}}{\partial\theta_1} & \frac{\partial p^0_{4_y}}{\partial\theta_2} & \frac{\partial p^0_{4_y}}{\partial\theta_3} \\ \frac{\partial p^0_{4_z}}{\partial\theta_1} & \frac{\partial p^0_{4_z}}{\partial\theta_2} & \frac{\partial p^0_{4_z}}{\partial\theta_3} \end{bmatrix} \tag{4-7}$$

$$= \begin{bmatrix} -s1s23WE-s1s2ES-c1SN & c1c23WE+c1c2ES & c1c23WE \\ c1s23WE+c1s2ES-s1SN & s1c23WE+s1c2ES & s1c23WE \\ 0 & -s23WE-s2ES & -s23WE \end{bmatrix} . \quad (4\text{-}8)$$

and if the Jacobian is not singular the inverse Jacobian is (since for this arm WE = ES)

$$\underline{J}_1^{-1} = \frac{1}{|\underline{J}_1|} \begin{bmatrix} -s1s3ES & c1s3ES & 0 \\ \begin{matrix} c1s23(s2+s23)ES \\ -s1s23SN \end{matrix} & \begin{matrix} s1s23(s2+s23)ES \\ +c1s23SN \end{matrix} & c23(s2+s23)ES \\ \begin{matrix} -c1(s2+s23)^2ES \\ +s1(s2+s23)SN \end{matrix} & \begin{matrix} -s1(s2+s23)^2ES \\ -c1(s2+s23)SN \end{matrix} & \begin{matrix} -c2(s2+s23)ES \\ -c23(s2+s23)ES \end{matrix} \end{bmatrix} . \quad (4\text{-}9)$$

where $|\underline{J}_1| = (s2+s23)s3ES^2$.

The first three joint velocities can then be calculated as

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = \underline{J}_1^{-1} \begin{bmatrix} v^0_{4_x} \\ v^0_{4_y} \\ v^0_{4_z} \end{bmatrix} . \quad (4\text{-}10)$$

The possible singularities of $\underline{J}_1$ (s2+s23 = 0 and s3 = 0) are discussed in Section 4.2.2.1.

Now that the first three joint velocities are known, the last three joint velociites can be calculated from the end effector rotat onal velocity. This can be done by calculating the rotational velocity which must be provided by the last three joints ($\underline{\omega}_6^3$), which is equal to the desired end effector rotational velocity ($\underline{\omega}_6^0$) minus the rotational contribution of the first three joints ($\underline{\omega}_3^0$), all transformed into coordinate system ($\hat{\underline{x}}_3$, $\hat{\underline{y}}_3$, $\hat{\underline{z}}_3$); that is,

$$\underline{\omega}_6^3 = {}^{2}\underline{R}_3^{-1}\ {}^{1}\underline{R}_2^{-1}\ {}^{0}\underline{R}_1^{-1}\ (\underline{\omega}_6^0 - \underline{\omega}_3^0) \quad . \tag{4-11}$$

$$\begin{bmatrix} \omega^3_{6_x} \\ \omega^3_{6_y} \\ \omega^3_{6_z} \end{bmatrix} = \begin{bmatrix} c1c23 & s1c23 & -s23 \\ -s1 & c1 & 0 \\ c1s23 & s1s23 & c23 \end{bmatrix} \begin{bmatrix} \omega^0_{6_x} + (\dot{\theta}_2+\dot{\theta}_3)s1 \\ \omega^0_{6_y} + (\dot{\theta}_2+\dot{\theta}_3)c1 \\ \omega^0_{6_z} - \dot{\theta}_1 \end{bmatrix} \quad . \tag{4-12}$$

Thus to calculate the last three joint velocities, the Jacobian matrix must be found whose elements are the partial derivatives of the components of the end effector orientation with respect to the last three joint angles. The columns of that Jacobian are simply the vectors describing the last three joint axes in coordinate system ($\hat{\underline{x}}_3$, $\hat{\underline{y}}_3$, $\hat{\underline{z}}_3$); thus

$$\underline{J}_2 = \begin{bmatrix} 0 & -s4 & c4s5 \\ 0 & c4 & s4s5 \\ 1 & 0 & c5 \end{bmatrix}$$

and if the Jacobian is not singular the inverse Jacobian is

$$\underline{J}_2^{-1} = \frac{1}{-s5}\begin{bmatrix} c4c5 & s4s5 & -s5 \\ s4s5 & -c4s5 & 0 \\ -c4 & -s4 & 0 \end{bmatrix} . \qquad (4\text{-}14)$$

The last three joint velocities can then be calculated as

$$\begin{bmatrix} \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix} = \underline{J}_2^{-1} \begin{bmatrix} \omega^3_{6_x} \\ \omega^3_{6_y} \\ \omega^3_{6_z} \end{bmatrix} . \qquad (4\text{-}15)$$

The possible singularity of $\underline{J}_2$ (s5 = 0)is discussed in Section 4.2.2.2.

4.2.2. Jacobian Singularities.

Both the Jacobians discussed above can be singular (have a determinant equal to zero). Under those conditions the joint velocities cannot be calculated using the inverse Jacobian since it does not exist. There are two ways of dealing with this problem: Another method can be used to calculate the joint velocities under those conditions or artificial limits on the joints can be imposed to make those conditions impossible. The latter approach is adopted for the singularities of $\underline{J}_1$, and the former is used to the singularity of $\underline{J}_2$. Those approaches are discussed in the following two sections.

4.2.2.1. Singularities of $\underline{J}_1$.

The Jacobian matrix $\underline{J}_1$ is singular when s3 = 0 or s2+s23 = 0. The condition s3 = 0 occurs when the arm is fully extended (i.e., when the elbow joint is straight). At this point the wrist cannot move farther away from the shoulder, nor can it move directly towards the shoulder; thus any velocity command which includes components in those directions will be unattainable. This condition was eliminated by imposing an artificial joint limit of $\theta_3 \leq -1°$. This limit does not allow the arm to be fully extended, but since it can be almost fully extended, the radius of the reachable space is only decreased by about 0.01 in. The artificial limit also makes some otherwise reachable points near the arm base unreachable but for walking those points are unimportant. If one of the arms were being used for something other than walking (e.g., manipulating a tool), the limit could be overridden and the approach outlined in Section 4.2.2.2. could be used.

The approach used for the singularity of s2+s23 = 0 is similar. At this point the wrist is directly under the shoulder and cannot move either toward or away from the base-neck axis. This condition was eliminated by imposing the limit $\theta_2 \geq -\theta_3/2 + 1°$. This limit increases the minimum radius for the arm by only 0.03 in. It also makes an area underneath the robot unreachable, but for walking that area should be avoided anyway.

At any rate, if it were necessary to reach that area, the limit could be overridden and the approach in the next section used.

4.2.2.2 Singularity of $\underline{J}_2$.

The Jacobian matrix $\underline{J}_2$ is singular when s5 = 0. This singularity cannot be handled as the two in the previous section were because it can occur at many positions and orientations in the reachable area, not just at its edges. Thus elimination of this singularity would place severe restrictions on the motion of the arm, and a methcd must be found to solve for the joint velocities when s5 = 0.

When s5 = 0, $\theta_5 = 0°$ and the axes for θ_4 and θ_6 are the same. Any change in θ_4 or θ_6 will affect only $\omega^3_{6_z}$ (see Equation (4-12)), so either $\dot{\theta}_4$ or $\dot{\theta}_6$ can be chosen arbitrarily so long as the other is chosen so their sum is $\omega^3_{6_z}$. Thus let $\dot{\theta}_6$ remain constant; then

$$\ddot{\theta}_6 = 0 \tag{4-16}$$

and

$$\dot{\theta}_4 = \omega^3_{6_z} - \dot{\theta}_6 \quad . \tag{4-17}$$

Then to determine $\dot{\theta}_5$ note that the configuration is as shown in Figure 5. Any change in θ_5 can cause a rotation only is the direction of $\underline{\hat{z}}_4$. Thus the only portion of the desired rotational velocity (whose components are $\omega^3_{6_x}$ and $\omega^3_{6_y}$) which can be implemented is the projection of that velocity on $\underline{\hat{z}}_4$;

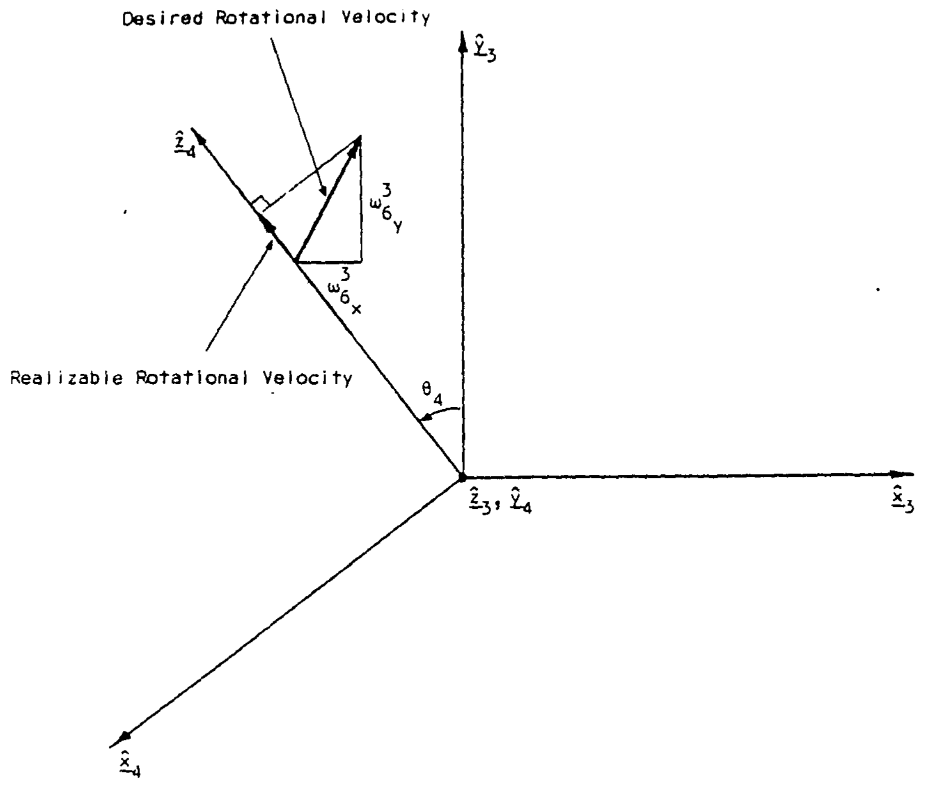

Figure 5. Configurations of Coordinate Systems ($\hat{\underline{x}}_3$, $\hat{\underline{y}}_3$, $\hat{\underline{z}}_3$) and ($\hat{\underline{x}}_4$, $\hat{\underline{y}}_4$, $\hat{\underline{z}}_4$) for Jacobian Matrix Singularity of $\sin \theta_5 = 0$ Showing Realizable Component of Desired Velocity.

that is,

$$\dot{\theta}_5 = -s4 \times \omega^3_{6_x} + c4 \times \omega^3_{6_y} \quad . \tag{4-18}$$

Equations (4-16), (4-17), and (4-18) are used when $\underline{J}_2$ is singular to implement as much of $\underline{\omega}^3_6$ as possible; as soon as the arm is in a configuration in which $\underline{J}_2$ is not singular the joint velocity calculation method described in Section 4.2.1. is again used. As mentioned earlier, the approach described here could also be applied to the singularities of $\underline{J}_1$ if desired.

4.3. Trajectory Calculation.

The problem of calculating a time-optimal trajectory for a robot arm to follow in passing through a sequence of points is an important one for this work. For reliability purposes, the robot will have two arms grasping handholds at all times, so only one arm can be moving in the air. This means that on the average the time it takes the robot to move one unit can be no less than three times the time it takes an arm in the air to travel the same unit with respect to the ground. An increase in the average velocity of the arm with respect to the ground will be reflected by three times that increase in the robot's velocity, so the degree of time optimality of the arms' trajectories will determine how close the robot's maximum velocity approaches its theoretical limit.

Some work has been done in the area of arm trajectory time optimization by Luh and Walker [30]. They consider translational motion only in which the end effector path is made up of straight line segments connected by smooth arcs. The arcs are "rounding" of the corners from an exactly point-to-point path, and are subject to a maximum error constraint which is related to the length of the path segments which form the corners. The path is not actually a minimum-time trajectory because a restriction of constant velocity is used in the straight segments between the connecting arcs.

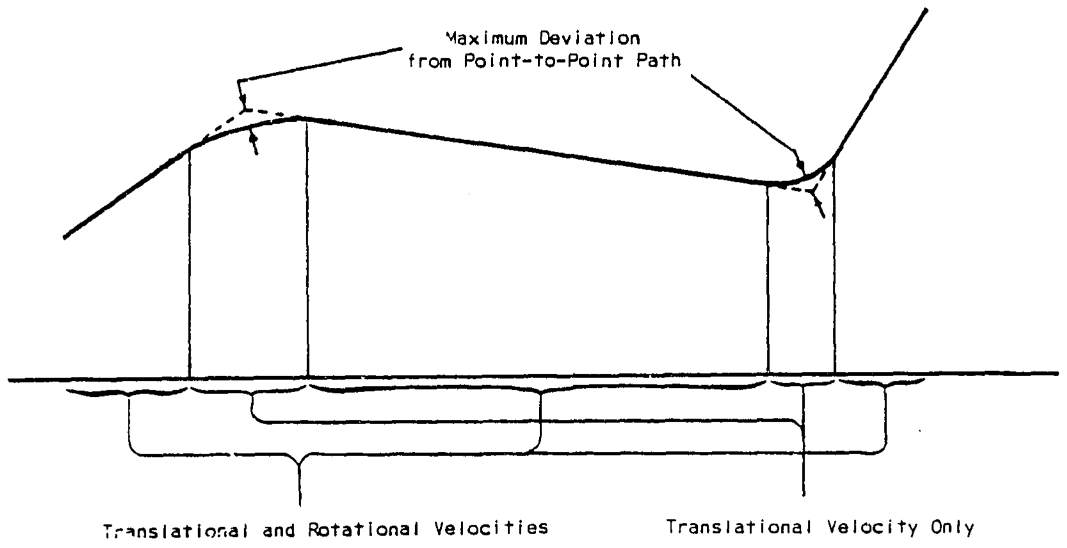
Maximum Deviation from Point-to-Point Path
Translational and Rotational Velocities
Translational Velocity Only

Figure 6. Typical Arm Path Showing How a Point-to-Point Specification is Modified for a Near-Minimum-Time Trajectory.

The approach used here considers both translational and rotational velocities. The translational path is similar to that of Luh and Walker in that it is composed of straight segments connected by arcs (see Figure 6), but the error constraint is absolute rather than a function of the segment lengths. Another difference from the previous work is that accelerations and decelerations are permitted in the straight segments, but constant acceleration is specified in the transition arcs, making them parabolic arcs. Since most of a path is usually straight segments, this approach allows the arm to travel faster in those segments, slowing down only to "turn the corners." The algorithm also assumes zero rotational velocity in the transition (see Figure 6), but since the desired translational displacement of

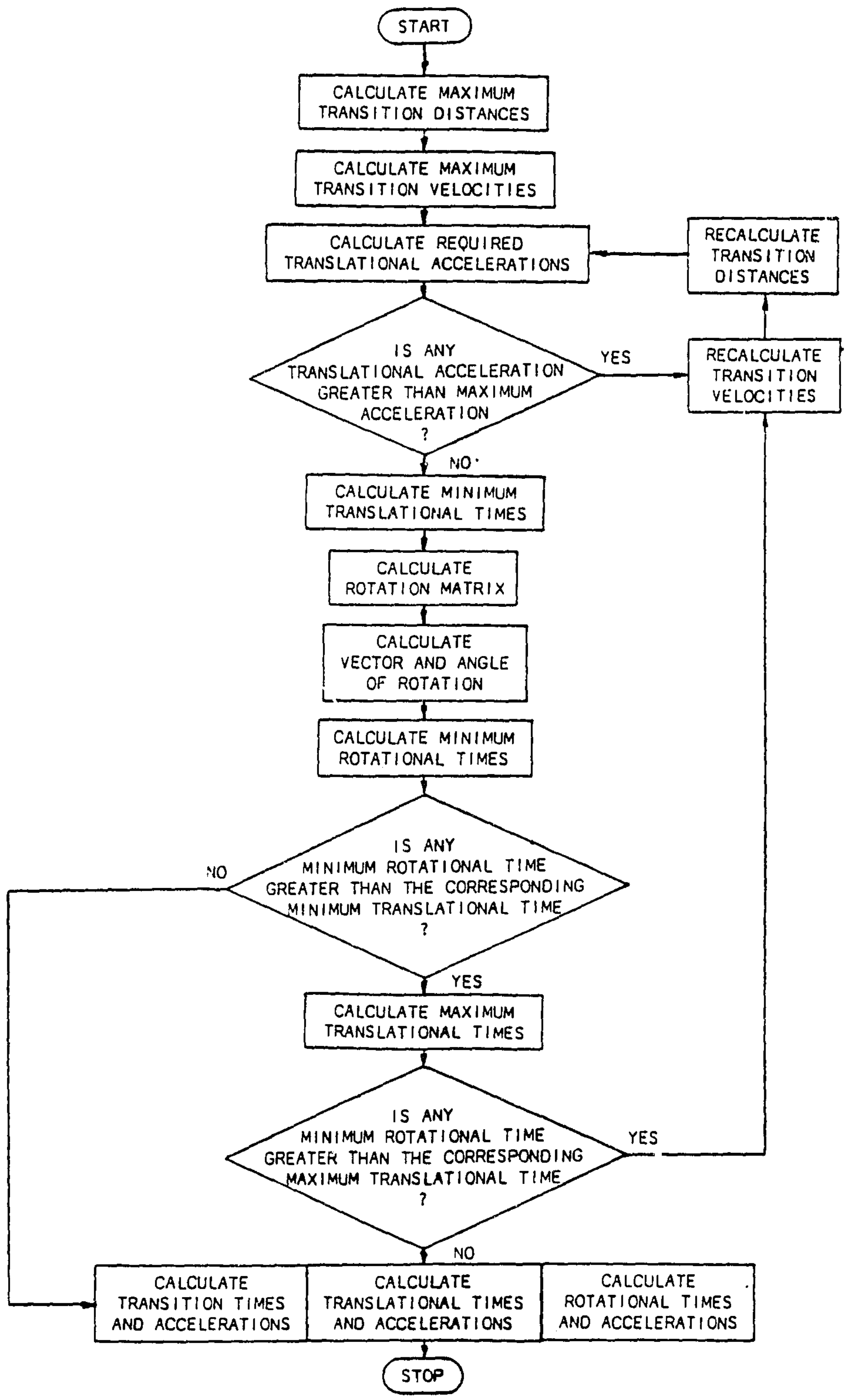
START
CALCULATE MAXIMUM TRANSITION DISTANCES
CALCULATE MAXIMUM TRANSITION VELOCITIES
CALCULATE REQUIRED TRANSLATIONAL ACCELERATIONS
RECALCULATE TRANSITION DISTANCES
IS ANY TRANSLATIONAL ACCELERATION GREATER THAN MAXIMUM ACCELERATION ?
YES
RECALCULATE TRANSITION VELOCITIES
NO
CALCULATE MINIMUM TRANSLATIONAL TIMES
CALCULATE ROTATION MATRIX
CALCULATE VECTOR AND ANGLE OF ROTATION
CALCULATE MINIMUM ROTATIONAL TIMES
IS ANY MINIMUM ROTATIONAL TIME GREATER THAN THE CORRESPONDING MINIMUM TRANSLATIONAL TIME ?
NO
YES
CALCULATE MAXIMUM TRANSLATIONAL TIMES
IS ANY MINIMUM ROTATIONAL TIME GREATER THAN THE CORRESPONDING MAXIMUM TRANSLATIONAL TIME ?
YES
NO
CALCULATE TRANSITION TIMES AND ACCELERATIONS
CALCULATE TRANSLATIONAL TIMES AND ACCELERATIONS
CALCULATE ROTATIONAL TIMES AND ACCELERATIONS
STOP

Figure 7. Flow Chart of Arm Trajectory Optimization Algorithm

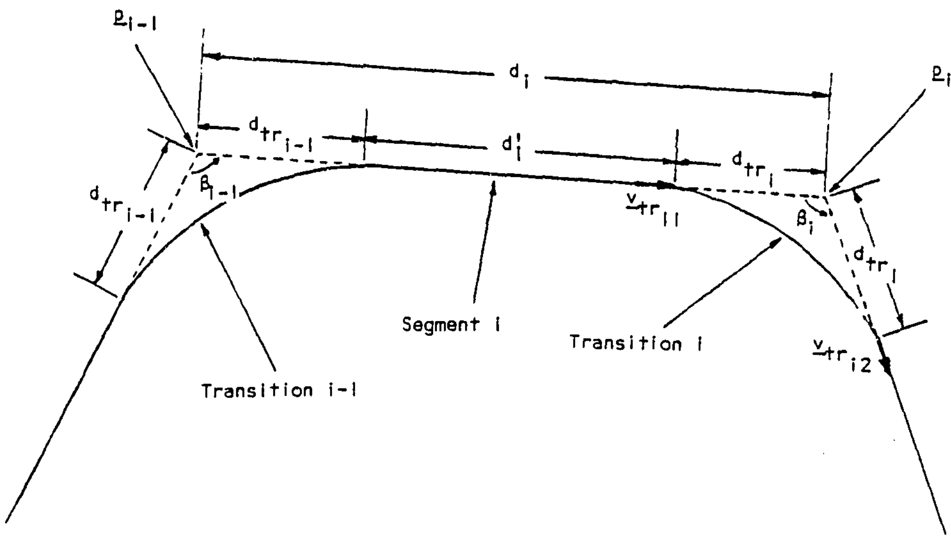

Figure 8. Definition of Arm Translational Trajectory Variables

an arm is usually relatively greater than the rotational displacement, this will not often slow down the arm. A flow chart of the algorithm is shown in Figure 7 and it is discussed in detail in the following paragraphs.

4.3.1. Translational Trajectory.

Figure 8 shows a segment of the translational trajectory with some of the terms which will be used below defined. The figure shows segment i of the trajectory along with transitions i-1 and i at the ends of that segment. The angle made at the corner of transition i is β_i. The distance from that corner to the point where the actual path breaks away from an exact point-to-point path is d_{tr_i}. The total length of segment i is d_i, and that length minus the two transition distances is d'_i.

The velocity on entrance to transition i is $\underline{v}_{tr_{i1}}$ and the velocity on exit from that transition if $\underline{v}_{tr_{i2}}$. The magnitude of both those velocities is v_{tr_i}. The time it takes for the arm to pass through transition i will be denoted by τ_{tr_i}.

The first steps in the algorithm are the calculation of the maximum transition distances and velocities (see Figure 8). The transition distance is calculated using the general formula

$$\Delta s = \left|\underline{v}_0(\Delta t) + \frac{1}{2}\,\underline{a}(\Delta t)^2\right| \tag{4-19}$$

where Δs is the distance travelled, $\underline{v}_0$ is the initial velocity, $\underline{a}$ is the acceleration, and Δt is the time interval; from that formula it can be calculated that

$$2d_{tr_i}\sin(\beta_i/2) = \left|\underline{v}_{tr_{i1}}\tau_{tr_i} + \frac{1}{2}\left(\frac{\underline{v}_{tr_{i2}} - \underline{v}_{tr_{i1}}}{\tau_{tr_i}}\right)\tau^2_{tr_i}\right| \tag{4-20}$$

$$= \tau_{tr_i} v_{tr_i} \sin(\beta_i/2) \tag{4-21}$$

$$d_{tr_i} = \frac{\tau_{tr_i} v_{tr_i}}{2} \; . \tag{4-22}$$

Since a constant acceleration is specified over the entire transition, the time taken to make the transition can be shown to be the same as if the arm followed the segment paths with an instantaneous change of direction at the intersection. For the transition distance to be maximum the difference in those two paths at the midpoint of the transition should be equal to the maximum allowable error; thus

$$\varepsilon_{max} = \left[\underline{v}_{tr_{i1}}(\tau_{tr_i}/2) + \frac{1}{2}\left(\frac{\underline{v}_{tr_{i2}} - \underline{v}_{tr_{i1}}}{\tau_{tr_i}}\right)(\tau_{tr_i}/2)^2\right] - \left[\underline{v}_{tr_{i1}}(\tau_{tr_i}/2)\right] \tag{4-23}$$

$$= \frac{\tau_{tr_i} v_{tr_i} \cos(\beta_i/2)}{4} \quad . \tag{4-24}$$

Then

$$d_{tr_i} = \frac{4\varepsilon_{max}}{\sqrt{2(1+\cos\beta_i)}} \tag{4-25}$$

but for simplicity later it is convenient to assume that the transition distance is less than one-half the segment lengths on either side of the transition, so

$$d_{tr_i} = \min\left(\frac{d_i}{2}, \frac{d_{i+1}}{2}, \frac{4\varepsilon_{max}}{\sqrt{2(1+\cos\beta_i)}}\right) \quad . \tag{4-26}$$

The maximum transition velocity can be calculated from

$$d_{tr_i} = \frac{\left|\underline{v}_{tr_{i2}} - \underline{v}_{tr_{i1}}\right| v_{tr_i}}{2a_{max}} \tag{4-27}$$

to be

$$v_{tr_i} = \sqrt{\frac{2d_{tr_i} a_{max}}{\sqrt{2(1+\cos\beta_i)}}} \quad . \tag{4-28}$$

The next step is calculation of the segment accelerations that the transition distances and velocities require. For segment i the required acceleration is

$$a_i = \left| \frac{v^2_{tr_i} - v^2_{tr_{i-1}}}{2(d_i - d_{tr_i} - d_{tr_{i-1}})} \right| \quad . \tag{4-29}$$

If any of the segment accelerations is greater than the maximum acceleration allowed, the transition distances and velocities must be recalculated. This is done by lowering the higher of the two velocities to the value which requires the maximum acceleration. The equations below are simplified if, for segment i which has a required accleration greater than the maximum, the transition with the higher of the two velocities is designated "hi" and the one with the lower is designated "lo"; then

$$v^2_{tr_{hi}} - v^2_{tr_{lo}} = 2a_{max}(d_i - d_{tr_{hi}} - d_{tr_{lo}}) \tag{4-30}$$

$$v_{tr_{hi}} = \sqrt{\frac{v^2_{tr_{lo}} + 2a_{max}(d_i - d_{tr_{lo}})}{1 + \sqrt{2(1+\cos\beta_{hi})}}} \quad . \tag{4-31}$$

The transition distance can then be calculated as

$$d_{tr_{hi}} = \frac{v^2_{tr_{hi}} \sqrt{2(1+\cos\beta_{hi})}}{2a_{max}} \quad . \tag{4-32}$$

The final step in the translational portion of the algorithm is calculation of the minimum translational times for the segments. These times will later be compared with rotational times to determine if either needs to be adjusted to synchronize the translational and rotational motions. The first step in the calculation of the translational times is calculation of the highest velocity reached in the segment as

$$v_{hi_i} = \min\left(v_{max}, \sqrt{a_{max}\left(d'_i - \frac{v^2_{tr_{hi}} - v^2_{tr_{lo}}}{2a_{max}}\right) + v^2_{tr_{hi}}}\right) \quad (4\text{-}33)$$

The time can then be calculated as

$$\tau_{trans_i} = \frac{2v_{hi_i} - v_{tr_{hi}} - v_{tr_{lo}}}{a_{max}} + \frac{d'_i - \dfrac{2v^2_{hi_i} - v^2_{tr_{hi}} - v^2_{tr_{lo}}}{2a_{max}}}{v_{max}} \quad (4\text{-}34)$$

4.3.2. Rotational Trajectory.

The object of the rotational portion of the algorithm is to determine for each segment a unit vector $\hat{\underline{r}}$ and an angle ϕ such that a rotation of ϕ about $\hat{\underline{r}}$ makes the desired rotational change over that segment. The first step in this procedure is calculation of a rotation matrix $e^{\underline{S}_i\phi}$ for segment i which describes the rotational motion for that segment. That matrix can be calculated from the rotation matrices of the two endpoints of the segment as

$$e^{\underline{S}_i\phi} = \begin{bmatrix} s_{i_{11}} & s_{i_{12}} & s_{i_{13}} \\ s_{i_{21}} & s_{i_{22}} & s_{i_{23}} \\ s_{i_{31}} & s_{i_{32}} & s_{i_{33}} \end{bmatrix} \quad (4\text{-}35)$$

$$= \underline{R}_i \underline{R}^{-1}_{i-1} \quad (4\text{-}36)$$

The vector and angle of rotation can then be derived from Euler's theorem [31], which states

$$\underline{R}_1\underline{R}_{i-1}^{-1} = e^{\underline{S}_i\phi} \tag{4-37}$$

$$= \underline{I}\cos\phi + \underline{\hat{r}}_i\underline{\hat{r}}_i^T(1-\cos\phi) + \underline{S}_i\sin\phi \tag{4-38}$$

where $\underline{S}_i = \begin{bmatrix} 0 & -\hat{r}_{i_z} & \hat{r}_{i_y} \\ \hat{r}_{i_z} & 0 & -\hat{r}_{i_x} \\ -\hat{r}_{i_y} & \hat{r}_{i_x} & 0 \end{bmatrix}$

From

$$\underline{\hat{r}}_i\sin\phi_i = \frac{1}{2}\begin{bmatrix} s_{i_{32}} - s_{i_{23}} \\ s_{i_{13}} - s_{i_{31}} \\ s_{i_{21}} - s_{i_{12}} \end{bmatrix} \tag{4-39}$$

it can be seen that (since $\underline{r}_i$ is a unit vector)

$$\sin\phi_i = \frac{1}{2}\left[(s_{i_{32}}-s_{i_{23}})^2 + (s_{i_{13}}-s_{i_{31}})^2 + (s_{i_{21}}-s_{i_{12}})^2\right]^{1/2} \tag{4-40}$$

Then

$$\underline{\hat{r}}_i = \frac{1}{2}\begin{bmatrix} s_{i_{32}} - s_{i_{23}} \\ s_{i_{13}} - s_{i_{31}} \\ s_{i_{21}} - s_{i_{12}} \end{bmatrix} /\sin\phi_i \tag{4-41}$$

Also,

$$\cos\phi_i = \frac{s_{i_{11}} - \hat{r}_{i_x}}{1-\hat{r}^2_{i_x}} = \frac{s_{i_{22}} - \hat{r}^2_{i_y}}{1-\hat{r}^2_{i_y}} = \frac{s_{i_{33}} - \hat{r}^2_{i_z}}{1-\hat{r}^2_{i_z}} \tag{4-42}$$

so ϕ_i can be calculated as

$$\phi_i = \tan^{-1}\left(\frac{\sin\phi_i}{\cos\phi_i}\right) . \tag{4-43}$$

The minimum rotational time can then be calculated as

$$\tau_{rot_i} = 2\sqrt{\frac{\phi_i}{\dot{\omega}_{max}}} . \tag{4-44}$$

4.3.3. Combination of Trajectories.

Now the minimum translational and rotational times for all the segments are known and the times must be reconciled to synchronize the translational and rotational motions. If the rotational time is less than the translational time, this can be accomplished easily because the rotation can be slowed down by using

$$\dot{\omega} = \dot{\omega}_{max}\left(\frac{\tau_{rot}}{\tau_{trans}}\right)^2 . \tag{4-45}$$

If the rotational time is greater than the translational time, the problem is not so simple because the translational velocity is not zero at the ends of the segment. The first step is to determine if, with the present transition distances and velocities, the translational motion can be slowed enough to take as long as the rotational motion. This can be checked by first calculating the lowest velocity which can be attained in the segment as

$$v_{lo_i} = \sqrt{\max\left(0,\ v_{tr_{lo}}^2 - a_{max}\left(d_i' - \frac{v_{tr_{hi}}^2 - v_{tr_{lo}}^2}{2a_{max}}\right)\right)} \qquad (4\text{-}46)$$

If v_{lo} is zero, the translational motion can be slowed down as much as necessary; if not, the maximum translational time must be calculated as

$$\tau'_{trans_i} = \frac{v_{tr_{hi}} + v_{tr_{lo}} - 2v_{lo_i}}{a_{max}} \ . \qquad (4\text{-}47)$$

If that time is less than the rotational time, one or both of the transition velocities must be reduced. In the program implemented they are reduced to the point where the arm velocity can be reduced to zero; this is not necessarily a minimum-time solution but it is time-consuming to obtain the minimum-time solution and this situation arises seldom so it should not significantly affect the effectiveness of the algorithm.

If the minimum rotational time is less than the maximum translational time but greater than the minimum translational time, the translational velocity profile is determined by calculating the average velocity which must be maintained over the segment (excluding the portion in which the arm is accelerated from the lower transition velocity to the higher transition velocity) as

$$v_{ave_i} = \frac{d_i' - \dfrac{v_{tr_{hi}}^2 - v_{tr_{lo}}^2}{2a_{max}}}{\tau_{rot_i} - \dfrac{v_{tr_{hi}} - v_{tr_{lo}}}{a_{max}}} \quad . \tag{4-48}$$

4.4. Summary.

This chapter has described the algorithms which are used in this work to implement motion of the robot's arms. These are the groundwork for the procedures which control the robot's walking, which are discussed in the next chapter.

Chapter 5
WALKING ALGORITHMS

5.1. Introduction.

The robot discussed in this work walks over the terrain shown in Figure 2. The surface is a cylinder on which there is a grid of handholds for the robot's arms to grasp. The program allows any of these handholds to be removed to simulate a hole, trench, or similar obstacle. It is assumed that these obstacles have no height so a leg can pass over them.

The algorithm used to control the robot's walking is basically similar to that of McGhee and Iswandhi [26]. It is based on the notion of kinematic margin (defined in Section 5.3.1.); the algorithm seeks to maximize the value of kinematic margin over all the supporting legs so as to maximize the distance the robot can move before one of the arms reaches the limit of its joints. The algorithm is described in detail in Section 5.3.2.

5.2. Robot Velocity Command Generation.

The basic input specification to the free gait algorithm described in the next section is a robot velocity command vector. This command consists of three translational components and three rotational components. Only three of those components, two translational and one rotational, are specified by the operator. The other two degrees of freedom are supplied by requiring the robot to remain at a specified height and tangent to the cylinder. The height specification supplies a translational velocity and the tangency requirement supplies two rotational velocities, so with the three operator-specified velocities the robot's motion is fully specified.

The operator-specified velocities are the robot's tangential and axial velocities on the cylinder and the velocity of rotation of its body. These commands are generated in one of two ways, depending on the mode of operation.

When the system is in Robot Velocity Mode, a three-axis joystick is used, and velocity commands are read each time the free gait algorithm is executed. Movement of the joystick gives the translational components of the robot velocity and rotation of the joystick about its axis gives the rotational component.

When the system is in Robot Destination Mode, the velocity commands are computed from a stored trajectory. This trajectory is calculated from a sequence of robot positions and orientations entered on entry into Destination Mode. The program first calculates the minimum translational and rotational times between each pair of points and determines which of those two times is greater. The greater of the times is used to compute the translational and rotational velocities required to move between those points in that time. Those velocities are then stored and constitute the trajectory for that sequence of points.

5.3. Free Gait Algorithm.

A flow chart of the walking algorithm is shown in Figure 9. As mentioned above, the key concept of the free gait algorithm is that of kinematic margin. That concept is defined in Section 5.3.1., and its use in the algorithm is described in Section 5.3.2.

5.3.1. Kinematic Margin.

The kinematic margin of an arm grasping a handhold is defined as the distance the base of that arm could travel in the direction of its current velocity before a joint of that arm reached the limit of its travel. The smallest kinematic margin of the three arms is thus the distance the robot could travel in its current direction before it had to stop because of arm limits. The algorithm seeks to maximize the value of this smallest margin because that will maximize the distance the robot can travel in its current configuration, thus maximizing the chance that it will be able to find another configuration in which to continue.

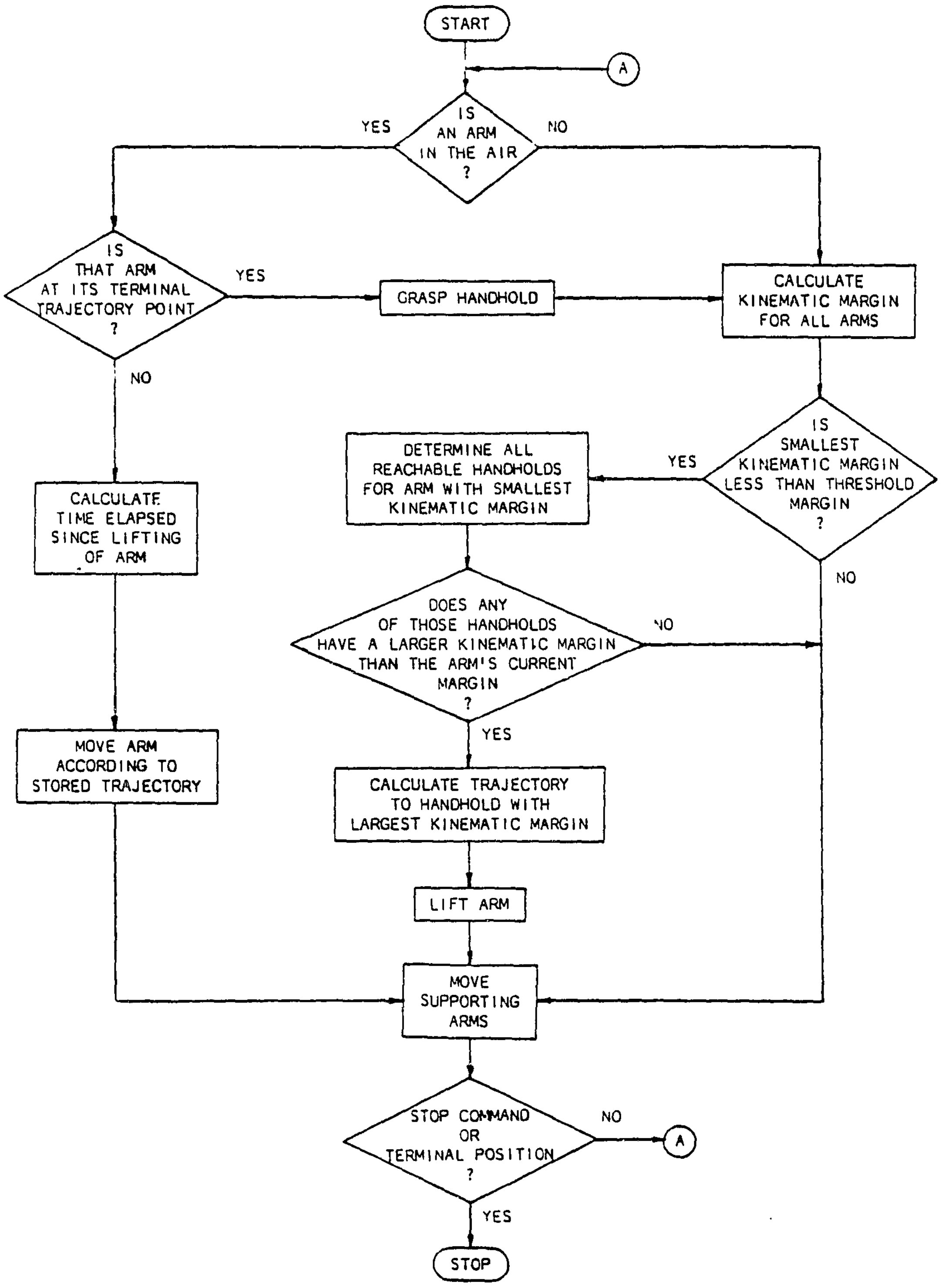
START
A
IS AN ARM IN THE AIR ?
YES
NO
IS THAT ARM AT ITS TERMINAL TRAJECTORY POINT ?
YES
GRASP HANDHOLD
CALCULATE KINEMATIC MARGIN FOR ALL ARMS
NO
IS SMALLEST KINEMATIC MARGIN LESS THAN THRESHOLD MARGIN ?
YES
DETERMINE ALL REACHABLE HANDHOLDS FOR ARM WITH SMALLEST KINEMATIC MARGIN
CALCULATE TIME ELAPSED SINCE LIFTING OF ARM
NO
DOES ANY OF THOSE HANDHOLDS HAVE A LARGER KINEMATIC MARGIN THAN THE ARM'S CURRENT MARGIN ?
NO
YES
MOVE ARM ACCORDING TO STORED TRAJECTORY
CALCULATE TRAJECTORY TO HANDHOLD WITH LARGEST KINEMATIC MARGIN
LIFT ARM
MOVE SUPPORTING ARMS
STOP COMMAND OR TERMINAL POSITION ?
NO
A
YES
STOP

Figure 9. Flow Chart of Free Gait Algorithm

The most accurate method of calculating kinematic margin would be to use knowledge of the three-dimensional reachable space of the arm. The boundaries of that space are not easily described, though, so two-dimensional limits are used in this work. Figure 10 shows the limits of the arm's reach in both the tangential and axial directions on the cylinder. These limits can be used to determine the approximate shape of the reachable area of the cylinder, which is shown in Figure 11. This area is not easily described, so an inscribed circle centered at the base of the arm is used in the program to calculate kinematic margin.

The calculations for kinematic margin assume that the base of the arm will continue at its present velocity indefinitely. The kinematic margin is then the length of a vector in a direction opposite to the arm base velocity from the current handhold to an intersection with the boundary of the reachable area (see Figure 12). The next section describes the use of the kinematic margin in the free gait algorithm.

5.3.2. Foot Lifting and Placing.

The first step performed by the algorithm is a check to see if one of the arms is moving from one handhold to another. If one is, the algorithm calculates the time elapsed since the arm was lifted. This elapsed time is used to determine from the stored trajectory of the arm its desired velocity, $\underline{v}_d$, and

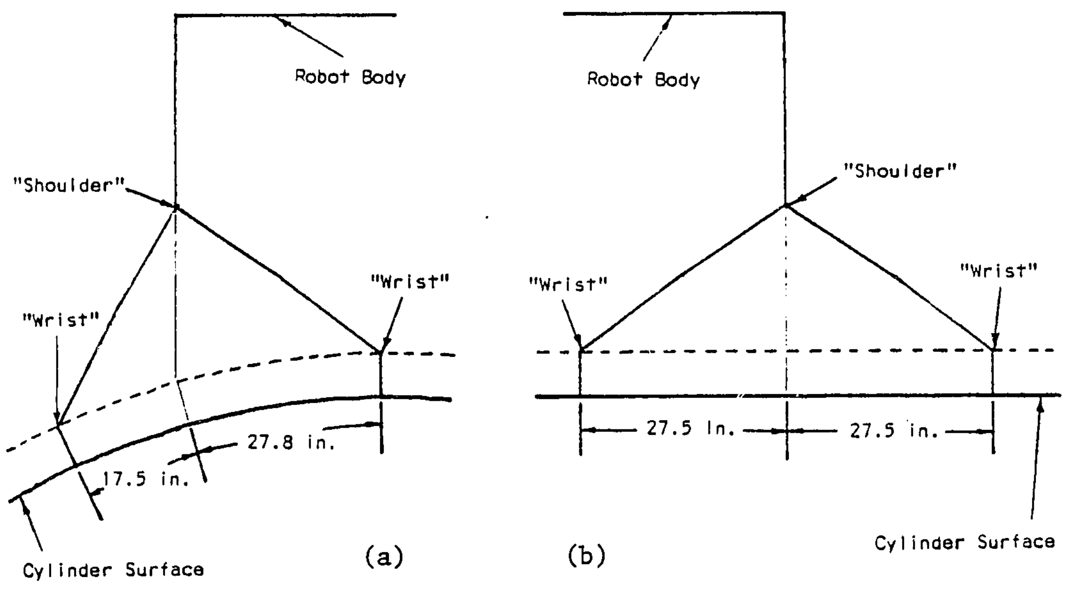

Figure 10. Arm Extension Limits for Vehicle at a Fixed Height (52 in.) on Cylinder. (a) Tangential Limits, (b) Axial Limits.

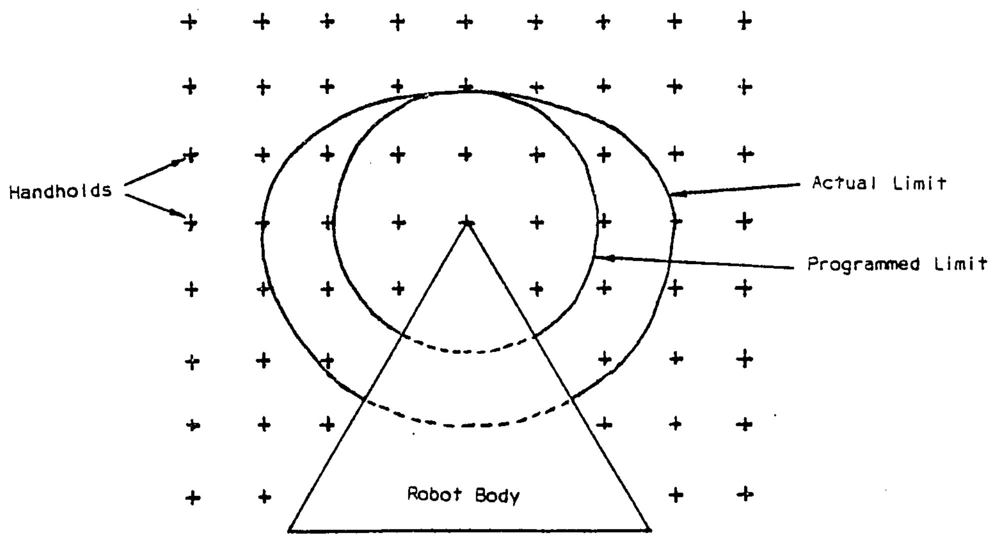

Figure 11. Actual and Programmed Reachable Areas for Arm of Vehicle on Cylinder

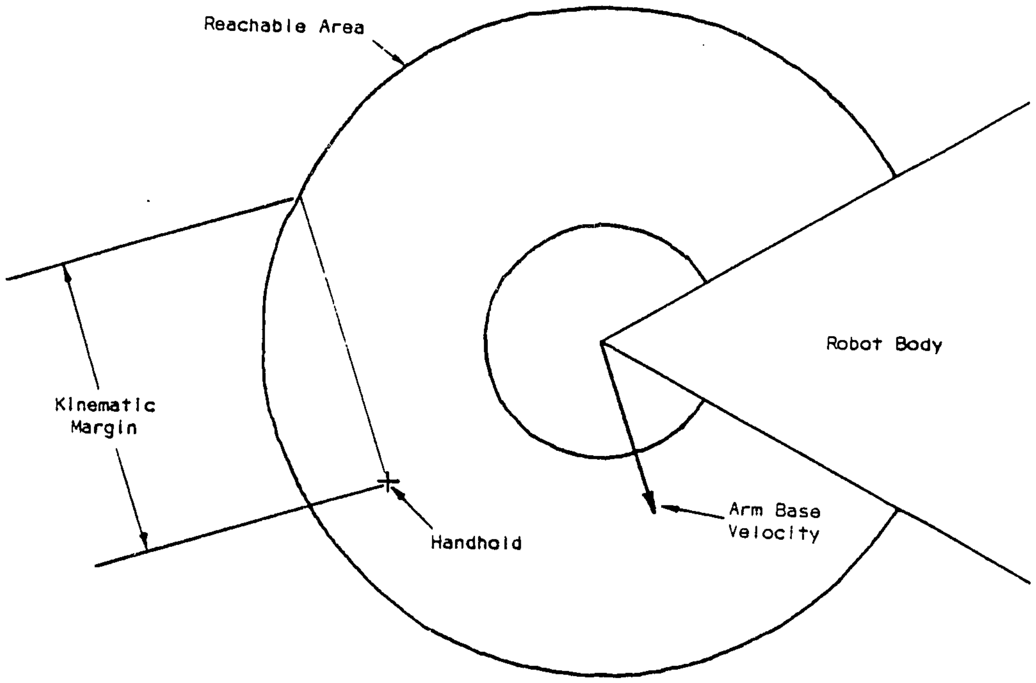

Figure 12. Kinematic Margin Definition Showing Distance Arm Base Can Travel at Present Velocity before Arm Reaches Limit

desired position, $\underline{p}_d$. That velocity and position information is then used with the current actual position, $\underline{p}_a$, to calculate a velocity command, $\underline{v}_c$, for the arm, as

$$\underline{v}_c = \underline{v}_d + (\underline{p}_d - \underline{p}_a)/\Delta t \qquad (5\text{-}1)$$

where Δt is the time interval between successive executions of the free gait algorithm. Thus the command is the sum of the desired velocity and a correction velocity intended to correct the position error in the time interval Δt. The command is then executed and a check is made to see if the arm is at the terminal point of its trajectory. If not, the algorithm moves on to calculate velocity commands for the supporting arms; if so

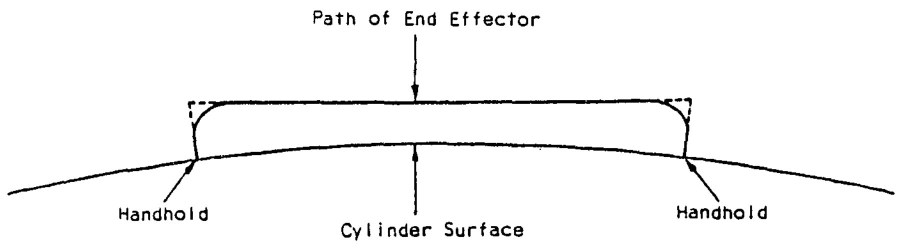

Figure 13. Typical Arm Handhold-to-Handhold Trajectory

the arm is sent a command to grasp the handhold.

If all of the arms are grasping handholds, the algorithm calculates the kinematic margin for each of the arms. The algorithm then checks to see if the minimum kinematic margin over the three arms is less than some operator-determined threshold value. A threshold is used to prevent the arms from lifting and setting down more often than necessary, which would consume more energy than required. If the minimum kinematic margin is larger than the threshold, the algorithm moves on to calculate velocity commands for the supporting arms; if the minimum kinematic margin is smaller than the threshold, the algorithm determines which of the reachable handholds for the arm with the minimum margin has the greatest kinematic margin. It then calculates (as described in Section 4.3.) a trajectory (as shown in Figure 13) for the arm to reach that handhold and lifts the arm.

The final step in the free gait algorithm is calculation of the velocity commands for the supporting arms, which are simply the opposites (in the vector sense) of the desired arm base velocities, which are determined by the desired robot velocity. The algorithm terminates when a stop command is received (for Robot Velocity Mode) or the terminal position is reached (for Robot Destination Mode). Control is then returned to a supervisory program.

5.4. Summary.

This chapter has described the operation of the algorithms used to control the robot's locomotion. The next chapter discusses the performance of the simulation which uses those algorithms.

Chapter 6
RESULTS AND CONCLUSIONS

6.1. Simulation Results.

The algorithms described in Chapters 4 and 5 have been implemented on a PDP 11/45 minicomputer with a Vector General graphics display. All of the routines are written in Fortran except for a few matrix manipulation subroutines, which are written in assembly language. Two photographs of the display which were taken during program execution are reproduced in Figures 14 and 15.

The approximate maximum speeds for the robot with respect to the cylinder are 3 in/sec for translational motion and 0.05 rad/sec for rotational motion. The rotational speed seems slow, but at that speed the end of the arm at full extension is moving 3 in/sec just as with translational motion. The speed is limited to 3 in/sec because at higher speeds the robot's arms cannot maintain positions close enough to the handholds to grasp them. This occurs because the simulation requires that the end of the arm be no more than one inch from a handhold when it grasps it. When the arm's base is moving rapidly with respect

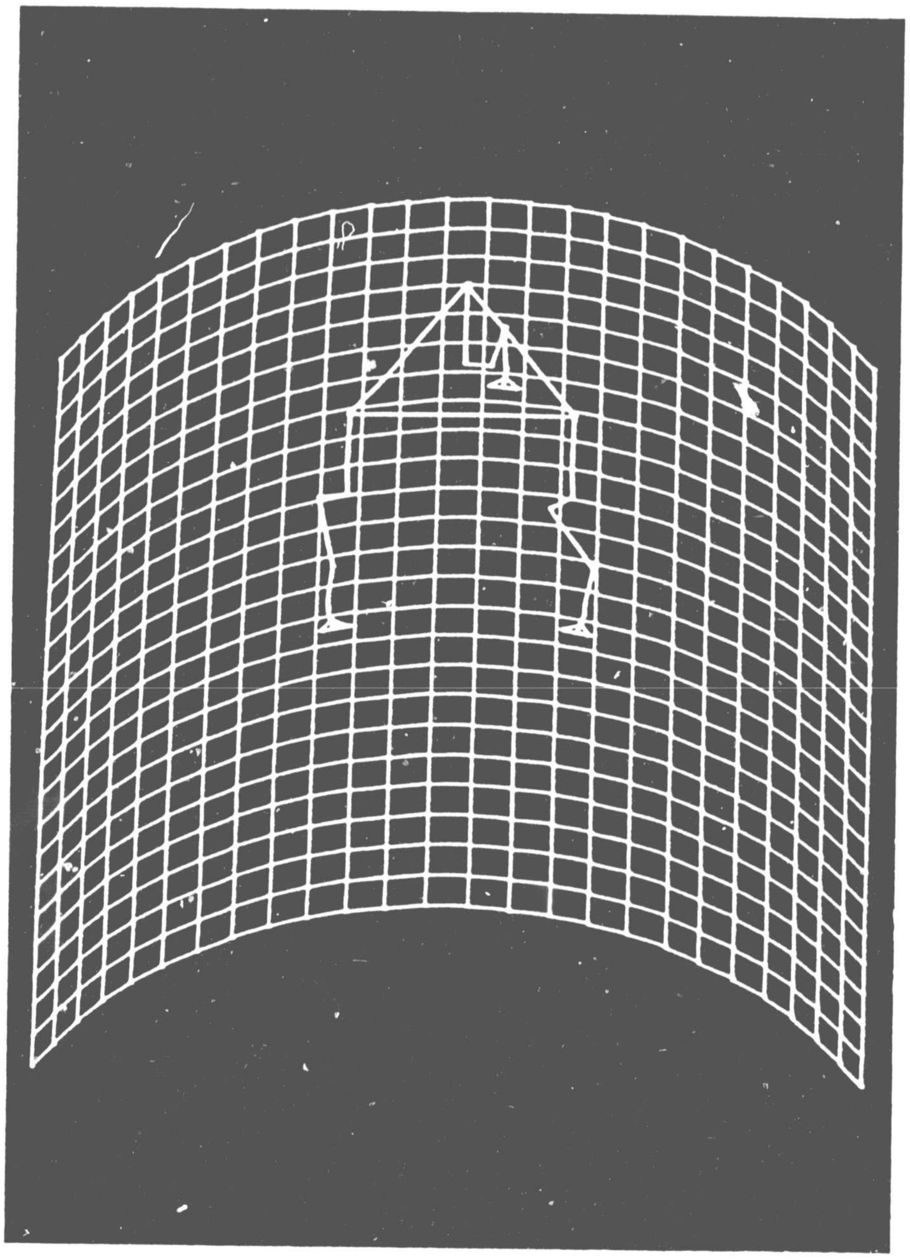

Figure 14. Simulation Display of Vehicle with All Three Arms Grasping Handholds

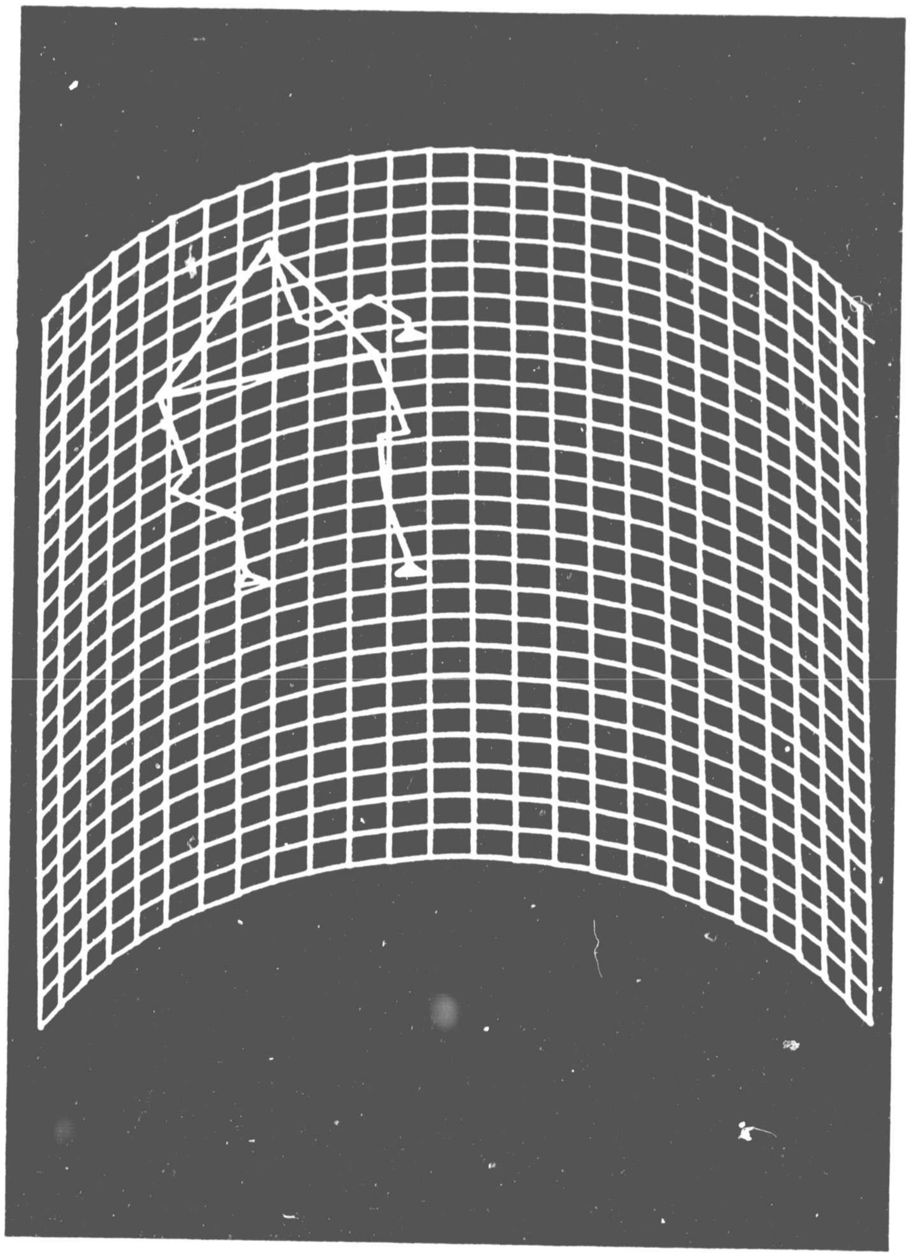

Figure 15. Simulation Display of Vehicle with Arm at Lower Left Moving to New Handhold

to the cylinder, the end effector cannot meet the positional tolerance, so the arm cannot grasp the handhold.

For whatever maximum speed of operation is chosen, rapid changes in robot velocity create situations in which already-selected handholds are no longer practical, or even reachable. This occurs because, since handholds are chosen based on the current arm base velocities, rapid changes in those velocities make the original choices undesirable. This does not create a serious problem except in those cases in which the originally chosen handholds are unreachable after the change in velocity. To deal with this problem, an additional step was added to the free gait algorithm. The added step computes the difference between the time elapsed since the arm was lifted and the expected time for the current arm trajectory which was calculated by the trajectory algorithm. If that difference is more than one second, a new handhold is chosen based on the new arm base velocity. Thus the additional step allows the program to "reconsider" its choices based on changing commands from the operator.

As mentioned in Chapter 3, handholds can be removed from the terrain to simulate obstacles of zero height such as holes or trenches. The robot has successfully crossed trenches as wide as four-fifths of the diameter of an arm's reachable area. This shows that handholds on the terrain do not have to be placed closely together for the robot to be able to move over the surface, so long as the robot has an accurate internal model of where the available handholds are.

6.2. Suggestions for Future Work.

The major weakness of this work is that many of the calculations (that of kinematic margin, for instance) are made by taking advantage of the fact that the terrain is a cylinder. An improvement could be made by using the same basic approach but using methods of calculations which are applicable for more general terrains. This would seem to require use of the "reachable volume" of the arm rather than the artificial "reachable area" approach.

Another improvement could be made in the arms themselves. As discussed in Section 4.2.2., when one or both of the Jacobian matrices for the arms are singular, some velocities are unattainable, which could be a serious problem at times. Use of arms with more than six degrees of freedom would allow Jacobian matrix singularities to be avoided entirely.

At any rate, the robot's success in walking over both terrain with regularly spaced handholds and terrain with obstacles seems to confirm the validity of the basic approach used. Although improvements could be made in the robot's speed and in applying the approach used here to more general terrain, this work shows that a robot similar to the one described may be a practical possibility in the not-too-distant future.

APPENDIX: COMPUTER PROGRAMS

The following pages contain the computer programs which were used to implement the algorithms described in Chapters 4 and 5. Other programs used in the simulation are not included due to space limitations.

```
C
C****
C
C  THIS IS THE MAIN ROBOT CONTROL PROGRAM.
C
C  M.R. PATTERSON
C
C****
C
      INTEGER CYL(25,24),TRANS(12),ARM,ARMSTS(4,3),PROB(3),PARM(3,3),
     +  JSTK(3),NUMSEG,AFLAG(3)
      REAL TH(6,3),THV(6,3),CTH(7,3),STH(7,3),JOINTS(3,8,3),END(3,4,3),
     +  ROBMAT(3,4),IRBMAT(3,4),RADIAL,AXIAL,HEIGHT,ROBVEL(6),RVELC(6),
     +  TIME,TIMINC,VELC(3),TAO,AXO,GAO,TA(4),AX(4),GA(4),ELTIM(4),
     +  RTAVEL(4),RAXVEL(4),RGAVEL(4),INSEGT,RELTIM
      DIMENSION IDISP(3,10),EULER(3),TRMAT(3,4),VECT1(3),VECT2(3)
C
      COMMON CYL
      COMMON /ARMS/ARMSTS
      COMMON /DISP/PROB,PARM
C
      DATA IBEGIN,IDEST,IVELOC,IACTRL,IEXIT/'B','D','V','A','E'/
      DATA LTCLR,LT7,LT14,PI/0,"000400,"000002,3.14159265/
C
C
      CALL VGINI
C
      WRITE(5,100)
  100 FORMAT(///'$THE VG HAS BEEN INITIALIZED.  ENTER ''B'' TO BEGIN: ')
   10 READ(5,101) IREPLY
  101 FORMAT(A1)
      IF(IREPLY.NE.IBEGIN) GOTO 10
C
C*  SET VG TRANSFORMATION MATRIX AND DISPLAY CYLINDER.
C
      CALL VGSEN(3)
      CALL VGSEN(1)
      CALL VGSEN(7,0)
      DO 1 I=1,12
        TRANS(I)=0
    1   CONTINUE
      WRITE(5,200)
  200 FORMAT(//'$ENTER DISPLAY ANGLE (IN DEGREES): ')
      READ(5,201) THETA
  201 FORMAT(F8.3)
      THETA=THETA*PI/180.
      TRANS(1)=32767
      TRANS(5)=32767*COS(THETA)
      TRANS(6)=32767*SIN(THETA)
      TRANS(8)=32767*(-SIN(THETA))
      TRANS(9)=32767*COS(THETA)
      TRANS(12)=16384
      CALL WRCOM(4,12,TRANS)
      CALL VGSEN(4)
C
```

```
      CALL SETHLD
C
C*  SET JOINT ANGLES AND ARM STATES TO THEIR INITIAL VALUES.
C
      DO 2 ARM=1,3
        TH(1,ARM)=0.*PI/180.
        TH(2,ARM)=90.*PI/180.
        TH(3,ARM)=-85.*PI/180.
        TH(4,ARM)=0.*PI/180.
        TH(5,ARM)=-5.*PI/180.
        TH(6,ARM)=0.*PI/180.
        THV(6,ARM)=0.
        ARMSTS(1,ARM)=0
    2   CONTINUE
C
C*  SET ROBOT MATRIX.
C
      WRITE(5,300)
  300 FORMAT(///' ENTER CENTER COORDINATES (IN POINTS RADIAL AND ',
     +  'AXIAL) AND HEIGHT OF'/'$  ROBOT BODY: ')
      READ(5,301) RADIAL,AXIAL,HEIGHT
  301 FORMAT(3F11.4)
      ANGLE=25.+5.*RADIAL
      EULER(1)=-ANGLE
      EULER(2)=90.
      EULER(3)=0.
      CALL MATEUL(ROBMAT(1,1),EULER)
      ROBMAT(1,4)=(100.+HEIGHT)*COS((180.-ANGLE)*PI/180.)
      ROBMAT(2,4)=(100.+HEIGHT)*SIN((180.-ANGLE)*PI/180.)
      ROBMAT(3,4)=-9.25*AXIAL
C
C*  DISPLAY ROBOT BODY.
C
      DO 3 ARM=1,3
        CALL TRIG(TH(1,ARM),CTH(1,ARM),STH(1,ARM))
        CALL POSIT(CTH(1,ARM),STH(1,ARM),JOINTS(1,1,ARM),END(1,1,ARM))
        CALL TRANSM(0,ARM,TRMAT)
        CALL MM3431(4,TRMAT,VECT1,JOINTS(1,1,ARM))
        CALL MM3431(4,ROBMAT,VECT2,VECT1)
        IDISP(1,ARM)=VECT2(1)*100.+8192.
        IDISP(2,ARM)=VECT2(2)*100.+8192.
        IDISP(3,ARM)=VECT2(3)*100.+8192.
    3   CONTINUE
      IDISP(1,4)=IDISP(1,1)
      IDISP(2,4)=IDISP(2,1)
      IDISP(3,4)=IDISP(3,1)
      CALL WRCOM(64,12,IDISP)
      DO 4 I=1,3
        CALL RDCOM(39,1,FROB(I))
        CALL VGSEN(9,4,1)
    4   CONTINUE
C
C*  DISPLAY ARMS.
C
      DO 5 ARM=1,3
```

```
          CALL TRANSM(0,ARM,TRMAT)
          DO 6 I=1,8
            CALL MM3431(4,TRMAT,VECT1,JOINTS(1,I,ARM))
            CALL MM3431(4,ROBMAT,VECT2,VECT1)
            IDISP(1,I)=VECT2(1)*100.+8192.
            IDISP(2,I)=VECT2(2)*100.+8192.
            IDISP(3,I)=VECT2(3)*100.+8192.
    6       CONTINUE
          IDISP(1,9)=IDISP(1,6)
          IDISP(2,9)=IDISP(2,6)
          IDISP(3,9)=IDISP(3,6)
          IDISP(1,10)=IDISP(1,7)
          IDISP(2,10)=IDISP(2,7)
          IDISP(3,10)=IDISP(3,7)
          CALL WRCOM(64,30,IDISP)
          DO 7 I=1,3
            CALL RDCOM(39,1,FARM(ARM,I))
            CALL VGSEN(9,10,1)
    7       CONTINUE
    5     CONTINUE
C
   20 CALL SETDWN(TH,THV,CTH,STH,JOINTS,END,ROBMAT,HEIGHT)
      DO 8 ARM=1,3
        DO 9 J=1,6
          THV(J,ARM)=0.
    9     CONTINUE
    8   CONTINUE
      CALL MOVROB(TH,THV,CTH,STH,JOINTS,END,ROBMAT,TIMINC,AFLAG)
      WRITE(5,400)
  400 FORMAT(///' ENTER ''D'' FOR DESTINATION MODE, ''V'' FOR VELOCITY',
     +  ' MODE, ''A'' FOR ARM'/'$CONTROL MODE, OR ''E'' TO EXIT: ')
   30 READ(5,401) IREPLY
  401 FORMAT(A1)
      IF(IREPLY.EQ.IDEST) GOTO 80
      IF(IREPLY.EQ.IVELOC) GOTO 50
      IF(IREPLY.EQ.IACTRL) GOTO 40
      IF(IREPLY.NE.IEXIT) GOTO 30
      CALL VGSEN(16)
      STOP
C
C
C   **********************
C   *                    *
C   *  ARM CONTROL MODE  *
C   *                    *
C   **********************
C
C
   40 CALL ARMCTL(TH,THV,CTH,STH,JOINTS,END,ROBMAT)
      GOTO 20
C
C
C   *******************
C   *                 *
C   *  VELOCITY MODE  *
```

```
C    *                    *
C    ********************
C
C
   50 TIME=-0.1
   60 CALL RDCOM(17,1,LIGHT)
      CALL WRCOM(17,1,LTCLR)
      IF(LIGHT.EQ.LT7) GOTO 20
      TIMINC=0.1
      TIME=TIME+TIMINC
      DO 11 ARM=1,3
        IF(ARMSTS(1,ARM).EQ.1) GOTO 70
   11   CONTINUE
C
C*  CALCULATE ROBOT POSITION AND VELOCITY.
C
   70 CALL ROBST(ARM,TH(1,ARM),THV(1,ARM),CTH(1,ARM),STH(1,ARM),
     +  JOINTS(1,1,ARM),END(1,1,ARM),ROBMAT,ROBVEL)
C
C*  CALCULATE COMMANDED ROBOT VELOCITIES.
C
      CALL RDCOM(18,3,JSTK)
      VELC(1)=FLOAT(ISIGN(IDIM(IABS(JSTK(1)+1), 640),JSTK(1)))/16000.
      VELC(2)=FLOAT(ISIGN(IDIM(IABS(JSTK(2)+1), 640),JSTK(2)))/16000.
      VELC(3)=FLOAT(ISIGN(IDIM(IABS(JSTK(3)+1),1280),JSTK(3)))/640000.
      CALL RVEL(ROBMAT,ROBVEL,HEIGHT,TIMINC,VELC,RVELC)
C
      CALL INVM34(IRBMAT,ROBMAT)
      CALL ARMS(TH,THV,CTH,STH,JOINTS,END,ROBMAT,HEIGHT,IRBMAT,RVELC,
     +  TIME,TIMINC)
      CALL MOVROB(TH,THV,CTH,STH,JOINTS,END,ROBMAT,TIMINC,AFLAG)
      IF(MAX0(AFLAG(1),AFLAG(2),AFLAG(3)).NE.0) GOTO 220
      GOTO 60
C
C
C    **********************
C    *                    *
C    *  DESTINATION MODE  *
C    *                    *
C    **********************
C
C
C*  READ SEQUENCE OF POINTS.
C
   80 DO 12 ARM=1,3
        IF(ARMSTS(1,ARM).EQ.1) GOTO 90
   12   CONTINUE
   90 CALL ROBST(ARM,TH(1,ARM),THV(1,ARM),CTH(1,ARM),STH(1,ARM),
     +  JOINTS(1,1,ARM),END(1,1,ARM),ROBMAT,ROBVEL)
      TAO=((PI-ATAN2(ROBMAT(2,4),ROBMAT(1,4)))*180./PI-25.)/5.
      AXO=ROBMAT(3,4)/-9.25
      CALL EULMAT(EULER,ROBMAT)
      GAO=EULER(3)
      WRITE(5,500) TAO,AXO,GAO
  500 FORMAT(///' THE ROBOT IS NOW AT ',3F11.4)
```

```
      WRITE(5,600)
  600 FORMAT(///'$ENTER THE NUMBER OF SEGMENTS: ')
      READ(5,601) NUMSEG
  601 FORMAT(I3)
      IF(NUMSEG.EQ.1) GOTO 110
      DO 13 I=1,NUMSEG-1
        WRITE(5,700) I
  700   FORMAT(//'$ENTER POINT ',I1,': ')
        READ(5,701) TA(I),AX(I),GA(I)
  701   FORMAT(3F11.4)
   13   CONTINUE
  110 WRITE(5,800)
  800 FORMAT(//'$ENTER TERMINAL POINT: ')
      READ (5,801) TA(NUMSEG),AX(NUMSEG),GA(NUMSEG)
  801 FORMAT(3F11.4)
C
C*  CALCULATE SEGMENT TIMES AND ACCELERATIONS.
C
      TIME1=ABS(TA(1)-TA0)/.25
      TIME2=ABS(AX(1)-AX0)/.25
      TIME3=ABS(GA(1)-GA0)/1.
      ELTIM(1)=AMAX1(TIME1,TIME2,TIME3)
      RTAVEL(1)=(TA(1)-TA0)/ELTIM(1)
      RAXVEL(1)=(AX(1)-AX0)/ELTIM(1)
      RGAVEL(1)=(GA(1)-GA0)/ELTIM(1)
      IF(NUMSEG.EQ.1) GOTO 120
      DO 14 ISEG=2,NUMSEG
        TIME1=ABS(TA(ISEG)-TA(ISEG-1))/.25
        TIME2=ABS(AX(ISEG)-AX(ISEG-1))/.25
        TIME3=ABS(GA(ISEG)-GA(ISEG-1))/1.
        ELTIM(ISEG)=ELTIM(ISEG-1)+AMAX1(TIME1,TIME2,TIME3)
        RTAVEL(ISEG)=(TA(ISEG)-TA(ISEG-1))/(ELTIM(ISEG)-ELTIM(ISEG-1))
        RAXVEL(ISEG)=(AX(ISEG)-AX(ISEG-1))/(ELTIM(ISEG)-ELTIM(ISEG-1))
        RGAVEL(ISEG)=(GA(ISEG)-GA(ISEG-1))/(ELTIM(ISEG)-ELTIM(ISEG-1))
   14   CONTINUE
C
C
  120 TIME=-0.1
  130 CALL RDCOM(17,1,LIGHT)
      CALL WRCOM(17,1,LTCLR)
      IF(LIGHT.EQ.LT7) GOTO 20
      TIMINC=0.1
      TIME=TIME+TIMINC
C
C*  CALCULATE ROBOT POSITION.
C
      DO 15 ARM=1,3
        IF(ARMSTS(1,ARM).EQ.1) GOTO 140
   15   CONTINUE
  140 CALL ROBST(ARM,TH(1,ARM),THV(1,ARM),CTH(1,ARM),STH(1,ARM),
     +  JOINTS(1,1,ARM),END(1,1,ARM),ROBMAT,ROBVEL)
      TAA=((PI-ATAN2(ROBMAT(2,4),ROBMAT(1,4)))*180./PI-25.)/5.
      AXA=ROBMAT(3,4)/-9.25
      CALL EULMAT(EULER,ROBMAT)
      GAA=EULER(3)
```

```
C
C*  CALCULATE COMMANDED ROBOT VELOCITIES.
C
      INSEGT=0.
      ISEG=1
  150 IF(TIME.LT.ELTIM(ISEG)) GOTO 160
        IF(ISEG.EQ.NUMSEG) GOTO 190
          INSEGT=ELTIM(ISEG)
          ISEG=ISEG+1
          GOTO 150
  160 RELTIM=TIME-INSEGT
      IF(ISEG.EQ.1) GOTO 170
        TAD=TA(ISEG-1)+RTAVEL(ISEG)*RELTIM
        AXD=AX(ISEG-1)+RAXVEL(ISEG)*RELTIM
        GAD=GA(ISEG-1)+RGAVEL(ISEG)*RELTIM
        GOTO 180
  170   TAD=TAO+RTAVEL(1)*RELTIM
        AXD=AXO+RAXVEL(1)*RELTIM
        GAD=GAO+RGAVEL(1)*RELTIM
  180 VELC(1)=RTAVEL(ISEG)+(1./TIMINC)*(TAD-TAA)
      VELC(2)=RAXVEL(ISEG)+(1./TIMINC)*(AXD-AXA)
      VELC(3)=(RGAVEL(ISEG)+(1./TIMINC)*(GAD-GAA))*PI/180.
      GOTO 210
  190 VELC(1)=(1./TIMINC)*(TA(NUMSEG)-TAA)
      VELC(2)=(1./TIMINC)*(AX(NUMSEG)-AXA)
      VELC(3)=(1./TIMINC)*(GA(NUMSEG)-GAA)*PI/180.
      IF((ABS(VELC(1)).GT.1.).OR.(ABS(VELC(2)).GT.1).OR.
     +  (ABS(VELC(3)).GT.1.)) GOTO 210
        WRITE(5,900)
  900   FORMAT(///' TERMINAL POSITION REACHED.')
        GOTO 20
  210 CALL RVEL(ROBMAT,ROBVEL,HEIGHT,TIMINC,VELC,RVELC)
C
C*  MOVE ROBOT.
C
      CALL INVM34(IRBMAT,ROBMAT)
      CALL ARMS(TH,THV,CTH,STH,JOINTS,END,ROBMAT,HEIGHT,IRBMAT,RVELC,
     +  TIME,TIMINC)
      CALL MOVROB(TH,THV,CTH,STH,JOINTS,END,ROBMAT,TIMINC,AFLAG)
      IF(MAX0(AFLAG(1),AFLAG(2),AFLAG(3)).NE.0) GOTO 220
      GOTO 130
C
  220 DO 16 ARM=1,3
        IF(AFLAG(ARM).EQ.0) GOTO 16
        WRITE(5,1100) AFLAG(ARM),ARM
 1100   FORMAT(///' JOINT',I2,' OF ARM',I2,' IS OUT OF LIMITS.')
        GOTO 20
   16   CONTINUE
C
      END
```

```
C
C****
C
C  THIS SUBROUTINE CONTROLS THE ARMS' MOTIONS WHEN THE ROBOT IS WALKING.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE ARMS(TH,THV,CTH,STH,JOINTS,END,ROBMAT,HEIGHT,IRBMAT,
     +  RVELC,TIME,TIMINC)
C
      INTEGER ARM,ARMSTS(4,3),TSEGCT,RSEGCT,TERFLG,HDHLD(2),
     +  HDHLDS(2,48),NUMSEG
      REAL TH(6,3),THV(6,3),CTH(7,3),STH(7,3),JOINTS(3,8,3),END(3,4,3),
     +  ROBMAT(3,4),HEIGHT,IRBMAT(3,4),RVELC(6),TIME,TIMINC,INTIME,
     +  ELTIME,EXTEND(3,4),ELTTIM(36),TACC(3,36),INTVEL(3,36),
     +  INTPOS(3,36),ELRTIM(12),RACC(12),INRPOS(9,12),ROTVEC(3,12),
     +  TERPOS(3,4),EXTVEL(6),EXISEG,POINTS(3,4,6),VEL(6)
      DIMENSION TRMAT(3,4),OTRMAT(3,4)
C
      COMMON /ARMS/ARMSTS
      COMMON /PATH/INTIME,TSEGCT,ELTTIM,TACC,INTVEL,INTPOS,RSEGCT,
     +  ELRTIM,RACC,INRPOS,ROTVEC,TERPOS
C
C
C*  FIND ARM IN AIR.
C
      DO 1 ARM=1,3
        IF(ARMSTS(1,ARM).EQ.0) GOTO 10
    1   CONTINUE
      GOTO 30
C
C*  ARM IS IN TRANSFER PHASE.
C
   10 ELTIME=TIME-INTIME
      FLAG=0.
      IF((ELTIME-ELTTIM(TSEGCT)).LT.1.) GOTO 20
        FLAG=1.
        GOTO 40
   20 CALL TRANSM(0,ARM,TRMAT)
      CALL MM3434(OTRMAT,ROBMAT,TRMAT)
      CALL MM3434(EXTEND,OTRMAT,END(1,1,ARM))
      CALL VELCOM(EXTEND,TSEGCT,ELTTIM,TACC,INTVEL,INTPOS,RSEGCT,
     +  ELRTIM,RACC,INRPOS,ROTVEC,TERPOS,ELTIME,TIMINC,EXTVEL,TERFLG)
      IF(TERFLG.EQ.0) GOTO 60
        ARMSTS(1,ARM)=1
C
C*  ALL ARMS ARE SUPPORTING.
C
   30 EXSGMN=99.
      DO 2 ARM=1,3
        CALL EXIST(ARM,ROBMAT,HEIGHT,RVELC,ARMSTS(2,ARM),EXISEG)
        IF(EXISEG.GE.EXSGMN) GOTO 2
          EXSGMN=EXISEG
          MESARM=ARM
    2   CONTINUE
      IF(EXSGMN.GT.1.) GOTO 70
      ARM=MESARM
C
C*  CALCULATE NEW TRAJECTORY.
C
   40 CALL TRANSM(0,ARM,TRMAT)
      CALL MM3434(OTRMAT,ROBMAT,TRMAT)
      CALL MM3434(EXTEND,OTRMAT,END(1,1,ARM))
      CALL HDHOLD(ARM,EXTEND,ROBMAT,HEIGHT,HDHLDS,NOHHS)
      IF(NOHHS.EQ.0) GOTO 70
      EXSGMX=-1.
      DO 3 I=1,NOHHS
        CALL EXIST(ARM,ROBMAT,HEIGHT,RVELC,HDHLDS(1,1),EXISEG)
        IF(EXISEG.LE.EXSGMX) GOTO 3
          EXSGMX=EXISEG
          MXESHH=I
    3   CONTINUE
      IF(FLAG.EQ.1.) GOTO 50
      IF(EXSGMX.LE.EXSGMN) GOTO 70
   50 HDHLD(1)=HDHLDS(1,MXESHH)
      HDHLD(2)=HDHLDS(2,MXESHH)
C
      CALL HHDPTH(ARM,OTRMAT,HDHLD,POINTS,NUMSEG)
      DO 4 I=1,4
      DO 5 J=1,3
        TERPOS(J,I)=POINTS(J,I,NUMSEG)
    5   CONTINUE
    4   CONTINUE
      CALL ARMPTH(EXTEND,NUMSEG,POINTS,TSEGCT,ELTTIM,TACC,INTVEL,INTPOS,
     +  RSEGCT,ELRTIM,RACC,INRPOS,ROTVEC)
C
      ARMSTS(1,ARM)=0
      INTIME=TIME
      GOTO 70
C
C*  MOVE TRANSFER PHASE ARM.
C
   60 CALL AVEL(ARM,ROBMAT,IRBMAT,RVELC,EXTVEL,VEL)
      CALL SOLVE(TH(1,ARM),THV(1,ARM),CTH(1,ARM),STH(1,ARM),
     +  JOINTS(1,1,ARM),END(1,1,ARM),VEL)
C
C*  MOVE SUPPORT PHASE ARMS.
C
   70 DO 6 ARM=1,3
        IF(ARMSTS(1,ARM).NE.1) GOTO 6
        CALL HLDPOS(ARM,END(1,1,ARM),ROBMAT,TIMINC,EXTVEL)
        CALL AVEL(ARM,ROBMAT,IRBMAT,RVELC,EXTVEL,VEL)
        CALL SOLVE(TH(1,ARM),THV(1,ARM),CTH(1,ARM),STH(1,ARM),
     +    JOINTS(1,1,ARM),END(1,1,ARM),VEL)
    6   CONTINUE
C
      RETURN
      END
```

```
C
C****
C
C  THIS SUBROUTINE RETURNS A LIST OF REACHABLE HANDHOLDS FOR A
C    GIVEN ARM.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE HDHOLD(ARM,EXTEND,ROBMAT,HEIGHT,HDHLDS,NOHHS)
C
      INTEGER ARM,HDHLDS(2,48),NOHHS,CYL(25,24),ARMSTS(4,3)
      REAL EXTEND(3,4),ROBMAT(3,4),HEIGHT,NO,SN,ES,HW,TRM33(9,3),
     +  TRM31(3,6)
      DIMENSION XYZ(3),TA(2),TACHH(2),TAHH(2),TAROB(2)
C
      COMMON CYL
      COMMON /ARMS/ARMSTS
      COMMON /ARMPAR/NO,SN,ES,HW
      COMMON /TRMATS/TRM33,TRM31
C
      DATA PI/3.14159265/
C
C
C*  CALCULATE ARM BASE COORDINATES IN (T,A).
C
      CALL MM3431(4,ROBMAT,XYZ,TRM31(1,ARM))
      CALL TACALC(ROBMAT,XYZ,TA)
      CALL TACALC(ROBMAT,EXTEND(1,4),TACHH)
C
C*  DETERMINE HANDHOLD LIST.
C
      CENDST=SQRT((100.+HEIGHT-NO)**2+TRM31(1,1)**2)
      ANGLE=ATAN(TRM31(1,1)/(100.+HEIGHT-NO))
      CRANGL=((100.+HW)**2+CENDST**2-(2.*ES)**2)/(2.*CENDST*(100.+HW))
      RANGLE=ATAN(SQRT(1.-CRANGL**2)/CRANGL)
      STANGL=TRM31(1,1)/(100.+HW)
      TANGLE=ATAN(STANGL/SQRT(1.-STANGL**2))
      RADIUS=(ANGLE+RANGLE-TANGLE)*(100.+HW)/9.25
C
      INDEX=1
      DO 1 I=1,25
      DO 2 J=1,24
        IF(CYL(I,J).NE.1) GOTO 2
        IF((ARMSTS(2,ARM).EQ.I).AND.(ARMSTS(3,ARM).EQ.J)) GOTO 2
        TAHH(1)=FLOAT(I)
        TAHH(2)=FLOAT(J)
        IF(ABS(TAHH(1)-TA(1)).GT.RADIUS) GOTO 2
        IF(ABS(TAHH(2)-TA(2)).GT.RADIUS) GOTO 2
        HHRAD2=(TAHH(1)-TA(1))**2+(TAHH(2)-TA(2))**2
        HHRAD=SQRT(HHRAD2)
        IF(HHRAD.GT.RADIUS) GOTO 2
        IF(HHRAD.LT.(SN+2.)/9.25) GOTO 2
        CALL TACALC(ROBMAT,ROBMAT(1,4),TAROB)
```

```
          DRB=SQRT((TA(1)-TAROB(1))**2+(TA(2)-TAROB(2))**2)
          DHH=HHRAD
          DTPROD=(TA(1)-TAROB(1))*(TAHH(1)-TA(1))+
     +      (TA(2)-TAROB(2))*(TAHH(2)-TA(2))
          IF((DTPROD/DRB/DHH).LT.-0.707) GOTO 2
          A12=(TAHH(1)-TACHH(1))**2+(TAHH(2)-TACHH(2))**2
          B12=(TACHH(1)-TA(1))**2+(TACHH(2)-TA(2))**2
          C12=HHRAD2
          A1=SQRT(A12)
          B1=SQRT(B12)
          C1=HHRAD
          COSB=(B12-C12-A12)/(-2.*A1*C1)
          COSC=(C12-A12-B12)/(-2.*A1*B1)
          IF((COSB.GT.0.).AND.(COSC.GT.0.)) GOTO 10
            IF(C1.LT.(SN+2.)/9.25) GOTO 2
            GOTO 20
   10       DIST=C1*SQRT(1.-COSB**2)
            IF(DIST.LT.(SN+2.)/9.25) GOTO 2
   20     A22=A12
          B22=(TACHH(1)-TAROB(1))**2+(TACHH(2)-TAROB(2))**2
          C22=(TAHH(1)-TAROB(1))**2+(TAHH(2)-TAROB(2))**2
          A2=A1
          B2=SQRT(B22)
          C2=SQRT(C22)
          COSB=(B22-C22-A22)/(-2.*A2*C2)
          COSC=(C22-A22-B22)/(-2.*A2*B2)
          IF((COSB.GT.0.).AND.(COSC.GT.0.)) GOTO 30
            IF(C2.LT.(TRM31(1,1)-SN-2.)/9.25) GOTO 2
            GOTO 40
   30       DIST=C2*SQRT(1.-COSB**2)
            IF(DIST.LT.(TRM31(1,1)-SN-2.)/9.25) GOTO 2
C
   40       HDHLDS(1,INDEX)=1
            HDHLDS(2,INDEX)=J
            INDEX=INDEX+1
    2     CONTINUE
    1     CONTINUE
C
        NOHHS=INDEX-1
C
        RETURN
        END
```

```
C
C****
C
C  THIS SUBROUTINE CALCULATES THE EXISTENCE SEGMENT FOR A GIVEN
C    HANDHOLD AND ARM.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE EXIST(ARM,ROBMAT,HEIGHT,RVELC,HNDHLD,EXISEG)
C
      INTEGER ARM,HNDHLD(2)
      REAL ROBMAT(3,4),HEIGHT,RVELC(6),EXISEG,NO,SN,ES,HW,TRM33(9,3),
     +  TRM31(3,6)
      DIMENSION XYZ(3),XYZO(3,2),TA(2),TAO(2,2),XYZV(3),TAV(2),
     +  VXYO(2,2),TAHH(2),VXYHH(2),VECT(3)
C
      COMMON /ARMPAR/NO,SN,ES,HW
      COMMON /TRMATS/TRM33,TRM31
C
      DATA PI/3.14159265/
C
C
C*  CALCULATE ARM BASE POSITIONS IN (T,A).
C
      CALL MM3431(4,ROBMAT,XYZ,TRM31(1,ARM))
      INDEX=1
      DO 1 I=1,3
        IF(I.EQ.ARM) GOTO 1
        CALL MM3431(4,ROBMAT,XYZO(1,INDEX),TRM31(1,I))
        INDEX=2
    1   CONTINUE
C
      TH=ATAN2(XYZ(2),XYZ(1))
      CALL TACALC(ROBMAT,XYZ,TA)
      CALL TACALC(ROBMAT,XYZO(1,1),TAO(1,1))
      CALL TACALC(ROBMAT,XYZO(1,2),TAO(1,2))
C
C*  CALCULATE ARM BASE VELOCITY IN (T,A).
C
      CALL MM3431(3,ROBMAT,VECT,TRM31(1,ARM))
      XYZV(1)=RVELC(5)*VECT(3)-RVELC(6)*VECT(2)+RVELC(1)
      XYZV(2)=RVELC(6)*VECT(1)-RVELC(4)*VECT(3)+RVELC(2)
      XYZV(3)=RVELC(4)*VECT(2)-RVELC(5)*VECT(1)+RVELC(3)
      TAV(1)=(SIN(TH)*XYZV(1)-COS(TH)*XYZV(2))/9.25
      TAV(2)=-XYZV(3)/9.25
C
C*  CONVERT POINTS TO (VX,VY).
C
      PHI=ATAN2(TAV(2),TAV(1))
      DO 2 I=1,2
        VXYO(1,I)=SIN(PHI)*(TAO(1,I)-TA(1))-COS(PHI)*(TAO(2,I)-TA(2))
        IF(VXYO(1,I).EQ.0.) VXYO(1,I)=1.0E-8
        VXYO(2,I)=COS(PHI)*(TAO(1,I)-TA(1))+SIN(PHI)*(TAO(2,I)-TA(2))
    2   CONTINUE
      TAHH(1)=FLOAT(HNDHLD(1))
      TAHH(2)=FLOAT(HNDHLD(2))
      VXYHH(1)=SIN(PHI)*(TAHH(1)-TA(1))-COS(PHI)*(TAHH(2)-TA(2))
      VXYHH(2)=COS(PHI)*(TAHH(1)-TA(1))+SIN(PHI)*(TAHH(2)-TA(2))
C
C*  CALCULATE THE FOUR ARM LIMITS.
C
      YLIM1=VXYHH(1)/VXYO(1,1)*VXYO(2,1)
      IF(((VXYHH(1)*VXYO(1,1)).LT.0.).OR.(VXYHH(2).LT.YLIM1)) YLIM1=-99.
C
      YLIM2=VXYHH(1)/VXYO(1,2)*VXYO(2,2)
      IF(((VXYHH(1)*VXYO(1,2)).LT.0.).OR.(VXYHH(2).LT.YLIM2)) YLIM2=-99.
C
      RAD1=(SN+2.)/9.25
      IF(ABS(VXYHH(1)).GT.RAD1) GOTO 10
      YLIM3=SQRT(RAD1**2-VXYHH(1)**2)
   10 IF((ABS(VXYHH(1)).GT.RAD1).OR.(VXYHH(2).LT.0.)) YLIM3=-99.
C
      CENDST=SQRT((100.+HEIGHT-NO)**2+TRM31(1,1)**2)
      ANGLE=ATAN(TRM31(1,1)/(100.+HEIGHT-NO))
      CRANGL=((100.+HW)**2+CENDST**2-(2.*ES)**2)/(2.*CENDST*(100.+HW))
      RANGLE=ATAN(SQRT(1.-CRANGL**2)/CRANGL)
      STANGL=TRM31(1,1)/(100.+HW)
      TANGLE=ATAN(STANGL/SQRT(1.-STANGL**2))
      RAD2=(ANGLE+RANGLE-TANGLE)*(100.+HW)/9.25
      YLIM4=-SQRT(RAD2**2-VXYHH(1)**2)
C
C*  CALCULATE EXISTENCE SEGMENT.
C
      EXISEG=VXYHH(2)-AMAX1(YLIM1,YLIM2,YLIM3,YLIM4)
C
      RETURN
      END
```

```
C
C****
C
C  THIS SUBROUTINE CALCULATES THE POINTS THROUGH WHICH AN ARM CAN
C    PASS FOR A TRAJECTORY WHICH WILL TAKE IT TO THE GIVEN HANDHOLD.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE HHDPTH(ARM,OTRMAT,HNDHLD,POINTS,NUMSEG)
C
      INTEGER ARM,ARMSTS(4,3),HNDHLD(2),NUMSEG
      REAL OTRMAT(3,4),POINTS(3,4,6)
      DIMENSION EULER(3)
C
      COMMON /ARMS/ARMSTS
C
      DATA PI/3.14159265/
C
C
      INDEX=1
C
      IF(ARMSTS(1,ARM).NE.1) GOTO 10
        ANGLE=25.+5.*ARMSTS(2,ARM)
        POINTS(1,4,INDEX)=103.*COS((180.-ANGLE)*PI/180.)
        POINTS(2,4,INDEX)=103.*SIN((180.-ANGLE)*PI/180.)
        POINTS(3,4,INDEX)=-9.25*ARMSTS(3,ARM)
        EULER(1)=-ANGLE
        EULER(2)=90.
        EULER(3)=0.
        CALL MATEUL(POINTS(1,1,INDEX),EULER)
        INDEX=INDEX+1
C
   10 ANGLE=25.+5.*HNDHLD(1)
      POINTS(1,4,INDEX)=103.*COS((180.-ANGLE)*PI/180.)
      POINTS(2,4,INDEX)=103.*SIN((180.-ANGLE)*PI/180.)
      POINTS(3,4,INDEX)=-9.25*HNDHLD(2)
      EULER(1)=-ANGLE
      EULER(2)=90.
      EULER(3)=0.
      CALL MATEUL(POINTS(1,1,INDEX),EULER)
      INDEX=INDEX+1
C
      POINTS(1,4,INDEX)=100.*COS((180.-ANGLE)*PI/180.)
      POINTS(2,4,INDEX)=100.*SIN((180.-ANGLE)*PI/180.)
      POINTS(3,4,INDEX)=-9.25*HNDHLD(2)
      EULER(1)=-ANGLE
      EULER(2)=90.
      EULER(3)=0.
      CALL MATEUL(POINTS(1,1,INDEX),EULER)
C
      ARMSTS(2,ARM)=HNDHLD(1)
      ARMSTS(3,ARM)=HNDHLD(2)
      ARMSTS(4,ARM)=0
      NUMSEG=INDEX
C
      RETURN
      END
```

```
C
C****
C
C  THIS SUBROUTINE CALCULATES THE TIMES AND ACCELERATIONS REQUIRED TO
C    MOVE AN ARM THROUGH A GIVEN PATH.  THE TIMES AND ACCELERATIONS
C    ARE CALCULATED SUBJECT TO MAXIMUM ACCELERATION AND VELOCITY
C    CONSTRAINTS.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE ARMPTH(END,NUMSEG,POINTS,TSEGCT,ELTTIM,TACC,INTVEL,
     +  INTPOS,RSEGCT,ELRTIM,RACC,INRPOS,ROTVEC)
C
      INTEGER NUMSEG,TSEGCT,RSEGCT
      REAL END(3,4),POINTS(3,4,6),ELTTIM(36),TACC(3,36),INTVEL(3,36),
     +  INTPOS(3,36),ELRTIM(12),RACC(12),INRPOS(9,12),ROTVEC(3,12),
     +  TRVEL(7),TRTIME(7),TRDIST(7),USX(6),USY(6),USZ(6),TSEGTM(6,5),
     +  TSEGAC(6,5),RSEGAC(6),RVEC(3,6),TRTMIN(6),ROTMIN(6),ERRMAX,
     +  MAXTRV,MAXTRA,MAXROV,MAXROA
      DIMENSION SX(6),SY(6),SZ(6),DSEG(6),UNACC(7),COSB(7),TRDMAX(7),
     +  SQTRV(7),REQSGA(6),CHANGE(6),DINSEG(6),DMATCH(6),SGVMX2(6),
     +  SEGVMX(6),ROTMAT(3,3),ROTDIF(3),PHI(6),ROTSUM(3)
C
      COMMON /PTHPAR/ERRMAX,MAXTRV,MAXTRA,MAXROV,MAXROA
C
C
      NUMTR=NUMSEG+1
C
C*  CALCULATE SEGMENT VECTORS AND UNIT VECTORS.
C
      SX(1)=POINTS(1,4,1)-END(1,4)
      SY(1)=POINTS(2,4,1)-END(2,4)
      SZ(1)=POINTS(3,4,1)-END(3,4)
      IF(NUMSEG.LT.2) GOTO 10
      DO 1 I=2,NUMSEG
        SX(I)=POINTS(1,4,I)-POINTS(1,4,I-1)
        SY(I)=POINTS(2,4,I)-POINTS(2,4,I-1)
        SZ(I)=POINTS(3,4,I)-POINTS(3,4,I-1)
    1   CONTINUE
C
   10 DO 2 I=1,NUMSEG
        DSEG(I)=SQRT(SX(I)**2+SY(I)**2+SZ(I)**2)
        IF(DSEG(I).GE.1.0E-6) GOTO 20
          DSEG(I)=0.
          USX(I)=0.
          USY(I)=0.
          USZ(I)=0.
          GOTO 2
   20     USX(I)=SX(I)/DSEG(I)
          USY(I)=SY(I)/DSEG(I)
          USZ(I)=SZ(I)/DSEG(I)
    2   CONTINUE
C
```

```
C*  CALCULATE TRANSITION PARAMETERS.
C
       UNACC(1)=2.
       UNACC(NUMTR)=2.
       IF((NUMTR-1).LT.2) GOTO 30
       DO 3 I=2,NUMTR-1
         COSB(I)=-(USX(I)*USX(I-1)+USY(I)*USY(I-1)+USZ(I)*USZ(I-1))
         UNACC(I)=SQRT(2.*(1.+COSB(I)))
    3    CONTINUE
C
C*  CALCULATE MAXIMUM TRANSITION DISTANCES.
C
   30 TRDIST(1)=0.
       TRDIST(NUMTR)=0.
       IF((NUMTR-1).LT.2) GOTO 40
       DO 4 I=2,NUMTR-1
         TRDMAX(I)=4.*ERRMAX/UNACC(I)
         TRDIST(I)=AMIN1(DSEG(I-1)/2.,DSEG(I)/2.,TRDMAX(I))
    4    CONTINUE
C
C*  CALCULATE SQUARES OF MAXIMUM TRANSITION VELOCITIES.
C
   40 DO 5 I=1,NUMTR
         SQTRV(I)=2.*TRDIST(I)*MAXTRA/UNACC(I)
    5    CONTINUE
C
C*  CALCULATE REQUIRED SEGMENT ACCELERATIONS.
C
   50 DO 6 I=1,NUMSEG
         IF((DSEG(I)-TRDIST(I+1)-TRDIST(I)).NE.0.) GOTO 70
           IF((SQTRV(I+1)-SQTRV(I)).NE.0.) GOTO 60
             REQSGA(I)=0.
             GOTO 80
   60        REQSGA(I)=1.0E+6
             GOTO 80
   70      REQSGA(I)=ABS((SQTRV(I+1)-SQTRV(I))/(2.*(DSEG(I)-
     +     TRDIST(I+1)-TRDIST(I))))
   80    IF(I.NE.1) GOTO 90
           SEGAMX=REQSGA(I)
           INSAMX=I
   90    IF(REQSGA(I).LE.SEGAMX) GOTO 6
           SEGAMX=REQSGA(I)
           INSAMX=I
    6    CONTINUE
C
C*  IF GREATEST SEGMENT ACCELERATION IS GREATER THAN MAXIMUM ALLOWABLE
C*   ACCELERATION, RECALCULATE TRANSITION VELOCITIES AND DISTANCES.
C
       IF((SEGAMX-MAXTRA).LE.1.0E-4) GOTO 130
         IF(SQTRV(INSAMX).GT.SQTRV(INSAMX+1)) GOTO 110
           ILO=INSAMX
           IHI=INSAMX+1
           GOTO 120
  110      IHI=INSAMX
           ILO=INSAMX+1
  120   SQTRV(IHI)=(SQTRV(ILO)+2.*MAXTRA*(DSEG(INSAMX)-TRDIST(ILO)))/
     +     (1.+UNACC(IHI))
        TRDIST(IHI)=SQTRV(IHI)*UNACC(IHI)/(2.*MAXTRA)
        GOTO 50
C
C*  CALCULATE TRANSITION VELOCITIES AND TIMES.
C
  130 TRVEL(I)=0.
      TRVEL(NUMTR)=0.
      IF((NUMTR-1).LT.2) GOTO 150
      DO 7 I=2,NUMTR-1
        TRVEL(I)=SQRT(SQTRV(I))
        IF(TRVEL(I).NE.0.) GOTO 140
          TRTIME(I)=0.
          GOTO 7
  140     TRTIME(I)=2.*TRDIST(I)/TRVEL(I)
    7   CONTINUE
C
C*  CALCULATE MINIMUM SEGMENT TIMES.
C
  150 DO 8 I=1,NUMSEG
        IF(SQTRV(I).GT.SQTRV(I+1)) GOTO 160
          CHANGE(I)=1.
          IF(SQTRV(I).EQ.SQTRV(I+1)) CHANGE(I)=0.
          ILO=I
          IHI=I+1
          GOTO 170
  160     CHANGE(I)=-1.
          IHI=I
          ILO=I+1
  170   DINSEG(I)=DSEG(I)-TRDIST(I)-TRDIST(I+1)
        DMATCH(I)=(SQTRV(IHI)-SQTRV(ILO))/(2.*MAXTRA)
        SGVMX2(I)=MAXTRA*(DINSEG(I)-DMATCH(I))+SQTRV(IHI)
        SEGVMX(I)=SQRT(SGVMX2(I))
        IF(SEGVMX(I).LE.MAXTRV) GOTO 180
          SEGVMX(I)=MAXTRV
          SGVMX2(I)=SEGVMX(I)**2
          DPEAK=DINSEG(I)-DMATCH(I)-(SGVMX2(I)-SQTRV(IHI))/MAXTRA
          TRTMIN(I)=(2.*SEGVMX(I)-TRVEL(IHI)-TRVEL(ILO))/MAXTRA+
     +      DPEAK/SEGVMX(I)
          GOTO 8
  180     TRTMIN(I)=(2.*SEGVMX(I)-TRVEL(IHI)-TRVEL(ILO))/MAXTRA
    8   CONTINUE
C
C*  CALCULATE ROTATIONAL MATRIX AND ANGLE AND UNIT VECTOR OF ROTATION.
C
      CALL MT3333(ROTMAT(1,1),POINTS(1,1,1),END(1,1))
      DO 9 I=1,NUMSEG
        IF(I.EQ.1) GOTO 190
          CALL MT3333(ROTMAT(1,1),POINTS(1,1,I),POINTS(1,1,I-1))
  190   ROTDIF(1)=(ROTMAT(3,2)-ROTMAT(2,3))/2.
        ROTDIF(2)=(ROTMAT(1,3)-ROTMAT(3,1))/2.
        ROTDIF(3)=(ROTMAT(2,1)-ROTMAT(1,2))/2.
        SINPHI=SQRT(ROTDIF(1)**2+ROTDIF(2)**2+ROTDIF(3)**2)
        COSPHI=(ROTMAT(1,1)+ROTMAT(2,2)+ROTMAT.3,3)-1.)/2.
      PHI(I)=ATAN2(SINPHI,COSPHI)
      IF(SINPHI.LT.1.0E-6) GOTO 210
        RVEC(1,I)=ROTDIF(1)/SINPHI
        RVEC(2,I)=ROTDIF(2)/SINPHI
        RVEC(3,I)=ROTDIF(3)/SINPHI
        GOTO 9
210   IF(COSPHI.LT.0.9999) GOTO 220
        PHI(I)=0.
        RVEC(1,I)=0.
        RVEC(2,I)=0.
        RVEC(3,I)=0.
        GOTO 9
220   ROTSUM(1)=(ROTMAT(3,2)+ROTMAT(2,3))/2.
      ROTSUM(2)=(ROTMAT(1,3)+ROTMAT(3,1))/2.
      ROTSUM(3)=(ROTMAT(2,1)+ROTMAT(1,2))/2.
      IF((ABS(ROTSUM(1)).LT.1.0E-6).AND.(ABS(ROTSUM(2)).LT.1.0E-6))
     +  GOTO 230
        ALPHA=ATAN2(ROTSUM(1),ROTSUM(2))
        COS2B=ROTMAT(3,3)
        IF(ABS(SIN(ALPHA)).GT.1.0E-6) SIN2B=ROTSUM(1)/SIN(ALPHA)
        IF(ABS(COS(ALPHA)).GT.1.0E-6) SIN2B=ROTSUM(2)/COS(ALPHA)
        BETA=ATAN2(SIN2B,COS2B)/2.
        RVEC(1,I)=COS(ALPHA)*SIN(BETA)
        RVEC(2,I)=SIN(ALPHA)*SIN(BETA)
        RVEC(3,I)=COS(BETA)
        GOTO 9
230   IF(ROTMAT(3,3).LT.-0.9999) GOTO 240
        RVEC(1,I)=0.
        RVEC(2,I)=0.
        RVEC(3,I)=1.
        GOTO 9
240   COS2A=ROTMAT(1,1)
      SIN2A=ROTSUM(3)
      ALPHA=ATAN2(SIN2A,COS2A)/2.
      RVEC(1,I)=COS(ALPHA)
      RVEC(2,I)=SIN(ALPHA)
      RVEC(3,I)=0.
    9 CONTINUE
C
C*  CALCULATE MINIMUM ROTATIONAL TIMES.
C
      DO 11 I=1,NUMSEG
        ROTMIN(I)=2.*SQRT(PHI(I)/MAXROA)
   11 CONTINUE
C
C*  TEST FOR SUFFICIENT TIME FOR ROTATION.
C
      TSFLMX=0.
      ITSFMX=0
      DO 12 I=1,NUMSEG
        IF(ROTMIN(I).LE.TRTMIN(I)) GOTO 12
          TTVLO2=AMIN1(SQTRV(I),SQTRV(I+1))
          TRVMN2=TTVLO2-MAXTRA*(DINSEG(I)-DMATCH(I))
          IF(TRVMN2.LE.0.) GOTO 12
            TRTMAX=(TRVEL(I+1)+TRVEL(I)-2.*SQRT(TRVMN2))/MAXTRA
```

```
              IF((ROTMIN(I)-TRTMAX).LE.TSFLMX) GOTO 12
                TSFLMX=ROTMIN(I)-TRTMAX
                ITSFMX=I
   12   CONTINUE
C
C*  IF THERE IS A ROTATIONAL TIME SHORTFALL, RECALCULATE TRANSITION
C*   VEILOCITIES AND DISTANCES.
C
      IF(ITSFMX.EQ.0) GOTO 280
        IF(CHANGE(ITSFMX).EQ.-1.) GOTO 250
          ILO=ITSFMX
          IHI=ITSFMX+1
          GOTO 260
  250     IHI=ITSFMX
          ILO=ITSFMX+1
  260   IF((SQTRV(ILO)/MAXTRA).GT.(DSEG(ITSFMX)-2.*TRDIST(ILO)))
     +    GOTO 220
          SQTRV(IHI)=(2.*MAXTRA*(DSEG(ITSFMX)-TRDIST(ILO))-SQTRV(ILO))/
     +      (1.+UNACC(IHI))
          TRDIST(IHI)=SQTRV(IHI)*UNACC(IHI)/(2.*MAXTRA)
          GOTO 50
  270     SQTRV(IHI)=MAXTRA*DSEG(ITSFMX)/(1.+UNACC(IHI)+UNACC(ILO))
          SQTRV(ILO)=SQTRV(IHI)
          TRDIST(IHI)=SQTRV(IHI)*UNACC(IHI)/(2.*MAXTRA)
          TRDIST(ILO)=SQTRV(ILO)*UNACC(ILO)/(2.*MAXTRA)
          GOTO 50
C
C*  CALCULATE SEGMENT TIMES AND ACCELERATIONS.
C
  280 DO 13 I=1,NUMSEG
        TSEGTM(I,1)=0.
        TSEGTM(I,2)=0.
        TSEGTM(I,3)=0.
        TSEGTM(I,4)=0.
        TSEGTM(I,5)=0.
        TSEGAC(I,1)=CHANGE(I)*MAXTRA
        TSEGAC(I,3)=0.
        TSEGAC(I,5)=CHANGE(I)*MAXTRA
        IF(CHANGE(I).EQ.-1.) GOTO 290
          ILO=I
          IHI=I+1
          GOTO 310
  290     IHI=I
          ILO=I+1
  310   TMATCH=(TRVEL(IHI)-TRVEL(ILO))/MAXTRA
        IF(ROTMIN(I).GT.TRTMIN(I)) GOTO 320
          IF(CHANGE(I).EQ. 1.) TSEGTM(I,1)=TMATCH
          IF(CHANGE(I).EQ.-1.) TSEGTM(I,5)=TMATCH
          TSEGTM(I,2)=(SEGVMX(I)-TRVEL(IHI))/MAXTRA
          TSEGAC(I,2)=MAXTRA
          TSEGTM(I,4)=TSEGTM(I,2)
          TSEGAC(I,4)=-TSEGAC(I,2)
          TREM=TRTMIN(I)-TSEGTM(I,1)-TSEGTM(I,2)-TSEGTM(I,4)-TSEGTM(I,5)
          IF(TREM.GE.1.0E-6) TSEGTM(I,3)=TREM
          RSEGAC(I)=(ROTMIN(I)/TRTMIN(I))**2*MAXROA
          GOTO 13
 320    TRVAVE=(DINSEG(I)-DMATCH(I))/(ROTMIN(I)-TMATCH)
        IF(TRVAVE.LE.((MAXTRV+TRVEL(IHI))/2.)) GOTO 330
          IF(CHANGE(I).EQ. 1.) TSEGTM(I,1)=TMATCH
          IF(CHANGE(I).EQ.-1.) TSEGTM(I,5)=TMATCH
          T234=ROTMIN(I)-TMATCH
          TSEGTM(I,2)=T234*(TRVAVE-MAXTRV)/(TRVEL(IHI)-MAXTRV)
          TSEGAC(I,2)=(MAXTRV-TRVEL(IHI))/TSEGTM(I,2)
          TSEGTM(I,4)=TSEGTM(I,2)
          TSEGAC(I,4)=-TSEGAC(I,2)
          TSEGTM(I,3)=T234-TSEGTM(I,2)-TSEGTM(I,4)
          GOTO 370
 330    IF(TRVAVE.LE.TRVEL(IHI)) GOTO 340
          IF(CHANGE(I).EQ. 1.) TSEGTM(I,1)=TMATCH
          IF(CHANGE(I).EQ.-1.) TSEGTM(I,5)=TMATCH
          TSEGTM(I,2)=(ROTMIN(I)-TMATCH)/2.
          TSEGAC(I,2)=2.*(TRVAVE-TRVEL(IHI))/TSEGTM(I,2)
          TSEGTM(I,4)=TSEGTM(I,2)
          TSEGAC(I,4)=-TSEGAC(I,2)
          GOTO 370
 340    IF(TRVAVE.LT.TRVEL(ILO)) GOTO 350
          TSEGTM(I,1)=ABS(TRVAVE-TRVEL(I))/MAXTRA
          TSEGTM(I,3)=ROTMIN(I)-TMATCH
          TSEGTM(I,5)=ABS(TRVAVE-TRVEL(I+1))/MAXTRA
          GOTO 370
 350    IF(TRVAVE.LT.(TRVEL(ILO)/2.)) GOTO 360
          IF(CHANGE(I).EQ. 1.) TSEGTM(I,5)=TMATCH
          IF(CHANGE(I).EQ.-1.) TSEGTM(I,1)=TMATCH
          TSEGTM(I,2)=(ROTMIN(I)-TMATCH)/2.
          TSEGAC(I,2)=-2.*(TRVEL(ILO)-TRVAVE)/TSEGTM(I,2)
          TSEGTM(I,4)=TSEGTM(I,2)
          TSEGAC(I,4)=-TSEGAC(I,2)
          GOTO 370
 360      IF(CHANGE(I).EQ. 1.) TSEGTM(I,5)=TMATCH
          IF(CHANGE(I).EQ.-1.) TSEGTM(I,1)=TMATCH
          T234=ROTMIN(I)-TMATCH
          TSEGTM(I,2)=T234*TRVAVE/TRVEL(ILO)
          TSEGAC(I,2)=-TRVEL(ILO)/TSEGTM(I,2)
          TSEGTM(I,4)=TSEGTM(I,2)
          TSEGAC(I,4)=-TSEGAC(I,2)
          TSEGTM(I,3)=T234-TSEGTM(I,2)-TSEGTM(I,4)
 370      RSEGAC(I)=MAXROA
  13    CONTINUE
C
C*  SET UP PATH TRAJECTORY.
C
      CALL SETPTH(END,NUMSEG,POINTS,TSEGCT,ELTTIM,TACC,INTVEL,
     +  INTPOS,RSEGCT,ELRTIM,RACC,INRPOS,ROTVEC,TRVEL,TRTIME,TRDIST,
     +  USX,USY,USZ,TSEGTM,TSEGAC,RSEGAC,RVEC,TRTMIN,ROTMIN)
C
      RETURN
      END
```

```
C
C****
C
C  THIS SUBROUTINE SETS UP THE DATA FOR AN ARM TRAJECTORY, INCLUDING
C    TIMES, ACCELERATIONS, VELOCITIES, AND POSITIONS.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE SETPTH(END,NUMSEG,POINTS,TSEGCT,ELTTIM,TACC,INTVEL,
     +  INTPOS,RSEGCT,ELRTIM,RACC,INRPOS,ROTVEC,TRVEL,TRTIME,TRDIST,
     +  USX,USY,USZ,TSEGTM,TSEGAC,RSEGAC,RVEC,TRTMIN,ROTMIN)
C
      INTEGER NUMSEG,TSEGCT,RSEGCT
      REAL END(3,4),POINTS(3,4,6),ELTTIM(36),TACC(3,36),INTVEL(3,36),
     +  INTPOS(3,36),ELRTIM(12),RACC(12),INRPOS(9,12),ROTVEC(3,12),
     +  TRVEL(7),TRTIME(7),TRDIST(7),USX(6),USY(6),USZ(6),TSEGTM(6,5),
     +  TSEGAC(6,5),RSEGAC(6),RVEC(3,6),TRTMIN(6),ROTMIN(6)
C
C
C*  SET UP TRANSLATIONAL PATH.
C
      ITSEG=1
      ELTTIM(ITSEG)=0.
      TACC(1,ITSEG)=0.
      TACC(2,ITSEG)=0.
      TACC(3,ITSEG)=0.
      INTVEL(1,ITSEG)=0.
      INTVEL(2,ITSEG)=0.
      INTVEL(3,ITSEG)=0.
      INTPOS(1,ITSEG)=END(1,4)
      INTPOS(2,ITSEG)=END(2,4)
      INTPOS(3,ITSEG)=END(3,4)
      ITSEG=ITSEG+1
      DO 1 I=1,NUMSEG
        IF(I.EQ.1) GOTO 30
          IF(TRTIME(I).GE.1.0E-6) GOTO 10
            ELTTIM(ITSEG)=ELTTIM(ITSEG-1)
            TACC(1,ITSEG)=0.
            TACC(2,ITSEG)=0.
            TACC(3,ITSEG)=0.
            GOTO 20
   10       ELTTIM(ITSEG)=ELTTIM(ITSEG-1)+TRTIME(I)
            TACC(1,ITSEG)=(TRVEL(I)/TRTIME(I))*(USX(I)-USX(I-1))
            TACC(2,ITSEG)=(TRVEL(I)/TRTIME(I))*(USY(I)-USY(I-1))
            TACC(3,ITSEG)=(TRVEL(I)/TRTIME(I))*(USZ(I)-USZ(I-1))
   20       INTVEL(1,ITSEG)=TRVEL(I)*USX(I-1)
            INTVEL(2,ITSEG)=TRVEL(I)*USY(I-1)
            INTVEL(3,ITSEG)=TRVEL(I)*USZ(I-1)
            INTPOS(1,ITSEG)=POINTS(1,4,I-1)-TRDIST(I)*USX(I-1)
            INTPOS(2,ITSEG)=POINTS(2,4,I-1)-TRDIST(I)*USY(I-1)
            INTPOS(3,ITSEG)=POINTS(3,4,I-1)-TRDIST(I)*USZ(I-1)
            ITSEG=ITSEG+1
   30   DO 2 J=1,5
```

C-2

```
            IF(TSEGTM(I,J).EQ.0.) GOTO 2
              ELTIME=ELTTIM(ITSEG-1)-ELTTIM(ITSEG-2)
              IF(ITSEG.EQ.2) ELTIME=0.
              ELTTIM(ITSEG)=ELTTIM(ITSEG-1)+TSEGTM(I,J)
              TACC(1,ITSEG)=TSEGAC(I,J)*USX(I)
              TACC(2,ITSEG)=TSEGAC(I,J)*USY(I)
              TACC(3,ITSEG)=TSEGAC(I,J)*USZ(I)
              INTVEL(1,ITSEG)=INTVEL(1,ITSEG-1)+TACC(1,ITSEG-1)*
     +          ELTIME
              INTVEL(2,ITSEG)=INTVEL(2,ITSEG-1)+TACC(2,ITSEG-1)*
     +          ELTIME
              INTVEL(3,ITSEG)=INTVEL(3,ITSEG-1)+TACC(3,ITSEG-1)*
     +          ELTIME
              INTPOS(1,ITSEG)=INTPOS(1,ITSEG-1)+INTVEL(1,ITSEG-1)*
     +          ELTIME+TACC(1,ITSEG-1)*ELTIME**2/2.
              INTPOS(2,ITSEG)=INTPOS(2,ITSEG-1)+INTVEL(2,ITSEG-1)*
     +          ELTIME+TACC(2,ITSEG-1)*ELTIME**2/2.
              INTPOS(3,ITSEG)=INTPOS(3,ITSEG-1)+INTVEL(3,ITSEG-1)*
     +          ELTIME+TACC(3,ITSEG-1)*ELTIME**2/2.
              ITSEG=ITSEG+1
    2       CONTINUE
    1     CONTINUE
C
        TSEGCT=ITSEG-1
C
C*    SET UP ROTATIONAL PATH.
C
        IRSEG=1
        ELRTIM(IRSEG)=0.
        RACC(IRSEG)=0.
        INRPOS(1,IRSEG)=END(1,1)
        INRPOS(2,IRSEG)=END(2,1)
        INRPOS(3,IRSEG)=END(3,1)
        INRPOS(4,IRSEG)=END(1,2)
        INRPOS(5,IRSEG)=END(2,2)
        INRPOS(6,IRSEG)=END(3,2)
        INRPOS(7,IRSEG)=END(1,3)
        INRPOS(8,IRSEG)=END(2,3)
        INRPOS(9,IRSEG)=END(3,3)
        IRSEG=IRSEG+1
        DO 3 I=1,NUMSEG
          IF(I.EQ.1) GOTO 40
            ELRTIM(IRSEG)=ELRTIM(IRSEG-1)+TRTIME(I)
            RACC(IRSEG)=0.
            INRPOS(1,IRSEG)=POINTS(1,1,I-1)
            INRPOS(2,IRSEG)=POINTS(2,1,I-1)
            INRPOS(3,IRSEG)=POINTS(3,1,I-1)
            INRPOS(4,IRSEG)=POINTS(1,2,I-1)
            INRPOS(5,IRSEG)=POINTS(2,2,I-1)
            INRPOS(6,IRSEG)=POINTS(3,2,I-1)
            INRPOS(7,IRSEG)=POINTS(1,3,I-1)
            INRPOS(8,IRSEG)=POINTS(2,3,I-1)
            INRPOS(9,IRSEG)=POINTS(3,3,I-1)
            IRSEG=IRSEG+1
   40       ELRTIM(IRSEG)=ELRTIM(IRSEG-1)+AMAX1(ROTMIN(I),TRTMIN(I))
```

```
     RACC(IRSEG)=RSEGAC(I)
     ROTVEC(1,IRSEG)=RVEC(1,I)
     ROTVEC(2,IRSEG)=RVEC(2,I)
     ROTVEC(3,IRSEG)=RVEC(3,I)
     DO 4 J=1,9
      INRPOS(J,IRSEG)=INRPOS(J,IRSEG-1)
  4     CONTINUE
     IRSEG=IRSEG+1
  3  CONTINUE
C
   RSEGCT=IRSEG-1
C
   RETURN
   END
```

```
C****
C
C  THIS SUBROUTINE CALCULATES THE COMMANDED ARM VELOCITY FROM THE
C     TRAJECTORY DATA.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE VELCOM(EXTEND,TSEGCT,ELTTIM,TACC,INTVEL,INTPOS,RSEGCT,
     +  ELRTIM,RACC,INRPOS,ROTVEC,TERPOS,ELTIME,TIMINC,VEL,TERFLG)
C
      INTEGER TSEGCT,RSEGCT,TERFLG
      REAL EXTEND(3,4),ELTTIM(36),TACC(3,36),INTVEL(3,36),INTPOS(3,36),
     +  ELRTIM(12),RACC(12),INRPOS(9,12),ROTVEC(3,12),TERPOS(3,4),
     +  ELTIME,TIMINC,VEL(6),TTRERR,TTRVEL,TROERR,TROVEL
      DIMENSION VELD(6),POSD(3,4),ROTMAT(3,3),ERROR(6)
C
      COMMON /TERPAK/TTRERR,TTRVEL,TROERR,TROVEL
C
C
C*  CALCULATE TRANSLATIONAL ERROR.
C
      RINSGT=0.
      ITSEG=1
   10 IF(ELTIME.LT.ELTTIM(ITSEG)) GOTO 20
        IF(ITSEG.EQ.TSEGCT) GOTO 30
          RINSGT=ELTTIM(ITSEG)
          ITSEG=ITSEG+1
          GOTO 10
C
   20 ELSEGT=ELTIME-RINSGT
      VELD(1)=INTVEL(1,ITSEG)+TACC(1,ITSEG)*ELSEGT
      VELD(2)=INTVEL(2,ITSEG)+TACC(2,ITSEG)*ELSEGT
      VELD(3)=INTVEL(3,ITSEG)+TACC(3,ITSEG)*ELSEGT
      POSD(1,4)=INTPOS(1,ITSEG)+INTVEL(1,ITSEG)*ELSEGT+
     +  TACC(1,ITSEG)*ELSEGT**2/2.
      POSD(2,4)=INTPOS(2,ITSEG)+INTVEL(2,ITSEG)*ELSEGT+
     +  TACC(2,ITSEG)*ELSEGT**2/2.
      POSD(3,4)=INTPOS(3,ITSEG)+INTVEL(3,ITSEG)*ELSEGT+
     +  TACC(3,ITSEG)*ELSEGT**2/2.
      ERROR(1)=POSD(1,4)-EXTEND(1,4)
      ERROR(2)=POSD(2,4)-EXTEND(2,4)
      ERROR(3)=POSD(3,4)-EXTEND(3,4)
      GOTO 40
   30 VELD(1)=0.
      VELD(2)=0.
      VELD(3)=0.
      ERROR(1)=TERPOS(1,4)-EXTEND(1,4)
      ERROR(2)=TERPOS(2,4)-EXTEND(2,4)
      ERROR(3)=TERPOS(3,4)-EXTEND(3,4)
C
C*  CALCULATE ROTATIONAL ERROR.
C
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
   40 RINSGT=0.
      IRSEG=1
   50 IF(ELTIME.LT.ELRTIM(IRSEG)) GOTO 60
        IF(IRSEG.EQ.RSEGCT) GOTO 90
          RINSGT=ELRTIM(IRSEG)
          IRSEG=IRSEG+1
          GOTO 50
C
   60 ELSEGT=ELTIME-RINSGT
      RTINT2=(ELRTIM(IRSEG)-ELRTIM(IRSEG-1))/2.
      IF(ELSEGT.GT.RTINT2) GOTO 70
        RVELD=RACC(IRSEG)*ELSEGT
        PHI=RACC(IRSEG)*ELSEGT**2/2.
        GOTO 80
   70   RVELD=RACC(IRSEG)*RTINT2-RACC(IRSEG)*(ELSEGT-RTINT2)
        PHI=RACC(IRSEG)*RTINT2**2/2.+RACC(IRSEG)*RTINT2*(ELSEGT-RTINT2)-
     +    RACC(IRSEG)*(ELSEGT-RTINT2)**2/2.
   80 VELD(4)=RVELD*ROTVEC(1,IRSEG)
      VELD(5)=RVELD*ROTVEC(2,IRSEG)
      VELD(6)=RVELD*ROTVEC(3,IRSEG)
      CPHI=COS(PHI)
      SPHI=SIN(PHI)
      R111CP=ROTVEC(1,IRSEG)*ROTVEC(1,IRSEG)*(1.-CPHI)
      R221CP=ROTVEC(2,IRSEG)*ROTVEC(2,IRSEG)*(1.-CPHI)
      R331CP=ROTVEC(3,IRSEG)*ROTVEC(3,IRSEG)*(1.-CPHI)
      R121CP=ROTVEC(1,IRSEG)*ROTVEC(2,IRSEG)*(1.-CPHI)
      R131CP=ROTVEC(1,IRSEG)*ROTVEC(3,IRSEG)*(1.-CPHI)
      R231CP=ROTVEC(2,IRSEG)*ROTVEC(3,IRSEG)*(1.-CPHI)
      R1SP=ROTVEC(1,IRSEG)*SPHI
      R2SP=ROTVEC(2,IRSEG)*SPHI
      R3SP=ROTVEC(3,IRSEG)*SPHI
      ROTMAT(1,1)=R111CP+CPHI
      ROTMAT(2,1)=R121CP+R3SP
      ROTMAT(3,1)=R131CP-R2SP
      ROTMAT(1,2)=R121CP-R3SP
      ROTMAT(2,2)=R221CP+CPHI
      ROTMAT(3,2)=R231CP+R1SP
      ROTMAT(1,3)=R131CP+R2SP
      ROTMAT(2,3)=R231CP-R1SP
      ROTMAT(3,3)=R331CP+CPHI
      CALL MM3333(POSD,ROTMAT,INRPOS(1,IRSEG))
      CALL ROERRC(EXTEND,POSD,ERROR(4))
      GOTO 110
   90 VELD(4)=0.
      VELD(5)=0.
      VELD(6)=0.
      CALL ROERRC(EXTEND,TERPOS,ERROR(4))
C
C*  CALCULATE VELOCITY COMMANDS.
C
  110 DO 1 I=1,6
        VEL(I)=VELD(I)+(1./TIMINC)*ERROR(I)
    1   CONTINUE
C
C*  CHECK FOR TERMINAL POSITION.
```

```
C
      TERFLG=0
      IF(ELTIME.LT.ELTTIM(TSEGCT)) GOTO 120
        TRERR=SQRT(ERROR(1)**2+ERROR(2)**2+ERROR(3)**2)
        IF(TRERR.GT.TTRERR) GOTO 120
        ROERR=SQRT(ERROR(4)**2+ERROR(5)**2+ERROR(6)**2)
        IF(ROERR.GT.TROERR) GOTO 120
        TRVEL=SQRT(VEL(1)**2+VEL(2)**2+VEL(3)**2)
        IF(TRVEL.GT.TTRVEL) GOTO 120
        ROVEL=SQRT(VEL(4)**2+VEL(5)**2+VEL(6)**2)
        IF(ROVEL.GT.TROVEL) GOTO 120
          TERFLG=1
C
  120 RETURN
      END
```

```
C
C****
C
C  THIS SUBROUTINE CALCULATES THE JOINT VELOCITIES REQUIRED TO
C    IMPLEMENT THE GIVEN END EFFECTOR TRANSLATIONAL AND ROTATIONAL
C    VELOCITIES.
C
C  M.R. PATTERSON
C
C****
C
      SUBROUTINE SOLVE(TH,THV,CTH,STH,JOINTS,END,VEL)
C
      REAL TH(6),THV(6),CTH(7),STH(7),JOINTS(3,8),END(3,4),VEL(6),
     +  NO,SN,ES,HW,JLIM(8),JVLIM(6)
      DIMENSION WRVEL(3),ROTVEL(3),RINVJ(3,3)
C
      COMMON /ARMPAR/NO,SN,ES,HW
      COMMON /JNTPAR/JLIM,JVLIM
C
C
      C9=CTH(2)+CTH(7)
      S9=STH(2)+STH(7)
C
C*  COMPUTE WRIST VELOCITY.
C
      WRVEL(1)=VEL(1)+(END(2,3)*VEL(6)-END(3,3)*VEL(5))*HW
      WRVEL(2)=VEL(2)+(END(3,3)*VEL(4)-END(1,3)*VEL(6))*HW
      WRVEL(3)=VEL(3)+(END(1,3)*VEL(5)-END(2,3)*VEL(4))*HW
C
C*  COMPUTE INVERSE JACOBIAN FOR FIRST THREE JOINTS.
C
      S3ES=STH(3)*ES
      RINVJ(1,1)=-STH(1)*S3ES
      RINVJ(1,2)=CTH(1)*S3ES
      RINVJ(1,3)=0.
      RINVJ(2,1)=JOINTS(1,5)*STH(7)
      RINVJ(2,2)=JOINTS(2,5)*STH(7)
      RINVJ(2,3)=(JOINTS(3,5)-JOINTS(3,4))*S9
      S1S9=STH(1)*S9
      C1S9=CTH(1)*S9
      S9ES=S9*ES
      RINVJ(3,1)=S1S9*SN-C1S9*S9ES
      RINVJ(3,2)=-C1S9*SN-S1S9*S9ES
      RINVJ(3,3)=-(JOINTS(3,5)-NO)*S9
C
C*  COMPUTE FIRST THREE JOINT VELOCITIES.
C
      DIV=S9ES*S3ES
      THV(1)=(RINVJ(1,1)*WRVEL(1)+RINVJ(1,2)*WRVEL(2))/DIV
      THV(2)=(RINVJ(2,1)*WRVEL(1)+RINVJ(2,2)*WRVEL(2)+RINVJ(2,3)*
     +    WRVEL(3))/DIV
      THV(3)=(RINVJ(3,1)*WRVEL(1)+RINVJ(3,2)*WRVEL(2)+RINVJ(3,3)*
     +    WRVEL(3))/DIV
C
```

```
C*  COMPUTE END EFFECTOR ROTATIONAL VELOCITIES.
C
      VEL(4)=VEL(4)+STH(1)*(THV(2)+THV(3))
      VEL(5)=VEL(5)-CTH(1)*(THV(2)+THV(3))
      VEL(6)=VEL(6)-THV(1)
      ROTVEL(1)=CTH(1)*CTH(7)*VEL(4)+STH(1)*CTH(7)*VEL(5)-
     +  STH(7)*VEL(6)
      ROTVEL(2)=-STH(1)*VEL(4)+CTH(1)*VEL(5)
      ROTVEL(3)=CTH(1)*STH(7)*VEL(4)+STH(1)*STH(7)*VEL(5)+
     +  CTH(7)*VEL(6)
C
C*  COMPUTE LAST THREE JOINT VELOCITIES.
C
      IF(ABS(STH(5)).LT.1.0E-2)  GOTO 10
        THV(6)=(CTH(4)*ROTVEL(1)+STH(4)*ROTVEL(2))/STH(5)
        THV(5)=-STH(4)*ROTVEL(1)+CTH(4)*ROTVEL(2)
        THV(4)=ROTVEL(3)-CTH(5)*THV(6)
        GOTO 20
C
C*  COMPUTE LAST THREE JOINT VELOCITIES FOR S5=0.
C
   10   THV(6)=ROTVEL(3)
        THV(5)=-STH(4)*ROTVEL(1)+CTH(4)*ROTVEL(2)
        THV(4)=0.
C
C*  TEST FOR EXCESSIVE JOINT VELOCITIES.
C
   20 RED=1.
      IF(ABS(THV(1)).GT.JVLIM(1)) RED=AMIN1(RED,JVLIM(1)/ABS(THV(1)))
      IF(ABS(THV(2)).GT.JVLIM(2)) RED=AMIN1(RED,JVLIM(2)/ABS(THV(2)))
      IF(ABS(THV(3)).GT.JVLIM(3)) RED=AMIN1(RED,JVLIM(3)/ABS(THV(3)))
      IF(ABS(THV(4)).GT.JVLIM(4)) RED=AMIN1(RED,JVLIM(4)/ABS(THV(4)))
      IF(ABS(THV(5)).GT.JVLIM(5)) RED=AMIN1(RED,JVLIM(5)/ABS(THV(5)))
      IF(ABS(THV(6)).GT.JVLIM(6)) RED=AMIN1(RED,JVLIM(6)/ABS(THV(6)))
      IF(RED.EQ.1.) GOTO 30
        THV(1)=RED*THV(1)
        THV(2)=RED*THV(2)
        THV(3)=RED*THV(3)
        THV(4)=RED*THV(4)
        THV(5)=RED*THV(5)
        THV(6)=RED*THV(6)
C
   30 RETURN
      END
```

REFERENCES

1. Paine, G. "The Automation of Remote Vehicle Controls." Proc. of 1977 Joint Automatic Control Conference, San Francisco, CA. June, 1977.

2. Heer, E. "Robot and Automation Technology Requirements for Space Industrialization." Proc. of Ninth International Symposium on Industrial Robots, Washington, D.C. March, 1979.

3. Large Space Systems Technology. NASA Conference Publication 2035, Scientific and Technical Information Office, NASA, Washington, D.C. January, 1978.

4. Proc. of Ninth International Symposium on Industrial Robots, Washington, D.C. March, 1979.

5. Mosher, R.S. "Exploring the Potential of a Quadruped." SAE Paper No. 690191. International Automotive Engineering Conference, Detroit, MI. January, 1969.

6. Horn, B.K.P., and Inoue, H. "Kinematics of the MIT-AI-Vicarm Manipulator." M.I.T. Artificial Intelligence Laboratory Working Paper 69. May, 1974.

7. Whitney, D.E. "Resolved Motion Rate Control of Manipulators and Human Protheses." IEEE Transactions on Man-Machine Systems, Vol. MMS-10, No. 2. June, 1969.

8. Renaud, M. "Coordinated Control of Robots-Manipulators; Determination of the Singularities of the Jacobian Matrix." Proc. of First Yugoslav Symposium on Industrial Robots and Artificial Intelligence, Dubrovnik, Yugoslavia. September, 1979.

9. Whitney, D.E. "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators." Journal of Dynamic Systems, Measurement, and Control, Transactions ASME, Series G, Vol. 94, No. 4. December, 1972.

10. Liégeois, A. "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms." IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-7, No. 12. December, 1977.

11. Albus, J.S. "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)." Journal of Dynamic Systems, Measurement, and Control, Transactions ASME, Series G, Vol. 97, No. 3. September, 1975.

12. Raibert, M.H. "A State Space Model for Sensorimotor Control and Learning." M.I.T. Artificial Intelligence Memo No. 351. January, 1976.

13. Horn, B.K.P., and Raibert, M.H. "Configuration Space Control." M.I.T. Artificial Intelligence Memo No. 458. December, 1977.

14. Denavit, J., and Hartenberg, R.S. "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices." Journal of Applied Mechanics, Trans. ASME, Vol. 22, No. 5. June, 1955.

15. Peiper, D.L. "The Kinematics of Manipulators under Computer Control." Stanford Artificial Intelligence Laboratory Memo AI-72. October, 1968.

16. Paul, R.C. "Modeling, Trajectory Calculation, and Servoing of a Computer Controlled Arm." Stanford Artificial Intelligence Laboratory Memo AIM-177. November, 1972.

17. McGhee, R.B. "Control of Legged Locomotion Systems." Proc. 1977 Joint Automatic Control Conference, San Francisco, CA. June, 1977.

18. Tomovic, R. "A General Theoretical Model of Creeping Displacement." Cybernetica, Vol. 4, No. 2. 1961 (English translation.)

19. Tomovic, R., and McGhee, R.B. "A Finite State Approach to the Synthesis of Bioengineering Control Systems." IEEE Transactions on Human Factors in Electronics, Vol. HFE-7, No. 2. June, 1966.

20. Frank, A.A. Automatic Control Systems for Legged Locomotion Machines. Ph.D. dissertation, University of Southern California, Los Angeles, CA. May, 1968.

21. McGhee, R.B., and Pai, A.L. "An Approach to Computer-Control for Legged Vehicles." Journal of Terramechanics, Vol. 11, No. 2. March, 1974.

22. Vukobratovic, M. "How to Control Artificial Anthropomorphic Systems." IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-3, No. 5. September, 1973.

23. Sun, S.S. A Theoretical Study of Gaits for Legged Locomotion Systems. Ph.D. dissertation, Ohio State University, Columbus, OH. March, 1974.

24. Kugushev, E.I., and Jaroshevskij, V.S. "Problems of Selecting a Gait for an Integrated Locomocion Robot." Proc. Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgian SSR, USSR. September, 1975.

25. Bessonov, A.P., and Umnov, N.V. "The Analysis of Gaits in Six Legged Vehicles According to Their Static Stability." Proc. of Symposium for the Theory and Practice of Robots and Manipulators, Udine, Italy. September, 1973.

26. McGhee, R.B., and Iswandhi, G.I. "Adaptive Locomotion of a Multilegged Robot over Rough Terrain." IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-9, No. 4. April, 1979.

27. Ferrell, W.R., and Sheridan, T.B. "Supervisory Control of Remote Manipulation." IEEE Spectrum, V. 4, No. 10. October, 1967.

28. Albus, J.S., private communication.

29. Narinyani, A.S., Pyatkin, V.P, Kim, P.A., and Dementyev, V.N. "Walking Robot: A Non-deterministic Model of Control." Proc. Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgian SSR, USSR. September, 1975.

30. Luh, J.Y.S., and Walker, M.W. "Minimum-Time along the Path for a Mechanical Arm." Proc. of 1977 IEEE Conference on Decision and Control, New Orleans, LA. December, 1977.

31. Noble, B. Applied Linear Algebra, Prentice-Hall, 1969. pp. 420-422.