INVERSE KINEMATICS AND DESIGN CONSIDERATIONS FOR KINEMATICALLY REDUNDANT ROBOTS

By

JAMES FRANK JONES

Bachelor of Science

in Mechanical Engineering

Oklahoma State University

Stillwater, Oklahoma

1984

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 1987

Thesis
1987
J771
Cop. 2

OKLAHOMA STATE UNIVERSITY LIBRARY

INVERSE KINEMATICS AND DESIGN CONSIDERATIONS FOR KINEMATICALLY REDUNDANT ROBOTS

Thesis Approved:

A. H. Soni
Thesis Adviser

P. M. Mosier

R L Lowery

Norman N. Durham
Dean of the Graduate College

1275670

PREFACE

Inverse kinematics of kinematically redundant robots has become an area of active research in the past few years. A redundant robot is theoretically capable of avoiding all degenerate configurations. This study discusses design considerations of the redundant manipulator and presents a technique to perform the inverse kinematics without degeneracies within the manipulator's workspace. At the early stages of this study I thought that I had "solved the problem". Now I realize that we, as researchers in robotics, have only begun. Each new solution inspires new applications which then inspires a greater number of unanswered questions.

I wish to express my thanks to the people who helped me during this study. First, Dr. A. H. Soni who has been an inexhaustible source of new ideas and who has helped me to mature professionally in the past few years. Second, Dr. G. Naganathan who has given me a great deal of technical assistance and has served as a constant source of inspiration. Third, Bin Fang who did some of the preliminary work for this study. Also, Palaniswamy Sathyadev who has given me a great deal of assistance in understanding some of the mathematical theory used in this study.

I would like to thank my parents, Frank and Maryella Jones, for their years of understanding and undying confidence that someday I would succeed. Also, to my sister, Karen VanSchoyck, who finally inspired me to attend college.

Finally, I would like to thank the most important person in my life, my wife Mary. Everyone should have an understanding friend as wonderful as she.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

$\mathbf{A}$	matrix
$\mathbf{A}^+$	pseudo-inverse of $\mathbf{A}$
$\mathbf{a}$	direction cosine of Z-axis
a	link length
$\mathbf{b}$	vector
C_i	cosine of θ_i
C_{ij}	cosine of $(\theta_i + \theta_j)$
$\mathbf{c}$	vector
$\mathbf{d}$	incremental move in cartesian coordinates
d	link offset
$\mathbf{d}$	vector
$\mathbf{g}$	vector
H	least squares function
∇H	gradient of H
$\mathbf{h}$	vector
$\mathbf{I}$	identity matrix
$\mathbf{J}$	Jacobian matrix
$\mathbf{j}$	one column of Jacobian matrix
$\mathbf{n}$	direction cosine of X-axis
$\mathbf{o}$	direction cosine of Y-axis
$\mathbf{p}$	position of the end of the last link
S_i	sine of θ_i
$\mathbf{T}$	transformation matrix

x	local coordinate system
$\mathbf{x}$	vector
α	link twist angle
β	real constant
θ	joint variable
$\Delta\theta_i$	maximum one sided excursion for θ_i
θ_{ci}	median value for θ_i
$d\boldsymbol{\theta}$	incremental move in joint coordinates
$*$	complex conjugate transpose

CHAPTER I

INTRODUCTION

Inverse Kinematics

This thesis deals with the inverse kinematics of robots and how the inverse kinematics affects their design. Inverse kinematics is the solving for the value of each of the joints of a robot to obtain a desired position and/or orientation of the last link of the robot. Currently robots usually contain a number of joints equal to the number of desired components of motion of the end-effector. For example, if the only concern is to position the end-effector in space and the end-effectors orientation in space is of no concern, the robot would need three components of motion. Hence, the robot would need a minimum of three joints. The inverse kinematic problem for this three degree of freedom robot would be to solve for the value of each joint so that the end-effector would be at a desired location in space.

Inconsistent Robots

Inconsistent robots are ones in which the number of joints that the robot contains is less than the desired components of motion. Since in general inconsistent robots will not be able to be reach a desired position they will

not be considered further in this thesis.

Unique Robots

A unique robot is one in which the number of joints that the robot contains equals the number of desired components of motion. Therefore, there are at most a finite number of solutions to the inverse kinematics for a unique robot.

One popular method of performing inverse kinematics of a unique robot is the Jacobian method [1]. The Jacobian matrix relates incremental joint motions to incremental motions in a more convenient coordinate system, usually Cartesian. Once the Jacobian is determined it is inverted and premultiplied by the desired incremental move to obtain a vector containing the incremental joint motions. A problem with the Jacobian method is that of singularities. If the Jacobian becomes singular then the inverse of the Jacobian does not exist and the robot becomes uncontrollable.

Another method is to invert and premultiply successive transformation matrices to obtain a closed form solution [2,3]. A closed form solution would allow for the joint values to be solved directly instead of in terms of several incremental moves. The closed form solution will also have a problem with singularities since it will require division by trigonometric functions of joint variables; hence, a division by zero could result. Another problem with the

closed form solution is that the path of the end-effector is not defined. A path may be defined by breaking the path into several incremental moves; however, using several increments would negate any computational advantage the closed form solution has over the Jacobian method.

Redundant Robots

Mathematical singularities or robot degeneracies has become an active area of research in the past few years. Paul extensively studied the cause and effects of degeneracies of a robot's orientation structure [4]. If a robot reaches a degenerate position large joint velocities will occur resulting in unreliable solutions. Most of the recent work attempts to solve the problem of degeneracies by adding extra joints to the robot. A robot with more joints than desired components of motion is referred to as a redundant robot. A redundant robot has at least one infinity of solutions for the inverse kinematics.

Hopefully, a redundant robot would have no degeneracies, or at least avoid any degenerate position by picking the proper solution to the inverse kinematics from the infinity available. However, it is difficult to select one solution from the infinity available.

One method of inverse kinematics for redundant robots is to divide the Jacobian matrix into several submatrices [5]. Since the Jacobian matrix for a redundant robot is rectangular, representing a consistent set of simultaneous

equations, the Jacobian matrix could be divided into several square submatrices. Each submatrix could be evaluated and the best suited submatrix would be used to perform the inverse kinematics, thereby never allowing a singular solution. However, evaluating all the possible submatrices prior to solving the inverse kinematics for each incremental move will add a great deal of additional computation.

Another method is to add additional constraining equations to the Jacobian [6]. By adding constraining equations the Jacobian can be made square; therefore, allowing inversion by traditional means. This method will result in a unique solution to the inverse kinematics; however, it does not guarantee a non-singular inversion of the Jacobian.

Optimization of performance criterion has shown promising results [7,8,9]. Optimization techniques can be used to optimize any performance criterion such as path length or robot dynamics. Additionally, they are not restricted to piece wise linerization of the nonlinear inverse kinematics problem. However, the excessively large number of computations required for an optimization problem along with the inherent stability problem of nonlinear iterative techniques makes optimization methods impractical for on-line inverse kinematics.

The pseudoinverse method has the best potential to perform inverse kinematics quickly and near optimally since the solution of the pseudoinverse yields the entire solution

space of the inverse kinematics with respect to an arbitrary vector. The mathematical properties of the pseudoinverse have been studied in great detail and are well understood [10,11,12]. The pseudoinverse method has been used, in the laboratory, to perform the inverse kinematics for a six jointed robot executing a task requiring only three components of motion [13]. In addition, the properties of the pseudoinverse when applied to a robot have been studied extensively [14].

Problem Statement

The goal of this study is develop a robot configuration and method of inverse kinematics that will not allow the existence of mathematical singularities or geometrical degenerate configurations to degrade the performance of a robot. First a method of inverse kinematics must be selected, then a way to identify types of degeneracies and how degeneracies effect the method used to perform inverse kinematics. Finally, it will have to be proven that a proposed robot will either avoid or not be affected by degeneracies within its workspace.

CHAPTER II

INVERSE KINEMATICS

When selecting a method of inverse kinematics many factors must be considered. The method must be fast to allow on-line programming of the robot. The method must never allow a mathematical singularity to degrade the performance of the robot. Lastly, the method must give dynamically acceptable solutions. Since the pseudoinverse method seems best suited to obtain the above mentioned criterion, the pseudoinverse method will be the method used in this study.

Jacobian

To use the pseudoinverse method the path of the end-effector must be broken up into incremental moves. The Jacobian matrix relates incremental joint motions of a manipulator to incremental motions in a more convenient coordinate system.

The Jacobian consists of the partial derivatives of each local coordinate system with respect to the joint variable. Mathematically the Jacobian may be written -

$$\mathbf{J} \equiv \left| \frac{\partial X_i}{\partial \theta_j} \right|$$

where i = 1 to the number of degrees of freedom

j = 1 to the number of joint variables

$\mathbf{J}$ = The Jacobian matrix

X_i = The i th local coordinate system

θ_j = the j th joint variable

Paul demonstrates a method to construct a Jacobian matrix for any manipulator consisting of revolute joints by using transformation matrices [1]. A transformation matrix $\mathbf{T}_i$ for a prizmatic or revolute joint my be written -

$$\mathbf{T}_i = \begin{vmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & \cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

where θ_i = the i th joint angle

d_i = the i th link offset

a_i = the i th link length

α_i = the i th link twist angle

The $\mathbf{T}_i$ matrix relates the positions and directions of link i to link i-1 as shown in figure 1.

In general the transformation matrix may be written -

$$^{j}\mathbf{T}_i = \begin{vmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

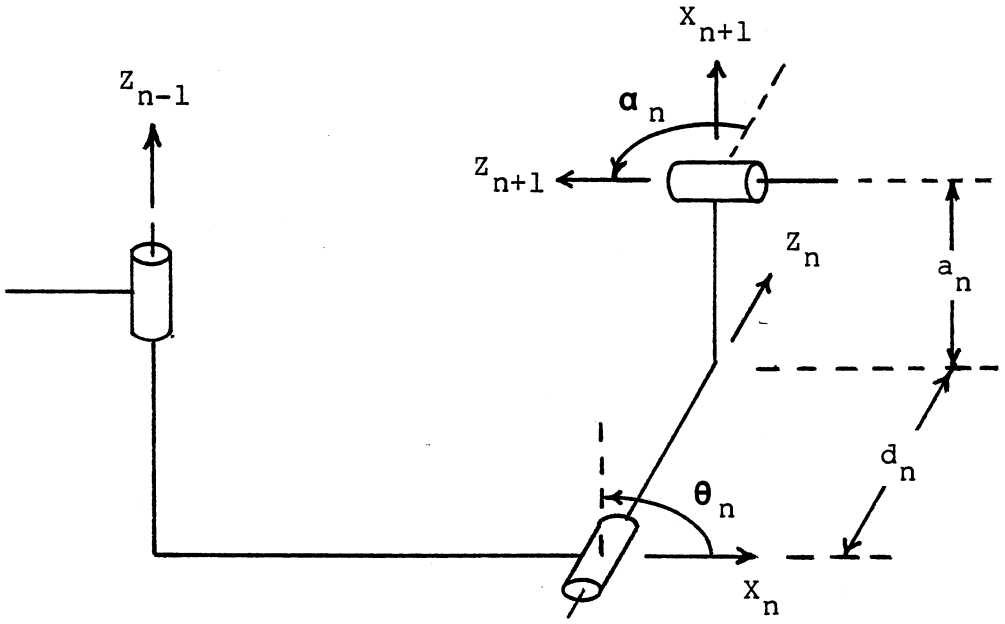

Figure 1. Link Parameters

where **n** = the direction cosines of the i th X axis coordinate system with respect to the j th coordinate system

o = the direction cosines of the i th Y axis coordinate system with respect to the j th coordinate system

a = the direction cosines of the i th Z axis coordinate system with respect to the j th coordinate system

p = the position of the end of the i th link with respect to the j th coordinate system

Once a transformation matrix has been formed one column of the Jacobian may be computed, the i th column of the Jacobian matrix for a link with a revolute joint moving in XYZ space can be computed from the ${}^{j}\mathbf{T}_{i}$ transformation matrix as follows -

$$\mathbf{j}_i = \begin{vmatrix} -n_x p_y + n_y p_x \\ -o_x p_y + o_y p_x \\ -a_x p_y + a_y p_x \\ n_z \\ o_z \\ a_z \end{vmatrix}$$

After determining all i columns of the Jacobian matrix the differential change in position and orientation as a function of differential rotations of the revolute joints may be written as -

$$\begin{vmatrix} {}^{j}d_x \\ {}^{j}d_y \\ {}^{j}d_z \\ {}^{j}\delta_x \\ {}^{j}\delta_y \\ {}^{j}\delta_z \end{vmatrix} = \mathbf{J} \begin{vmatrix} d\theta_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ d\theta_i \end{vmatrix}$$

where ${}^{j}d_x$ = an incremental move in the X direction in the j th coordinate system

${}^{j}d_y$ = an incremental move in the Y direction in the j th coordinate system

${}^{j}d_z$ = an incremental move in the Z direction in the j th coordinate system

${}^{j}\delta_x$ = an incremental rotation about the X axis in the j th coordinate system

${}^{j}\delta_y$ = an incremental rotation about the Y axis in the j th coordinate system

${}^{j}\delta_z$ = an incremental rotation about the Z axis in the j th coordinate system

$\mathbf{J}$ = the Jacobian matrix

$d\theta_i$ = an incremental rotation of the i th revolute joint

or in matrix notation as-

$$\mathbf{d} = \mathbf{J}\, d\boldsymbol{\theta}$$

Now the incremental joint motions for an incremental change in position and orientation may be obtained by inverting the Jacobian and premultipling the above equation

by the inverse of the Jacobian. Resulting in -

$$d\theta = J^{-1}\, d$$

Pseudoinverse

The above method of inverse kinematics using the inverse of the Jacobian works well if the Jacobian is a well behaved and square. In general the inverse of the Jacobian may be found by determining the adjoint of the Jacobian, then dividing the adjoint by the determinant of the Jacobian.

$$J^{-1} = \frac{1}{\det |J|} \left| \text{Adj } J \right|$$

However, using the adjoint divided by the determinant to calculate the inverse of the Jacobian has two major drawbacks. One, if the determinant of the Jacobian becomes zero (degenerates) the solution for the joint velocities will become unreliable. Two, the components of motion of the end-effector must equal the number of independent joints so that a square Jacobian is obtained. The ideal solution would be to find a method to invert any matrix whether degenerate or rectangular.

Such an inversion technique does exist, called the Moore-Penrose pseudoinverse [11,12,13,14]. The pseudoinverse must have certain properties. These properties are -

$$AA^{+}A = A$$

$$A^{+}AA^{+} = A^{+}$$

$$(AA^{+})* = AA^{+}$$

$$(A^{+}A)* = A^{+}A$$

where A = any matrix

A^{+} = the pseudoinverse of A

* indicates the complex conjugate transpose

We would also expect that if A has m rows and n columns the pseudoinverse A^{+} would have n rows and m columns.

When using the pseudoinverse to perform inverse kinematics of a kinematicaly redundant manipulator the Jacobian will have more columns than rows; hence, a set of consistent equations. The entire solution space for a consistent set of linear equations

$$g = A\,x$$

may be written [10,11,12] -

$$x = A^{+}g + (I - A^{+}A)h$$

where I = the identity matrix

h = any vector

Rao and Mitra go on to prove that the use of the non-homogenous part of the solution space for a consistent set of linear equations will always yield the minimum norm solution [11]. Or -

$$|| A^{+}g || \leq || A^{+}g + (I - A^{+}A)h ||$$

The minimum norm solution can be of great value in the inverse kinematics of kinematically redundant manipulators

since the inverse Jacobian relation -

$$d\Theta = J^{+}d$$

Will yield the minimum norm of the joint velocities for a given incremental move **d**.

Avoiding Joint Limits

If one is willing to sacrifice the minimum norm solution for joint velocities, the arbitrary vector **h** may be used in the pseudoinverse solution. One possible application is avoiding joint limitations in the case of non-ideal revolute joints. If **h** is picked correctly the solution of the consistent set of linear equations for joint velocities can cause the joints to tend toward the median of their travel in a least-squares norm fashion [13,14]. The solution for joint velocities would then be -

$$d\Theta = J^{+}d + \beta(J^{+}J - I)\nabla H(\theta)$$

where β = a real constant

$$H = \Sigma((\theta_i - \theta_{ci})/\Delta\theta_i)^2$$

for i = 1 to number of joints

θ_{ci} = median value for joint i

$\Delta\theta_i$ = the maximum one sided excursion for joint i

Although the use of the gradient vector to cause the joint travels to tend toward the median of their travel is sub-optimal it does lead to a easily applicable solution to deal with the limitations of non-ideal revolute joints.

Pseudoinverse Computation

There a several methods available to compute a pseudoinverse. One method is to take advantage of the complex conjugate transpose properties of the pseudoinverse [14]. where -

$$A^{+} = A^{*}(AA^{*})^{-1}$$

for the underdetermined case and -

$$A^{+} = (A^{*}A)^{-1}A^{*}$$

for the overdetermined case
However, in the undetermined case of a redundant manipulator problems do occur near degeneracies due to the ill-conditioned state of the Jacobian resulting in large joint velocities [14].

Boullion and Odell present an efficient recursive method [12]. Their method requires one recursion for each column of the original matrix and does not have any problem inverting ill-conditioned matrices because a division by zero is never allowed.

Let a_k denote the k th column of a matrix A. Let A_k be the first k columns of the matrix A. To begin let -

$A_1^{+} = 0$

if a_k equals the zero vector, otherwise let -

$A_1^{+} = (a_1^{T} a_1)^{-1} a_1^{T}$

then for k = 2 to the number of columns in A compute

$d_k = A_{k-1}^{+} a_k$

$c_k = a_k - A_{k-1} d_k$

if c_k is not equal to the zero vector

$$b_k = c_k^+$$

if c_k is equal to the zero vector

$$b_k = (1 + d_k^T d_k)^{-1} d_k^T A_{k-1}^+$$

then

$$A_k^+ = \begin{vmatrix} A_{k-1}^+ - d_k b_k \\ b_k \end{vmatrix}$$

The above method to determine the pseudoinverse is easy to follow and to translate into computer code.

CHAPTER III

PREVENTING DEGENERACIES

Types of Degeneracies

There are two types of degeneracies, mathematical and geometric. Mathematical degeneracies occur when the equations governing the inverse kinematics do not define one or more variables. Geometric degeneracies occur when a manipulator is not physically capable of performing a desired move. Both mathematical and geometric degeneracies may be observed in the Jacobian matrix.

Mathematical

Mathematical degeneracies occur when one or more column vectors of the Jacobian are zero. When a column vector becomes zero it means that the coefficient for a particular joint variable is zero in all of the equations relating incremental joint motions to incremental moves of the end of the last link. Since all of coefficients are zero, any incremental change in that joint variable would satisfy the equations of motion, resulting in an infinity of solutions.

If the inversion of the Jacobian matrix is performed by calculating the adjoint then dividing the adjoint by the determinant of the Jacobian, problems will occur near and at

a degeneracy. As the manipulator approaches a degeneracy the determinant will begin to vanish resulting in large coefficients in the inverse Jacobian matrix; hence, causing unreliable results near a degeneracy and no solution at a degeneracy.

However, if the pseudoinverse method is used to determine the inverse of the Jacobian the problem of erratic joint velocities does not occur. As stated earlier the pseudoinverse will always return the minimum norm solution to a set of linear equations. In the case of a mathematical degeneracy since any incremental move for the affected joint variable will be valid, the minimum solution would be zero.

Therefore, mathematical degeneracies do not pose a problem when solving for the inverse kinematics of a manipulator as long as the pseudoinverse technique is used to find the inverse of the Jacobian.

Geometrical

Geometrical degeneracies appear in the Jacobian as a row of zeros. A row of zeros means that all of the coefficients in an equation relating incremental joint motions to incremental moves of the last link are zero. Hence, no matter what values are given to the incremental joint motions the manipulator will not be able the perform the desired move.

There is no inversion technique that will allow the manipulator to move through a geometric degeneracies because

this type of degeneracy is a property of the geometry of the manipulator. Therefore, care must be taken when designing a manipulator to insure it has no geometric degeneracies or an least only a few discrete degeneracies that can easily be avoided.

Degeneracies of Position Structures

In this section several different types of position structures will be presented. Since mathematical degeneracies are of no concern, the position structures will be analyzed for geometric degeneracies only.

3-R Position Structure

A popular 3-R position structure is shown in figure 2. The position structure allows the end of the last link to be position anywhere in space that is within its reach but allows no control over orientation of the last link. This position structure has been studied extensively for its workspace characteristics [15] and slight variations are currently used in many commercially available robots. The kinematic parameters for this position structure are shown in table I.

Using the method to derive the Jacobian described in chapter II the Jacobian can be shown to be -

$$\begin{vmatrix} 0 & a_2 S_3 & 0 \\ 0 & a_3 + a_2 C_3 & a_3 \\ -a_3 C_{23} - a_2 C_2 & 0 & 0 \end{vmatrix}$$

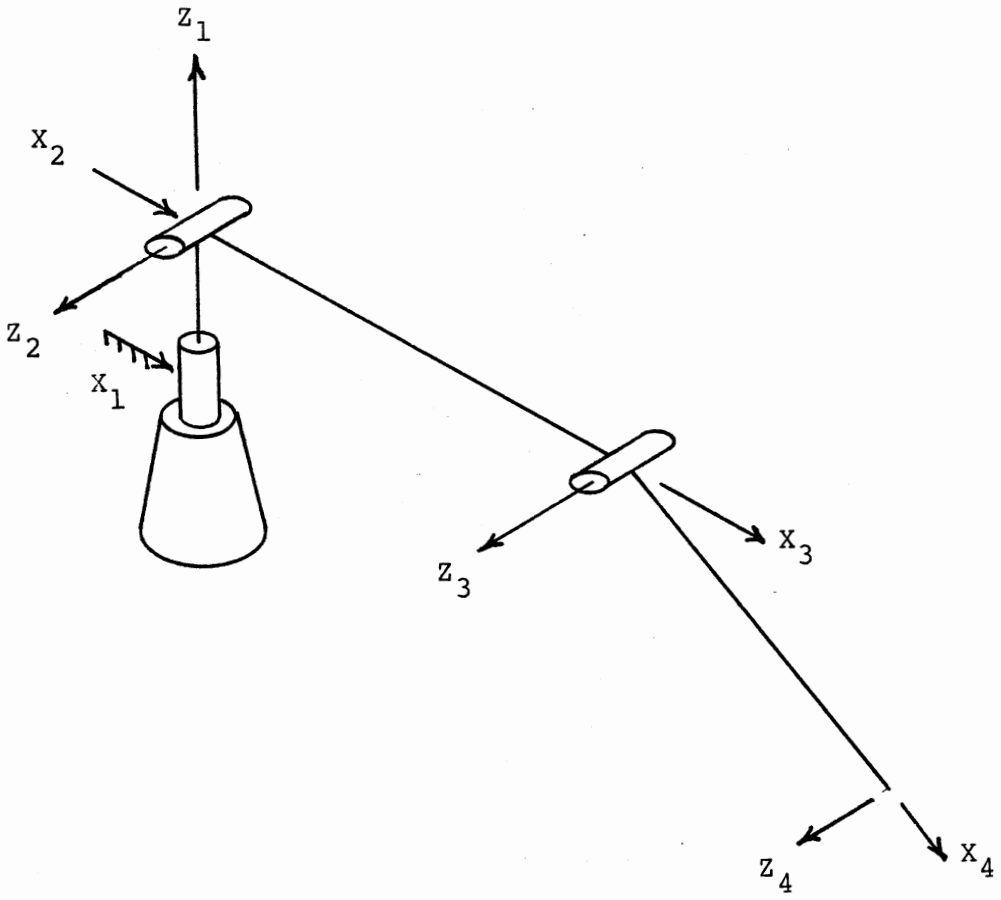

Figure 2. 3-R Position Structure

TABLE I

Kinematic Parameters for 3-R Position Structure

origin	a	α	d	θ
1	0	90°	0	variable
2	a_2	0	0	variable
3	a_3	0	0	variable

By observing row one of the Jacobian if θ_3 equals 0° or 180° a geometric degeneracy occurs. However, θ_3 equal to 0° corresponds to the outer edge of the workspace where a degeneracy would be expected regardless of the geometry of the manipulator. Also, θ_3 equal to 180° creates a degeneracy, this corresponds to the inner edge of the workspace and a degeneracy would be expected there also.

Row two can never degenerate as long as link three has a length not equal to zero.

Row three will degenerate when $-a_3C_{23}$ equals a_2C_2. The above relation will hold true when the end of the third link is on line with the Z_1 axis. When this alignment occurs the end of the third link is not able to move in the Z_4 direction regardless of the incremental joint rotations.

Note that a mathematical degeneracy occurs at the same time as the row three geometrical degeneracy. The column one mathematical degeneracy is of no consequence if the pseudo-inverse technique is used to invert the Jacobian. The end of the third link would be able to move in the X_3 and Y_3 directions but would be prevented from moving in the Z_3 direction due to the geometrical degeneracy.

4-R Position Structure

A proposed 4-R position structure is shown in figure 3. Like the 3-R position structure the 4-R position structure may position the end of the last link anywhere within its reach but has no control of the orientation of the last

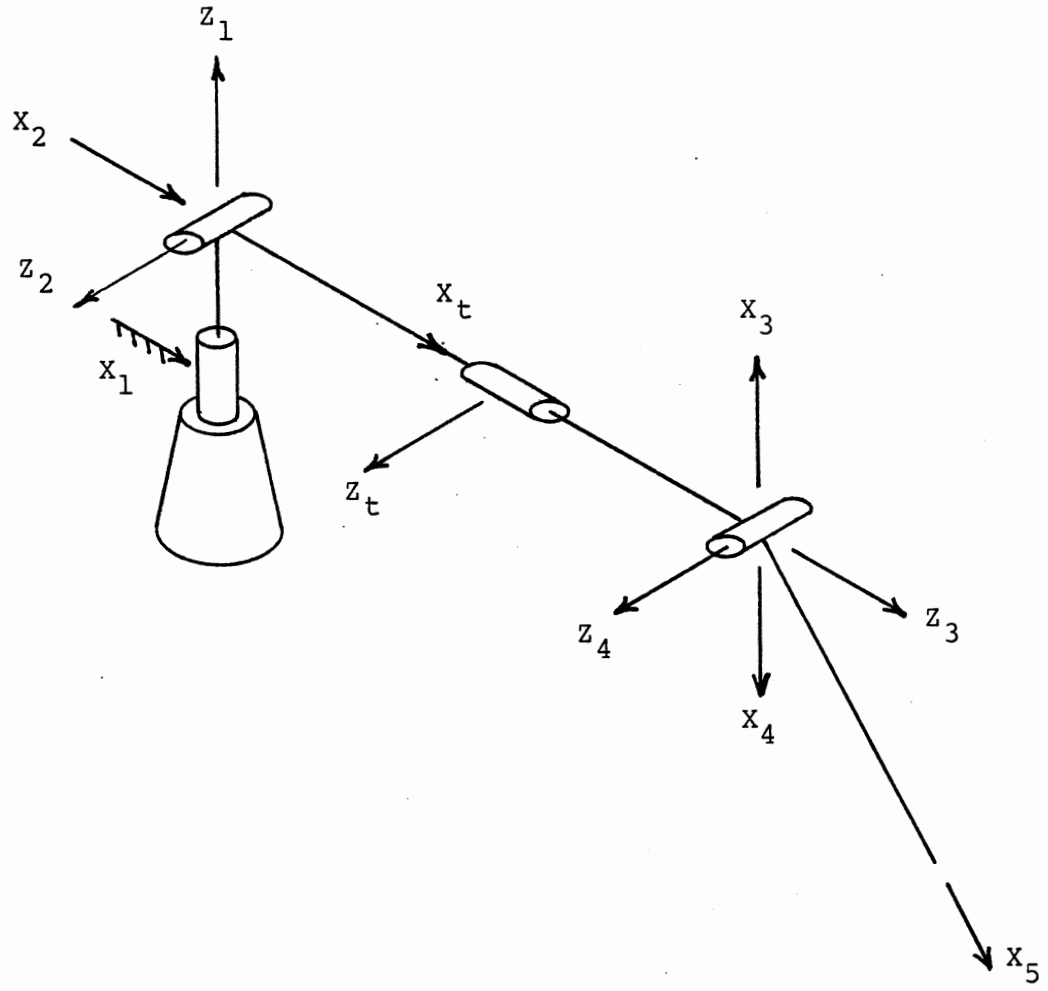

Figure 3. 4-R Position Structure

link. The kinematic parameters for the proposed 4-R position structure are shown in table II.

Note the an additional transformation used between the second and third joint. This additional transformation is not needed to mathematically describe the position of the end of the last link, but is needed to describe the position of joint 3 which is needed to accurately animate the proposed 4-R position structure. Therefore, the additional transformation is included here for the sake of completeness.

Using the procedure in chapter II to derive the Jacobian matrix and the data in table II the Jacobian matrix can be shown to be.

$$\begin{vmatrix} AA & [\,(d_3+a_2)C_3C_4\,] & [\,0\,] & [\,0\,] \\ BB & [a_4S_3S_4+(d_3+a_2)S_3] & [-a_4C_4] & [\,0\,] \\ CC & [a_4C_3+(d_3+a_2)C_3S_4] & [\,0\,] & [-a_4] \end{vmatrix}$$

where $AA = -(d_3+a_2)C_2S_3C_4$

$BB = a_4(C_2C_3S_4-S_2C_4) + (d_3+a_2)C_2C_3$

$CC = a_4(S_2C_3C_4-C_2C_4-C_2S_4)S_3C_4 - (d_3+a_2)C_2S_3C_4$

By observing the Jacobian it is apparent that if the length of the fourth link is nonzero, row three may never become degenerate.

Looking at row two if θ_4 becomes $\pm 90°$ at the same time θ_3 becomes 0 or 180° row two will degenerate. However, θ_4 of $\pm 90°$ corresponds to the outer and inner edge of the workspace where a geometrical degeneracy is expected.

The problem occurs in row one of the Jacobian. Besides

TABLE II

Kinematic Parameters for 4-R Position Structure

origin	type	a	α	d	θ
1	joint	0	90°	0	variable
2	joint	a_2	0	0	variable
t	trans	0	90°	0	90°
3	joint	0	90°	d_3	variable
4	joint	a_4	90°	0	variable

the edge of the workspace degeneracy, if θ_3 becomes ±90° at the same time θ_2 becomes ±90° row one degenerates meaning the end of the last link cannot move in the X_5 direction (see figure 4). The row one degeneracy in general does not occur at the edge of the workspace; therefore, may cause a problem in the inverse kinematics of the 4-R position structure.

However, if θ_3 is not allowed to become ±90° a row one degeneracy would only occur at the edge of the workspace. θ_3 may be kept away from the ±90° positions by using the entire solution for a consistent set of linear equations.

$$d\boldsymbol{\theta} = \mathbf{J}^{+}\mathbf{d} + \beta(\mathbf{J}^{+}\mathbf{J} - \mathbf{I})\nabla H(\theta)$$

The least-squares function would be -

$$H = (\theta_3 / \Delta\theta_3)^2$$

making the gradient vector -

$$\nabla H = \begin{vmatrix} 0 \\ 0 \\ (2\theta_3)/(\Delta\theta_3)^2) \\ 0 \end{vmatrix}$$

Figure 5 shows the motion of the joints of the 4-R position structure performing an incremental move with the gain constant ß equal to zero. Notice θ_3 moves away from its center position of zero degrees.

Figure 6 shows the motion of the joint variables performing the same incremental move as in figure 5 except for figure 6 ß is set to 0.05. Notice now θ_3 moves toward its center position; hence, the non-edge of workspace degeneracies can be avoided.

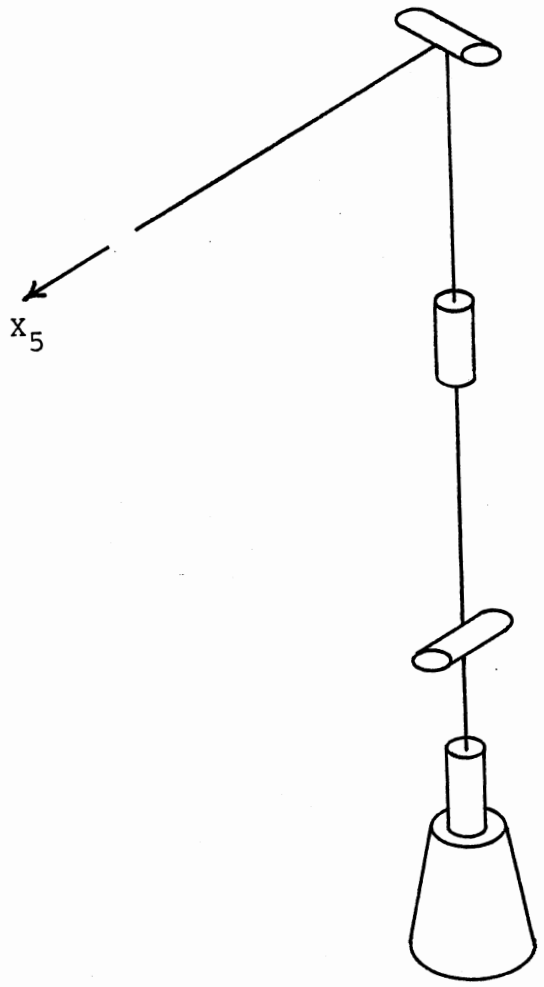

Figure 4. 4-R Position Structure at Geometrical Degeneracy

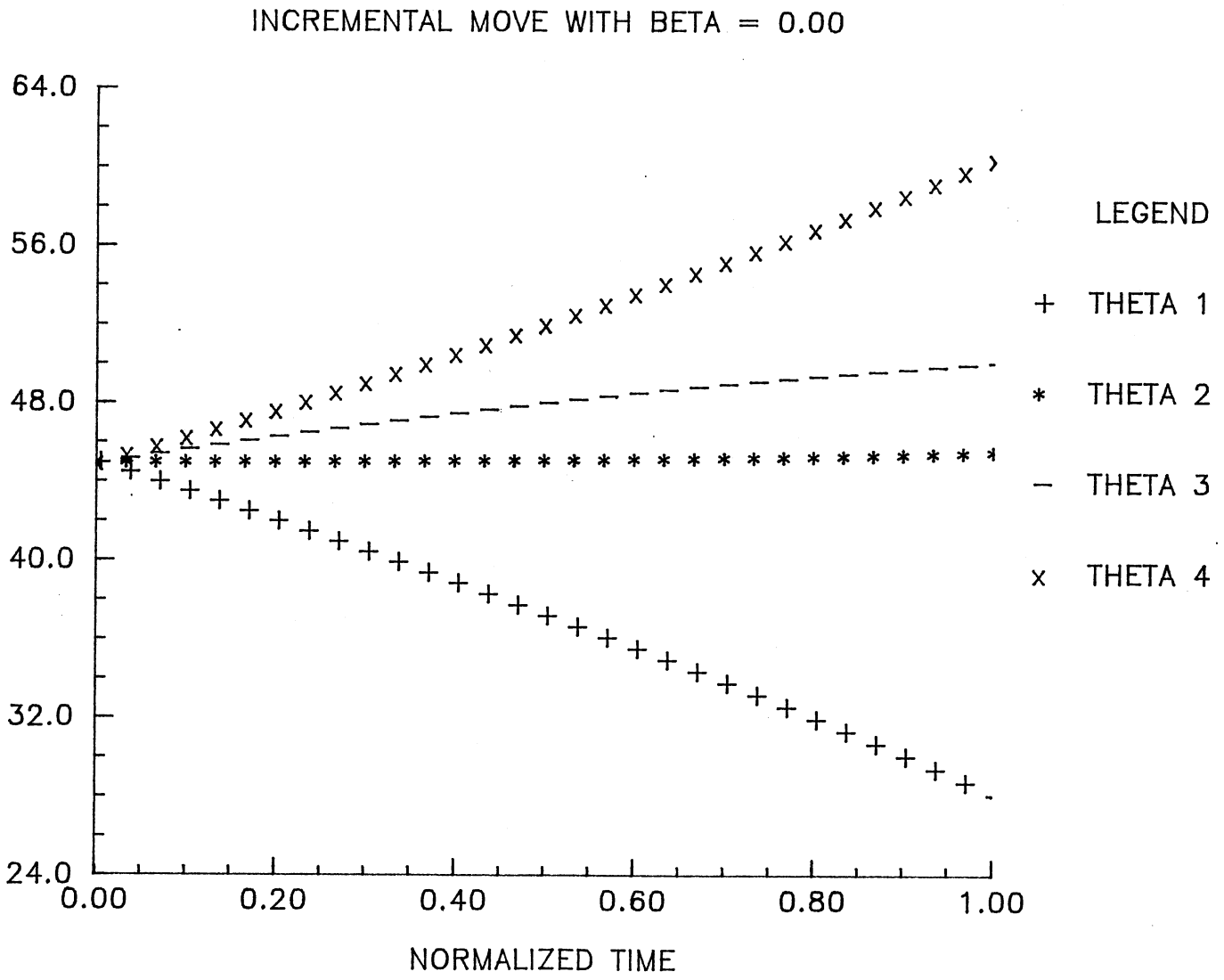

Figure 5. Incremental Move with BETA = 0.00

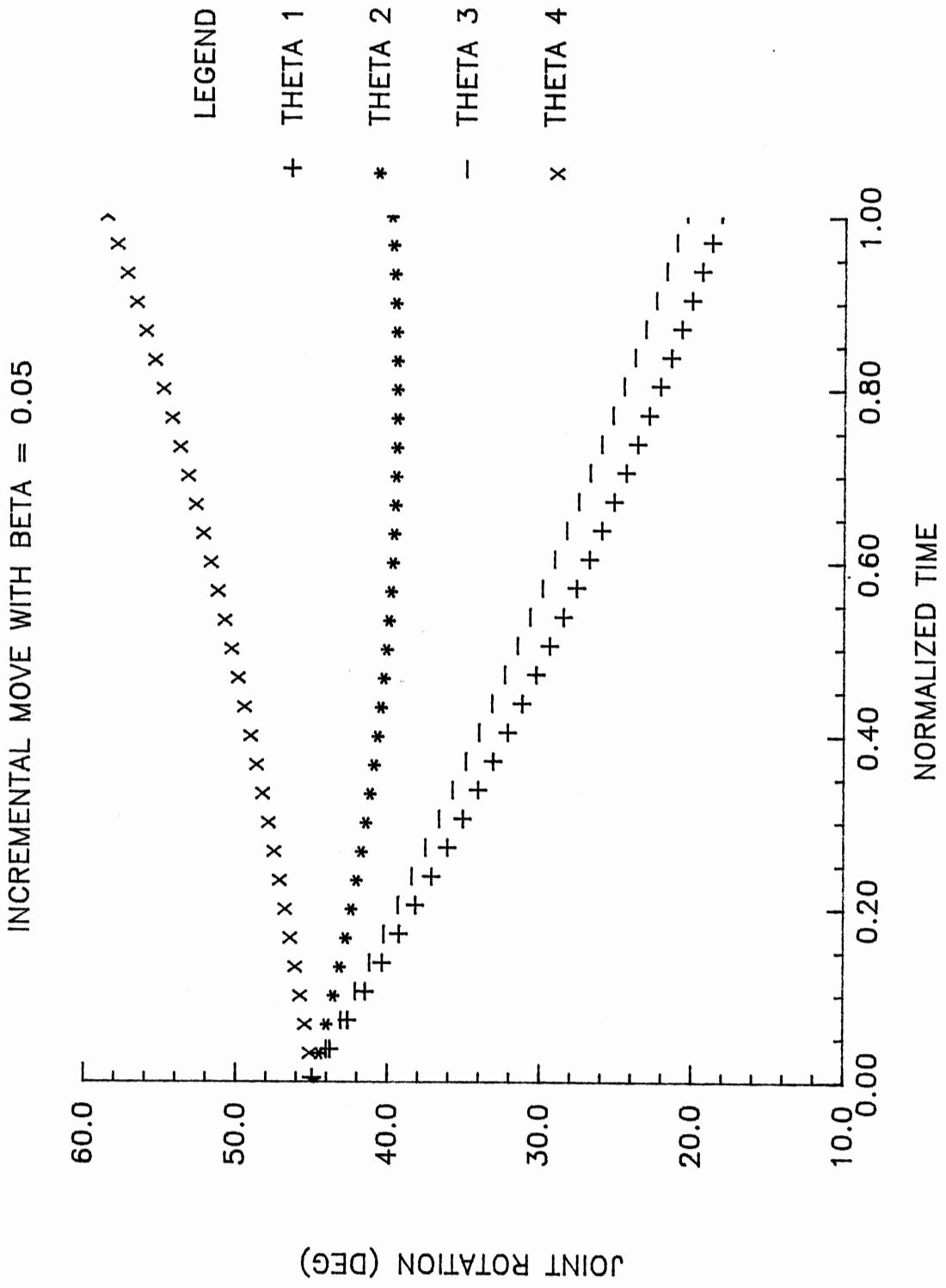

Figure 6. Incremental Move with BETA = 0.05

5-R Position Structure (type 1)

Figure 7 shows a proposed 5-R position structure. This position structure is similar to the 4-R position structure studied by Laughlin [16] shown in figure 8. Laughlin proved that the position structure shown in figure 8 would have excellent workspace characteristics with non-ideal joints. However, from the analysis of the 3-R position structure it is obvious that a geometric shoulder degeneracy would occur with the Laughlin position structure. Hence, the addition of a joint placed axially between the second and third joint of the Laughlin position structure may eliminate the shoulder degeneracy.

Table III shows the kinematic parameters for the 5-R position structure. Note additional transformations are not included to reduce computation.

Using the data in table III the Jacobian is -

$$\begin{vmatrix} AA & DD & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} a_4S_5 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \\ BB & EE & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} a_5+a_4C_5 \end{bmatrix} & \begin{bmatrix} a_5 \end{bmatrix} \\ CC & FF & \begin{bmatrix} -a_5C_{45}-a_4C_4 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \end{vmatrix}$$

where AA = is irrelevant to discussion

BB = is irrelevant to discussion

$CC = a_5(C_2C_{45}+S_2C_3S_{45}) + a_4(C_2C_4+S_2C_3S_4) + d_3(S_2C_3)$

$DD = -a_4(C_3S_5) + d_3(C_3C_{45})$

EE = is irrelevant to discussion

$FF = a_5(S_3S_{45}) + a_4(S_3S_4) + d_3(S_3)$

By observing row 1 of the Jacobian θ_5 must equal 0 or

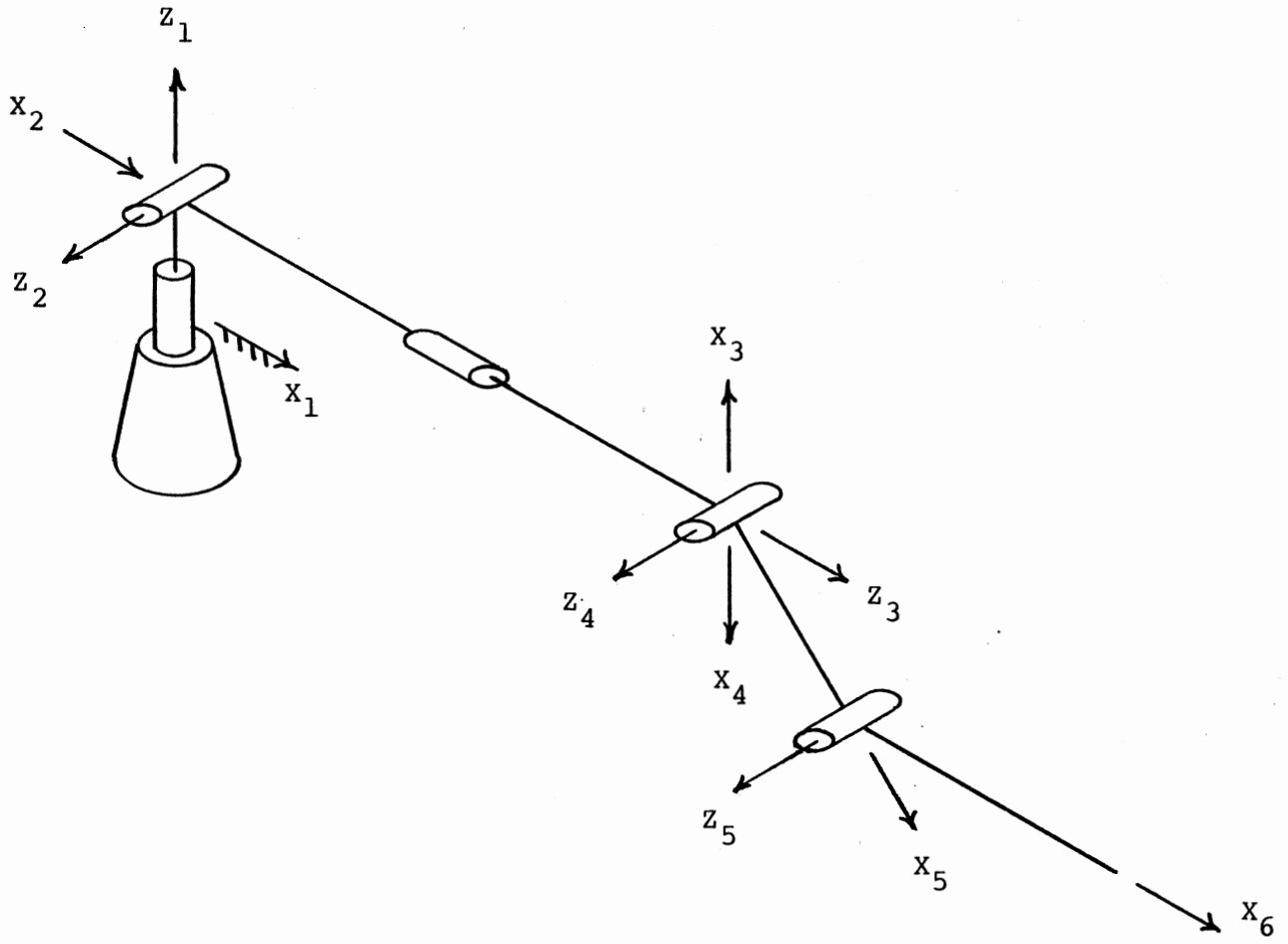

Figure 7. 5-R Position Structure (type 1)

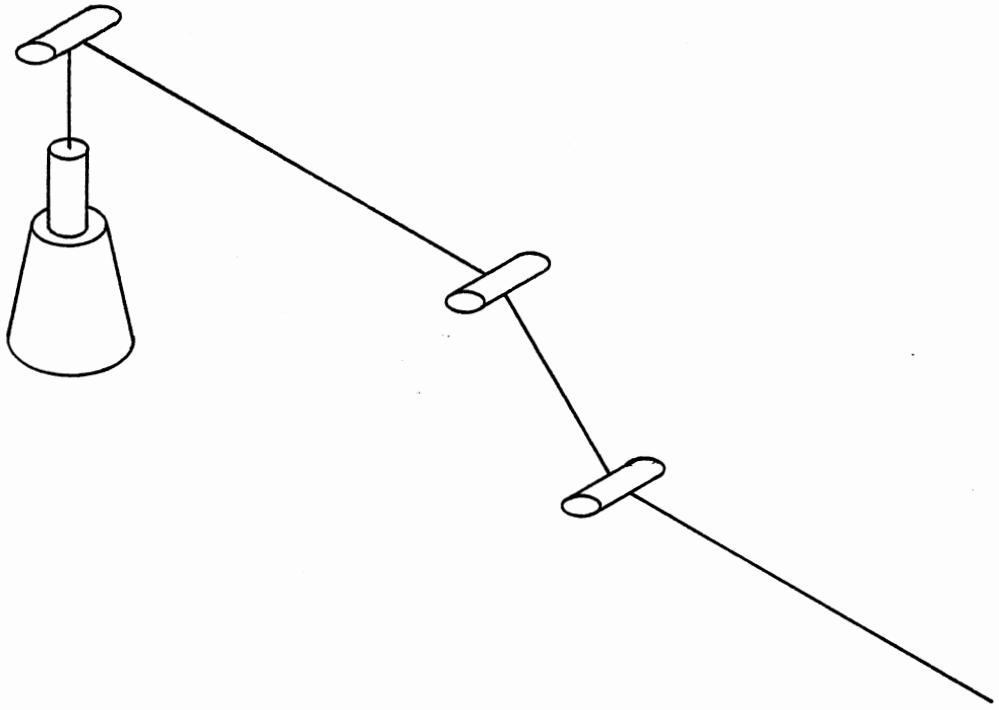

Figure 8. Laughlin Position Structure

TABLE III

Kinematic Parameter for 5-R Position Structure
(type 1)

origin	a	α	d	θ
1	0	90°	0	variable
2	0	90°	0	variable
3	0	90°	d_3	variable
4	a_4	0	0	variable
5	a_5	0	0	variable

180° at the same time either θ_3 or θ_4 equals ±90° for row 1 to have the possibility to equal the zero vector. However, with the coordinate systems chosen if θ_5 equals 0 or 180° at the same time θ_4 equals ±90° the end of the last link is at the boundary of the workspace where a degeneracy is expected. This leaves the possibility of θ_3 equal to ±90°. However, if θ_3 is treated in the same way as the in the proposed 4-R position structure a row 1 degeneracy will not occur within the position structures workspace.

Since the row 2, column 5 entry of the Jacobian is a constant, a row 2 degeneracy will not occur as long as link 5 is of non-zero length.

Row 3 of the Jacobian requires close observation. The row 3, column 3 entry will equal zero when the end of the last link aligns with the Z_3 axis. The row 3, column 2 entry will equal zero when θ_3 equals 0 or 180° or when the end of the last link touches a plane defined by a normal to the Z_3 axis passing through the origin of the second coordinate system. Realistically, due to finite link dimensions, the end of the last link will never reach the two positions stated above simultaneously; however, it is very possible that θ_3 may equal 0 when the end of the last link aligns with the Z_3 axis since θ_3 will tend toward 0 due to the gradient used to keep row 1 from degenerating.

Observing the row 3, column 1 entry and remembering that θ_3 will tend toward 0, and the end of the last link will not align with the Z_3 axis at the same time it touches

a plane normal to Z_3 passing through the origin of the second coordinate system there will be an infinity of row 3 degeneracies. This infinity will occur when the Z_1, Z_3 and end of the last link are all colinear, see figure 9.

The row 3 degeneracy may be avoided by using the gradient vector to keep θ_2 between 0 and 180°; however, part of the workspace will be sacrificed.

5-R Position Structure (type 2)

Figure 10 shows a different configuration for a 5-R position structure. The second 5-R position structure has a joint placed axially between the third and fifth joint instead of between the second and fourth joint as in the type 1 5-R position structure. The kinematic parameters for the type 2 5-R position structure are shown in table IV.

Using the data in table IV the Jacobian is -

$$\begin{vmatrix} AA & DD & \begin{bmatrix} d_4C_4C_5 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \\ BB & EE & \begin{bmatrix} -a_5C_4 & -d_4C_4S_5 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} a_5 \end{bmatrix} \\ CC & FF & \begin{bmatrix} a_5S_4S_5 & +d_4S_4 \end{bmatrix} & \begin{bmatrix} -a_5C_5 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \end{vmatrix}$$

where AA = is irrelevant to discussion

BB = is irrelevant to discussion

$CC = a_5(C_2C_3C_5 - S_2S_3C_5 + S_{23}C_4S_5) + d_4S_{23}C_4 + a_2C_2C_4$

$DD = d_4C_4C_5 + a_2(S_3C_4C_5 - C_3S_5)$

EE = is irrelevant to discussion

$FF = a_5S_4S_5 + d_4S_4 + a_2S_3S_4$

Row 1 will degenerate when θ_3 becomes ±90° and either θ_4 or θ_5 become ±90°. However, if θ_3 and θ_5 become ±90° the

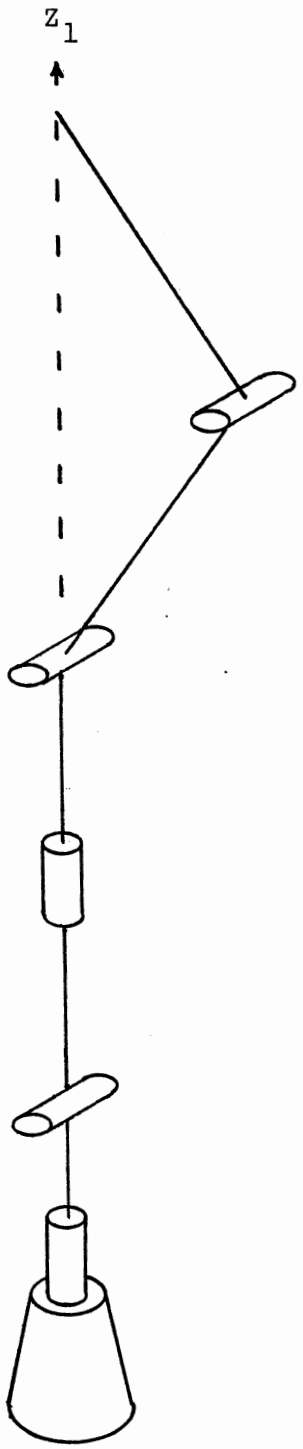

Figure 9. 5-R Position Structure (type 1) at Geometrical Degeneracy

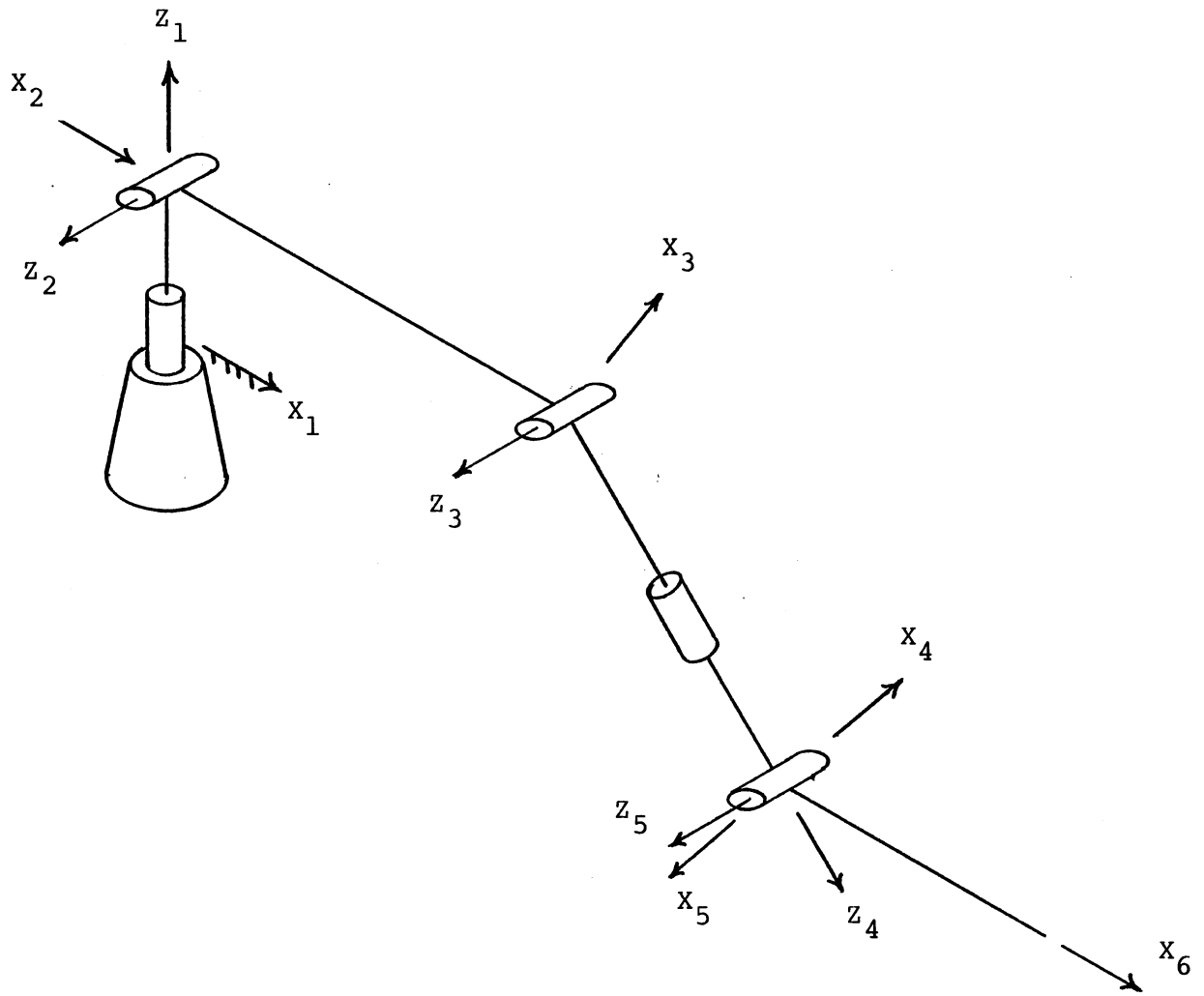

Figure 10. 5-R Position Structure (type 2)

TABLE IV

Kinematic Parameters for 5-R Position Structure
(type 2)

origin	a	α	d	θ
1	0	90°	0	variable
2	a_2	0	0	variable
3	0	90°	0	variable
4	0	90°	d_4	variable
5	a_5	0	0	variable

position structure is at the edge of its workspace where a degeneracy is expected. Therefore, if the gradient vector is used to keep θ_4 toward the middle of its range a row 1 degeneracy will not occur within the position structures workspace.

Since the row 2, column 5 entry is a constant a row 2 degeneracy will not occur as long as link 5 is of non-zero length.

For a row 3 degeneracy to occur θ_2, θ_3 and θ_5 must become $\pm 90°$ at the same time θ_4 becomes 0 or 180°. Hence, a row 3 degeneracy will occur when all three links become co-axial. If all three links become coaxial the position structure is at the end of its workspace where a degeneracy is expected. The degeneracies within the workspace require two or more links to occupy the same space which will not occur for two reasons. First, it is physically impossible for two links to occupy the same space. Second, joints 2, 3 and 5 need not have entire mobility to have excellent workspace characteristics [16].

Therefore, it may be more prudent to use the type 2 5-R position structure than the type 1 because the type 2 will have a finite number of geometric degeneracies that are easily avoidable without sacrificing workspace.

Orientational Degeneracies

In the previous section several proposed position structures were analyzed for geometric degeneracies. In

this section 3-R orientation structures will be added to each of the position structures and the entire manipulators will be analyzed for orientational geometric degeneracies. Since the orientation structure will not add to the workspace, if a manipulator can be shown to have no orientational geometric degeneracies it can be deduced that the manipulator has no unavoidable geometric degeneracies.

Figure 11 shows the four different types of 3-R orientation structures that allow full dexterity [15]. Since each joint in these orientation structures is positioned at a 90° alpha angle from each other they are kinematically similar. Therefore, if the first joint in the orientation structure is position at a 90° alpha angle with respect to the last joint of the position structure it will be sufficient to study one orientation structure with each position structure.

6-R Manipulator

Figure 12 shows a 6-R manipulator consisting of the 3-R position structure studied above and a 3-R orientation structure. The kinematic parameters for the 6-R manipulator are shown in table V.

To study orientational degeneracies the Jacobian will be derived to relate incremental joint motions to incremental rotations of the end of the last link as shown in chapter II. Using the kinematic parameters in table V the orientational Jacobian for the 6-R manipulator is -

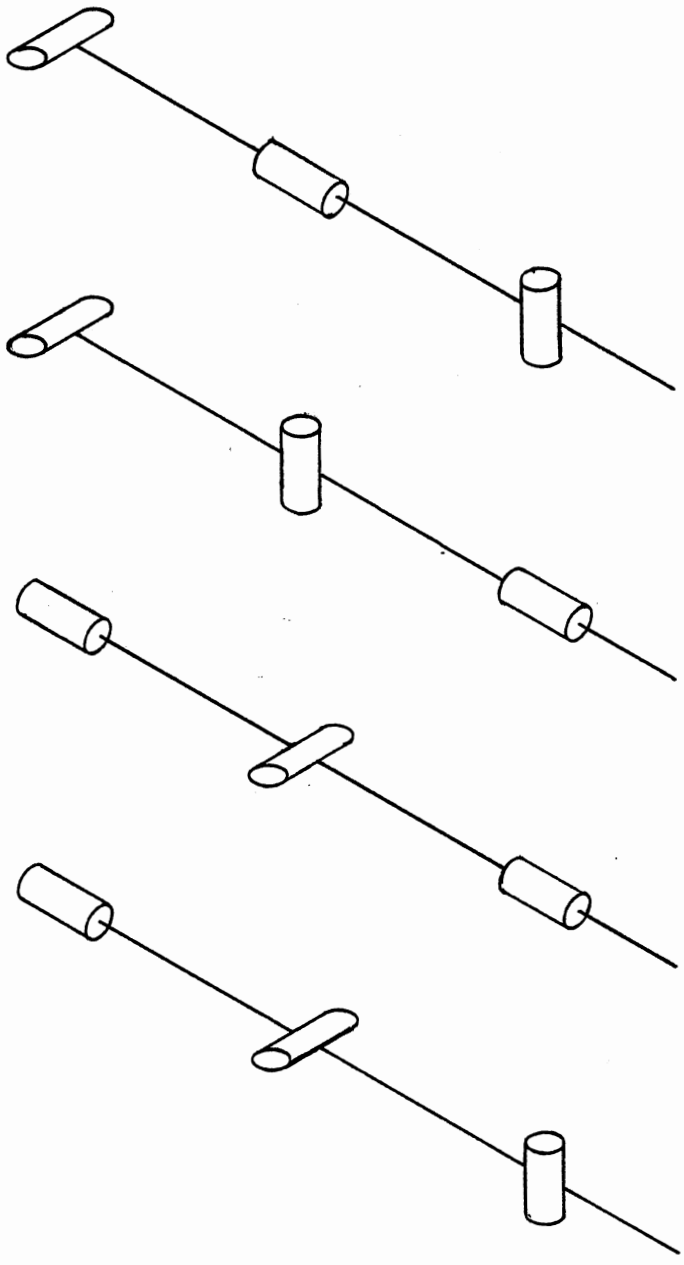

Figure 11. 3-R Orientation Structures

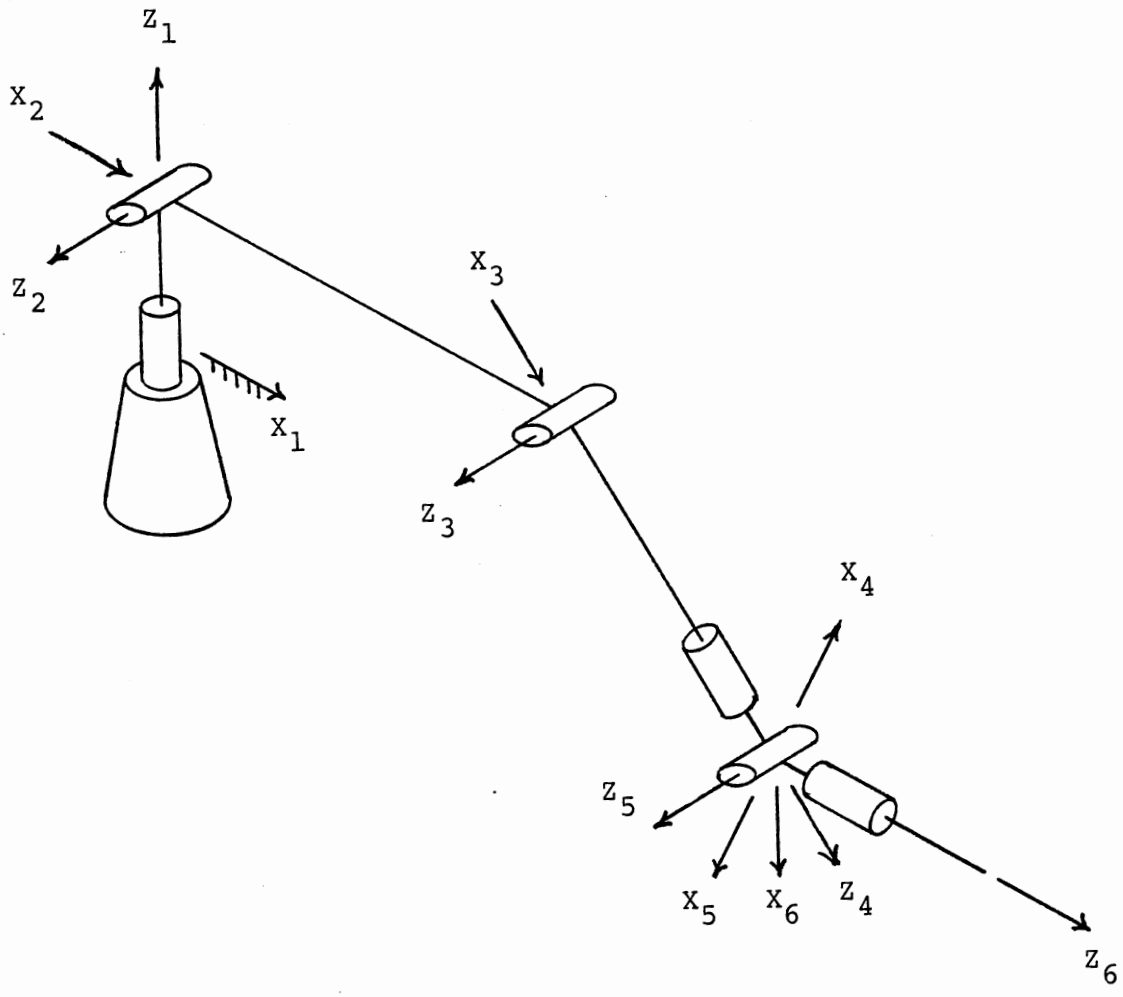

Figure 12. 6-R Manipulator

TABLE V

Kinematic Parameters for 6-R Manipulator

origin	a	α	d	θ
1	0	90°	0	variable
2	a_2	0	0	variable
3	a_3	90°	0	variable
4	0	90°	0	variable
5	0	90°	0	variable
6	0	0	0	variable

$$\begin{vmatrix} AA & [\,S_4C_5C_6-C_4S_6] & [\,S_4C_5C_6-C_4S_6] & [\,S_5C_6] & [S_6] & [0] \\ BB & [-S_4C_5S_6-C_4C_6] & [-S_4C_5S_6-C_4C_6] & [-S_5S_6] & [C_6] & [0] \\ CC & [\quad S_4S_5\quad] & [\quad S_4S_5\quad] & [\,-C_6\,] & [\,0] & [1] \end{vmatrix}$$

where $AA = S_2[C_3(\,C_4C_5C_6+S_4S_6) + S_3S_5C_6] + C_2[S_3(\,C_4C_5C_6+S_4S_6) - C_3S_5C_6]$

$BB = S_2[C_3(-C_4C_5S_6+S_4C_6) - S_3S_5S_6] + C_2[S_3(-C_4C_5S_6+S_4C_6) + C_3S_5S_6]$

$CC = S_2[C_3C_4S_5-S_3C_6] + C_2[S_3C_4S_5+C_3C_6]$

Observing row 1 if θ_4, θ_5 and θ_6 become 0 or 180° and the same time the sine of θ_2 plus θ_3 becomes zero a row 1 geometrical degeneracy will result.

In row 2 if θ_4 and θ_5 become 0 or 180°, and θ_6 is ±90° at the same time the sine of θ_2 plus θ_3 becomes zero a row 2 geometrical degeneracy will result.

Row 3 will never degenerate due to the constant at column 6.

From rows 1 and 2 it is obvious that the 6-R manipulator has an infinity of orientationally degenerate configurations making it very difficult to guarantee a non-degenerate solution for the inverse kinematics.

7-R Manipulator

Figure 13 shows a 7-R manipulator that is capable of positioning and orienting the last link of the manipulator anywhere in its workspace. The 7-R manipulator consists of the 4-R position structure studied in the previous section and a 3-R orientation structure. As demonstrated in the previous section the position structure contains a finite

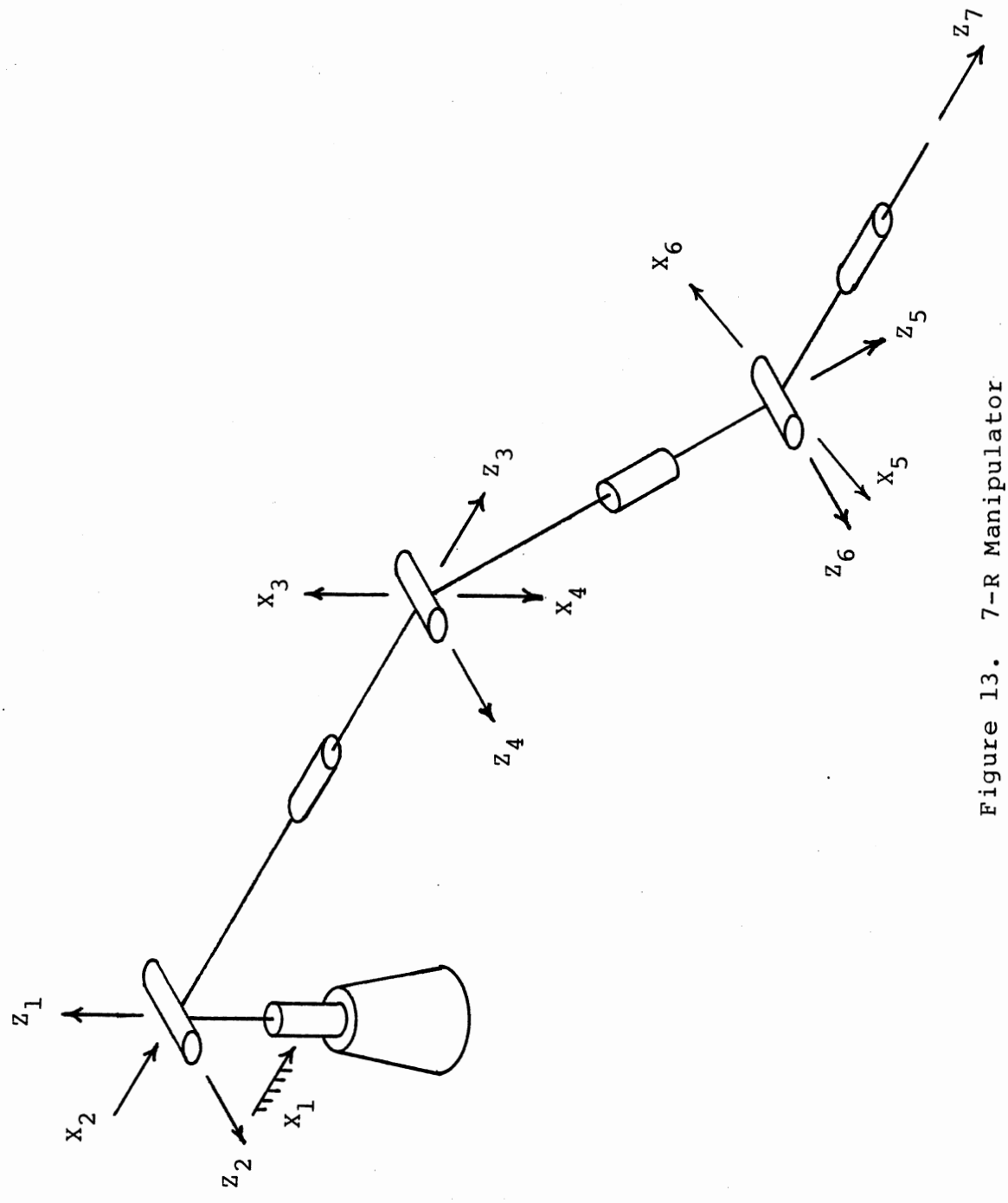

Figure 13. 7-R Manipulator

number of geometric degeneracies that may be avoided using the entire solution to the consistent set of linear equation for inverse kinematics. Therefore, if the proposed 7-R manipulator contains no geometric orientational degeneracies the 7-R manipulator will have no unavoidable degeneracies because it has already been demonstrated that the geometric degeneracies associated with the positioning can easily be avoided.

Table VI contains the link parameters for the 7-R manipulator. Note this time additional transformations to describe the position of all the joints are not included since the derivation of the Jacobian will become less tractable.

Using the method described in Chapter 2 to derive the orientational Jacobian the Jacobian can be shown to be -

$$\begin{vmatrix} AA & DD & [S_4(C_5C_6C_7+S_5S_7)-C_4S_6C_7] & [S_5C_6C_7-C_5S_7] \\ BB & EE & [S_4(-C_5C_6S_7+S_5C_7)+C_4S_6S_7] & [-S_5C_6S_7-C_5C_7] \\ CC & FF & [S_4C_5S_6+C_4C_6] & [S_5S_6] \end{vmatrix} \begin{matrix} S_5C_7 & S_7 & 0 \\ -S_6S_7 & C_7 & 0 \\ -C_6 & 0 & 1 \end{matrix} \Bigg|$$

where $AA = S_2\{C_3[C_4(C_5C_6C_7+S_5S_7)+S_4S_6C_7] + S_3[S_5C_6C_7-C_5S_7]\} - C_2\{S_4(C_5C_6C_7+S_5S_7)-C_4S_6C_7\}$

$BB = S_2\{C_3[C_4(-C_5C_6C_7+S_5S_7)+S_4S_6S_7] - S_3[S_5C_6S_7+C_5C_7]\} + C_2\{S_4(-C_5C_6C_7+S_5C_7)+C_4S_6S_7\}$

$CC = S_2\{C_3(C_4C_5S_6-S_4C_6)+S_3S_5S_6\} - C_3\{S_4C_5S_6+C_4C_6\}$

$DD = S_3[C_4(C_5C_6C_7+S_5S_7)+S_4S_6C_7] - C_3[S_5C_6C_7-C_5S_7]$

$EE = S_3[C_4(-C_5C_6S_7+S_4S_7)-S_4S_6S_7] + C_3[S_5C_6S_7+C_5C_7]$

$FF = S_3(C_4C_5S_6-S_4C_6) - C_3S_5S_6$

TABLE VI

Kinematic Parameters for 7-R Manipulator

origin	a	α	d	θ
1	0	90°	0	variable
2	0	90°	0	variable
3	0	90°	d_3	variable
4	0	90°	0	variable
5	0	90°	d_5	variable
6	0	90°	0	variable
7	0	0	0	variable

Observing row 1 of the Jacobian, θ_2 through θ_7 must all equal 0 or 180° for row 1 to geometrically degenerate. But, if θ_4 is equal to 0 or 180° the manipulator is at the outer or inner edge of the workspace where a degeneracy of the position structure is expected.

Observing row 2 of the Jacobian, θ_2 through θ_6 must all equal 0 or 180° and θ_7 must equal ±90° for row 2 to geometrically degenerate. Here again, θ_4 must equal 0 or 180° where the position structure is at the outer or inner edge of its workspace and a degeneracy of the position structure is expected.

Row 3 of the Jacobian of the Jacobian will never degenerate because of the constant in column 7.

Therefore, the 7-R manipulator will geometrically degenerate at the edge of its workspace and at a finite number of joint values within its workspace. The degeneracies at the edge of the workspace are to be expected and cannot be eliminated because a manipulator with finite link lengths will always have a finite reach. The geometric degeneracies that lie within the workspace of the manipulator occur at a finite number of joint values and can be easily avoided using the entire solution of the consistent set of linear equations as demonstrated above.

8-R Manipulator

Figure 14 shows the 5-R position structure type 1 with a 3-R orientation structure. Figure 15 shows the 5-R

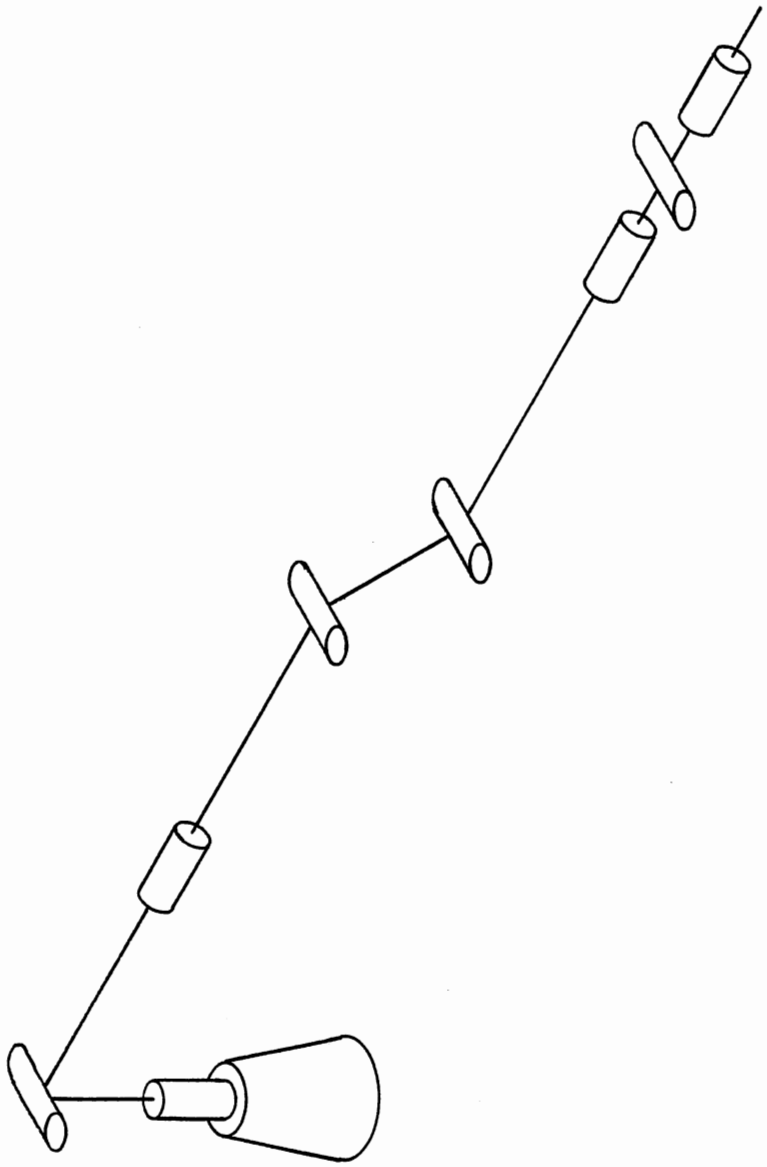

Figure 14. 8-R Manipulator (type 1)

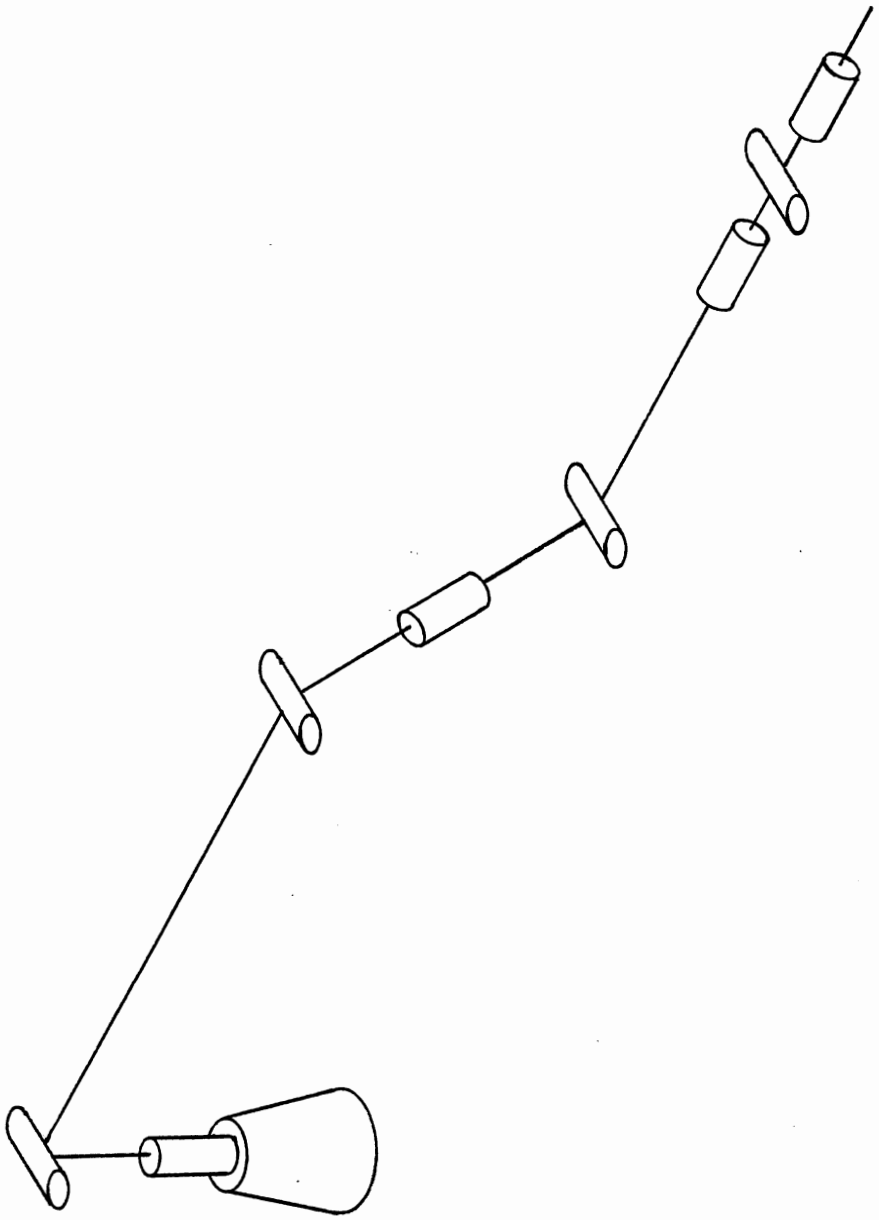

Figure 15. 8-R Manipulator (type 2)

position structure type 2 with a 3-R orientation structure. The orientational Jacobian will not be derived here since a logical argument will be sufficient to prove the orientational non-degeneracy of the proposed 8-R manipulators.

It was proved above that the 7-R manipulator was in danger of an orientational degeneracy only when all the links aligned in a specific fashion. However, it was also noted that when the alignment occurred the manipulator had reached the edge of the workspace where a degeneracy is expected.

Now since the 8-R manipulators are constructed by adding a joint to the 7-R manipulator no new orientational degeneracies can occur. Therefore, the 8-R manipulator is in danger of an orientational degeneracy only when all the links align in a specific fashion. As stated in the sections on the 5-R position structures the alignment of all the links cannot and need not occur anywhere but at the edge of the workspace where a degeneracy is to be expected. Hence, the 8-R manipulators will have no orientational degeneracies within their workspace.

CHAPTER IV

WORKSPACE PERFORMANCE

In this chapter the 6-R, 7-R and 8-R manipulators will be analyzed for their performance in a workspace environment. By analyzing each manipulator in its working environment it can be determined under which circumstances a particular manipulator should be used. In general the simplest acceptable solution would be the preferred solution; therefore, there must exist a practical reason why a 7-R manipulator is preferred over a 6-R manipulator if a 7-R manipulator is to be used. Likewise, there must exist a practical reason why an 8-R manipulator is preferred over a 7-R manipulator if an 8-R manipulator is to be used.

Workspace Simulation

The most simple and inexpensive way to evaluate the workspace performance of a manipulator is a computer simulation. By animating a manipulator as it performs a variety of tasks the user can get valuable qualitative information on a manipulators performance near and at degenerate configurations.

For this study a computer package was written that performs the inverse kinematics and graphical animation for

any N-R manipulator. First the user creates a data base containing the geometry of the manipulator. Then the manipulator can be sent through a series of straight line moves. The user can evaluate the performance of the manipulator by watching the graphical animation and by noting the norm of the joint motions which is calculated at the end of each move. Since the program numerically derives and inverts the Jacobian matrix a variety of manipulators can be quickly tested by merely revising the data base.

6-R Manipulator vs 7-R Manipulator

In this section the 6-R manipulator is compared to the 7-R manipulator in a workspace environment. Figures 16 and 17 show the 6-R and 7-R manipulators respectively. Note the two cartesian reference frames depicted, the large axes depict the global reference frame which remains stationary. The small axes, attached to the end of the last link is the tool reference frame which moves with the end of the last link. The computer package will allow straight line motion with respect to either the global or tool reference frames.

Figures 18 and 19 show a trace animation of the 6-R and 7-R manipulators executing a move in the tool Z direction only, while not allowing a reorientation of the reference frame attached to the end of the last link.

Figures 20 and 21 show a trace animation of the 6-R and 7-R manipulators executing a move in the global Z direction along the Z_1 axis. Note from chapter III that when the end

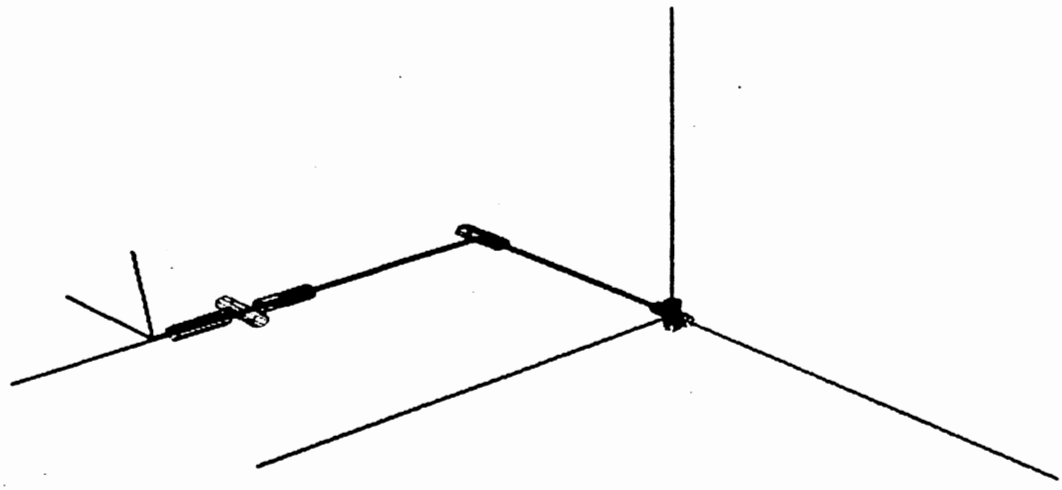

Figure 16. Computer Simulation of 6-R Manipulator

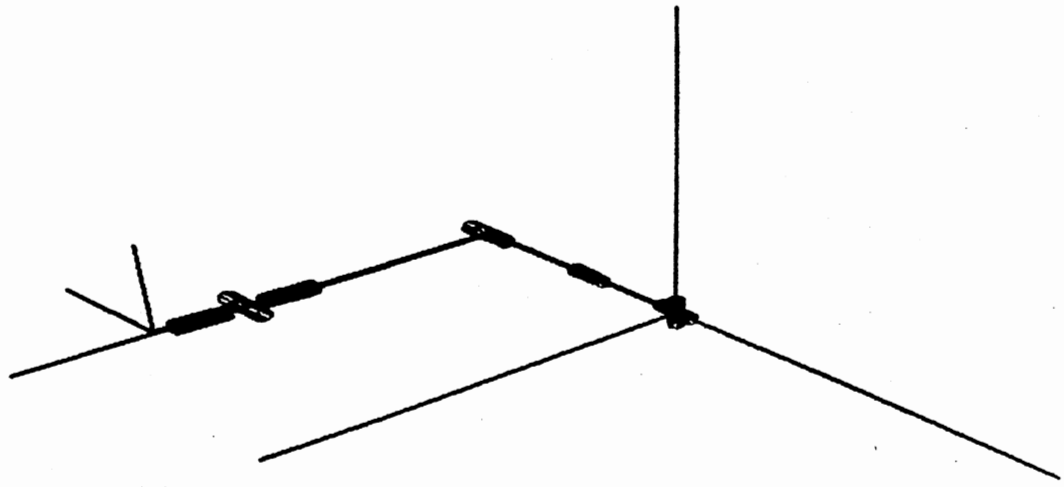

Figure 17. Computer Simulation of 7-R Manipulator

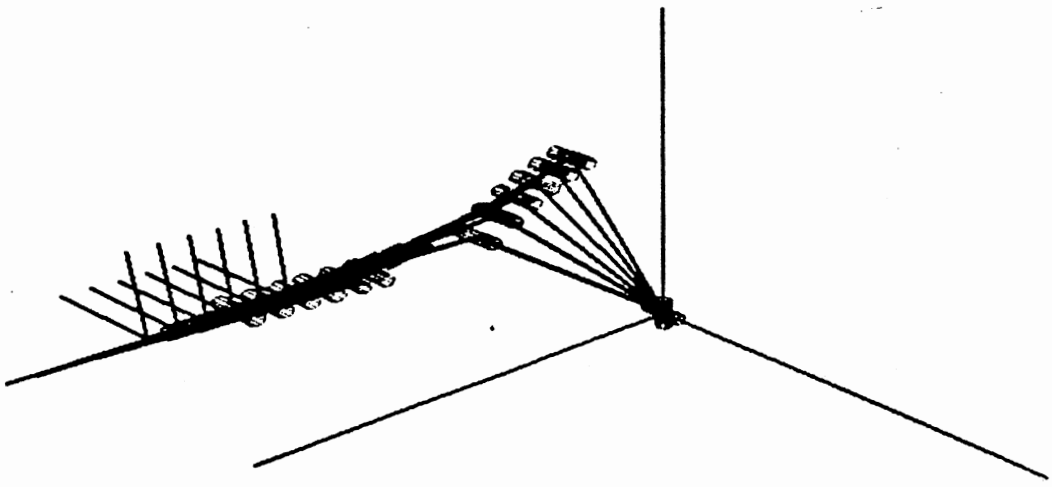

Figure 18. Trace Animation of 6-R Manipulator moving in Tool Z Direction

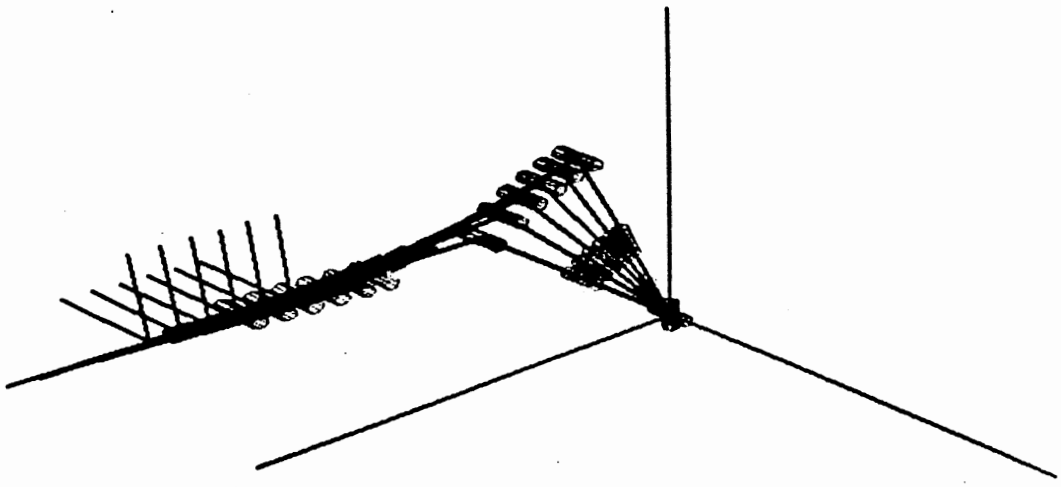

Figure 19. Trace Animation of 7-R Manipulator moving in Tool Z Direction

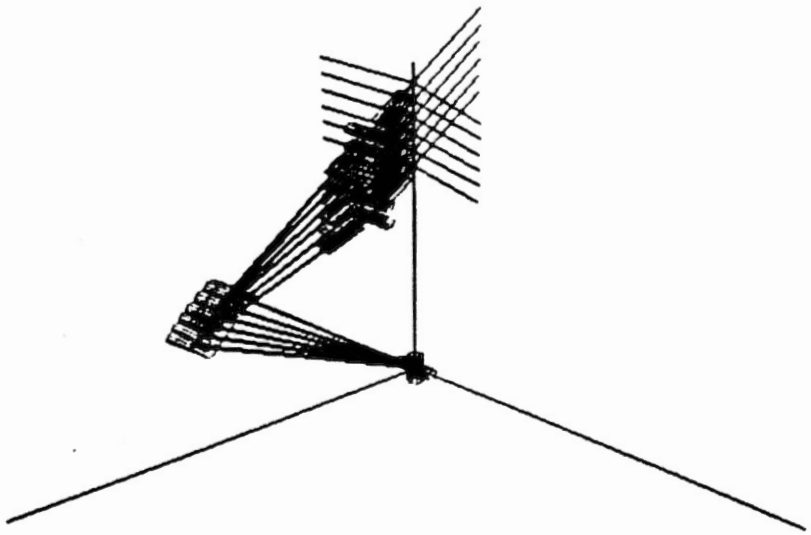

Figure 20. Trace Animation of 6-R Manipulator moving along Z_1 Axis

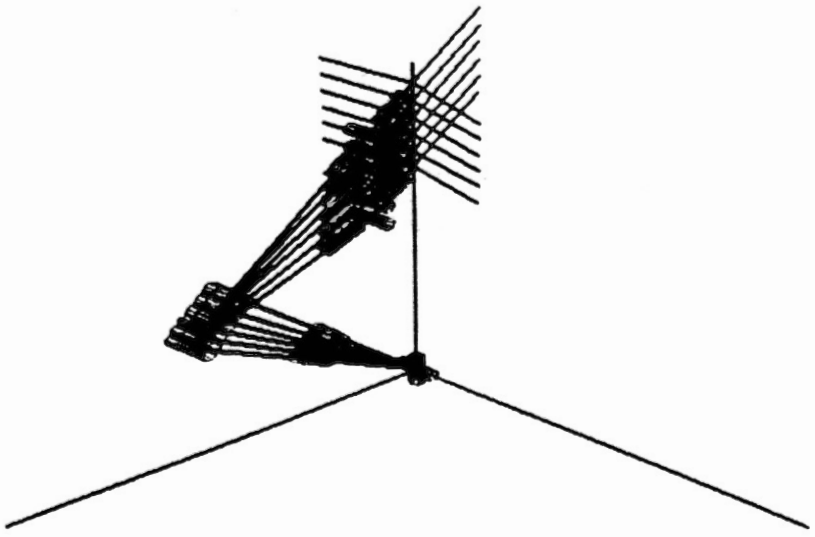

Figure 21. Trace Animation of 7-R Manipulator moving along Z_1 Axis

of the last link aligns with the Z_1 axis both the 6-R and 7-R manipulators are mathematically degenerate; however, since the inversion of the Jacobian matrix is done using the pseudo-inverse method the performance of the manipulators is not degraded.

Figures 22 and 23 show a trace animation of the 6-R and 7-R manipulators attempting to execute a straight line motion in the global Z and Y direction simultaneously with the end of the last link located initially along the Z_1 axis. Note the 6-R manipulator is not able to execute a move in the Y direction when the end of the last link is located along the Z_1 axis do to the geometric degeneracy described in chapter III. The 7-R manipulator has no problem executing the desired move.

Figures 24 and 25 show a trace animation of the 6-R and 7-R manipulators attempting a reorientation of the end of the last link. The 6-R manipulator is not able to reorient the end of the last link in the depicted configuration because it is in a geometrically degenerate configuration. The 7-R manipulator is able to reorient the end of the last link since it has no orientational geometric degeneracies.

From the above analysis it can be concluded that there are two situations where a 7-R manipulator would be desired over a 6-R manipulator. The first situation, if the manipulator is required to reach the space along the Z_1 axis where the 6-R manipulator is geometrically degenerate and may not be able to execute a desired move. The second

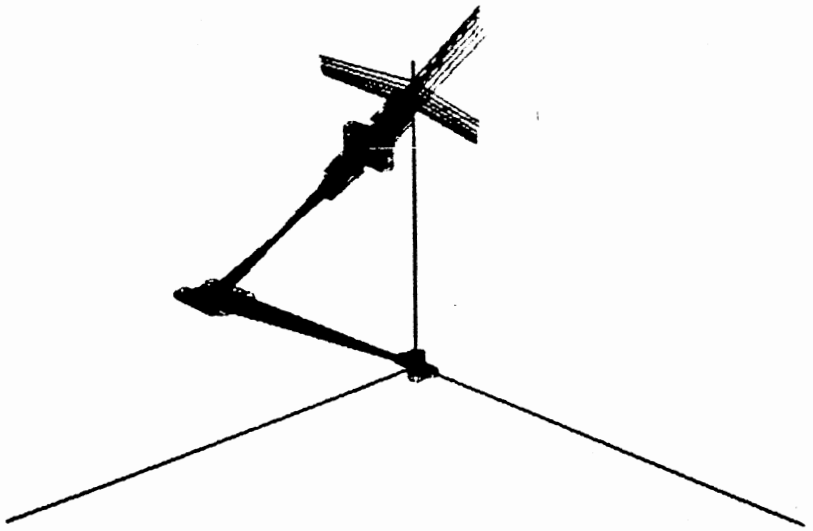

Figure 22. Trace Animation of 6-R Manipulator Attempting a Global ZY Move

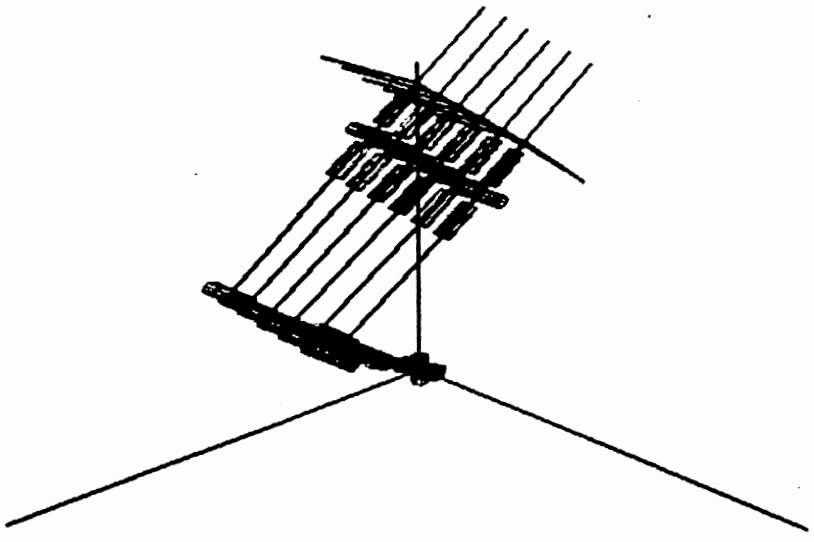

Figure 23. Trace Animation of 7-R Manipulator Making a Global ZY Move

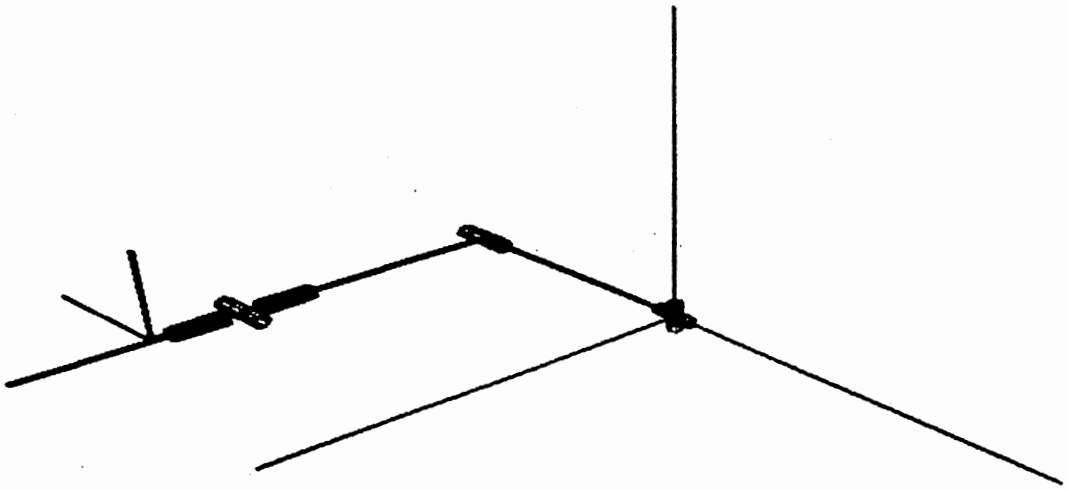

Figure 24. Trace Animation of 6-R Manipulator Attempting to Re-orient the End of the Last Link

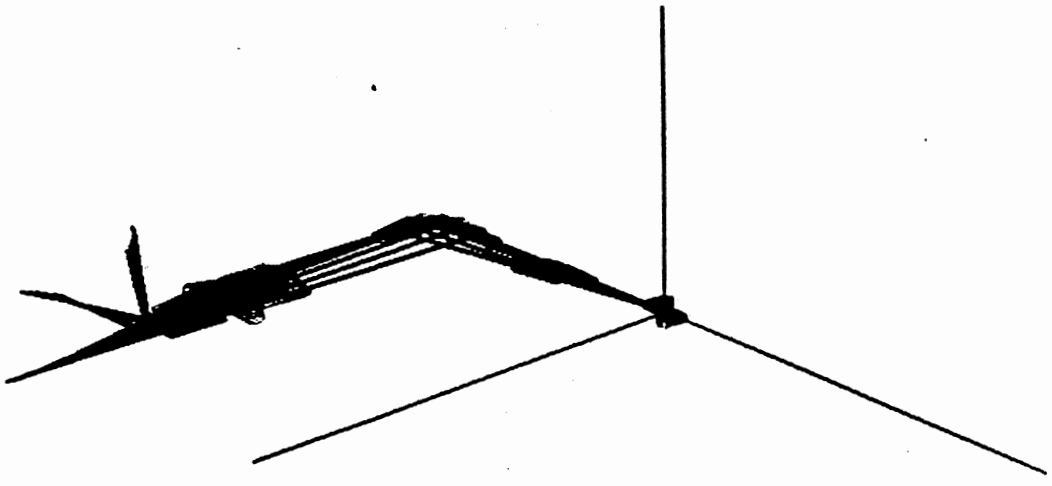

Figure 25. Trace Animation of 7-R Manipulator Re-orienting the End of the Last Link

situation, if the 6-R manipulator needs to obtain an orientational geometric degenerate configuration. The second situation, although not as apparent as the first situation, may not be a problem since the 6-R manipulator can reach any given position and orientation within its workspace by at least two configurations. However, each move the 6-R manipulator makes needs to be checked in advance to insure it does not pass through the orientational geometric degenerate configuration.

7-R Manipulator vs 8-R Manipulator

In this section the 7-R and 8-R manipulators will be compared in a workspace environment to see what advantage the 8-R manipulator has over the 7-R manipulator. Figures 26 and 27 show the 7-R and 8-R manipulators respectively. Figures 28 and 29 show the 7-R and 8-R manipulators moving in a straight line motion in the global Z and Y directions simultaneously with the end of the last link located initially above the Z_1 axis. Notice neither manipulator has trouble executing the desired move.

Figures 30 and 31 show a trace animation of the 7-R and 8-R manipulators executing a large straight line move in the global Z direction with the end of the last link located along the Z_1 axis. Observing figure 30 closely, notice the 7-R manipulator is approaching a configuration that would require two links to occupy the same space, while in figure 31 the links of the 8-R manipulator are in no danger of

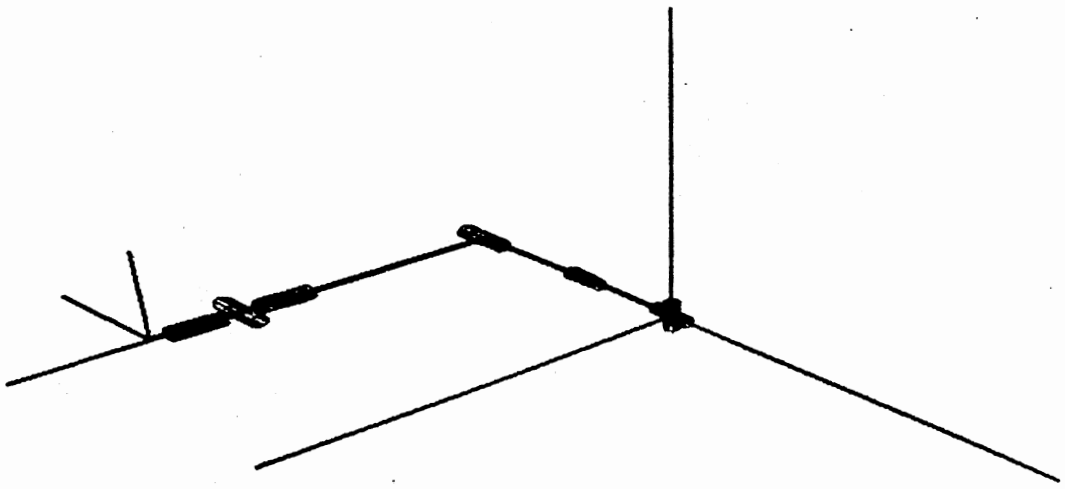

Figure 26. Computer Animation of 7-R Manipulator

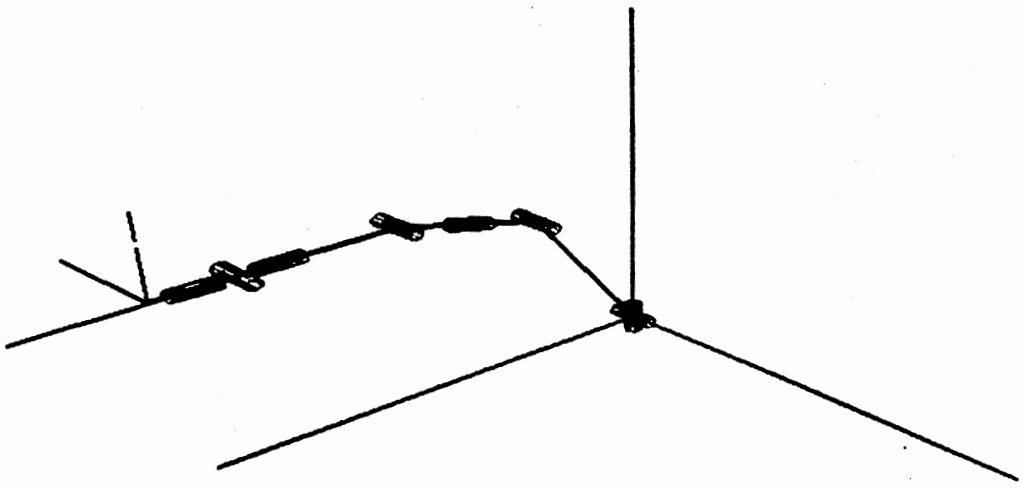

Figure 27. Computer Animation of 8-R Manipulator

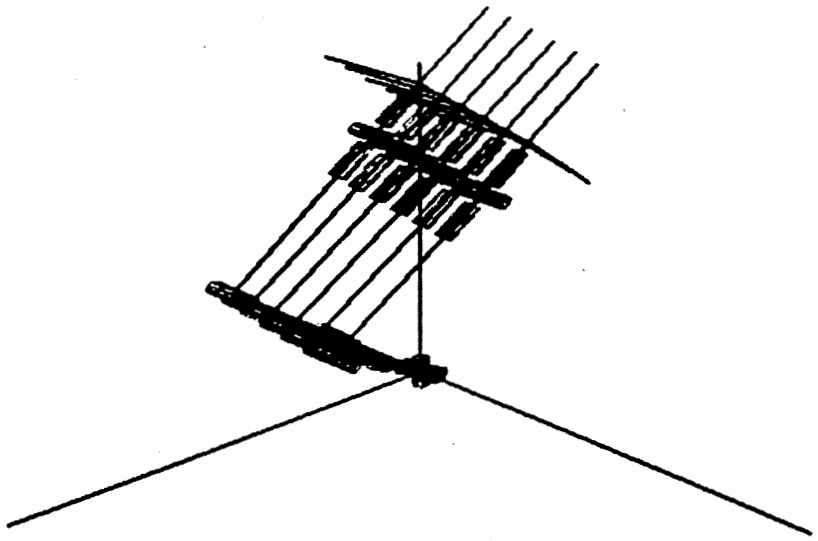

Figure 28. Trace Animation of 7-R Manipulator Making a Global ZY Move

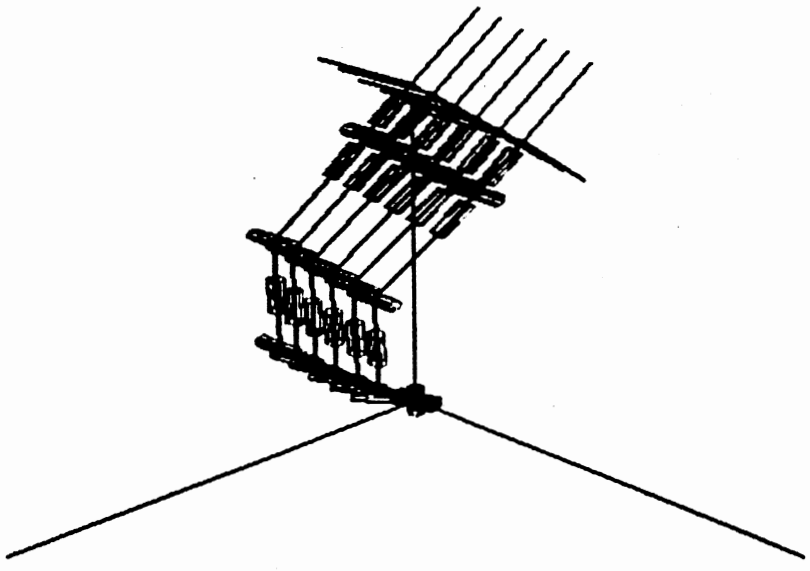

Figure 29. Trace Animation of 8-R Manipulator Making a Global ZY Move

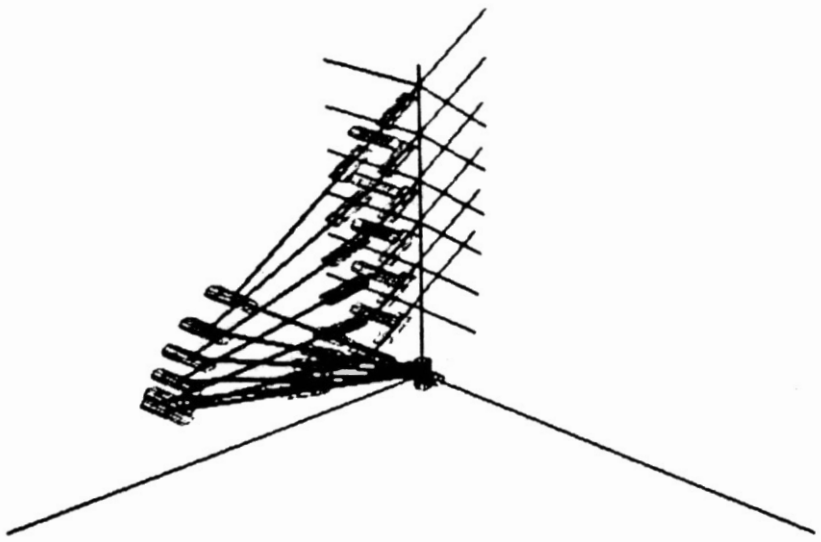

Figure 30. Trace Animation of 7-R Manipulator Making Large Move in Global Z Direction

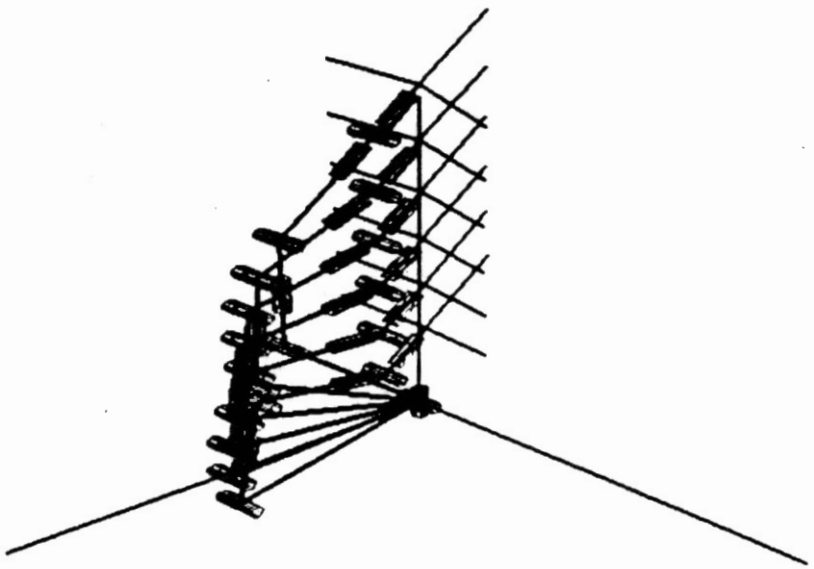

Figure 31. Trace Animation of 8-R Manipulator Making Large Move in Global Z Direction

requiring two link to occupy the same space. Realistically, a manipulator would have joint travel limit switches to prevent two links from trying to occupy the same space thus, avoiding any damage to the manipulator. The resulting non-ideal joints would cause the 7-R manipulator to have a smaller workspace.

Therefore, from the above analysis the 8-R manipulator may be preferred over the 7-R manipulator if the manipulators have non-ideal joints. Non-ideal joints in the 7-R manipulator will cause a significant reduction in the workspace while non-ideal joints in the 8-R manipulator will not affect its workspace [16].

CHAPTER V

SUMMARY

Preventing degeneracies from degrading the performance of manipulators has become an area of active research in the past few years. All previous work requires the use of kinematically redundant manipulators and appropriate mathematical theory to prevent degeneracies within the workspace from degrading the performance of a manipulator.

Using the pseudoinverse method for inverse kinematics has two major limitations. First, the solution of the pseudo-inverse does not yield an optimum solution unless the optimum is considered to be the minimum norm of the joint velocities. However, the pseudoinverse does give a non-iterative solution; hence, is better suited for on line programming. Second, the solution to the inverse kinematics using the pseudoinverse method requires a piecewise solution. But, if a continuous path solution is desired even a closed form solution will require the path to be defined in a piecewise fashion.

Past research has attempted to avoid all degenerate configurations regardless of the physical significance of the type of degeneracy. In this study a distinction between purely mathematical and geometrical degeneracies was

presented. Mathematical degeneracies, represented by a zero column vector in the Jacobian matrix, will not degrade the performance of the manipulator if an appropriate inversion method is used. Geometrical degeneracies, represented by a zero row vector in the Jacobian matrix, must be eliminated by proper design or avoided using the entire solution to the set of consistent linear equations defining the solution space for the inverse kinematics.

In addition, 6-R and proposed 7-R and 8-R manipulators were investigated for geometric degeneracies. The 6-R manipulator contained several infinities of geometric degeneracies, some of which could not be avoided. The 7-R manipulator contained a finite number of geometric degeneracies that could be avoided by using the entire solution to the consistent set of linear equations defining the inverse kinematics. The 8-R manipulator contained an infinity of degenerate configurations that could also be avoided using the entire solution of the inverse kinematics.

An algorithm was developed and computer program written to perform the inverse kinematics and animate any N-R manipulator. The computer program was then used to study the 6-R, 7-R and 8-R manipulators performance to determine each manipulators best suited application. A 6-R manipulator is sufficient if the space along the Z_1 axis need not be accessed and if each move the the manipulator makes can be studied prior to execution. The 7-R manipulator has no unavoidable degenerate configurations

within its workspace; therefore, moves need not be studied before hand; however, the 7-R manipulator will have void areas in its workspace if non-ideal joint constraints are imposed. The 8-R manipulator has no unavoidable degenerate configurations within its workspace and non-ideal joint constraints will not necessarily cause void areas in its workspace.

Although this study solves many of the problems concerning manipulator degeneracies and inverse kinematics of redundant manipulators there still remains a great deal of potential research. In the near term, the theory presented in this study can be extended to include prismatic joints. Also, the use of the entire solution space to the inverse kinematics for a redundant manipulator could be further investigated to include things such as near optimal solutions or obstacle avoidance.

In the long term, the computational speed of the general inverse kinematic algorithm could be improved through the use of parallel processing. Improved computational speed would allow the development of a standardized controller and programming language for a wide range of manipulators.

A very interesting study would be to extend the algorithm to inverse kinematics of a damaged manipulator in an inaccessible environment. If a manipulator is working in an inaccessible environment, such as a radioactive area, and incurred damage it would be desirable for the manipulator to

continue operation, if possible, without human intervention. For the damaged manipulator to function it would need the ability to determine the extent of its damage, modify the inverse kinematic and control algorithms, and have the ability to determine if the desired task can still be executed properly.

REFERENCES

[1] Paul, R. P., Robot Manipulators: Mathematics, Programming, and Control, MIT Press, Cambridge, Massachusetts, 1981.

[2] Paul, R. P., Shimano, B. and Mayer G. E., "Kimematic Control Equations for Simple Manipulators", IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-11, No. 6, June 1981, pp. 449-455.

[3] Takano, M., "A New Effective Solution for Inverse Kinematics Problem (Synthesis) of a Robot with Any Type of Configuration", Journal of the Faculty of Engineering, The University of Tokyo Vol. XXXVIII, No. 2, 1985, pp.107-135.

[4] Paul, R. P. and Stevenson C. N., "Kinematics of Robot Wrists", International Journal of Robotics Research, Vol. 2, No. 1, Spring 1983, pp. 31-38.

[5] Stanisic M. M. and Pennock G. R., "A Nondegenerate Kinematic Solution of a Seven-Jointed Robot Manipulator", The International Journal of Robotics Research, Vol. 4, No. 2, Summer 1985, pp. 10-20.

[6] Waldron, K. J. and Reidy J., "A Study of a Kinematically Redundant Manipulator Structure", IEEE International Conference on Robotics and Automation, 1986, CH 2282-2, pp. 1-8.

[7] Shih, L. Y., "Redundant Anthropomorphic Manipulator Subject to a Kinematic Constraint", IEEE International Conference on Robotics and Automation, 1983, CH 1962-0, pp. 676-679.

[8] Goldenberg, A. A., Benhabib, B. and Fenton, R. G., "A Complete Generalized Solution to the Inverse Kinematics of Robots", IEEE Journal of Robotics and Automation, Vol. RA-1, No. 1, March 1985, pp. 14-20.

[9] Chang, P. H., "A Closed-Form Solution fot the Control of Manipulators with Kinematic Redundancy", IEEE International Conference on Robotics and Automation, 1986, CH 2282-2, pp. 9-14.

[10] Graybill, F. A., Matrices with Applications in Statistics, Wadsworth Publishing Co., Belmont, California, 1969.

[11] Rao, C. R. and Mitra, S. K., Generalized Inverse of Matrices and its Applications, John Wiely & Sons Inc., New York, New York, 1971.

[12] Boullion, T. L. and Odell, P. L., Generalized Inverse Matrices, John Wiley & Sons Inc., New York, New York, 1971.

[13] Klein, C. A. and Huang, C., "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators", IEEE Transaction on Systems, Man and Cybernetics, Vol. SMC-13, No. 3, March/April 1983, pp. 245-250.

[14] Liegeois, 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, pp. 868-871.

[15] Tsai, Y. C. and Soni, A. H., "The Effect of Link Parameter on the Working Space of General 3R Robot Arms", Mechanism and Machine Theory, Vol. 19, No. 1, 1984, pp. 9-16.

[16] Laughlin, Gary Lynn, Position Robot With One Degree Of Redundancy, Oklahoma State University, Masters Thesis, Stillwater, OK, 1984.

APPENDIXES

APPENDIX A

NUMERIC EXAMPLE OF PSEUDOINVERSE

Given the matrix $\mathbf{A}$

$$\mathbf{A} = \begin{vmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix}$$

Find the pseudoinverse $\mathbf{A}^+$ using the method shown in chapter II.

<u>k = 1</u>

$$\mathbf{A}_1^+ = \left| \left| \begin{matrix} 1 & 1 & 0 \end{matrix} \right| \left| \begin{matrix} 1 \\ 1 \\ 0 \end{matrix} \right| \right|^{-1} \left| \begin{matrix} 1 & 1 & 0 \end{matrix} \right|$$

$$\mathbf{A}_1^+ = \left| \begin{matrix} (1/2) & (1/2) & (0) \end{matrix} \right|$$

<u>k = 2</u>

$$\mathbf{d}_2 = \left| \begin{matrix} (1/2) & (1/2) & (0) \end{matrix} \right| \left| \begin{matrix} 0 \\ 1 \\ 1 \end{matrix} \right| = \left| \begin{matrix} (1/2) \end{matrix} \right|$$

$$\mathbf{c}_2 = \left| \begin{matrix} 0 \\ 1 \\ 1 \end{matrix} \right| - \left| \begin{matrix} 1 \\ 1 \\ 0 \end{matrix} \right| \left| \begin{matrix} (1/2) \end{matrix} \right| = \left| \begin{matrix} (-1/2) \\ (1/2) \\ (1) \end{matrix} \right|$$

$$\mathbf{b}_2 = \left| \left| \begin{matrix} (-1/2) & (1/2) & (1) \end{matrix} \right| \left| \begin{matrix} (-1/2) \\ (1/2) \\ (1) \end{matrix} \right| \right|^{-1} \left| \begin{matrix} (-1/2) & (1/2) & (1) \end{matrix} \right|$$

$$\mathbf{A}_2^+ = \frac{1}{3} \left| \begin{matrix} (2) & (1) & (-1) \\ (-1) & (1) & (2) \end{matrix} \right|$$

k = 3

$$\mathbf{d}_3 = \begin{vmatrix} (1/3) \\ (1/3) \end{vmatrix}$$

$$\mathbf{c}_3 = \begin{vmatrix} (2/3) \\ (-2/3) \\ (2/3) \end{vmatrix}$$

$$\mathbf{b}_3 = \begin{vmatrix} (1/2) & (-1/2) & (1/2) \end{vmatrix}$$

$$\mathbf{A}_3^+ = \frac{1}{2} \begin{vmatrix} (1) & (1) & (-1) \\ (-1) & (1) & (1) \\ (1) & (-1) & (1) \end{vmatrix}$$

k = 4

$$\mathbf{d}_4 = \begin{vmatrix} (1/2) \\ (1/2) \\ (1/2) \end{vmatrix}$$

$$\mathbf{c}_4 = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix}$$

$$\mathbf{b}_4 = \left| 1 + (\mathbf{d}_4^T)(\mathbf{d}_4) \right|^{-1} (\mathbf{d}_4^T)(\mathbf{A}_3^+)$$

$$\mathbf{b}_4 = \begin{vmatrix} (1/7) & (1/7) & (1/7) \end{vmatrix}$$

$$\mathbf{A}_4^+ = \frac{1}{7} \begin{vmatrix} (3) & (3) & (-4) \\ (-4) & (3) & (4) \\ (3) & (-4) & (3) \\ (1) & (1) & (1) \end{vmatrix}$$

APPENDIX B

LISTING OF MOVEJTS MODULE

```
/* subprogram movejts calculates the                                    */
/* incremental move for each joint of                                   */
/* a general robot consisting of                                        */
/* revolute joints only                                                 */
/*                                                                      */
/*  the function call is:                                               */
/*    movejts(a,al,d,theta,jtlim,dx,alpha                               */
/*            ,coor,num,jts,free,typ,dth)                               */
/*      where:                                                          */
/*    a      - is array consisting of the linklenghts                   */
/*    al     -                          twist angles                    */
/*    d      -                          link offsets                    */
/*    theta  -                          joint angles                    */
/*    jtlim  - is array consisting of the maximum one                   */
/*             sided excursion for each joint in radians                */
/*               if there is no limit the entry                         */
/*               should be zero                                         */
/*    dx     - is array[7] consisting of the desired                    */
/*             incremental move, location                               */
/*                1 - X direction                                       */
/*                2 - Y                                                 */
/*                3 - Z                                                 */
/*                4 - rotation about X axis                             */
/*                5 -                 Y                                 */
/*                6 -                 Z                                 */
/*       note: if not all components of motion are used                 */
/*             null entries should NOT be used in dx                    */
/*    alpha  - is real and is the gain constant                         */
/*    coor   - int array  and is the corrdinate                         */
/*             system for the move                                      */
/*                  0 - not applicable                                  */
/*                  1 - tool coordinates                                */
/*                  2 - global                                          */
/*               coor[1] - position                                     */
/*               coor[2] - orientation                                  */
/*    num    - int, it is the total number of                          */
/*             transformations                                          */
/*    jts    - int, is the total number of joints                      */
/*    free   - char array, contains 'y' or 'n' and                     */
/*             corrosponds to the degrees of freedoms                  */
/*             for the robot  ie.                                      */
/*             if  free[1] = 'y' then the robot can                    */
```

```
  /*                          move in the X direction          */
  /*     typ     - char array, contains 'j' or 't'             */
  /*               to tell is the corrosponding                */
  /*               transformation with a,al,d,theta is a       */
  /*               joint variable or a coordinate              */
  /*               transformation                              */
  /*     dth     - is array containing the resulting           */
  /*               incremental joint moves in radians          */
  /*                                                           */
  /*    note: during compilation this module requires          */
  /*          linking to the math library and to modules       */
  /*          ginv and jacob                                   */
  /*    note: for a robot with more than 9 joints the          */
  /*          defined variable size must be increased in       */
  /*          all modules                                      */


#include <math.h>
#include <stdio.h>
#define size 10


tool_pos(dx,tr_dx,free)
  /* routine to transform dx into tool coordinates */
  /* for position structure                        */
 float dx[],tr_dx[];
 char  free[];
 {
  int j,ct = 1;
   for(j = 1; j <= 3; ++j)
    {
      if(free[j] == 'y')
       {
         tr_dx[ct] = dx[ct];
         ++ct;
       }
    }
 }

tool_orient(dx,tr_dx,free)
  /* routine to transform dx into tool coordinates */
  /* for orientation structure                     */
 float dx[],tr_dx[];
 char  free[];
 {
  int j,ct = 4;
  for(j = 1; j <= 3; ++j)
   {
     if(free[j] == 'n')
       --ct;
   }
  for(j = ct; j <= (ct+3); ++j)
     tr_dx[j] = dx[j];
 }
```

```
global_pos(dx,tr_dx,t)
  /* routine to transform dx into global coordinates */
  /* for position structure                          */
 float dx[],tr_dx[],t[][5];
 {
  int j,k;
   for(j = 1; j <= 3; ++j)
    {
      tr_dx[j] = 0.0;
      for(k = 1; k <= 3; ++k)
         tr_dx[j] = t[k][j]*dx[k] + tr_dx[j];
    }
 }

global_orient(dx,tr_dx,t,free)
  /* routine to transform dx into global coordinates */
  /* for position structure                          */
 float dx[],tr_dx[],t[][5];
 char  free[];
 {
  int j,k,ct,init,pos = 4;
   for(j = 1; j <= 3; ++j)
    {
      if(free[j] == 'n')
        --pos;
    }
   init = pos;
   for(j = 1; j <= 3; ++j)
    {
      ct = init;
      tr_dx[pos] = 0.0;
      for(k = 1; k <= 3; ++k)
       {
         tr_dx[pos] = t[k][j]*dx[ct] + tr_dx[pos];
         ++ct;
       }
      ++pos;
    }
 }

non_homo_solution(jinv,dx,dth,dof,jts)
      /* procedure to calcualte the non-homogenous */
      /* solution for the joint rates              */
 float jinv[][size],dx[],dth[];
 int   dof,jts;
  {
   int   k,l;
   float carry;
      for (k = 1; k <= jts; ++k)
       {
          dth[k] = 0.0;
          for (l = 1; l <= dof; ++l)
              dth[k] = jinv[k][l]*dx[l] + dth[k];
       }
```

```
  }

jinvXj(solution,jinv,j,dof,jts)
   /* procedure to multiply jinv times j */
 float solution[][size],jinv[][size],j[][size];
 int   dof,jts;
  {
   int k,l,m;
    for (k = 1; k <= jts; ++k)
     {
        for (l = 1; l <= jts; ++l)
         {
            solution[k][l] = 0.0;
            for (m = 1; m <= dof; ++m)
                solution[k][l] = jinv[k][m]*j[m][l]
                                   + solution[k][l];
         }
     }
  }

subtractI(mat,jts)
     /* procedure to perform mat=mat-I */
 float mat[][size];
 int   jts;
  {
   int k;
    for (k = 1; k <= jts; ++k)
        mat[k][k] = mat[k][k] - 1.0;
  }

getH(H,th,thlim,jts)
    /* procedure to create H matrix */
 float H[],th[],thlim[];
 int   jts;
  {
   int k;
    for (k = 1; k <= jts; ++k)
     {
        H[k] = 0.0;
        if ( thlim[k] > 0.0 )
           H[k] = 2*th[k]/(thlim[k]*thlim[k]);
     }
  }

multcol(mat,sol,h,jts)
     /* procedure to multiply sol = mat*h    */
     /*      where sol => #jts by 1          */
     /*            mat => #jts by #jts       */
     /*            h   => #jts by 1          */
 float mat[][size],sol[],h[];
 int   jts;
  {
   int k,l;
    for (k = 1; k <= jts; ++k)
```

```
      {
         sol[k] = 0.0;
         for (l = 1; l <= jts; ++l)
             sol[k] = mat[k][l]*h[l] + sol[k];
      }
  }

mult(mat,a,jts)
/* procedure to multiply a column matrix by a real number */
 float mat[],a;
 int   jts;
  {
   int k;
    for (k = 1; k <= jts; ++k)
         mat[k] = a*mat[k];
  }

homo_solution(j,jinv,sol,th,thlim,alpha,dof,jts)
/* procedure to determine the matrix for the homogenous  */
/*  solution part                                        */
 float j[][size],jinv[][size],sol[],th[],thlim[],alpha;
 int   dof,jts;
  {
   float temp[size][size],H[size];
    jinvXj(temp,jinv,j,dof,jts);
    subtractI(temp,jts);
    getH(H,th,thlim,jts);
    multcol(temp,sol,H,jts);
    mult(sol,alpha,jts);
  }

jointrates(j,jinv,dx,dth,th,thlim,alpha,dof,jts)
  /* procedure to determine joint rates from jacobian,   */
  /* inverse jacobian, and desired incremental motion    */
  /*                                                     */
  /* equation :  dtheta = jinv*dx + alpha*(jinv*j - I)*H */
 float j[][size],jinv[][size],dx[],dth[],th[],thlim[],alpha;
 int   dof,jts;
  {
   int   k;
   float tempvect[size];
    non_homo_solution(jinv,dx,dth,dof,jts);
    homo_solution(j,jinv,tempvect,th,thlim,alpha,dof,jts);
    for (k = 1; k <= jts; ++k)
         dth[k] = dth[k] + tempvect[k];
  }

prtjacob(j,dof,jts)
  /* routine to print jacobian matrix */
 float j[][size];
 int   dof,jts;
 {
  int k,l;
  FILE *data,*fopen();
```

```
  data = fopen("j","a");
   for(k = 1; k <= dof; ++k)
    {
      for(l = 1; l <= jts; ++l)
         fprintf(data,"  % .3e",j[k][l]);
      fprintf(data,"\n");
    }
   fprintf(data,"\n");
   fclose(data);
}

movejts(a,al,d,theta,thlim,dx,alpha,coor,num
        ,jts,free,typ,dth)
             /* main function */
 float a[],al[],d[],thlim[],theta[],dx[],alpha,dth[];
 int   coor[],num,jts;
 char  free[],typ[];
  {
   float j[size][size],jinv[size][size],t[5][5];
   float tr_dx[7];
   int   dof,l,k;
      /*                                    */
      /* determine degrees of freedom */
    dof = 0;
    for(l = 1; l <= 6; ++l)
     {
       if(free[l] == 'y')
         ++dof;
     }
      /*                                 */
      /* find jacobian and inverse */
    jacobian(j,t,a,al,d,theta,free,typ,num,jts);
    ginverse(j,jinv,dof,jts);
      /*                                                    */
      /*  find and convert move to proper coordinates */
    if(coor[1] == 1)
       tool_pos(dx,tr_dx,free);
    if(coor[1] == 2)
       global_pos(dx,tr_dx,t);
    if(coor[2] == 1)
       tool_orient(dx,tr_dx,free);
    if(coor[2] == 2)
       global_orient(dx,tr_dx,t,free);
      /*             */
      /* move joint */
    jointrates(j,jinv,tr_dx,dth,theta,thlim,alpha,dof,jts);
  }
```

APPENDIX C

LISTING OF GINV MODULE

```
    /*  This funtion will take the                          */
    /*     generalized inverse of a matrix                  */
    /*     specifically the Moore-Penrose                   */
    /*     pseudoinverse                                    */
    /*  The required function call is :                     */
    /*       ginverse(mat,matinv,rows,cols)                 */
    /*          where -                                     */
    /*     mat     - is the matrix to be inverted           */
    /*     matinv - is the inverse of mat                   */
    /*     rows     - is the number of rows in mat          */
    /*     cols     - is the number of columns in mat       */
    /*  note: the size of mat may not exceed                */
    /*        10 X 10 without changing the                  */
    /*        declaration of size                           */
    /*  note: during operation the program must             */
    /*        check for values of zero.  tolerence          */
    /*        defines zero.  If the martix you are          */
    /*        inverting has very large or small             */
    /*        entries tolerence may need to be              */
    /*        adjusted.                                     */
#define size 10
#define tolerence 0.01


ainit(mat,matinv,m)
    /* procedure to obtain the initial inverse */
 int     m;
 float mat[][size],matinv[][size];
  {
    float test,dum;
    int  j;
    test = 0.0;
      for (j = 1; j <= m; ++j)
         test = abs(mat[j][1]) + test;
      if (test <= tolerence)
       {
          for (j = 1; j <= m; ++j)
              matinv[1][j] = 0.0;
        }
      if (test > tolerence)
       {
          dum = 0.0;
```

```
            for (j = 1; j <= m; ++j)
                dum = mat[j][1]*mat[j][1]+dum;
            dum = 1.0/dum;
            for (j = 1; j <= m; ++j)
                matinv[1][j] = dum*mat[j][1];
        }
    }

getak(mat,ak,m,k)
      /* procedure to get the ak vector */
 float mat[][size],ak[];
 int   m,k;
  {
   int  j;
     for (j = 1; j <= m; ++j)
         ak[j] = mat[j][k];
  }

getdk(matinv,ak,dk,k,m)
      /* procedure to calculate the dk vector  */
 float matinv[][size],ak[],dk[];
 int   k,m;
  {
   int   j,l,q;
       j = k - 1;
       for (l = 1; l <= j; ++l)
        {
           dk[1] = 0.0;
           for (q = 1; q <= m; ++q)
               dk[1] = matinv[1][q]*ak[q] + dk[1];
        }
  }

getck(mat,ak,ck,dk,k,m)
      /* procedure to calculate ck vector */
 float mat[][size],ak[],ck[],dk[];
 int   k,m;
  {
   int j,l;
      for (j = 1; j <= m; ++j)
       {
          ck[j] = 0.0;
          for (l = 1; l <= (k-1); ++l)
              ck[j] = mat[j][1]*dk[1] + ck[j];
       }
      for (j = 1; j <= m; ++j)
          ck[j] = ak[j] - ck[j];
  }

getbk(matinv,bk,ck,dk,k,m)
      /* procedure to calculate the bk vector */
 float matinv[][size],bk[],ck[],dk[];
 int   k,m;
  {
```

```
    int    j,l;
    float dum;
         dum = 0.0;
         for (j = 1; j <= m; ++j)
              dum = ck[j]*ck[j] + dum;
                 /* if ck is NOT zero vector */
         if (dum > tolerence)
          {
            dum = 1.0/dum;
            for (j = 1; j <= m; ++j)
                bk[j] = ck[j]*dum;
          }
               /* if ck IS zero vector */
         if (dum <= tolerence)
          {
            dum = 0.0;
            for (j = 1; j <= (k-1); ++j)
                dum = dk[j]*dk[j] + dum;
            dum = 1.0/(1.0+dum);
            for (j = 1; j <= m; ++j)
             {
                bk[j] = 0.0;
                for (l = 1; l <= (k-1); ++l)
                    bk[j] = dk[l]*matinv[l][j] + bk[j];
             }
            for (j = 1; j <= m; ++j)
                bk[j] = bk[j] * dum;
          }
   }

assembleinv(matinv,bk,dk,k,m)
 /* procedure to assemble the generalized inverse matrix */
 float matinv[][size],bk[],dk[];
 int   k,m;
  {
   int   j,l;
   float temp[size][size];
        for (j = 1; j <= (k-1); ++j)
         {
            for (l = 1; l <= m; ++l)
                temp[j][l] = matinv[j][l] - dk[j]*bk[l];
         }
        for (j = 1; j <= m; ++j)
            temp[k][j] = bk[j];
        for (j = 1; j <= k; ++j)
         {
            for (l = 1; l <= m; ++l)
                matinv[j][l] = temp[j][l];
         }
   }

ginverse(mat,matinv,rows,cols)
   /* procedure to take the generalized inverse  mat is  */
   /* the original matrix having  # rows  and  # cols    */
```

```
float mat[][size],matinv[][size];
int   rows,cols;
 {
  float ak[size],bk[size],ck[size],dk[size];
  int   k,l;
    ainit(mat,matinv,rows);
    for (k = 2; k <= cols; ++k)
     {
        getak(mat,ak,rows,k);
        getdk(matinv,ak,dk,k,rows);
        getck(mat,ak,ck,dk,k,rows);
        getbk(matinv,bk,ck,dk,k,rows);
        assembleinv(matinv,bk,dk,k,rows);
     }
 }
```

APPENDIX D

LISTING OF JACOB MODULE

```
    /*  this module derives the jacobian                    */
    /*  matrix for a general n-R robot                      */
    /*  the function call is -                              */
    /* jacobian(j,t,a,al,d,th,free,typ,num,jts)             */
    /*   where:                                             */
    /*     j     - Jacobian matrix                          */
    /*     t     - the transformation matrix for            */
    /*             the robot                                */
    /*     free  - char array for the types                 */
    /*             of degrees of freedom to                 */
    /*             be derived for the                       */
    /*             jacobian. (y/n)                          */
    /*           [1] - move X direction                     */
    /*           [2] -      Y                               */
    /*           [3] -      Z                               */
    /*           [4] - rotate about X axis                  */
    /*           [5] -              Y                       */
    /*           [6] -              Z                       */
    /*     a     - float array containing link              */
    /*             lengths                                  */
    /*     al    - float array containing joint             */
    /*             alpha angles                             */
    /*     d     - float array containing link              */
    /*             offsets                                  */
    /*     th    - float array containing joint             */
    /*             angles                                   */
    /*     typ   - char array defining a data set           */
    /*             a,al,d and th as either -                */
    /*              j - joint variable                      */
    /*              t - transformation                      */
    /*     num   - int, the number of j and t               */
    /*             data sets                                */
    /*     jts   - int, number of joints                    */
    /* note: maximum number of joint variables              */
    /*       is defined by size in the following            */
    /*       #define statement                              */
#include <math.h>
#include <stdio.h>
#define size 10

init_trans(t)
   /* routine to make initial transformation matrix */
```

```
   /*    ie.  identity matrix                                    */
 float t[][5];
 {
  int j,k;
   for(j = 1; j <= 4; ++j)
    {
      for(k = 1; k <= 4; ++k)
         t[j][k] = 0.0;
    }
   for(j = 1; j <= 4; ++j)
      t[j][j] = 1.0;
 }

multi(t,t1,t2)
  /* routine to multiply transformation matricies */
  /*       T = T1 * T2                                          */
 float t[][5],t1[][5],t2[][5];
 {
  int j,k,l;
   for(j = 1; j <= 4; ++j)
    {
      for(k = 1; k <= 4; ++k)
       {
         t[j][k] = 0.0;
         for(l = 1; l <= 4; ++l)
            t[j][k] = t1[j][l]*t2[l][k] +t[j][k];
       }
     }
 }

swap(t1,t2)
  /* routine to put transformation matrix T2 into */
  /*  transformation matrix T1                                  */
 float t1[][5],t2[][5];
 {
  int j,k;
   for(j = 1; j <= 4; ++j)
    {
      for(k = 1; k <= 4; ++k)
         t1[j][k] = t2[j][k];
    }
  }

make_trans(t,a,al,d,th)
   /* routine to make a transformation matrix */
 float t[][5],a,al,d,th;
 {
  t[1][1] =  cos(th);
  t[2][1] =  sin(th);
  t[3][1] =  0.0;
  t[4][1] =  0.0;

  t[1][2] = -sin(th)*cos(al);
  t[2][2] =  cos(th)*cos(al);
```

```
   t[3][2] =  sin(al);
   t[4][2] =  0.0;

   t[1][3] =  sin(th)*sin(al);
   t[2][3] = -cos(th)*sin(al);
   t[3][3] =  cos(al);
   t[4][3] =  0.0;

   t[1][4] =  a*cos(th);
   t[2][4] =  a*sin(th);
   t[3][4] =  d;
   t[4][4] =  1.0;
  }

make_col(j,t,free,col)
  /* routine to make a column for the jacobian */
 float j[][size],t[][5];
 int   col;
 char  free[];
 {
  int k,row,column;
   row = 1;
   for(k = 1; k <= 3; ++k)
    {
      if(free[k] == 'y')
        {
          j[row][col] = -t[1][k]*t[2][4] + t[2][k]*t[1][4];
          row = row + 1;
        }
    }
   for(k = 4; k <= 6; ++k)
    {
      if(free[k] == 'y')
        {
          column = k - 3;
          j[row][col] = t[3][column];
          row = row + 1;
        }
    }
 }

prtmat(t)
 float t[][5];
 {
  int j,k;
  FILE *data,*fopen();
  data = fopen("tran","a");
   fprintf(data,"\n");
   for(j = 1; j <= 4; ++j)
    {
      for(k = 1; k <= 4; ++k)
         fprintf(data,"  % e",t[j][k]);
      fprintf(data,"\n");
    }
```

```
 }

jacobian(j,t,a,al,d,th,free,typ,num,jts)
 float j[][size],a[],al[],d[],th[],t[][5];
 int   num,jts;
 char  free[],typ[];
 {
  float temp[5][5],link[5][5];
  int   k,l;
   l = jts;
   init_trans(temp);
   for(k = num; k >= 1; --k)
    {
      make_trans(link,a[k],al[k],d[k],th[k]);
      multi(t,link,temp);
      if(typ[k] == 'j')
       {
          make_col(j,t,free,l);
          --l;
       }
      swap(temp,t);
    }
 }
```

VITA

James Frank Jones

Candidate of the Degree of

Master of Science

Thesis: INVERSE KINEMATICS AND DESIGN CONSIDERATIONS FOR KINEMATICALLY REDUNDANT ROBOTS

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Tulsa, Oklahoma, October 28, 1959, the son of Frank O. and Maryella Jones. Married to Mary Lee Dereske-Jones.

Education: Graduated form Tulsa Edison High School, Tulsa, Oklahoma, 1978. Attended Tulsa University, Tulsa, Oklahoma, 1977-1978; Tulsa Jr. College, Tulsa, Oklahoma, 1980; received Bachelor of Science Degree in Mechanical Engineering from Oklahoma State University, Stillwater, Oklahoma in December, 1984; completed requirements for Master of Science degree at Oklahoma State University in May, 1987.

Professional Experience: Vice-President, Frank Jones and Associates Inc., 1982-1986; Teaching Assistant, Department of Mechanical Engineering, Oklahoma State University, 1984-1986; Research Assistant, Robot Design Center, Oklahoma State University, 1986.