Control of a Robot Arm Using Vision Feedback and Tiled Space

Henry Polley

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

August 1987

ISBN 0-315-37089-0

ABSTRACT

Control of a Robot Arm Using
Vision Feedback and Tiled Space

Henry Polley

In this thesis we present a method of moving a robot arm suitable for adaptive control with the use of visual feedback. The method is based on dividing the workspace into discrete, 3-dimensional segments called tiles, each associated with a set of precomputed values representing the inverse Jacobian in that region of the workspace. The arm control algorithm uses these values in its computation of motor movements, thus reducing considerably the computation effort for each step in a multi-step trajectory. The algorithm is designed specifically for adaptive control in the context of a system that would supply position information from a vision subsystem. We investigate the behavior of a simulated arm controlled by this method and compare the overhead involved with that of conventional methods. We show that an implementation of this method on a physical system is feasible and discuss the design parameters involved in its implementation.

Acknowledgements

I would like to thank Dr. T. Fancott for his long hours in the preparation of this thesis. Without his good humor and patience this thesis would not have been possible. Also, I would like to thank Dr. P. Grogono for his comments and suggestions.

Table of Contents **Page**

List of Figures **page**

CHAPTER 1: INTRODUCTION.

General methods exist for the transformation of the position and orientation of the end-effector of a manipulator from Cartesian coordinates to joint space (joint angles) **[3,6,10,12,14,19,21,23,25, 31,32,33,35]**. We use the term manipulator in this work to signify a robot arm of five or six degrees of freedom. By end-effector, we mean that part of the arm beyond the wrist, including the work tool (for example, a gripper).

Most of the present methods are very time consuming to compute which is due largely to the number of matrix multiplications that are involved. Also, if velocity computations are performed, there is the need to compute the inversion of the Jacobian matrix **[9]**.

Classical methods are based upon the use of homogeneous transformation matrices **[3,10,19,21,23,24,31,32,33]** to make the transitions between joint and Euclidean space and vice versa. In the geometric method, the arm geometry is resolved into sets of equations, which are subsequently used to translate displacement in space into a movement. Since there are more degrees of freedom than dimensions in space, constraints must be introduced to arrive at specific solutions. These constraints are again dependent on the nature of the arm. This approach to the kinematics problem has been used by many researchers **[5,6,7,9,13,15,16]** but has the disadvantage that specific solutions must be found for each type of manipulator.

In the classical method, a coordinate frame is associated with each joint of the manipulator. There is a homogeneous transformation matrix associated with each joint relating it to the previous coordinate frame. To solve the forward kinematic problem, that is to find the position of the end effector of the manipulator given its position in joint space (transformation from joint space to Cartesian space), as well as its orientation, the transformation matrices for each frame are multiplied together. The assignment of the coordinate frames to the joints is specified according to rules presented in **[3,15,23]**. The inverse kinematics problem is much more difficult to solve by this method since the inverse transformations must be found to yield the joint values when given the end-effector position in Euclidean space and its orientation.

Since it is most often the inverse kinematic solution that is required, we come to a second method of solving the kinematics problem. This method is a geometrical method where the relationships between joint and Euclidean space can be expressed by using trigonometric equations. These equations can be used to compute the joint parameters given the position and orientation of the end-effector in Euclidean space. A second set of equations can also be formulated relating the joint space position of the arm to the Euclidean position and orientation of the end-effector. This method offers improved computational overhead but suffers from the problem that it is not a general solution. This means that the kinematic equations must be derived each time a manipulator with a different geometry is to be used.

Featherstone's method **[9]**, which is a modification of the geometrical method, is one of the most efficient implementations of this method to date. It is still a geometrical analysis method but one that is specifically designed to take advantage of the spherical wrist found in many manipulators to reduce the computational overhead. A spherical wrist is a joint with three intersecting axes of rotation [13]. In this method, the wrist position is first computed from the end-effector position and orientation. The problem is then simplified by separating it into two sub-problems. The spherical wrist allows the computation of the inverse kinematics to be partitioned so that the first three joint positions are solved independently from the joints in the wrist portion of the manipulator. After solving the first three joints, the joint positions of the wrist are determined using spherical trigonometry. This type of computation is not possible without a spherical wrist. The type of computation done by Featherstone becomes much less efficient if the links of the manipulator are not coplanar since the method becomes iterative under these circumstances. Also, it may not converge near deadpoints which may present a problem.

Featherstone's method as applied by Hollerbach and Sahar [13] to the Stanford arm required 64 multiplications, 38 additions and 10 transcendental function calls. In this work, Featherstone's method has been reworked using the same method of specifying the end-effector position and orientation as Paul [23] to provide a common basis for comparison of the computational overhead involved in the two methods.

Much of the literature deals extensively with the dynamics of the manipulators [3,9,12,13,14,15,19,20,21,27, 35]. These methods deal with the variations in performance caused by loading of the manipulator. They are also concerned with controlling the velocity and acceleration of the joints during movements of the arm. They deal with computing torque and force requirements for the motors controlling the joints to produce some desired trajectory with a given velocity or acceleration. Since the force requirements constantly change with the changing configuration of the joints, these methods require feedback from the arm so the values can be updated to compensate for changes. Therefore, these methods require the rapid re-computation of the dynamic equations so the number of computations must be minimized. The number of computations in these dynamic methods is as high as 66,271 multiplications and 51,548 additions using the Ucker/Kahn **[12]** method for a six degree of freedom arm. Other methods have been successful in reducing the computation such as the Newton-Euler **[12]** method which reduces the computational overhead to 852 multiplications and 738 additions and the Horn, Raibert **[12]** method which yields 468 multiplications and 264 additions.

In adaptive control (for example, grasping a moving object) there is the need for feedback on the position of the target. A suitable method of providing this feedback is vision. Several papers describe the use of vision for feedback of the end-effector position of the arm **[1,4,15,22,26,28,31,32,33]**. The vision function in these systems was also used to obtain the position of the target. However, the actual control of the arm was done using conventional methods with high computational overhead. This overhead slows the recomputation of the control parameters which then reduces the accuracy of the system because the error that will accumulate before recomputation of a corrected trajectory will be larger.

The method that we present in this thesis is based on the precomputation of differential values from conventional methods of kinematic solutions at discrete locations in the manipulator workspace. We study the behavior of a manipulator controlled using this method and plot the variance and error in the system. The system has no need of geometrical information about the system since it relies on partial precomputation of arm movements. This method also has the advantage that it is not dependent on the links of the manipulator being coplanar as in the case of Featherstone's method and there is also no assumption that the wrist is spherical. Another advantage of this method is that the only sensory information that is required will be specified by the cameras so there is no need for position sensors at the joints of the manipulator. The input from the camera system is also capable of supplying the needed information on the velocity of the arm which can be used in the control of it's trajectory. In this work, we have not studied the vision system itself. We assume that vision input capable of supplying the position information to the control algorithm is available. Also, we have not considered the dynamics involved in the control of a robot arm but have developed a system suitable for point to point control of robot arms with stepper motors. Since the stepper motors are an open loop system, we use visual feedback to give a closed loop system.

In the system that we present, the inverse kinematics have been eliminated from the control of the robot manipulator. We have managed to reduce the computational effort to 20 multiplications and 21 additions with the elimination of all transcendental functions for moving the arm based on one frame of visual information.

CHAPTER 2: THEORY

In order to move a robot manipulator from one position to another, we must be able to compute the individual motor movements required at each joint of the manipulator. To compute these movements, it is necessary to know the effect the motor produces per step (or other input unit for types other than stepper motors) on a joint. In addition we must know the amount of movement required by the joint to reach the desired position.

This is the basic problem in robot arm control; making the transformation of a movement specification to the joint space of the manipulator. This is necessary since movements are generally described in terms of Cartesian coordinates while arm motion is controlled by inputs to joint motors. This is known as the **INVERSE KINEMATICS** problem. The reverse of this process (transformation from joint space to Cartesian space) is the **FORWARD KINEMATICS** problem.

In summary, the inverse kinematics problem is: given the position of the end-effector of the manipulator in Cartesian space, find the corresponding joint positions. Thc forward kinematic problem is: given the joint position of the manipulator (from sources such as joint position encoders), find the position of the end effector in Cartesian space.

Most often it is the inverse kinematics of a manipulator that must be solved, since the position of a target or a trajectory is likely to be specified using Cartesian coordinates which must then be transformed into joint space. Normally, the position of the end-effector of the manipulator is specified by the point in space **(x,y,z)** and the orientation of the end-effector. The orientation of the end-effector is a vital piece of information, since with a four or five degree of freedom manipulator, there is an infinite number of possible positions in joint space corresponding to a particular point in Cartesian space. By specifying the orientation that the end-effector must assume at this position, we narrow down the possibilities of joint positions. For some joints, such as rotational joints, there may still be a choice between two positions (a positive and negative angle) as is shown in **FIG—1**. This is no problem since we can establish a rule to always take either all positive angles or all negative angles in solving the inverse kinematics.

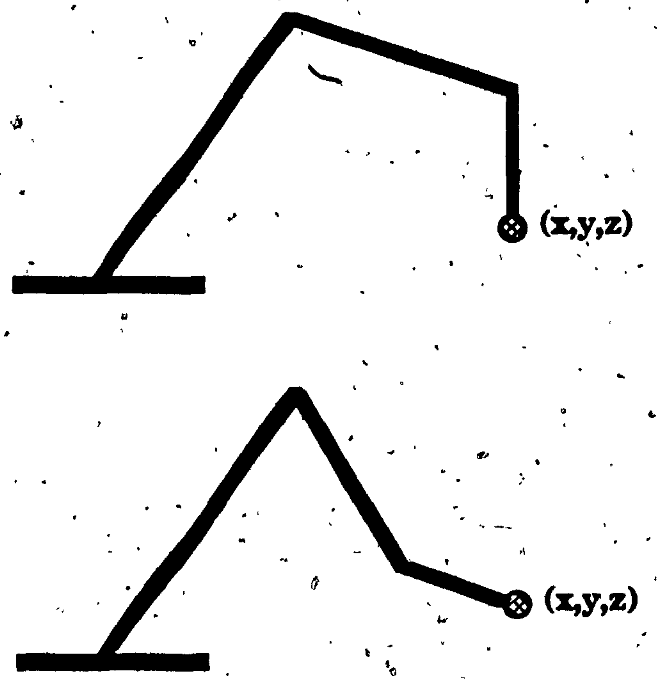


FIG-1

The orientation of the end effector and target can be specified using one of two methods. The first method involves specifying orientation by three angles, γ_x, γ_z and γ_r. γ_x is the rotation relative to the wrist of the end effector projected onto the XY-plane. If the wrist and end effector are projected onto the same point on the plane, the angle is measured relative to the origin, which is the base of the arm. γ_z is the angle that the end effector makes with the negative Z-axis of the coordinate system and γ_r is used to give the rotation of the end effector.

The second method replaces γ_x and γ_z with a unit vector that has its base at the end point of the arm. It points towards the wrist. The wrist position is computed by scaling this vector by the length of link L_3 which is the end effector of the arm. The vector system was chosen as it is easier for the vision system to measure a vector normal to the target rather than computing angles to specify the orientation of the target.

The kinematics of the manipulator may be solved by a number of methods. The conventional method, which is used in many industrial robots, is homogeneous transforms. This method is useful in that it provides a general solution to manipulator kinematics. It suffers however from a large amount of computational overhead.

Using this method, a coordinate frame is assigned to each of the links of the manipulator. A homogeneous transformation matrix is then used to describe each coordinate frame with respect to the coordinate frame of the previous link in the manipulator. The coordinate frame at the base of the arm then has a homogeneous transformation relating it to some reference coordinate system. A complete description of the methodology is given in Lee [15] or Paul [23].

To compute the position of the end-effector, the homogeneous transformations are then applied in series from the base of the manipulator to get the final result. The inverse kinematics solution is more difficult since the inverse transformations that describe the reference frame with respect to the current frame must be found.

Because of the high computational overhead of the homogeneous transformation method, a second method has been extensively used. This is the geometrical method **[5,6,7,9,13,15,16]**. In this method, the kinematics of the arm are solved using geometrical analysis. The disadvantage of this method is that it does not constitute a general solution since the solution is dependent on the manipulator geometry. Some researchers (**[9]**) have developed this method for a class of manipulators possessing a spherical wrist.

In Featherstone's method, the solution of the inverse kinematics is divided into two parts. The first part deals with the first three joints of the manipulator and the second part deals with the last three joints which serve to orient the end-effector. This partitioning of the manipulator geometry significantly reduces the amount of computation.

With this method, as applied by **Hollerbach & Sahar [13]** to the Stanford manipulator, the first step is to compute the position of the wrist by solving for the first three joint angles of the manipulator. In the second step the orientation of the hand relative to the forearm must be found. Lastly, the final three joint angles are computed on the basis of this orientation. Using this method, a computational complexity of 64 multiplications, 38 additions and 10 transcendental function calls was achieved in solving the inverse kinematics of the manipulator **[13]**.

In our investigation of the problem of manipulator control, we first examined the application of two variations of Featherstone's method: orientation of the end-effector specified using orientation angles and an orientation vector.

The manipulator that we have chosen is a modified Heathkit arm because our lab is already equipped with them. Also, because of its less than ideal geometry (due to the oblique shoulder joint), it serves to demonstrate that the computational overhead in our control algorithm is not dependent on the arrangement of the links or joints of the manipulator.

Our manipulator has 5 degrees of freedom. The motors in our arm are all stepper motor types. It is composed of 3 links, $(\mathbf{L_1}, \mathbf{L_2}, \mathbf{L_3})$, 5 joints, $(\mathbf{j_0}, \mathbf{j_1}, \ldots, \mathbf{j_4})$, and controlled by 5 motors, $(\mathbf{m_0}, \mathbf{m_1}, \ldots, \mathbf{m_4})$.

Link $\mathbf{L_1}$ (refer to **FIG-2**) is joined to the base of the manipulator by rotational joint $\mathbf{j_0}$. This joint allows $\mathbf{L_1}$ to rotate around the base in the xy-plane. (Note that the reference coordinate system has been placed so its origin coincides with this joint.) Then, link $\mathbf{L_2}$, which is a translational link (able to retract and extend), is connected to $\mathbf{L_1}$ by joint $\mathbf{j_1}$ at a fixed oblique angle. This is the shoulder of the manipulator. $\mathbf{j_1}$ allows $\mathbf{L_2}$ to rotate about an axis parallel to $\mathbf{L_1}$. Joint $\mathbf{j_2}$ is the translational joint of link $\mathbf{L_2}$.

The final link, $\mathbf{L_3}$ (the end-effector or gripper), is joined to $\mathbf{L_2}$ by joint $\mathbf{j_3}$. $\mathbf{L_3}$ is able to rotate about this joint in a vertical plane parallel to link $\mathbf{L_2}$. This joint forms the wrist of the manipulator. $\mathbf{L_3}$ is also able to rotate about an axis parallel to itself on joint $\mathbf{j_4}$ for the final degree of freedom. The wrist of our manipulator exhibits the same characteristics as the spherical wrist used by Featherstone except that it is lacking one degree of freedom.

Using orientation angles, we followed the same type of partitioning approach used by **Featherstone**. The analysis is as follows for solving the inverse kinematics of our manipulator. First, using three orientation angles and the point **(x,y,z)** to specify the position and orientation of the end–effector, we must find the position of the wrist. Referring to **FIG-2**, the length of **A** is:

$$A = |L_3 \sin(\gamma_z)|$$

similarly, the z-coordinate of the wrist (z_1) is:

$$z_1 = z + L_3 \cos(\gamma_z)$$

and the x and y coordinates of the wrist are found from:

$$x_1 = x - A \cos(\gamma_x)$$

$$y_1 = y - A \sin(\gamma_x)$$

Now that we have the wrist position ~~in~~ Cartesian space, we can solve for the first three joint values. We begin by solving for the coordinates (x_2,y_2) of the shoulder joint in Cartesian space. First, we find the slope (**m**) of the line L_{2xy} formed by projecting link l_2 onto the xy–plane:

$$m = \tan(\gamma_x)$$

and from the linear equation of a line (**y = mx + b**):

$$b_1 = y_1 - m^* x_1$$

which gives us a linear equation for L_{2xy}.

Then, since the slope of a line perpendicular to L_{2xy} is $-1/m$ and the line passes through the origin making **b = 0** for the equation of this perpendicular line, the intersection point (x_4,y_4) of the two lines can be found from:

$x4 = -(m*b1)/(1+m^2)$

Then, if the slope **m** is zero, we know that the **y** coordinate of the intersection point must be the same as the **y** coordinate of the end—effector of the arm. Otherwise, the **y** coordinate must be computed from:

$y4 = -x4/m$

since the equation of the perpendicular line is **y=(—1/m)x.**

Now we must find the distance **G** from the intersection point (x_4,y_4) on the line L_{2xy} to the shoulder (x_2,y_2). **G** is computed from Pythagoras theorem as:

$g = sqrt(L_1{}^2 - x_4{}^2 - y_4{}^2)$

Now the scaling factor **S** can be computed to scale the vector from (x_4,y_4) to (x,y) to a length of G. This scaled vector when added to (x_4,y_4) yields the shoulder point (x_2,y_2).

The scaling factor is found by taking the proportion of the length of the vector from the intersection point (x_4,y_4) to (x,y) that is represented by **g**.

$s = g/sqrt((x_1-x_4)^2 + (y_1-y_4)^2)$

Then, the shoulder position is:

$x_2 = x_4 + s*(x_1-x_4)$

$y_2 = y_4 + s*(y_1-y_4)$

Now, we can begin computing the actual joint values of the manipulator. Joint j_0 is a rotational joint so its value can be found using the two argument arctangent function and the shoulder position in Cartesian space.

$j_0 = atan2(y_2,x_2)$

Joint j_2, the length of link L_2 can be found as the distance between the shoulder and wrist points:

$$j_2 = \mathrm{sqrt}((x_1-x_2)^2 + (y_1-y_2)^2 + z_1^2)$$

Next we find the value of joint j_1, the shoulder rotation angle. The angle α_1 is the amount that the shoulder offset angle is greater than 90 degrees.

$$\alpha_1 = \pi/2 - \Theta_a$$

so this makes the length b:

$$b = L_2 * \cos(\alpha_1)$$

this allows us to use the arcsine function to obtain the value for joint j_1, the shoulder rotation as:

$$j_1 = \sin^{-1}(z_1/b)$$

Finally, we solve for joint j_3, the wrist rotation angle. We have part of this joint value in the orientation angle γ_z so we solve for the other part of the angle as α_2:

$$\alpha_2 = \cos^{-1}(z_1/L_2)$$

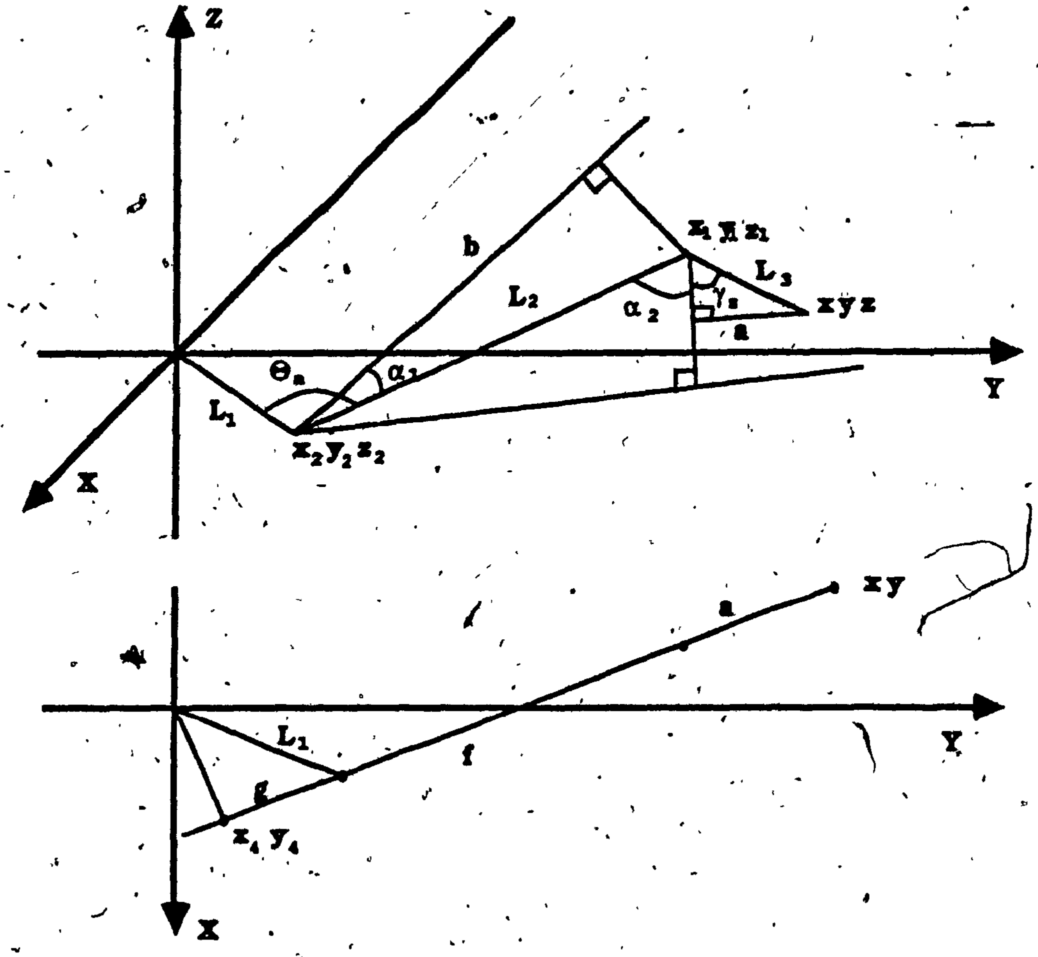
and the joint value is the sum of the two:

$$j_3 = \gamma_z + \alpha_2$$

The value of joint j_4 is given directly in the orientation specification as γ_r so:

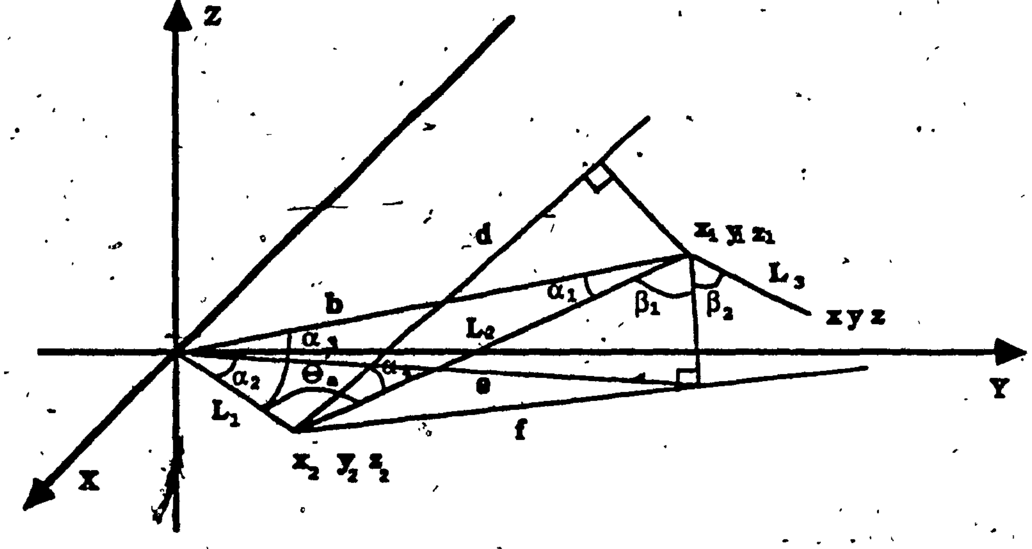
$$j_4 = \gamma_r$$

Our second variation, using an orientation vector, follows Featherstone's method much more closely than the first method. Using the unit vector method of orientation specification, we were able to eliminate all of the trigonometric functions in the computation of the wrist position. So, to find the wrist position, we scale the



Z
b
$x_1y_1z_1$
L_3
L_2
α_2
γ_2
xyz
a
Θ_a
α_1
Y
L_1
$x_2y_2z_2$
X
xy
a
L_1
Y
f
g
$x_4\ y_4$
X

FIG-2



Z
d
$x_1y_1z_1$
L_3
b
α_1
β_1
β_2
xyz
L_2
α_1
α_2
Θ_a
α_1
e
Y
L_1
f
X
$x_2\ y_2\ z_2$

FIG-3

orientation vector by the magnitude of link L_3 to obtain the position of the wrist, (x_1, y_1, z_1), in Cartesian coordinates (refer to **FIG-3**):

$$x_1 = x + L_3 * v_x$$

$$y_1 = y + L_3 * v_y$$

$$z_1 = z + L_3 * v_z$$

Then distance **b** is:

$$b = sqrt(x_1^2 + y_1^2 + z_1^2)$$

and the angle α_1 can be found from the law of sines as:

$$\alpha_1 = \sin^{-1}(L_1 * \sin(\Theta_a)/b)$$

and α_2 is:

$$\alpha_2 = \pi - \Theta_a - \alpha_1$$

now we can find the length of link L_2 (also the value of joint j_2) from the law of sines:

$$L_2 = b * \sin(\alpha_2)/\sin(\Theta_a)$$

Now the square of distance **f** (only the square of the distance is needed) can be found:

$$f = L_2^2 - z_1^2$$

then angle α_4 is:

$$\alpha_4 = \Theta_a - \pi/2$$

and the square of distance **e** is:

$$e = x_1^2 + y_1^2$$

and using the law of cosines we get angle α_3

$$\alpha_3 = \cos^{-1}((f - e - L_1^2)/(-2 * sqrt(e) * L_1))$$

Now, the value for joint j_0 can be found from:

$$j_0 = atan2(y_1, x_1) - \alpha_3)$$

If this value is greater than 360 degrees, subtract 360 degrees from it to get the proper value for joint j_0:

$$j_0 = j_0 - 2 * \pi$$

To get the value of joint j_1, the shoulder joint, we must first find the distance **d**:

$$d = L_2 * \cos(\alpha_4)$$

so the value of j_1 is found using the arcsine function:

$$j_1 = \sin^{-1}(z_1/d)$$

Now we must find the value of joint j_3, the wrist joint. First we find the angle β_1:

$$\beta_1 = \cos^{-1}(z_1/L_2)$$

and β_2 is:

$$\beta_2 = \cos^{-1}((z - z_1)/L_3)$$

Now, if the length of the vector to the wrist (x_1,y_1,z_1) is less than the length of the vector to the end-effector (x,y,z) then the value of joint j_3 is the sum of β_1 and β_2:

$$j_3 = \beta_1 + \beta_2$$

otherwise, link L_3 is positioned so that the angle formed with the vertical axis is negative and B_1 and B_2 must be subtracted to get the proper joint value for j_3:

$$j_3 = \beta_1 - \beta_2$$

As in the previous case, the value of joint j_4 is simply the value of the orientation angle γ_r.

In the case of the forward kinematic solution, both orientation methods use the same procedure to get the end-effector position but provide different orientation information.

The first step in the forward kinematics is to find the shoulder position. This is easily found from the length of link L_1 and the value of joint j_0 as:

$x_2 = L_1 * \cos(j_0)$

$y_2 = L_1 * \sin(j_0)$

and z_2 is known to be **0** since link L_1 lies in the xy-plane of Cartesian space. Then, to solve for the wrist position, (x_1, y_1, z_1), we first find the z-coordinate of the wrist. To do so, we need the angle α_1:

$\alpha_1 = \Theta_a - \pi/2$

and the length **b**:

$b = L_2 * \cos(\alpha_1)$

which gives z_1 as:

$z_1 = b * \sin(j_1)$

then we need to find the distance **d**:

$d = b * \cos(j_1)$

and **c** is:

$c = L_2 * \sin(\alpha_1)$

then we compute **g** from:

$g = \mathrm{sqrt}(d^2 + L_1^2)$

and find angles α_2 and α_3:

$\alpha_2 = \mathrm{atan2}(d, L_1)$

$\alpha_3 = j_0 + \alpha_2$

Then the intermediate point (x_0, y_0) is:

$x_0 = g * \cos(\alpha_3)$

$y_0 = g * \sin(\alpha_3)$

which gives the **x** and **y** coordinates of the wrist position as:

$x_1 = x_0 + c * \cos(j_0)$

$y_1 = y_0 + c * \sin(j_0)$

The final point that must be found is (x, y, z). to find this point we must compute the length of the vector from the shoulder to the end-effector when projected onto the xy–plane and compute a scaling factor to scale the vector from the shoulder to the wrist position in the xy–plane (which has already been solved) to give the x and y coordinates of the end–effector of the manipulator.

So, the angle α_4 is:

$$\alpha_4 = j_3 - \cos^{-1}(z_1/L_2)$$

and the z-coordinate of the end–effector is:

$$z = z_1 - L_3 * \cos(\alpha_4)$$

The length **h** of the vector from the wrist to the end–effector in the xy–plane is:

$$h = L_3 * \sin(\alpha_4)$$

and the length **i** of the shoulder to wrist vector in the xy–plane is:

$$i = \mathrm{sqrt}((x_1 - x_2)^2 + (y_1 - y_2)^2)$$

And the scaling factor **s** is the proportion of the total length to the length of i:

$$s = (h + i)/i$$

This gives the coordinates of the end–effector when the shoulder to wrist vector is scaled and added to the shoulder position:

$$x = x_2 + (x_1 - x_2) * s$$
$$y = y_2 + (y_1 - y_2) * s$$

The final item to be computed in the forward kinematics is the orientation specification. For the angular method, the γ_z value has already been computed as A_4. So:

$$\gamma_z = \alpha_4$$

The γ_x orientation angle is computed in two different ways. If link L_3 is positioned vertically, then the orientation angle is computed from the vector from the shoulder to the end–effector in the xy-plane using the 2 argument arctangent:

$\gamma_x = \mathrm{atan2}(y - y_2, x - x_2)$

Otherwise, the computation of γ_x is obtained from the direction of the vector from the wrist to the end–effector:

$\gamma_x = \mathrm{atan2}(y - y_1, x - x_1)$

In the case where a unit vector is used to specify the orientation of the end–effector, the vector from the end–effector to the wrist is scaled to a length of 1 to give the orientation vector **V**:

$V_x = (x_1 - x)/L_3)$

$V_y = (y_1 - y)/L_3)$

$V_z = (z_1 - z)/L_3)$

In both methods of orientation, the final orientation parameter, γ_r, is equivalent to the value of joint j_4.

The computational overhead for the forward and reverse kinematics using the two methods of orientation specification that have ben discussed are summarized in the following table:

METHOD	ADD	MULT	TRIG	INVERSE TRIG	SQUARE ROOT
Inv. kin.[1]	13	29	6	5	2
Inv. kin.[2]	16	37	2	6	2
Fwd. kin.[1]	15	21	16	3	2
Fwd. kin.[2]	17	27	12	2	2

The differences shown in the next table are the angular method minus the vector method.

DIFFERENCES(between two methods of orientation):

METHOD [2-1]	ADD	MULT	TRIG	INVERSE TRIG	SQUARE ROOT
Inv. kin.	3	8	-4	-1	0
Fwd. kin.	2	6	-4	-1	0

From this, we can see that there are fewer transcendental function calls in using the unit vector method of orientation specification in both the forward and reverse kinematics of our arm at the expense of several additions and multiplication.

CHAPTER 3: TILED METHOD.

In systems where the frequent recomputation of the manipulator kinematics is necessary, the overhead involved becomes a very important factor. This is true of systems that are using vision as a means of feedback for adaptive control and error correction as in the system that we study in this work.

In our system design, we have two cameras that provide visual information to the system on the position of the manipulator. The cameras are positioned so that one is viewing the xy-plane of the reference coordinate system and the other is viewing either the **xz** or **yz** plane. The cameras feed visual information to the vision processor which in turn supplies the control algorithm with the required positional information. The actual image processing involved in this portion of the system is assumed to be available. The image processing function is outside the scope of this work and is not examined here.

As can be seen from the above computations of the forward and inverse kinematics of our arm, there is a large overhead in the number of computations including many transcendental function calls. This represents a high computational burden in a system capable of adaptive movement. This is because in an adaptive system, the target may not be stationary but may move during the course of the manipulator movement to grasp it. In a non-adaptive system, this would cause the manipulator to be moved to where the target was supposed to be, missing the actual target. If we add visual feedback to the system and frequently recompute the required trajectory to the target, the movement of the target presents no problems in grasping the target (assuming that the manipulator is capable of moving faster than the target). In this type of system, it is therefore the computational overhead involved in the inverse kinematics will limit the rate at which the trajectory is to be recomputed from the visual information. This led us to develop a method that would reduce this overhead.

In our method, an object seen by the cameras will have position $\mathcal{P} = (p_1, p_2, \ldots, p_m)$ in pixel space. Since each camera provides the coordinate of a point in its image plane, m will usually be twice the number of cameras in the system. Since the manipulator has n joints driven by n motors, its position in joint space will be $\mathcal{R} = (r_1, r_2, \ldots, r_n)$ where r_i is the value of the ith joint of the manipulator.

In principle, we could derive a relation between the vectors $\mathcal{P}$ and $\mathcal{R}$ describing a point belonging to both pixel and joint space. This relation can be derived by the techniques discussed above, modified by the mapping of pixel space into Euclidean space. This method is not exclusive, as a form of self–calibration could also be used [4]. This is not developed in this thesis, but is discussed in **chapter 9**. We can assume therefore, if the gripper is at position $\mathcal{P}$ in pixel space, the corresponding joint configuration $\mathcal{R}$ is uniquely determined.

Two observations about the geometry of the system are of interest. The first is that we never actually use Euclidean space. The second is that both pixel space and arm space are discrete. This means that both can be represented as integers, resulting in potential savings of both computation and storage.

Assuming that the vision system can supply us with the position of the gripper and its orientation, the manipulator wrist position and the target position and orientation in pixel space, the problem is to move the gripper from $\mathcal{P}_h$ to $\mathcal{P}_t$ in pixel space. One way of accomplishing this is to make the transformation to joint space, producing vectors $\mathcal{R}_h$ and $\mathcal{R}_t$. The vector $\mathcal{R}_t - \mathcal{R}_h$ then defines the movement required by each joint. This, however, involves the computation of the coordinate transformations that we wish to avoid.

Suppose that $\mathcal{R}_{tw} - \mathcal{R}_w$ is small. Let $\Delta\mathcal{R} = \mathcal{R}_{tw} - \mathcal{R}_w$. The corresponding vector in pixel space is $\Delta\mathcal{P}$. We have a first order approximation, $\Delta\mathcal{R} = \mathcal{J}\bullet\Delta\mathcal{P}$ where $\mathcal{J}$ is the Jacobian matrix of joint space with respect to pixel space:

$$\mathcal{J}_{ij} = \frac{\delta r_j}{\delta p_i}$$

The value of $\mathcal{J}$ is a function of position. Thus $\Delta\mathcal{R}$ is a first order approximation to the movement required in joint space. Consequently, we can interpret $\Delta\mathcal{R}$ as an instruction to move the motors of the manipulator. The instructions will, of course, be accurate only if $|\Delta\mathcal{P}|$ is small enough to ensure that the non-linearity of the coordinate system and arm motion is unimportant. If $|\Delta\mathcal{P}|$ is large, this assumption does not hold. There is no reason to suppose that the arm movement $\Delta\mathcal{R}$ would move the gripper to the target because the transformation is non-linear. We hypothesize, however, that the direction indicated by $\Delta\mathcal{R}$ will be approximately correct. Thus $\Delta\mathcal{R}$ is used to move the arm through a small distance, after which $\mathcal{P}_w$ is measured again and the process repeated.

The same approach is repeated on the gripper of the manipulator to obtain the correct orientation.

The algorithm that we developed is based on the concept of dividing the workspace of the manipulator into non–overlapping, contiguous regions which we call tiles.

As in Featherstone's method, the system is partioned at the wrist of the manipulator which yields two types of tiling : Primary tiling for the joints positioning the wrist and secondary tiling for the orientation of the wrist. The size of the tiles is determined by a factor we define as the tile size. The tile size defines the length of the edge of a tile in the primary tiling and the height of the tile in the secondary tiling. To determine which tile a particular point in Cartesian space falls into, a mapping function is used to assign indices to the tile. For primary tiling, the mapping function will convert an **(x,y,z)** Cartesian coordinate into a tile index.

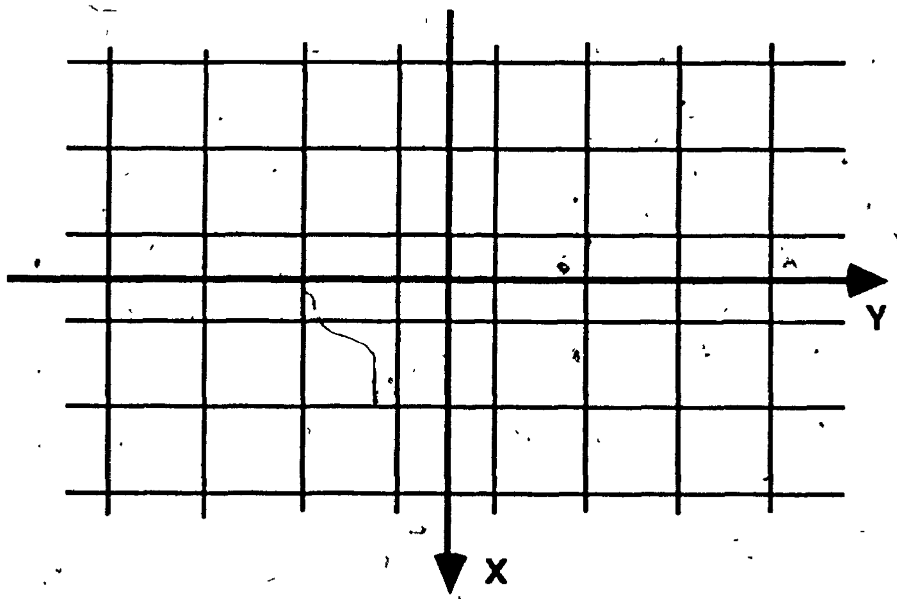


FIG-4

The positioning of the tiling with respect to the reference coordinate system is important. The method of dividing the space into tiles originally placed the tile **(0,0,0)** centered on the origin (base of the arm). Tiles that lay along the axis then had the axis passing through their central point as in **FIG-4**.

This method had a serious drawback for larger tile sizes as tile zero could cover some of the reachable space of the arm. The problem with this was that the central point of the tile where the coefficients for the tile are computed is also the center of the inner unreachable space of the arm. This meant that if the arm moved into this tile there would be no coefficients to compute the next portion of the move with. Also, since the tile covered all eight quadrants of the space, a coefficient could not be computed for it in the reachable portion of the tile since it would be different in each of the quadrants.

This problem is unlikely to occur in practice because the size of the tiles would be considerably smaller than the unreachable region. Nevertheless, we regard this as a problem that should be solved.

To overcome this problem, a new method of arranging the tiling was introduced. It involved moving the tiles so that they were no longer centered on the origin. The tiles along the axis now have their edges at the axis so they are distributed as in **FIG-5**.

An index for a tile contains an entry for each axis. This means that the **x** coordinate of a point is mapped to the **x** index for a tile with the same assignment for the other two axes. This mapping is done by the function **compute_tile** which takes a coordinate (any axis) and converts it to an index value. The tiles were originally indexed from zero. In this instance, the mapping function divides a coordinate on an axis by the tile size to produce the index to the tile on that axis. Since the indices to the tiles are integers, only the integer portion of the division is retained.

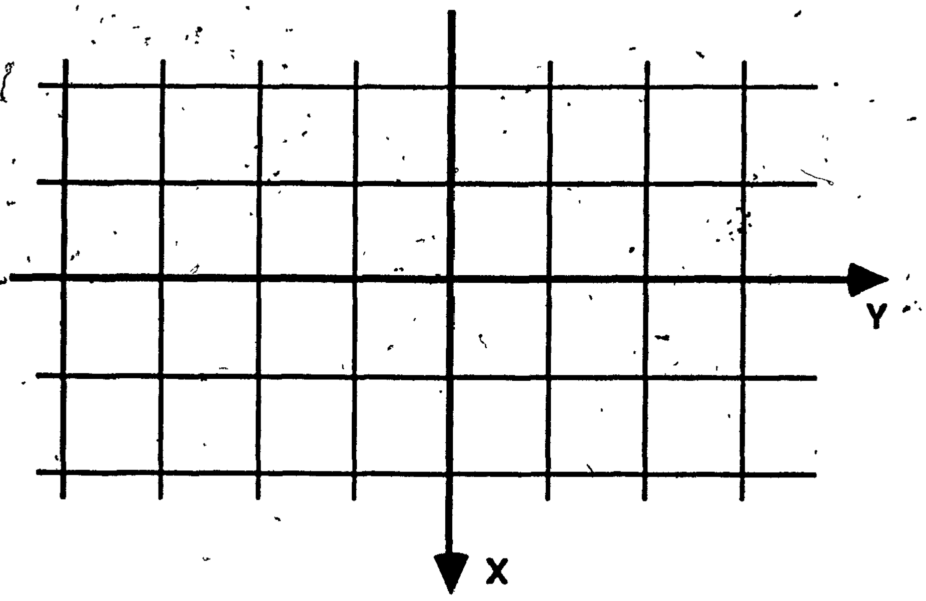


FIG-5

This method still had one undesirable feature in that it created eight tiles indexed by (0,0,0). To solve this problem, the indexing was changed to begin at 1 to avoid the confusion of multiple tiles for the same index. The new mapping function then became slightly more complicated in that there is a separate function for values less than zero. The function for values greater than or equal to zero increments the coordinate by the tile size before the division by tile size and for negative values it decrements. The purpose of this is to prevent any index from having an absolute value less than 1 which will give all tiles a unique set of indices.

The secondary tiling was introduced into the algorithm to reduce the computation required when the orientation of the end-effector is specified as a unit vector. The tiling for this motor is not relative to the base of the arm using the same tiles as the primary motors. Instead, this motor uses its own relocatable tiling system. If it had used the same tiles as the primary motors, then it would have been necessary to have a set of sub-tiles for each tile since the wrist motor effect is based on the current wrist joint position. This would lead to too many tiles and hence excessive storage requirements.

Therefore, a tiling system was developed that is base relative to the wrist position. In this way, the tiling system becomes relocatable. The tiles are actually arcs on the circumference of a circle, as in **FIG-6**, with a radius being a unit length vector like the one used for orientation specification.

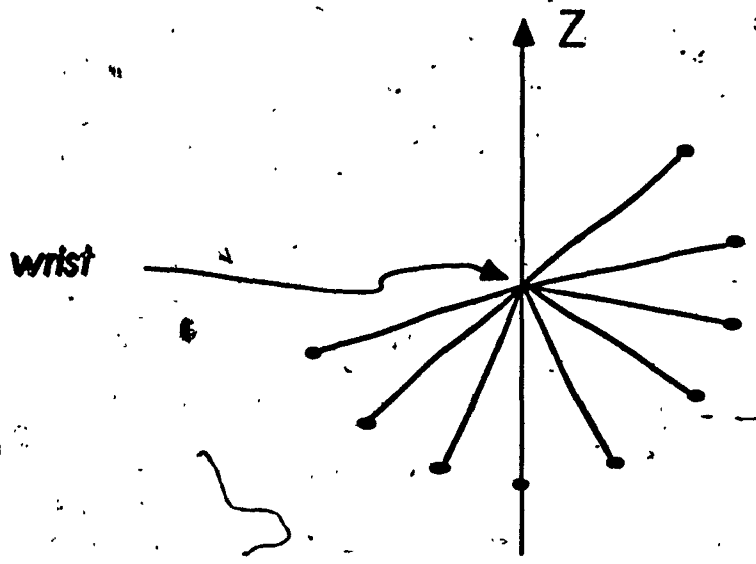


FIG-6

There are two indices for each of the tiles. The first index is the **z** component of the orientation vector for the end-effector. The **z** value however gives two possible tiles since it is symmetrical across the **z**-axis. A second index is then used to distinguish between these two tiles. It is derived from tiling the x-y plane into four large tiles as shown in **FIG-7**.

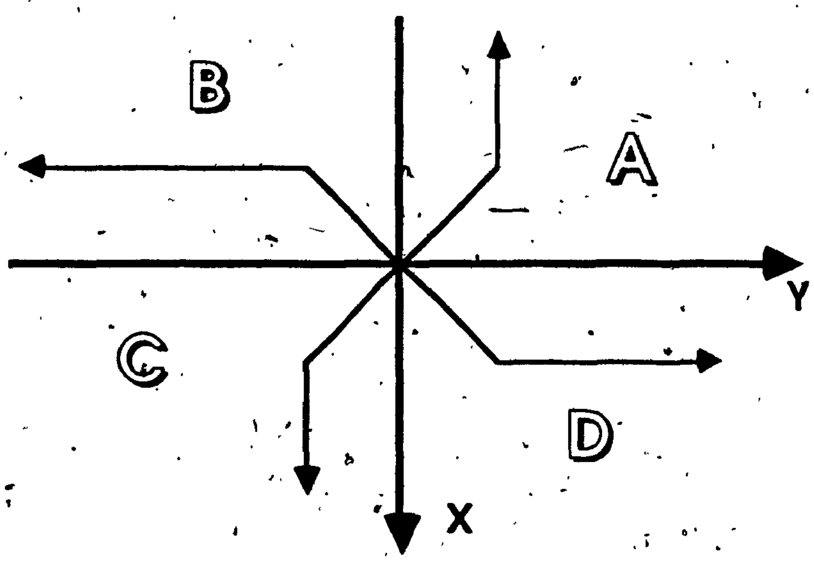


FIG-7

By looking at the signs of the **x** and **y** components of the orientation vector and the sign of the **x** and **y** coordinates of the wrist position, it is possible to determine which of the tiles that the wrist is in. Then the signs of the **x** and **y** components of the orientation vector can be used to determine which side of the circle that the current wrist motor tile belongs to. This is determined from the following table:

TILE	WRIST-x	WRIST-y	Vxy +	Vxy -
A		+	+x,-y	-x,+y
B	-		+x,+y	-x,-y
C		-	-x,+y	+x,-y
D	+		-x,-y	+x,+y

Then, the index to the tiling is simply the **z** component of the orientation vector for the end-effector and whether **Vxy** is a plus or a minus (which could be represented by a boolean value). The coefficients for each tile represent the change that is made to the z component of the orientation vector by the application of one pulse to the wrist motor. This is the only component that it is necessary to check since the circle that is made with link L_3 is always in a vertical position so the orientation of the link is dependent on the size of the **z** component of the orientation vector.

With each primary tile there is a corresponding set of coefficients (C_1, C_2, C_3) for motors (m_0, m_1, m_2) that represent the Jacobian of the manipulator in this region. The coefficients are obtained for the central point of the tile with each coefficient C_i being composed of (c_{ix}, c_{iy}, c_{iz}). It is assumed that the size of the tile will be small enough that the coefficients will be a good approximation of the actual values throughout the tile.

A coefficient represents the components of the vector from the center of the tile to the position resulting from the application of a single pulse to the corresponding motor. The primary tiling is used to position the wrist of the manipulator in the position required by the end–effector to reach the target with the correct orientation. We solve for scaling factors (p_1, p_2, p_3) corresponding to motors (m_0, m_1, m_2) such that the resulting vector **P•M** is equal to the vector from the current to the target position of the wrist. The vector **P** contains the scaling factors. It should be noted that the scaling factors are integer quantities representing the number of pulses to be sent to the motors.

The target position of the wrist is found by the same method as in the geometrical approach to the inverse kinematics by scaling the orientation vector by the magnitude of $\mathbf{L_3}$. The current wrist position is assumed to be supplied by the vision system. It is not expected that the pulse vector **P** applied to motors **M** will move the wrist to the target position but that the proportions of the pulses to each other will begin to move the wrist towards the target wrist position.

This brings us to the concept of frame size. The frame size **S** is defined to be the maximum number of pulses that can be sent to any motor during a frame. A frame is defined to be one iteration of the control algorithm which processes the information from one snapshot by the vision system.

Therefore, in order to ensure that the entire **P** vector can be delivered to the motors in a single frame, the maximum absolute value, $\mathbf{P_{max}}$, of any component of vector **P** is obtained. Then a scaling factor is computed that will reduce $\mathbf{P_{max}}$ to the frame size if it is larger than the frame size. If $\mathbf{P_{max}}$ is less than the frame size, then no scaling of the pulse vector **P** is done. In the first case, **P** is multiplied by the scaling factor to reduce the number of pulses to each motor. If the components of **P** are simply truncated to the frame size, the control algorithm may fail since the wrist may be moved in an entirely different direction from the target. Scaling **P** ensures that the proportion of pulses are the same as the original vector. This will cause the wrist to move on that part of the trajectory created by the original pulse vector that moves in the direction of the target. The smoothness of this movement depends on the frame size that is selected. A large frame size will allow the wrist to move farther along this path which may cause the resulting trajectory to the target wrist position to be ragged. In extreme cases, the wrist may never reach its intended destination due to oscillations induced by trying to move too far without recomputing the trajectory.

A second part to the algorithm deals with orienting the end-effector of the manipulator. Since our manipulator has only 5 degrees of freedom,the secondary tiling is only used to control motor $\mathbf{m_3}$ in our system. The indexing for the secondary tiling is different than the primary tiling as is explained above. The coefficient that is stored in the secondary tiling is for motor $\mathbf{m_3}$ which is the Δz produced by one pulse. The objective in the secondary tiling is to move the end-effector so that difference in the $\mathbf{z}$ components of the current and the target orientation vectors becomes zero.

The problem with this is that the motor may move in the wrong direction if the current and target orientations place link $\mathbf{L_3}$ on opposite sides of a vertical axis passing through the wrist. In this case, there is the need to insert a way-point into the trajectory within the secondary tiling. This way-point must be a point on the vertical position just out of reach of the end-effector so that it will not reach the way-point but pass it and continue on to the original target. The need for the way-point is explained by the fact that the $\mathbf{z}$ values are symmetric on both sides of the vertical axis. The algorithm always tries to reduce the $\mathbf{z}$ error (difference between the $\mathbf{z}$ components of the current and target orientation vectors) in the secondary tiling so it will not go through the part of the movement that will increase the $\mathbf{z}$

error in getting to the same side of the vertical axis, but will reduce the error to zero on the wrong side of the vertical axis without the way-point.

The number of pulses given to motor m_3 is truncated to the frame size if it is larger than the frame size since there is no other motor being used in this tiling system.

In summary, the manipulator is controlled by dividing the joints into two groups, primary and secondary. The primary joints, which position the wrist of the manipulator, are controlled by finding scaling factors for each of the coefficients in the current primary tile so that their sum is equal to the vector from the current to target wrist position. These factors are then scaled down to the frame size so the joints will begin to move in the direction of the target. Then, the scaling factors are recomputed using new visual information. Similarly, the secondary joints are controlled in a similar manner using the information in the secondary tiling to correctly orient the end-effector of the manipulator.

CHAPTER 4: THE ALGORITHM.

Instead of implementing our algorithm with an actual manipulator, we chose to simulate the system. The algorithm is similar to an actual implementation. This section describes the algorithm, the implementation problems encountered and their solutions.

In pseudo code format the algorithm is as follows:

```
STEP 1: Compute target wrist position.
STEP 2: Determine +/- index for target secondary tiling.
STEP 3: If current minus target position < PRECISION then STOP.
STEP 4: Compute primary move vector.
STEP 5: Compute primary tile index.
STEP 6: Compute primary pulses.
STEP 7: Compute secondary tile index.
STEP 8: Compute secondary pulses.
STEP 9: Scale primary pulses.
STEP 10: Send pulses to motors.
STEP 11: If target moving then GOTO 1
             Else GOTO 3.
```

4.1 EXPLANATION OF ALGORITHM:

STEP 1:

Determine the position of the wrist at the target. This gives the point where the wrist must be positioned in order to reach the target with the proper orientation. It is accomplished by scaling the orientation vector of the end-effector at the target position by the length of link $\overline{L_3}$ (the end-effector) to yield the target wrist position.

STEP 2:

The plus/minus index for the secondary tiling at the target wrist position must now be determined. It is determined as described in the section on tiling and is based upon the the wrist vector and the orientation vector at the target position.

STEP 3:

If the difference between the current position and the target position is less than **PRECISION**, then the target has been reached and the movement stops. **PRECISION** is a factor that defines the maximum error that can be tolerated on an axis and still be successful in reaching the target.

STEP 4:

The next step is to compute the primary move vector, which is the displacemen~~t vector fro~~m the current position of the wrist (known from the vision system) to the destination position of the wrist (computed in **step 1**).

STEP 5:

Now, the tile that the current wrist position is contained in must be determined. This is determined by computing the index for each axis of the tile. The indices will range from minus infinity to -1 and 1 to infinity. There is no tile zero due to the considerations described in **chapter 3** above. To compute an index, the coordinate is decremented by the tile size and then divided by the tile size if the coordinate is less than zero. Otherwise, if the coordinate is greater than zero, it is incremented by the tile size and then divided by the tile size. This yields the index for a coordinate. The three indices are then used to access the stored coefficients for that tile.

STEP 6:

We can now proceed to compute the total number of pulses (coefficient scaling factors) required for the primary joints (first three joints) of the arm that would produce the desired move vector at the wrist. This is accomplished by solving the following system of simultaneous equations which will determine scaling factors for the coefficients for each motor so that their sum will be the vector from the current to target wrist position:

$$\begin{aligned}\Delta x &= p_1 c_{1x} + p_2 c_{2x} + p_3 c_{3x}\\ \Delta y &= p_1 c_{1y} + p_2 c_{2y} + p_3 c_{3y}\\ \Delta z &= p_1 c_{1z} + p_2 c_{2z} + p_3 c_{3z}\end{aligned}$$

These can be solved by multiplying the inverted coefficient matrix by the displacement vector since the coefficient matrix has been inverted before it was stored. The solution then becomes:

$$\Delta V = C^{-1} P$$

where ΔV is the vector from the current to target wrist position.

STEP 7:

To compute the secondary tile index for the current position of the end-effector, we first consult the table for the plus/minus index to the tile. Then, the other index is computed from the z-component of the orientation vector. This is computed by adding half of the secondary tile size to the z value and dividing by the secondary tile size. Then, the table of secondary coefficients can be consulted to obtain the coefficients for the secondary motor.

STEP 8:

The pulses to be sent to the secondary motor are computed by first determining the difference in the z–components of the current and the destination orientation vectors. If the plus/minus index of the target orientation and the current orientation are not the same, then the way–point must be inserted as explained in the secondary tiling. The difference in the z-components is then relative to the way–point instead of the target. The number of pulses is then the **z** difference divided by the coefficient. If the **z** difference is less than zero and the plus/minus indices of the tiles are not the same, then the sign of the number of pulses must be switched to give the proper direction for the motor movement. Finally, if the absolute value of the number of pulses is greater than the frame size, then the value must be truncated to the frame size.

STEP 9:

The pulses obtained in **STEP 6** must now be scaled down to the frame size. This is accomplished by first finding the maximum absolute value of the number of pulses to be sent to the first three motors. Any motor that is at a limit and will not move away from the limit is not considered in this maximum. Then, the scaling factor is computed from this maximum divided by the frame size if the maximum is greater than the frame size. Otherwise, no scaling is done since the values would increase and make the move longer than it was supposed to be. All of the pulses for the first three motors are now divided by the scaling factor to get the actual number of pulses to be sent in this frame.

STEP 10:

The pulses are now sent to the motors.

STEP 11:

If the object is moving, then the target wrist position will have changed so it is necessary to include the first two steps into the loop. Otherwise, if the target is stationary, then we can proceed directly to **STEP 3** on subsequent iterations through the loop.

4.2 COMPUTATION OF PULSES: PRIMARY MOTORS

The number of pulses that is to be sent to each of the motors is computed as described in the description of the algorithm. In an earlier version, the pulses to be sent to the first three motors were determined by a different approach with results that were much poorer. In that system, the pulses were distributed by first computing the contribution that motor one could make to the move by determining the number of pulses that would be required to give the required x-displacement, then the y-displacement and finally the z displacement. The number of pulses that would be sent to motor $\mathbf{m_0}$ would be the amount which had the minimum absolute value of the three. Then, to go the rest of the required distance, the process was repeated for motor $\mathbf{m_1}$ and then again for motor $\mathbf{m_2}$. One of the major problems in using this method was that if a motor could not contribute to a specific direction of movement (such as z movement for motor $\mathbf{m_0}$, the base), then movement in that direction would have to be excluded in determining the number of pulses for that motor.

Besides the extra computational overhead involved in this method, there was also a considerable amount of extra logic involved to compute the number of pulses compared to the present method using matrix inversion to solve the system of equations.

The present method uses matrix inversion to solve for the number of pulses. The coefficient matrix is stored in its inverted form so all that is required to solve for the number of pulses required to produce a particular movement of the wrist is a matrix multiplication of the movement vector by the inverted coefficient matrix. Once the number of pulses has been obtained, it is necessary to scale them to the frame size if the maximum number of pulses is larger than the frame size. The scaling factor is computed so that the largest absolute value reduces to the frame size. In determining the largest absolute value in computing the scaling factor, motors that are against a limit are not included in determining the maximum unless they will move away from the limit in this frame. Without this feature, if a motor that could not move in the computed direction has a number of pulses that was much larger than the other motors, it would create a scaling factor that would reduce the remaining serviceable motors to zero and prevent the arm from moving. The arm would then deadlock in this position.

4.3 COMPUTATION OF PULSES: SECONDARY MOTORS

The pulses computed to motor m_3 were originally computed from the differences in th γ_z orientation angles at the current position and the target position. This method is not possible with the unit vector orientation method, since the angle difference is not known (this was a leftover from the inverse kinematics). Since the new method eliminated all of the inverse kinematics from the system, the wrist angle is not known. To compute the number of pulses sent to motor m_3, the tile was first determined. Then, if the plus/minus index for the current position was not equal to the index at the target, then a way-point was inserted into the local trajectory of motor m_3. This was necessary to make the motor rotate in the proper direction. Otherwise, the motor would rotate until it obtained the correct z component for its orientation vector which would be on the opposite side of the vertical axis from the target due to the symmetry in the system. The way point that was inserted in this case was a point that would be slightly below the position of link L_3 when it pointed straight down. It was slightly below this point so the arm could never reach it and stop. In trying to reach it, the arm will pass the vertical position and switch to the actual target for its destination. The number of pulses is determined by taking the difference in the z components of the orientation vectors at the target and the current positions and dividing by the coefficient for the current tile. If the number of pulses is larger than the frame size, then the number of pulses is truncated to the frame size. It is sufficient to truncate the pulses for this motor since there is only one motor involved in the secondary tiling. If a six degree of freedom arm was used, then it would be necessary to compute the scaling factor as in the primary tiling to ensure that the end effector moved along the desired trajectory. Also, if the current and target positions are on opposite sides of the vertical axis and the difference in the z components of the orientation vectors is negative, then the sign of the number of pulses is switched to create the proper direction.

CHAPTER 5: THE SIMULATION

By definition, a control program based on the segmentation of the mapping function of vision space into arm movement will exhibit nonlinear behavior. The full investigation of this behavior is greatly facilitated by simulation. Through simulation, not only typical behavior can be examined, but also extreme cases, both in trajectories and in the segmentation functions.

The results of this simulation not only assist with an understanding of the behavior of the system, but also highlight vision functions which must be examined analytically. In this chapter, we describe the simulation program and discuss the results.

5.1 THE SOFTWARE:

In the simulation program, the relevant robot arm parameters are stored as a linear array. These parameters are the lengths of the links of the manipulator and its joint positions. The measurement of the joint values is as follows: Joint J_0, the base rotation of the arm, is measured counter clockwise from the positive x-axis to link L_1. The shoulder rotation joint J_1, is measured from the x-y plane to a line that represents link L_2 if the angle of the shoulder, ***Theta-A***, was 90 degrees. The angle between link L_2 and link L_3 is the value of joint J_3, the wrist rotation. The rotation of the end effector, J_4, has not been included in the simulation. The program variables ***THETA-1***, ***THETA-2*** and ***THETA-3*** correspond to the values of joints J_0, J_1 and J_2 respectively. From the difference in the current and target gamma-r orientation angles, the pulses for this motor can be computed knowing the motor step size in degrees. The number of pulses to this motor is the angular movement required divided by the motor step size. The simulation neglects this rotational ability of the end effector since it is not involved in the movement algorithm.

There are three sets of arm description parameters in the system. The first, ***arm***, represents the current position of the arm. The second, ***start***, represents the position of the arm where all the motor register counts will be zero (all motors at their negative limits). The third, ***arm2***, represents the target position of the arm. This third set is required because the simulation asks for the joint parameters at the current position of the arm and at the target position. This is because with five degrees of freedom, it is difficult for the user to specify the correct orientation vector since the arm is not capable of reaching a point from any orientation. If the user specifies an incorrect orientation vector, the simulation would not work properly so the user input has been specified in joint space and the forward kinematic functions have been used to convert to Euclidean space.

Other parameters that must be specified as input to the simulation are the tile and frame sizes to be used. There is a separate tile size for primary and secondary tiling.

Point-1 is the current position of the end-effector of the arm and ***point-2*** is the position of the target. ***Wrist-1*** is the current wrist position and ***Wrist-2*** is the target wrist position. ***V1*** and ***V2*** are the orientation vectors for the current and target orientations respectively.

Delta-x, ***delta-y*** and ***delta-z*** are the **x**, **y** and **z** displacement respectively to reach the target wrist position from the current wrist position. ***Pulses*** is the number of pulses that are to be sent to each of the motors. ***MOTOR*** is an array of displacement values for each of the motors in joint space. ***Motor—register*** contains a register for each motor that includes a limiting number of pulses that can be sent to that motor. It contains a second register for the number of pulses that have been sent to a motor to reach its current position relative to the start position for that motor. Finally, there is ***TILE***, which is an array representing the differential coefficients for the motors in the current tile. There is only a set of coefficients for the current tile available at any one time in the simulation since they are computed when required in the simulation and are not stored.

The simulation begins by initializing the three sets of arm parameters, the ***MOTOR*** array, setting the motor register limits and the start position of the arm. Then, the frame size, tile size, and the current and destination positions (in joint space) must be entered. All of the angular values are accepted in degrees and then converted to radians by the simulation. From the joint space positions, the current and destination positions and orientations are computed in Euclidean space by the function called computepos. This function takes as arguments the current position in joint space and returns the current position in Euclidean space, the orientation vector, an array of Cartesian points representing the ends of each link which is used to drive a graphical display of the arm position. This function is an implementation of the forward kinematics of the arm discussed earlier in **chapter 2.**

Now we compute the at the target wrist position using the function ***compute_wrist***. The parameters to ***compute_wrist*** are the arm parameters (which provides the length of L_3), the Cartesian position of the end effector and the orientation vector. It returns the wrist position in Euclidean space by scaling the orientation vector by the length of link L_3 and adding it to the end-effector position.

Then, the motor registers must be initialized to reflect the position of the arm at the beginning of the move. The motor count is a counter that keeps track of the number of pulses that have been sent to that motor. This means that pulses that are to move the motor in the positive direction increment the counter and those in the other direction decrement the counter. The motor registers contain the upper limit of the number of pulses that a motor can receive with zero being the minimum. In this way software limit switches have been created since when the motor count is zero, the arm is at the limit of travel for that motor in one direction and when the count equals the motor register it is at the other extremity of movement for that motor. The current values of the motor counters can be determined by taking the difference between the current position and the zero position of each motor and dividing by the motor step value, which is the movement that one pulse to a motor creates (angular or linear). This gives the current value of the motor counter.

This is done by the function ***compute_registers*** with the arm parameters for the current position and the start position (*zero position*) of the arm as arguments. It initializes the motor register count for each motor using the global array ***MOTOR*** for the motor step sizes.

The last item to be done before entering the loop that is executed until the arm converges or stops moving, is to set the tile indexes for all of the axis to ***MAXINT***. This is to force a new tile when the first check for a tile change is made. This is necessary so that the coefficients for the starting tile will be evaluated since the coefficients are computed only when the wrist enters a new tile. This is to save computation in the simulation because when the wrist is still in the same tile as the previous frame, the coefficients do not have to be re-computed.

Now, we loop until the function called converging returns zero. Converging takes the current position of the end-effector and target (in Euclidean space and computes their difference as a vector. If all of the components of the vector are less than ***PRECISION***, which is the convergence tolerance on an axis, then the function returns zero. Otherwise, it returns non-zero.

Then, the current position of the wrist is evaluated by the function ***compute_wrist*** and the trajectory vector from ***WRIST1*** to ***WRIST2*** is computed by ***compute_vector***. The primary tile index is then computed for all of the axis by the function ***compute_tile***. If the wrist is no longer in the same tile, the coefficients fo the new tile are computed using the function ***compute_coeff***. The arguments to this function are the arm parameters for the current arm position, the trajectory vector of the wrist and the coefficients are returned in the third parameter.

The first step in ***compute_coeff*** is to make a local copy of the current arm parameters so they will not be altered. The reason they are required as parameters is so ***compute_coeff*** can access the constant parameters that are contained in them such as the lengths of the fixed links. Next, the center of the current tile is computed using the function ***tile_center***. ***Tile_center*** converts the axis index using the tile sizing into the coordinate of the center of the tile by reversing the ***compute_tile*** function. Using these coordinates for the center of the tile, the function ***compute_arm2*** computes the joint values for the manipulator up to the wrist based on the wrist being positioned at the center of the tile.

A copy of the arm parameters for the center of the tile is made so that the values at the center of the tile do not have to be recomputed for to obtain the coefficients for each primary motor. Then, the coefficients for each motor are determined by adding one pulse to the joint parameter for a motor. The new position of the wrist is then determined by the function ***compute_pos2*** which solves the forward kinematics up to the point of the wrist. This function is responsible for solving the forward kinematics of the arm up to the wrist. The vector from the center of the tile to the new position is then computed and its three components become the values of the coefficients for the motor. The procedure is repeated for the other two primary motors. The coefficients are then returned as the result of ***compute_coeff***.

Now, we must compute the number of pulses that are to be sent to the primary motors in this frame using the function ***compute_pulses***. The coefficients for the current tile and the wrist trajectory vector are the arguments and it returns the array pulses containing the number of pulses that would be required for the entire move.

In this simulation, the number of pulses has been solved for by using *Cramers Rule* instead of matrix inversion since there was no readily available matrix inversion routine on the system. All of the results in this routine are rounded off to the nearest integer.

The number of pulses required in the secondary tiling (the wrist joint, j_3) during this frame are computed by the function alter_theta3. The current and target orientation vectors, the current position of the end-effector and wrist position, the current arm parameters, and the plus/minus index for the target secondary tile are required as parameters. It places the number of pulses to be sent to the wrist motor in the proper position of the ***PULSES*** array.

First the tile index is computed by calling the function ***tile_theta3*** which will return the plus/minus index from the look-up of the table described in the previous section on secondary tiling. The other index component is computed using the function ***tile_3***, which implements the secondary tiling function. The coefficient is determined by computing the angle that link L_3 would make to reach the center of the secondary tile and then incrementing it by the motor step value. The new orientation vector is then computed from the cosine of the angle (which is equivalent to the z component of the new orientation vector) and the coefficient becomes the difference in the z index to the tile and the z component of the new orientation. The function also implements the insertion of the way point as discussed earlier in **section 4.3**. The number of pulses is then the difference in the z components of the orientation vectors divided by the coefficient. If the absolute value of the coefficient is larger than the frame size, it is truncated to the frame size. Also, if the plus/minus indices are not the same and the *z* difference is less than zero, the sign of the number of pulses is corrected.

The pulses for the primary motors must now be scaled so that the largest value is equal to the frame size. No scaling is done if the values are all less than the frame size. The function that is responsible for this operation is called ***compute_pulses_to_send*** which takes the number of pulses computed in ***compute_pulses*** as arguments and places the results in the array ***send_pulses*** with those for motor $\mathbf{m_3}$. The scaling of the pulses is as in the section on computing the pulses for the primary motors (**section 4.2**).

The pulse contained in the array ***send_pulses*** can now be sent to the motors. This is the function of ***send_pulse*** which takes the array of pulses to send and the arm parameters at the current position and updates the arm parameters to their resulting values after the pulses are sent. It is also responsible for updating the motor registers and enforcing the motor limits. If the number of pulses for a motor would exceed a limit, the number of pulses is adjusted so the motor will stop at the limit. Also, a flag is *or*-ed with the number of pulses that are being sent to a motor. This is an easy way to keep track of whether any pulses have been sent because the flag is initialized to zero and will still be zero if no pulses are sent. The flag is the return value for this function.

If ***send_pulse*** returns zero, the current position is displayed as well as the target position and then the program exits with the error code set to one because it was not able to reach the target within the limits set by ***PRECISION***. Otherwise, the new position is computed by computepos from the new joint values.

Then, the data is output for the graphical display and the next frame begins at the start of the while converging loop.

5.2 RECOMPUTATION:

When a motor is against a limit and the pulses to be sent to it will not move it away from the limit, it is not included in the computation of the scaling factor. It is very important to include this motor into the scaling factor when the motor will move away from the limit. This is due to the case where it is the largest value of the set of motions. If not included in determining the scaling factor, the resulting number of pulses for that motor would be larger than the frame size after scaling . When the motor is not going to receive pulses because they will keep it against the limit, it can be excluded from the scaling. This is due to the fact that it may cause the other motors to be scaled to zero and create an unnecessary deadlock situation. Also, there is the possibility in this case to recompute the pulses that are to be sent to the other motors that are not against a limit to achieve a trajectory that will be more linear, since the removal of a motor causes the resulting trajectory to change. This gives rise to the problem that the system of equations that must be solved reduces to three equations and two unknowns so there may not be a unique solution. We have tried this method and our experience has shown that it is not worth the extra computational effort, since in most cases there is no unique solution and the system must revert back to using the original values that were computed.

In conclusion, the simulation of the control algorithm has helped us learn about the parameters that influence the behavior of the system. The most important of these parameters are the tile and frame size. We have seen from the various types of plots that we have produced how these parameters vary throughout the work space of the manipulator.

5.3 ARM BEHAVIOR:

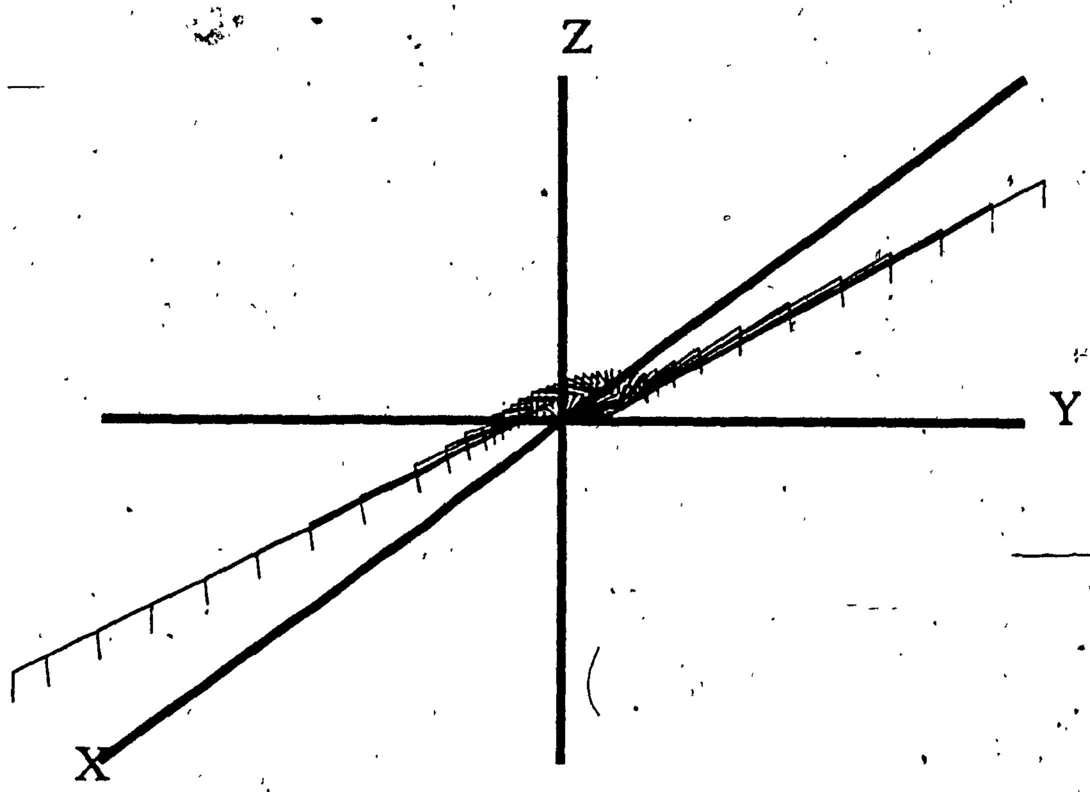
The arm will attempt to move from the current position of the wrist to the destination position of the wrist along a straight path. The linearity of the path is largely dependent on the frame size and the tile size. A large tile size will force the use of one approximation (coefficients are approximations that are assumed valid over the volume of the tile) over a larger area which means that the error will be greater. Similarly, a large frame size causes extrapolation of the coefficients over a large distance which would be equivalent to a large tile size. With small tile and frame sizes, the arm is well behaved, following a linear path to its destination except when the physical constraints of the arm force it to deviate. This is usually due to a motor reaching a limit of travel in a particular direction. In this case, the arm attempts to move towards the destination without the services of that motor which will cause the path to deviate from its linearity. As soon as it is able to move the motor that was against the limit,it does so but no attempt is made to regain the original path. Instead, the arm takes up a linear path from the current position of the wrist to the destination position of the wrist.

In the **FIG-8**, the arm hits the limit switch and then deviates to go around the space that the arm cannot reach.

This is a move on the x-y plane. It is a rotation of the arm about the base to a point on the other side of the origin. This move also illustrates the fact that the algorithm does not attempt to minimize the motor movements since this move could have been accomplished by rotating about the base with motor $\mathbf{m_0}$. Instead, the arm uses both motor $\mathbf{m_0}$ and motor $\mathbf{m_2}$ to retract the extender as well as do the rotation.

The algorithm is basically a feedback control mechanism for the arm and possesses no intelligence as is demonstrated in **FIG-9**.

In this case, the arm retracts until it hits a limit switch on the extender. It should now start rotating clockwise on motor $\mathbf{m_0}$ to reach its destination but since the linear path is shorter and lies in a counter-clockwise direction from the current position, it chooses this direction and encounters the limit switch. Then motor $\mathbf{m_1}$ starts moving upwards to get the wrist closer to the destination but it also encounters a limit. At this point it simply stops since it doesn't know how to reach the destination. This illustrates that the algorithm contains no intelligence. This illustrates that some higher level planner with intelligence such as a trajectory or task planner would be required in an operational system. This could direct it to some intermediate point, allowing it to complete the move if possible.



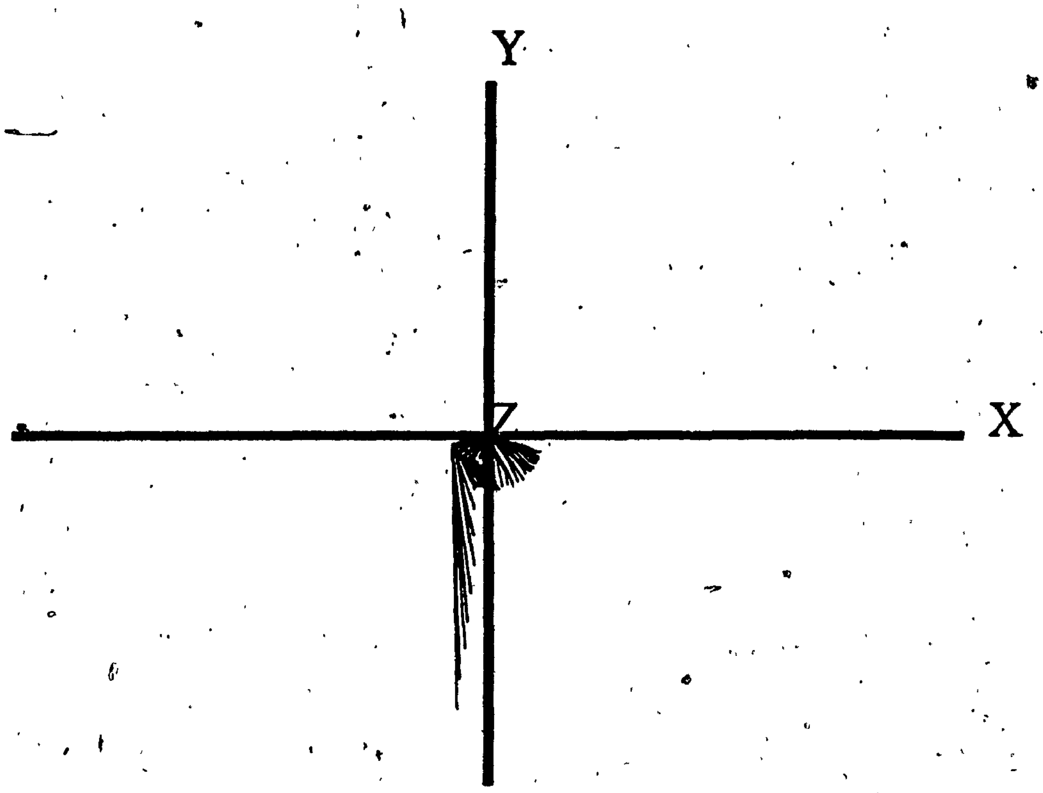
Plot of ARM POSITION. ---> Camera view is 3-dimensional.

FRAME SIZE = 20 TILE SIZE: PRIMARY = 1.0 SECONDARY = 0.0075

START JOINT POSITIONS; J1=90.00 J2= 0.00 J3=50.00 J4=90.00

END JOINT POSITIONS: J1=265.00 J2= 0.00 J3=50.00 J4=90.00

FIG-8



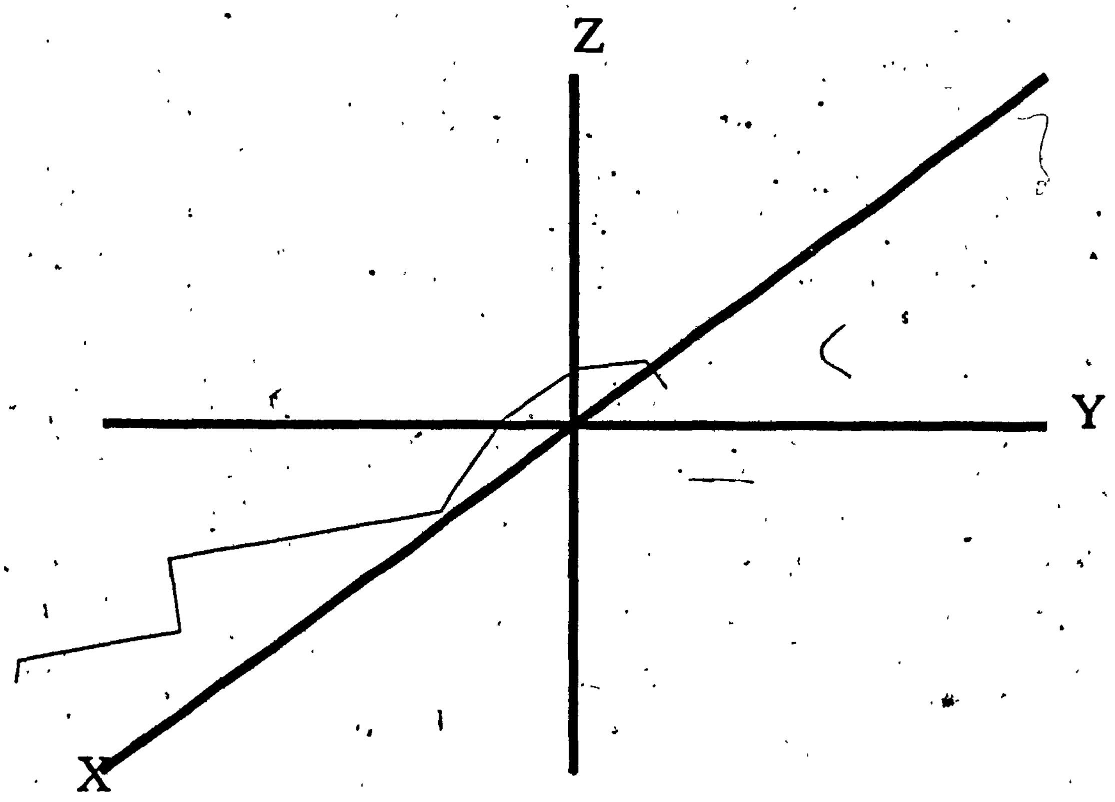
Plot of ARM POSITION. ---> Camera view is of XY plane.

FRAME SIZE = 20 TILE SIZE: PRIMARY = 5.0 SECONDARY = 0.0075

START JOINT POSITIONS: J1=200.00 J2= 0.00 J3=50.00 J4=90.00

END JOINT POSITIONS: J1=20.00 J2= 0.00 J3=50.00 J4=90.00

FIG-9



Plot of WRIST TRAJECTORY. ---> Camera view is 3-dimensional.
FRAME SIZE = 300 TILE SIZE: PRIMARY = 5.0 SECONDARY = 0.0100
START JOINT POSITIONS: J1=90.00 J2= 0.00 J3=50.00 J4=90.00
END JOINT POSITIONS: J1=265.00 J2= 0.00 J3=50.00 J4=90.00
FIG-10

It should be noted that this is an extreme condition that only occurs when a shortest line trajectory passes through unreachable regions in a way that brings the arm against a limit of movement, ie. the arm attempts to rotate in the wrong direction.

5.4 PARTIAL REACHABILITY.

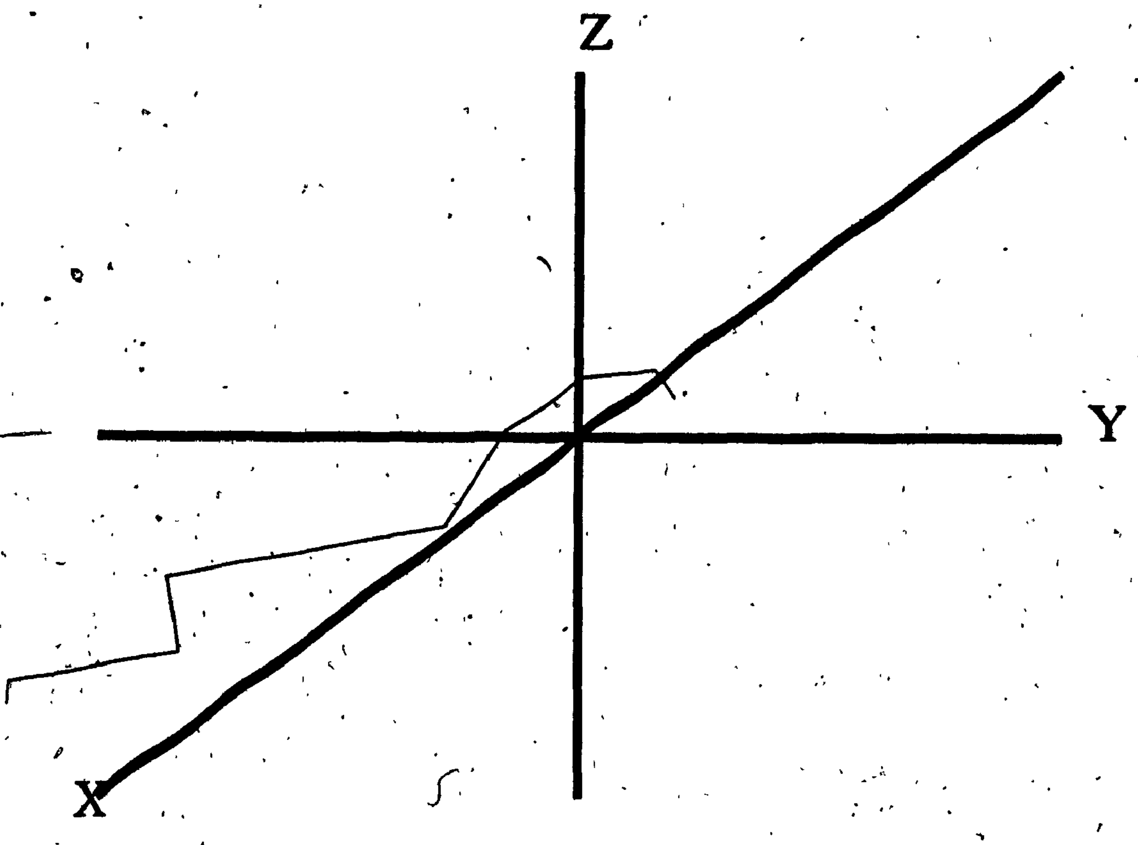
Originally there was some problems when the arm was navigating around the unreachable area. The arm behaved erratically in this area some times which was later traced to a problem in the coefficients that were being computed. The problem was arising from the fact that some of the tiles on the boundary of the inner unreachable space were only partially reachable. The centers of the tiles that were giving problems were on the unreachable side of the boundary. This meant that some of the values in the kinematics that were being used to compute the coefficients were becoming undefined which made the resulting coefficients undefined for the tile. This was then the reason for the erratic behavior in this area.

This problem has been solved by moving the position within the tile where the coefficients are computed when the center of the tile becomes unreachable. The position that has been chosen for these cases is the corner of the tile that is closest to the xy plane and farthest from the origin. The reason for this choice is that this position in the tile will be the last part to disappear into the unreachable space and it is easy to compute its location.

5.5 EFFECT OF FRAME SIZE:

The frame size is directly reflected in the accuracy of the arm movements. If the frame size is small, then the computations are done more frequently than in the case of a large frame size. This is due to the fact that for the duration of a frame (whose length is determined by the frame size), the motor movements are assumed to be linear. In reality this is not the case since most of the joints are revolute and not translational. This means that in the case of moving just one motor, the path that would be expected by the system is a tangent to the actual circular path taken by that joint. This is acceptable when the frame size is small because the basic direction of the movement will be towards the destination. With a large frame size, the path will no longer be towards the target at the end of the frame. This means that the arm may zig-zag or wander about on its path towards the target as in **FIG-10** or may even cause the arm to begin searching. In this case it may also be possible to encounter a situation where the arm oscillates back and forth about the target. This would be the case in a large frame size since the arm would follow a path for an extended distance compared to the smaller frame size.

Another important factor in the effect of the frame size is the magnitude of the motor steps. If the motor steps are large, then the frame size will be forced to be smaller than if the motor step size is very small. Therefore, in deciding the size of the frame, the motor step size must be considered since it is really the distance covered in a frame that is important and not the number of pulses sent to the



Plot of WRIST TRAJECTORY. ---> Camera view is 3-dimensional.
FRAME SIZE = 300 TILE SIZE: PRIMARY = 5.0 SECONDARY = 0.0100
START JOINT POSITIONS: J1=90.00 J2= 0.00 J3=50.00 J4=90.00
END JOINT POSITIONS: J1=265.00 J2= 0.00 J3=50.00 J4=90.00

FIG-10

motors. It also seems to make sense that in an application to a real system , there should be a separate frame size for each of the motors since all of the motor step sizes may vary as well as the speeds at which pulses may be delivered to the motors. In a case such as this, the scaling factor for the pulses would be more difficult to compute since it would have to insure that all of the motors were scaled within their frame size. The scaling factor could be found by finding the motor that was over the frame size by the largest amount instead of the maximum now in use.

In summary, the arm is well behaved within certain constraints. The simulation of the system was very useful in identifying and examining the situations where normal behavior of the arm degraded to unpredictable behavior. The constraints that effect the behavior of the system apply to the selection of the frame and tile sizes in the system since large tile sizes may cause inaccurate guidance and large frame sizes can cause irregular behavior which may even include oscillations preventing the arm from reaching the target. Also, if these parameters are too small, they may cause inefficiency in the system.

In this chapter, we have also looked at the problems associated with a motor reaching a limit of movement and their solutions. The handling of these situations is important to the navigation of the arm around unreachable areas of the work space. We have seen in the computation of pulses to the motors that it is important to use a scaling technique to reduce the movements of the individual motors within the frame size as opposed to truncating the number of pulses to the frame size.

CHAPTER 6: ANALYSIS

From the experience with the simulation of the control algorithm, we were able to gain a better understanding of the behavior of the manipulator under varying conditions and circumstances. From this, we were able to undertake the following analytical study of the positioning error introduced into the system by the control algorithm. The results of this analytical analysis coincided with our experiences from the simulation.

6.1 MOTOR ERROR ANALYSIS:

In this section we examine the sources of error within the algorithm as well as the overall performance of the system. Much of the performance information is derived from the graphical output of the simulation.

The error in the algorithm can be determined for each of the individual joints in the arm. This can be accomplished through a vector analysis of each joint movement. The error for motor zero would be expected to be a three dimensional parabolic surface. This has been confirmed from the plots of the error between the intended position and the actual position.

So, for motor zero, the coefficient, A_0, can be determined by rotating the wrist vector, W, by the motor zero step angle, λ_0, and subtracting the original wrist vector from the rotated version to give the coefficient that will be used by the algorithm.

$$A_0 = \begin{vmatrix} \cos(\lambda_0) & -\sin(\lambda_0) & 0 \\ \sin(\lambda_0) & \cos(\lambda_0) & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} W_x \\ W_y \\ W_z \end{vmatrix} - \begin{vmatrix} W_x \\ W_y \\ W_z \end{vmatrix}$$

Then the predicted position, $\mathbf{V_0}$, of the wrist for a frame sized move will be the wrist vector, **W**, plus the coefficient vector scaled by the frame size.

$$\mathbf{V_0} = \mathbf{W} + \lambda_0 * \mathbf{A_0}$$

The actual position, $\mathbf{P_0}$, that is reached by the wrist, since this is a rotational joint, will be the wrist vector, **W**, rotated by **f** times λ_0.

$$\mathbf{P_0} = \begin{vmatrix} \cos(f*\lambda_0) & -\sin(f*\lambda_0) & 0 \\ \sin(f*\lambda_0) & \cos(f*\lambda_0) & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} W_x \\ W_y \\ W_z \end{vmatrix}$$

This makes the error due to this joint, $\mathbf{E_0}$, the difference between the actual position and the predicted position.

$$\mathbf{E_0} = \mathbf{P_0} - \mathbf{V_0}$$

Motor one is expected to have a similar shape only in the vertical planes since it is also a rotational joint. Also, in the horizontal planes this joint will have the same shape. The plots have confirmed that this joint has the expected error plots.

The error for this joint can be determined by first computing a new wrist vector, W', by rotating it by the base rotation angle, Θ_1.

$$\mathbf{W} = \begin{vmatrix} \cos(\Theta_1) & \sin(\Theta_1) & 0 \\ -\sin(\Theta_1) & \cos(\Theta_1) & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} W_x \\ W_y \\ W_z \end{vmatrix}$$

Then, as in motor $\mathbf{m_0}$, the coefficient for motor $\mathbf{m_1}$, $\mathbf{A_1}$, can be determined by rotating the new wrist vector, $\mathbf{W}$, by the motor $\mathbf{m_1}$ step angle, λ_0 and subtracting $\mathbf{W}$ from it.

$$\mathbf{A_1} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos(\lambda_1) & -\sin(\lambda_1) \\ 0 & \sin(\lambda_1) & \cos(\lambda_1) \end{vmatrix} * \begin{vmatrix} W_x \\ W_y \\ W_z \end{vmatrix}$$

Then the predicted position, $\mathbf{V_1}$, can be determined as for motor one using:

$$\mathbf{V_1} = \mathbf{W} + \lambda_1 * \mathbf{A_0};$$

The actual position reached by moving this joint, $\mathbf{P_1}$, is:

$$\mathbf{P_1} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos(f*\lambda_1) & -\sin(f*\lambda_1) \\ 0 & \sin(f*\lambda_1) & \cos(f*\lambda_1) \end{vmatrix} * \begin{vmatrix} W_x \\ W_y \\ W_z \end{vmatrix}$$

The error for this motor is, as for motor $\mathbf{m_0}$, the difference between the actual and predicted positions of the wrist.

$$\mathbf{E_1} = \mathbf{P_1} - \mathbf{V_1}$$

The error plot in **FIG-15** is similar to that created by plotting this function.

For motor $\mathbf{m_2}$, there should be no error since it is a linear joint. This is because the algorithm tries to treat the rotational joints as if they were linear so for a linear joint the result is that no error occurs due to the algorithm. This has been confirmed by the error plots for movements of one frame. The coefficient for motor $\mathbf{m_2}$ is then a vector with magnitude equal to λ_2 and direction equal to $\mathbf{L_2}$. This makes the coefficient for motor $\mathbf{m_2}$:

$$A_2 = \lambda_2 / (|L_2| + \lambda_2) * L_2$$

The predicted (and actual) position of the wrist due to the application of **f** pulses to motor $\mathbf{m_2}$ will be:

$$P_2 = f*((|L_2| + \lambda_2) / \lambda_2) * L_2 + W$$

where $\mathbf{L_2}$ is the vector representing translational link.

6..2 COMBINED JOINT ERRORS.

Since the algorithms purpose is to compute scaling factors for the coefficient vectors for the first three motors so that their sum will be equal to the vector from the current wrist position to the target wrist position, the errors in each of the joints will add together. This will cause the arm to move away from its planned trajectory by the end of the frame. This means that the frame size should be set so that the deviation from the linear path to the target is no greater than some predefined tolerance. After setting the frame size,. the tile sizing should then be determined. The size of the tiling should be large enough that on average, each frame will terminate in a new tile which should be closer to the target and thus have more accurate information on how to reach the target.

So, the error in moving all of the motors by a full frame size number of pulses can be determined by combining the error at each joint. With this method we will be able to determine the error in the final position of the wrist. This will not give us the error in the number of pulses that should have been sent to the motors to achieve the intended destination. To do this, we would have to determine the joint positions at the current position and the target position and by taking there differences, determine the correct number of pulses that should have been sent to reach the target. Then, by comparing this to the number of pulses computed in the algorithm, determine the error in the number of pulses sent to the motors.

The errors we have examined are not the errors in the number of pulses to be delivered to the individual motors, but the positional error of the manipulator wrist due to errors in the number of pulses needed to reach a specific position.

Since we know that the error in the position of the wrist is due to the error in the rotational joints of the arm, we can determine the total error in the system by summing the individual joint errors to obtain the total. First, the expression determining the actual position after sending a number of pulses to each of the motors must be determined. If we know that each motor is to receive **i** pulses (i=0,1,2), and motor $\mathbf{m_0}$ transforms the wrist position vector, **W**, to $\mathbf{P_0}$, motor $\mathbf{m_1}$ transforms $\mathbf{P_0}$ to $\mathbf{P_1}$ and motor $\mathbf{m_2}$ transforms $\mathbf{P_1}$ to $\mathbf{P_2}$, the final position, we can determine the final position by applying each of the transformations in order to the wrist position vector, **W**, to give:

$$\mathbf{P_2 = f_2 * ((|L_2| + \lambda_2) / \lambda_2) * L_2 + P_1}$$

$$\mathbf{P_1 = T_4 * (T_5 * P_0)}$$

$$\mathbf{P_0 = T_2 * W}$$

Combining these three equations we get an expression for the actual position reached by the wrist.

$$P_2 = f_2*((|L_2| + \lambda_2) / \lambda_2) * L_2 + T_4*(T_5*(T_2*W))$$

The matrices, T_2, T_4, T_5 are from the matrices found for each of the individual joints for the arm.

$$T_2 = \begin{vmatrix} \cos(f_0*\lambda_0) & -\sin(f_0*\lambda_0) & 0 \\ \sin(f_0*\lambda_0) & \cos(f_0*\lambda_0) & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$T_4 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos(f_1*\lambda_1) & -\sin(f_1*\lambda_1) \\ 0 & \sin(f_1*\lambda_1) & \cos(f_1*\lambda_1) \end{vmatrix}$$

$$T_5 = \begin{vmatrix} \cos(\Theta_1) & \sin(\Theta_1) & 0 \\ -\sin(\Theta_1) & \cos(\Theta_1) & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Then, to compute the predicted position, V_2, we get:

$$V_2 = f_2*((|L_2| + \lambda_2) / \lambda_2) * L_2 + V_1$$

$$V_1 = T_5*V_0 + f_1*A_1$$

$$V_0 = W + f_0*A_0$$

The expression for the predicted position is then:

$$V_2 = f_2*((|L_2|+\lambda_2) / \lambda_2) * L_2 + T_5*V_0$$
$$+ f_1*A_1 + W + f_0*A_0$$

This finally leads to the error equation, which is the difference between the actual position and the predicted position of the wrist.

$$E = P_2 - V_2$$

This is only the error that would be encountered if the coefficients were determined for the current position of the wrist. If the wrist is not positioned in the center of the current tile, then the coefficient that is being used to determine the predicted position will also be in error. The error in the final position of the wrist from the predicted would need to be adjusted to reflect the error in the original coefficients that were used in predicting the position of the wrist. The error in each of the coefficients for the two rotational joints can be determined by computing the coefficient for the joint using the difference vector of the current wrist position and of the wrist vector to the center of the tile.

The total error in a movement due to the fact that the coefficients are not for the current position of the wrist, but for some nearby position instead, can be determined by realizing what the algorithm does in computing the number of pulses to be sent to move the wrist by the specified vector. It really computes the pulses for a vector that is parallel to the displacement vector from the wrist to the target. This parallel vector has its origin at the center of the tile. This will add a small error (proportional to the distance between the current position and the center of the tile) to the number of pulses that has been computed for the motors.

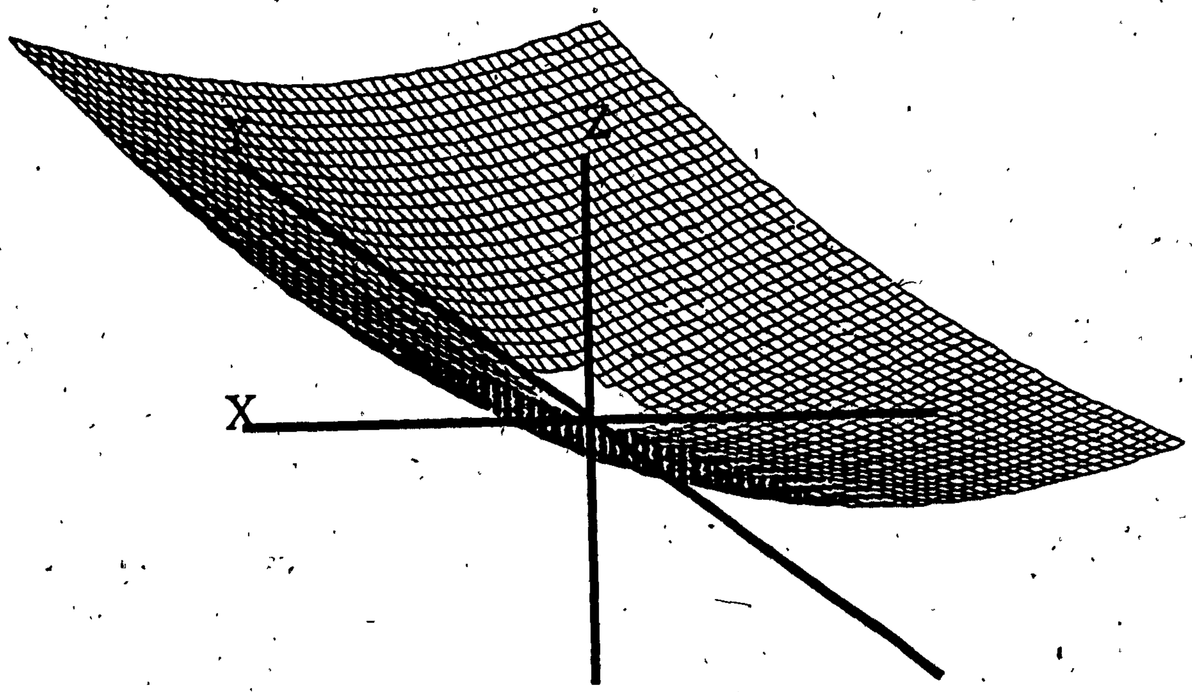
6.3 ERROR PLOTS.

Error plots based on frame size have been generated for each motor. They have been plotted as grid surface maps. The method used in creating these plots was to pick a horizontal or vertical plane. Then, for all of the tiles on that plane, the configuration of the arm needed to position the wrist at the center of the tile was determined using the inverse kinematics of the arm. Then the motor that was to be examined was moved by the frame size number of pulses. The position that the motor should have reached by extrapolating the coefficients for the tile was then determined and from the forward kinematics of the arm, the actual position that was reached was found. The magnitude of the difference vector between these two positions was then substituted for the coordinate that was constant for the plane to give the surface plot.

Then, as a double check, the error expressions derived above were used in determining the magnitude of the error to be plotted in each tile. The results of this set of plots was identical to that using the kinematics for the arm.

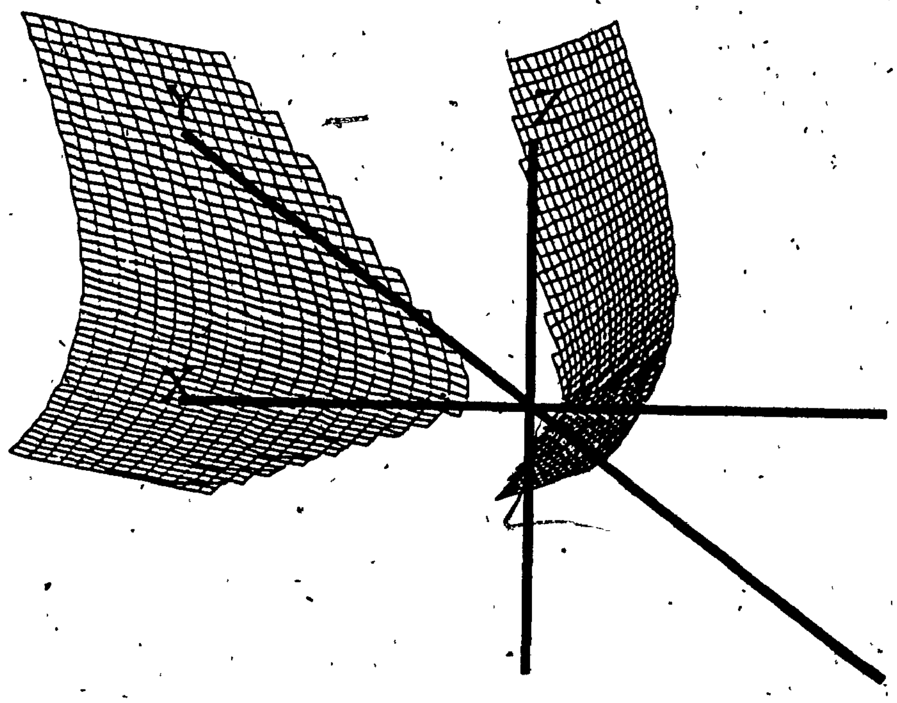
MOTOR m_0:

The error for this motor (**FIG-11**) is expected to increase as the length of the wrist vector increases. This is because the wrist position will be moved through a greater arc as the distance from



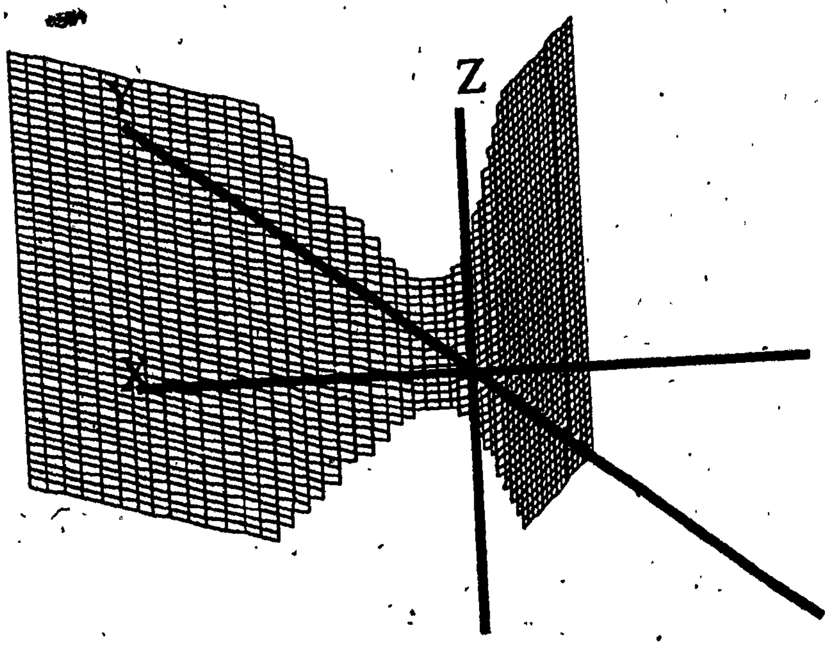
Frame Error Plot:
Error for XY plane 0.0 for motor 0:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-11



Frame Error Plot:
Error for XZ plane 0.0 for motor 1:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-12



Frame Error Plot:
Error for XZ plane 10.0 for motor 0:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-13

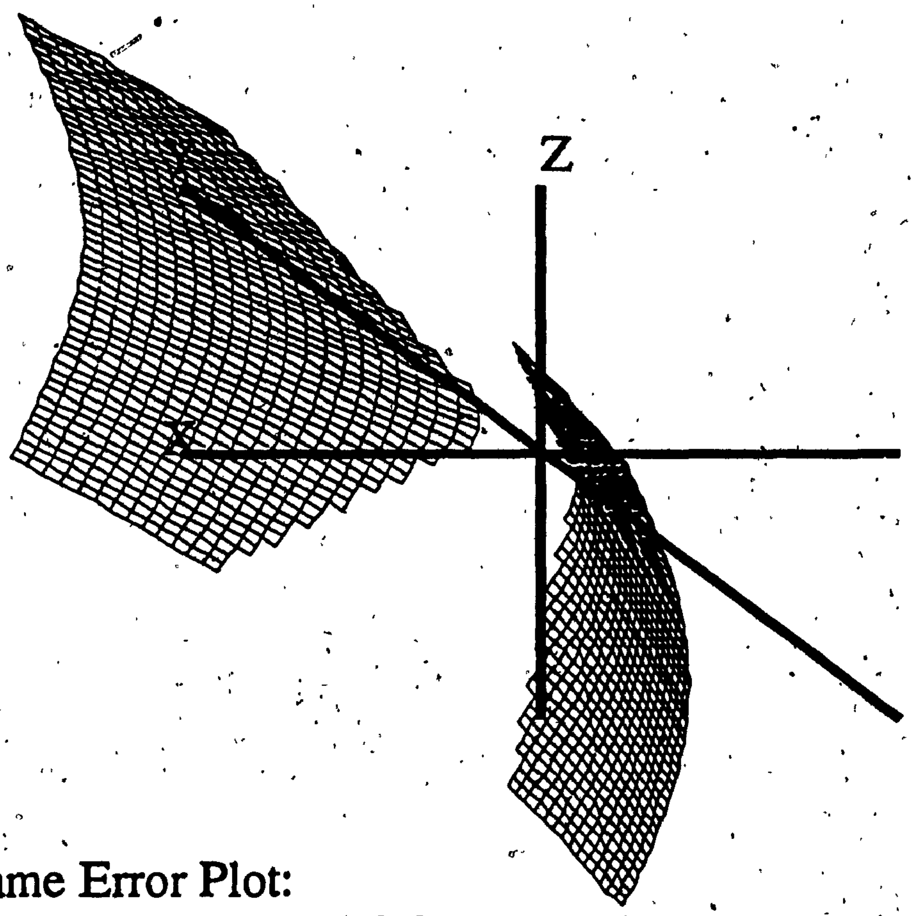
the center of rotation increases. There is only the need to plot one plane for this motor, since this motor rotates in the x-y plane. It only is concerned with the length of the wrist vector in the x-y plane. Other planes are taken at different heights, the z component doesn't enter into the error since it is not affected by this motor.

Also, the vertical planes are not of interest since they will be curved, with the error reaching a minimum at the vertical axis and increasing away from the axis. This is to be expected since these plots are the equivalent of taking a slice of the horizontal plot and replicating it at different vertical levels (**FIG-12**).

It is interesting to note that the plot has been divided into two completely separate sections by the inner unreachable space. As the vertical slices are taken farther from the vertical axis (**FIG-13**), the central part fills in as that region becomes reachable by the arm.

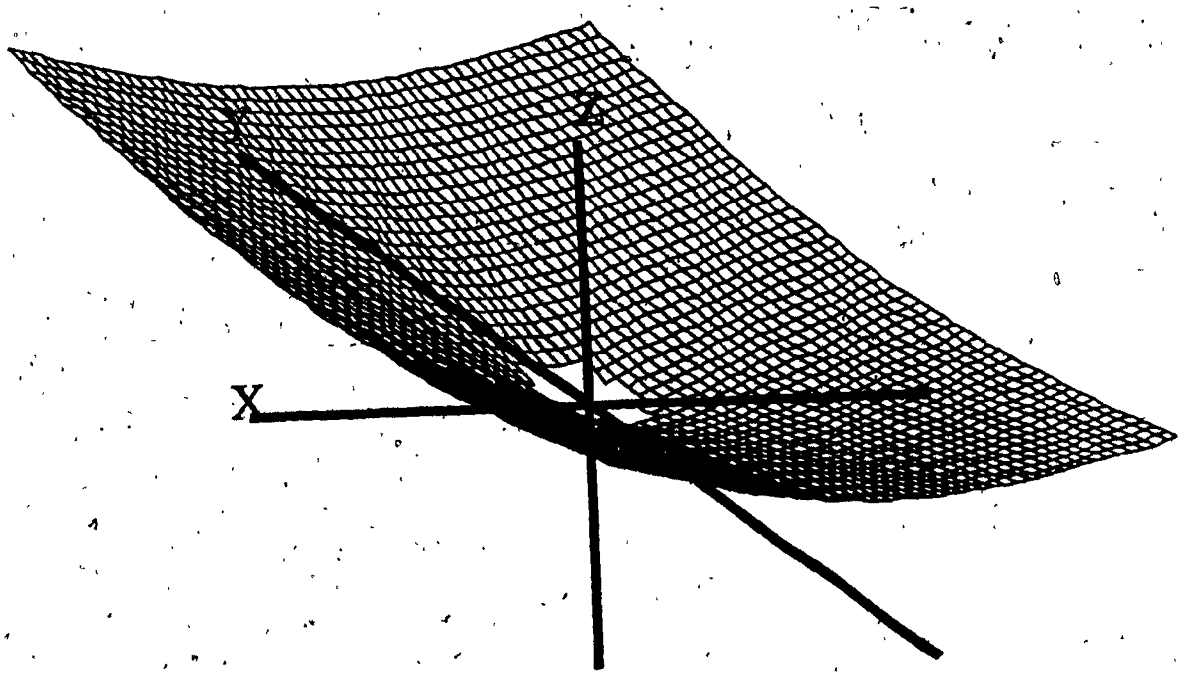
MOTOR m_1:

Motor m_1 is expected to have a vertical plot that resembles the horizontal plot of motor m_0 since it is also rotational but in the vertical plane (**FIG-14**). Also, since the base of reference for motor m_1 is rotated by motor m_0, it is expected that the vertical plots are made up of many of these parabola shaped functions. Therefore, the overall plot of the vertical plane will also be parabola shaped, with error increasing as the wrist moves away from the origin. However, since this motor's vertical plots can be thought of as an infinite series of parabola shaped plots, each one rotated slightly more about the z-axis than the previous one, the horizontal plane will take on a similar appearance to the vertical planes in shape. This is because the



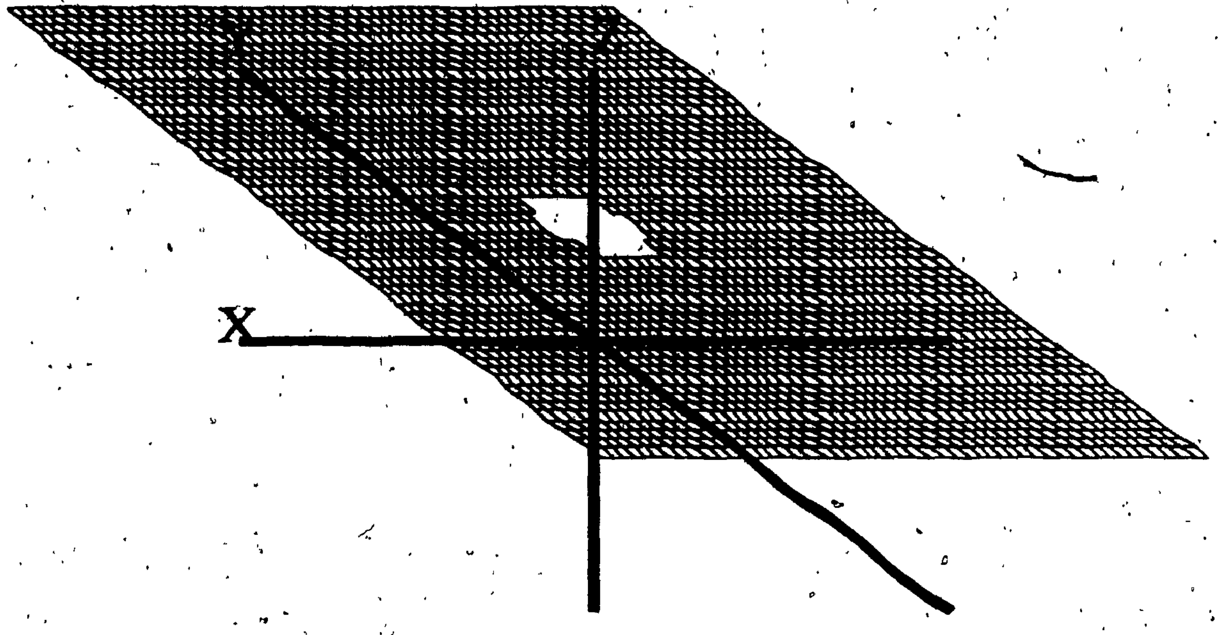
Frame Error Plot:
Error for YZ plane 0.0 for motor 1:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-14



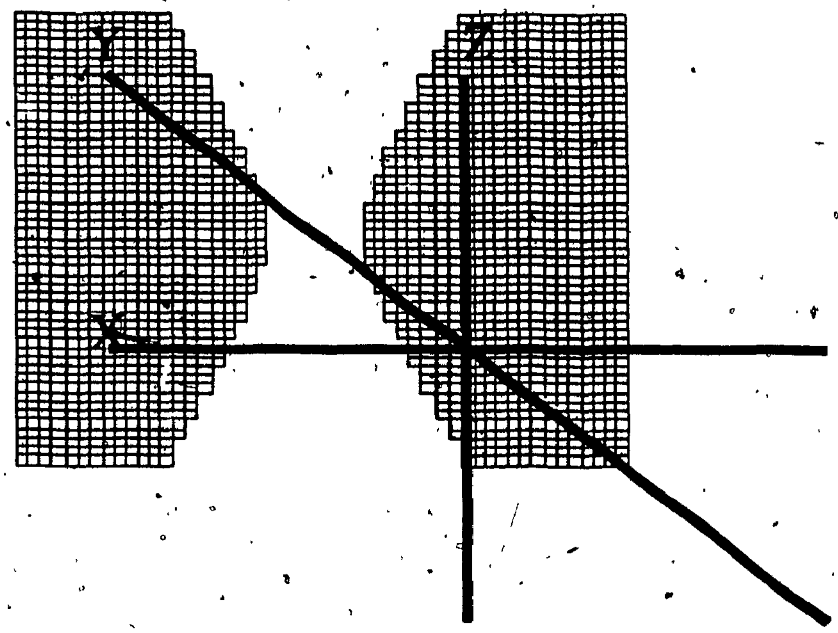
Frame Error Plot:
Error for XY plane 0.0 for motor 1:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-15



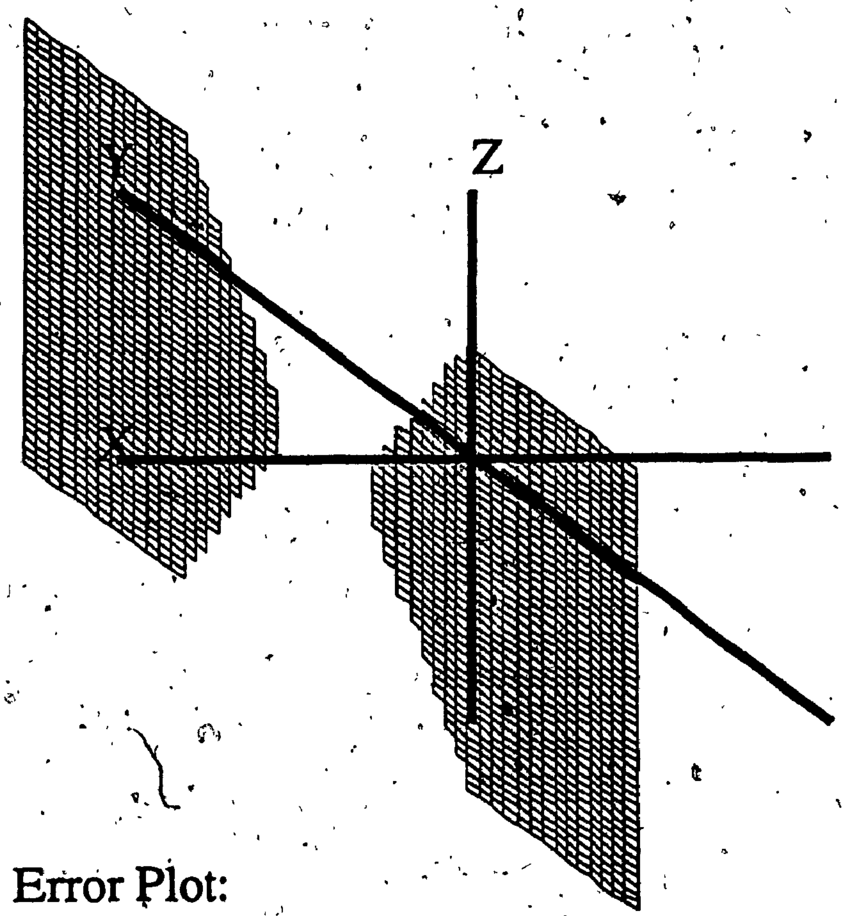
Frame Error Plot:
Error for XY plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-16



Frame Error Plot:
Error for XZ plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-17



Frame Error Plot:
Error for YZ plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
frame size = 20

FIG-18

horizontal planes will be the equivalent of composing a surface from pieces of the horizontal parabolas which becomes a parabola itself (**FIG-15**).

MOTOR m_2:

Motor $\mathbf{m_2}$ is a very uninteresting motor as far as error plots go. The plots for this motor are perfectly flat for all planes (**FIG-16, FIG-17, FIG-18**) since the motor exerts a linear motion. This means that the movement of more than one pulse based on the coefficients will reach the actual position that would have been the target.

6.4 COEFFICIENT VARIANCE.

The variance of the coefficients within a tile is proportional to the size of the the tile. The larger the tile, the greater the variance is expected to be since points farther apart are expected to have a larger difference in their coefficients.

To examine the variance of the coefficients across the space of a tile,surface plots have been generated which represent the variance of the coefficients within a tile in a specified direction. That is, the plots have been created by plotting the magnitude of the difference for the coefficients of a motor computed on parallel faces of the tile. The plots represent the difference in the direction between the two coefficients generated since all of the coefficients for a motor will have the same magnitude but their directions will be different. The plots were generated for the horizontal plane and the two vertical planes for each motor. There is a choice of three plots in a plane for each motor. The plots can represent the variance across the tile in the **x** direction, **y** direction or **z** direction. These plots revealed information on the rate of change of the coefficients across the tiled space of the arm.

MOTOR m_0:

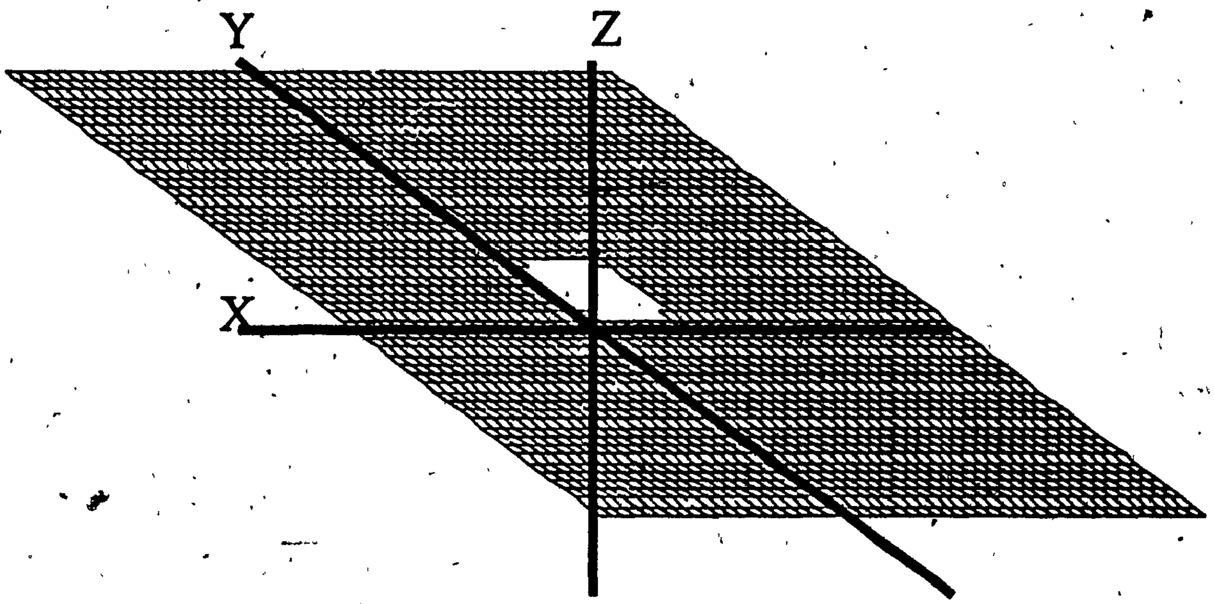
The plot for this motor was completely flat for all planes and directions (**FIG-19**). This was expected since the motor causes rotation around the base of the arm and the coefficient should vary at a constant rate.

Upon increasing the tile size for these plots, The magnitude of the variance increased but the surface remained flat. Also note that the variance in this plot has been *scaled by 1000*. So, it can be seen that the variance of the coefficients for this motor are very small.

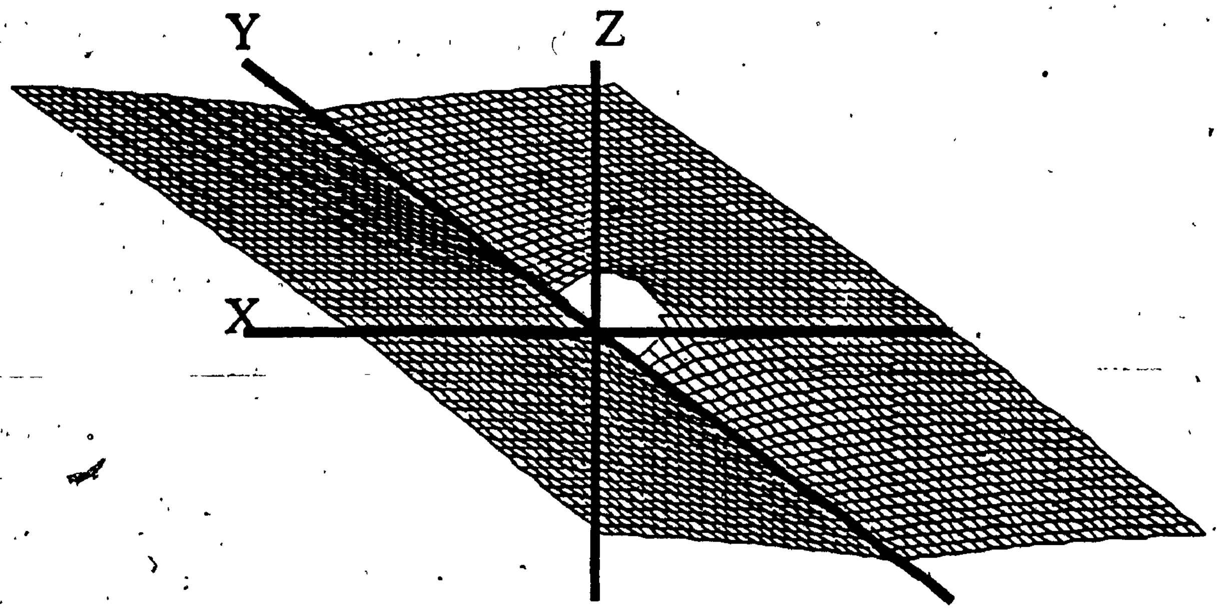
MOTOR m_1:

The variance for this motor is not as flat as that for motor m_0. This motor has a slightly curved variance plot for the **x** (**FIG-20**) and **y** (**FIG-21**) directions on the horizontal plane but a perfectly flat plot for the **z** direction (**FIG-22**).

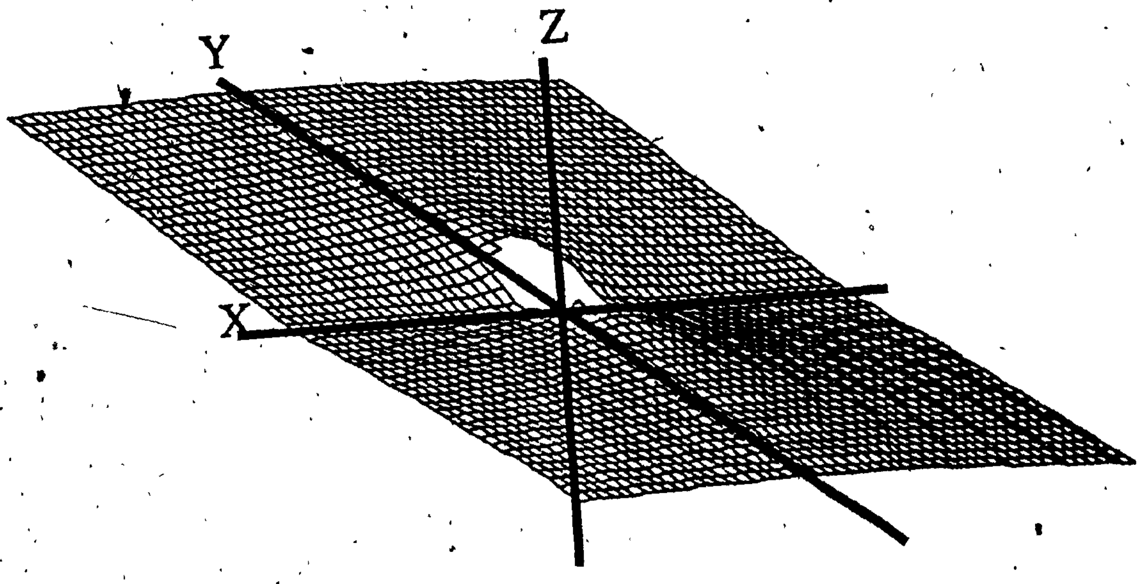
The **x** direction and **y** direction plots for this motor appear to be the same plot with the **y** plot being rotated by 90 degrees. The plot of the variance in the x– direction across the tiles approaches a minimum near the x– axis and a maximum near the y– axis. For the y– direction plot, the minimum and maximum occur in the other order. The reason for the shape of this plot is that the base rotation angle to reach the two points on opposite sides of the tile is different for each point. This means that the direction of the coefficients for this motor will be pointing in slightly different directions. As you go around the z-axis, the direction of the



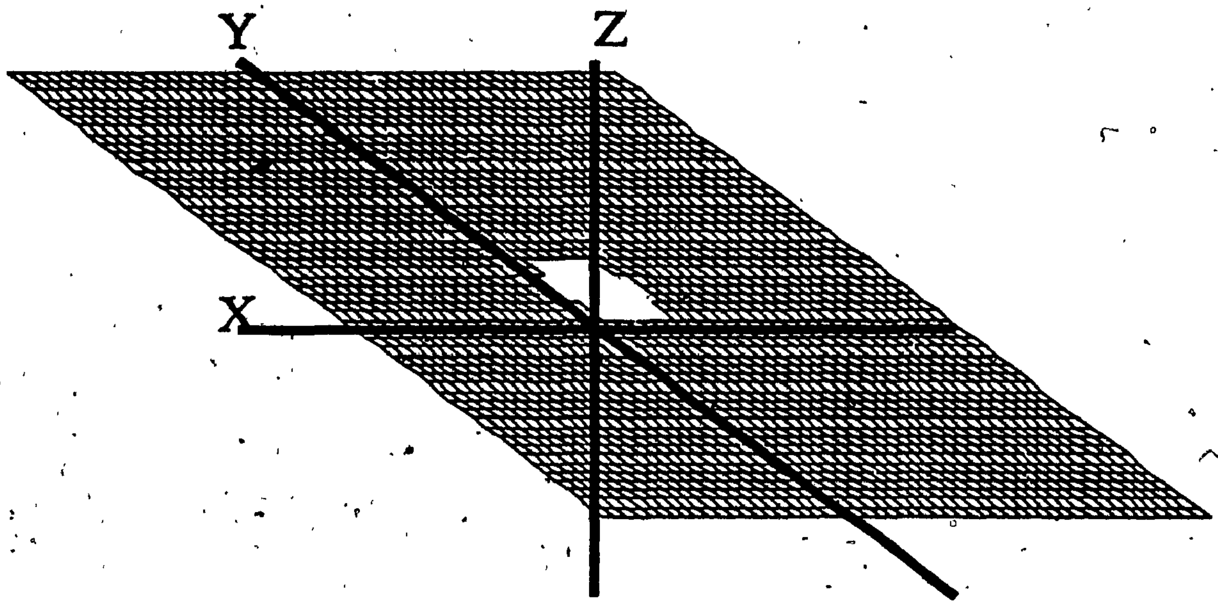
VARIANCE PLOT:
X-direction variance of XY plane 0.0 for motor 0.
tile size = 2.0 scaling factor = 1000.0
FIG-19



VARIANCE PLOT:
X-direction variance of XY plane 0.0 for motor 1.
tile size = 2.0 scaling factor = 1000.0
FIG-20



VARIANCE PLOT:
Y-direction variance of XY plane 0.0 for motor 1.
tile size = 2.0 scaling factor = 1000.0
FIG-21



VARIANCE PLOT:
Z-direction variance of XY plane 0.0 for motor 1.
tile size = 2.0 scaling factor = 1000.0
FIG-22

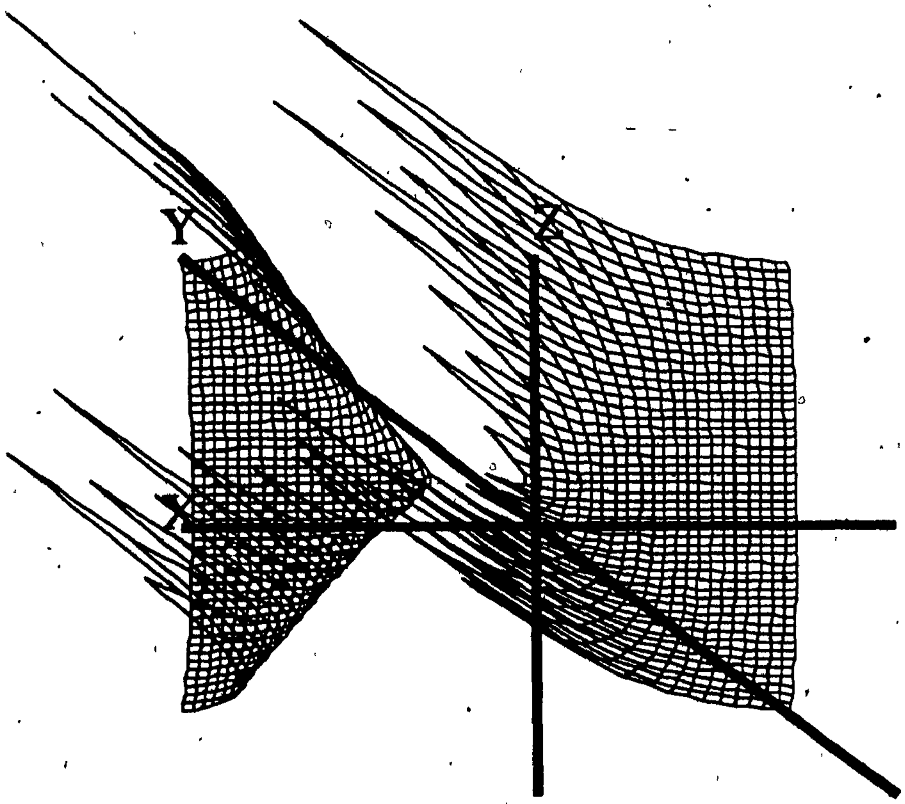
coefficients changes until they point in approximately the same direction near the x-axis and then begin to move apart again as you move away from the axis. The y direction is the same thing except the coefficients point in approximately the same direction near the y-axis instead of the x-axis.

The z-direction is expected to be flat, since the two points for the computation of the coefficients will be at the same base rotation angle. In the vertical planes (**FIG-23**, **FIG-24**), a similar plot appears for the **x** and **y** directions.

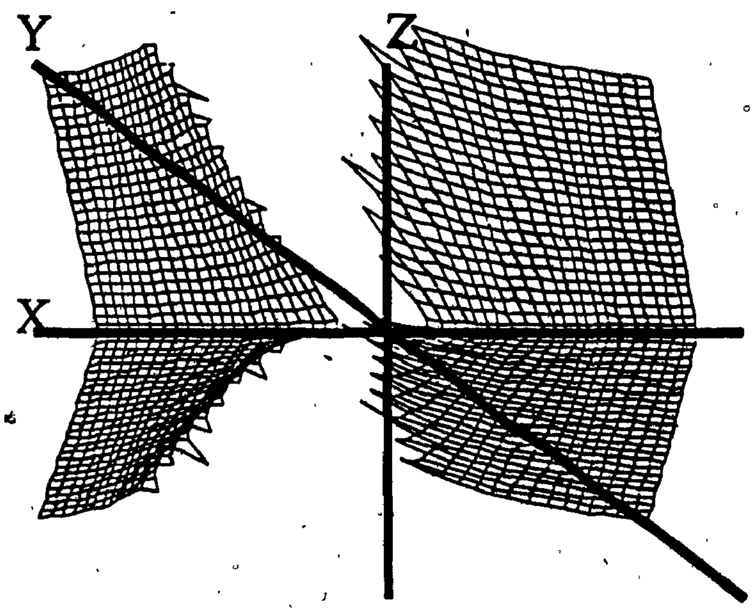
MOTOR m_2:

Motor m_2 has the variance plots (**FIG—25**, **FIG—26**, **FIG—27**) with the most features. These peaks and valleys are due to the same effect as was discussed for motor m_1 where the base rotation angle changes the direction of the coefficient. For this motor, there is the additional directional change due to the shoulder elevation which contributes to coefficient variance. Furthermore, the change in direction has a greater effect for this motor since it is controlling a linear translational joint so even a very small change completely changes the coefficients direction.

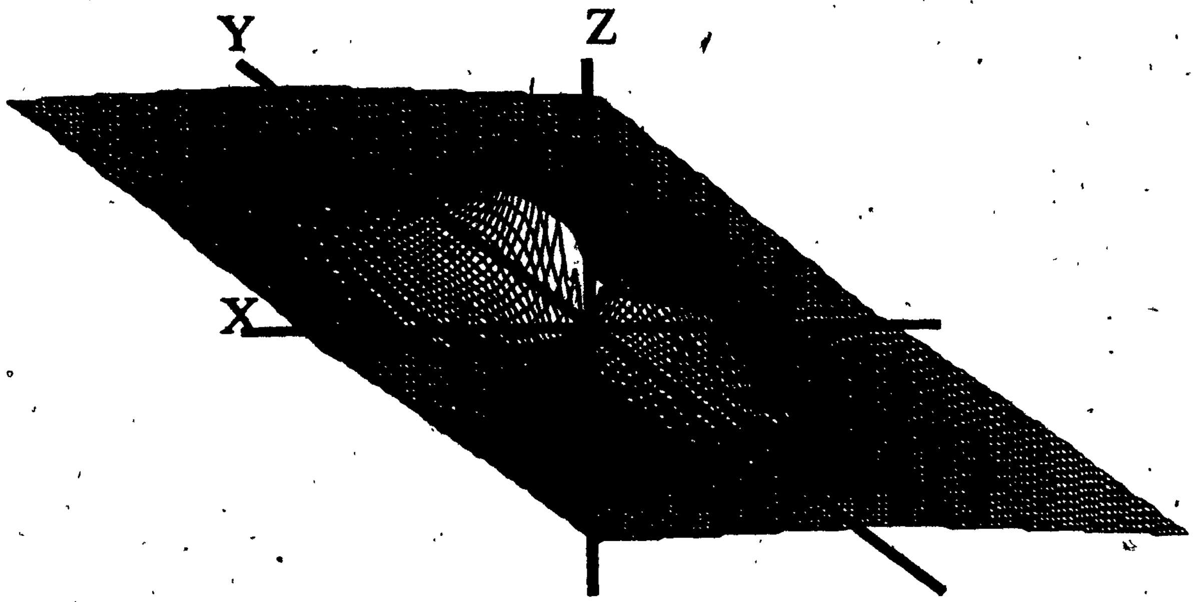
FIG-28 shows the type of plot obtained from the xz-plane for this motor. Note the break in the center of the plot due to the unreachable area. **FIG-29** is also the xz-plane but is taken at the point where **y** is 10. In this plot it can be seen that the unreachable area is begining to disappear as we move away from the origin. **FIG-30** shows that the other vertical plane (the yz-plane) for this motor are similar to the xz-plane.



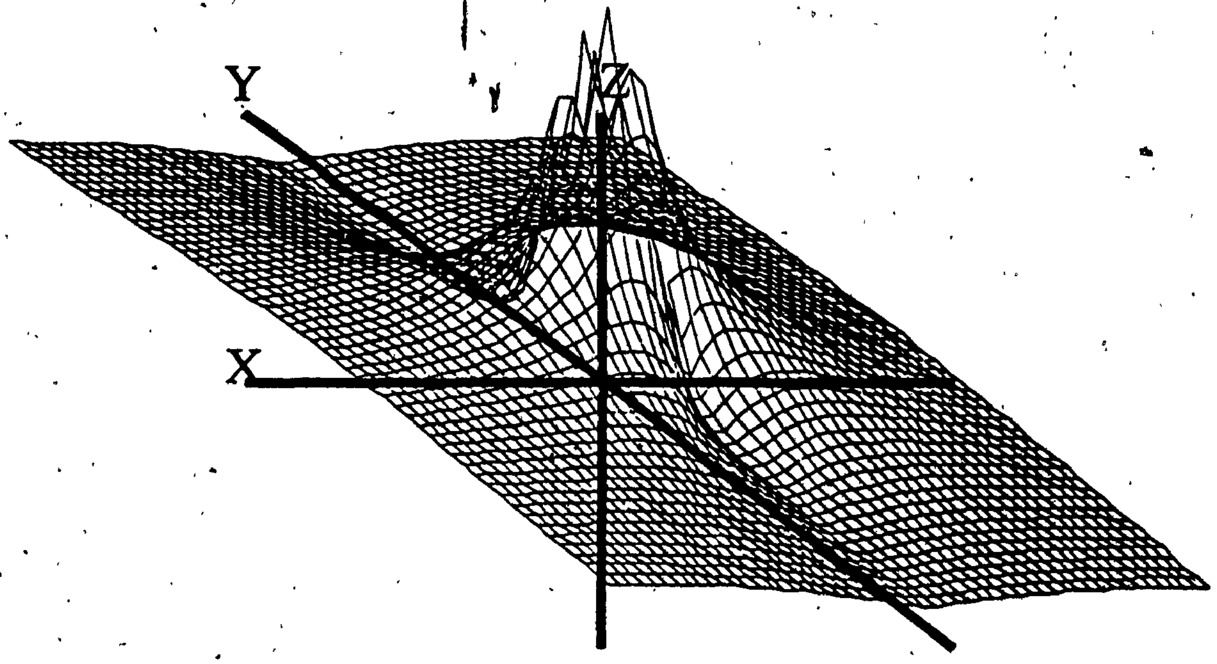
VARIANCE PLOT:
X-direction variance of XZ plane 0.0 for motor 1.
tile size = 2.0 scaling factor = 1000.0
FIG-23



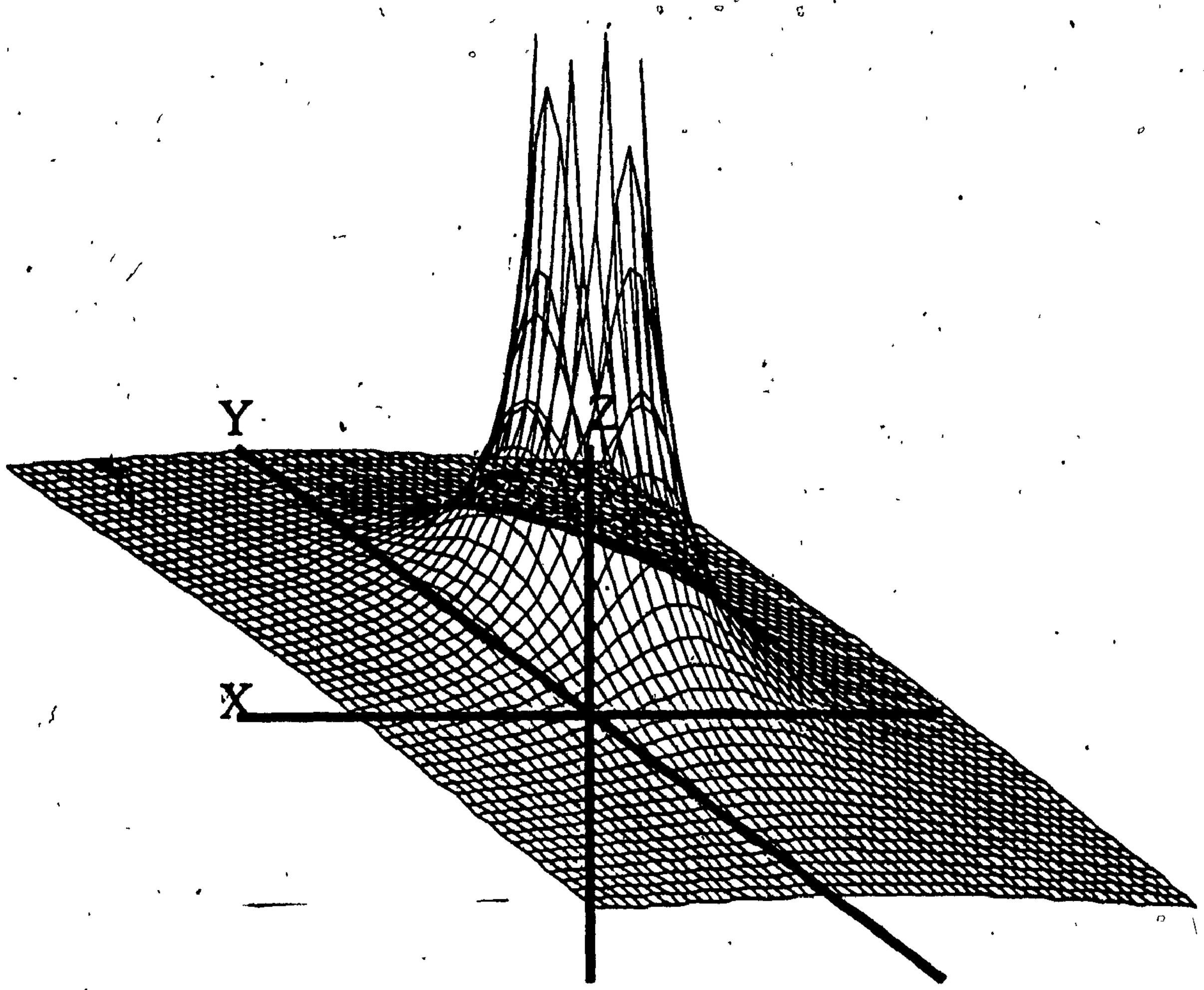
VARIANCE PLOT:
Y-direction variance of XZ plane 0.0 for motor 1.
tile size = 2.0 scaling factor = 1000.0
FIG-24



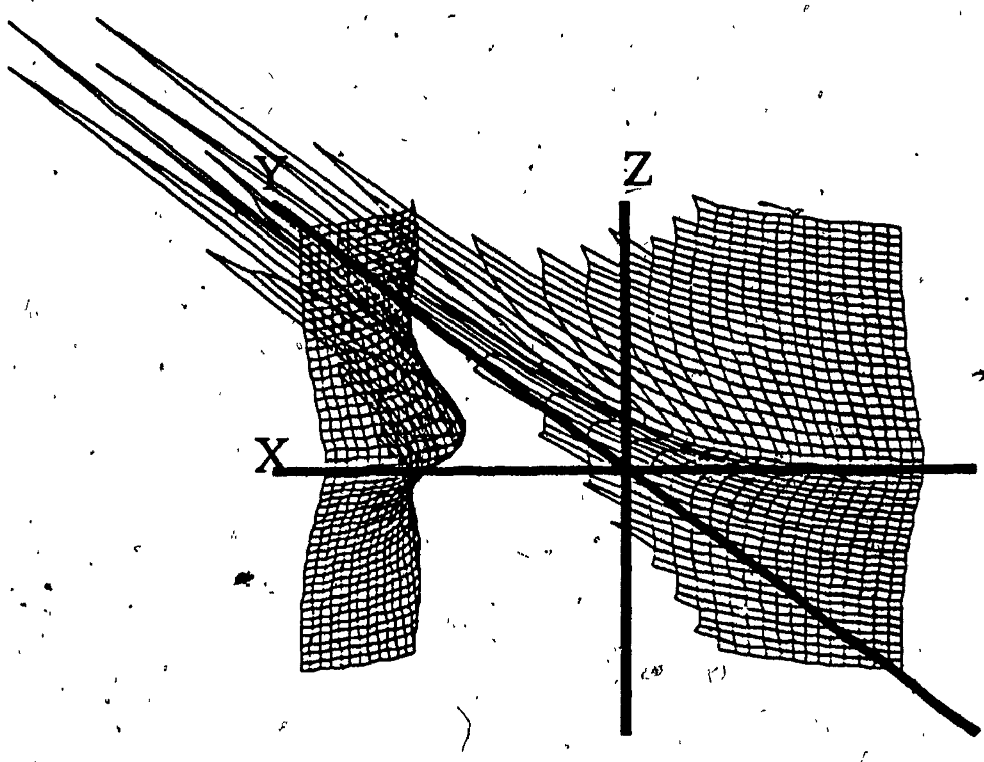
VARIANCE PLOT:
X-direction variance of XY plane 0.0 for motor 2.
tile size = 1.0 scaling factor = 1000.0
FIG-25



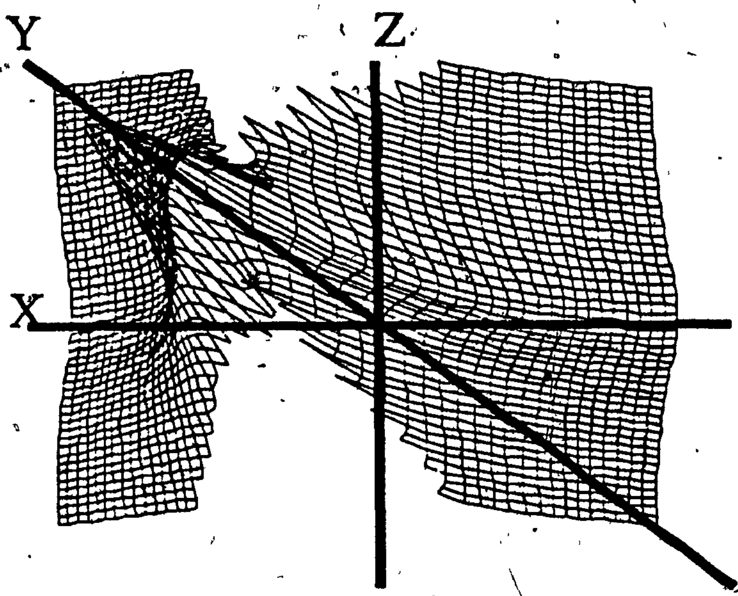
VARIANCE PLOT:
Y-direction variance of XY plane 0.0 for motor 2.
tile size = 2.0 scaling factor = 1000.0
FIG-26



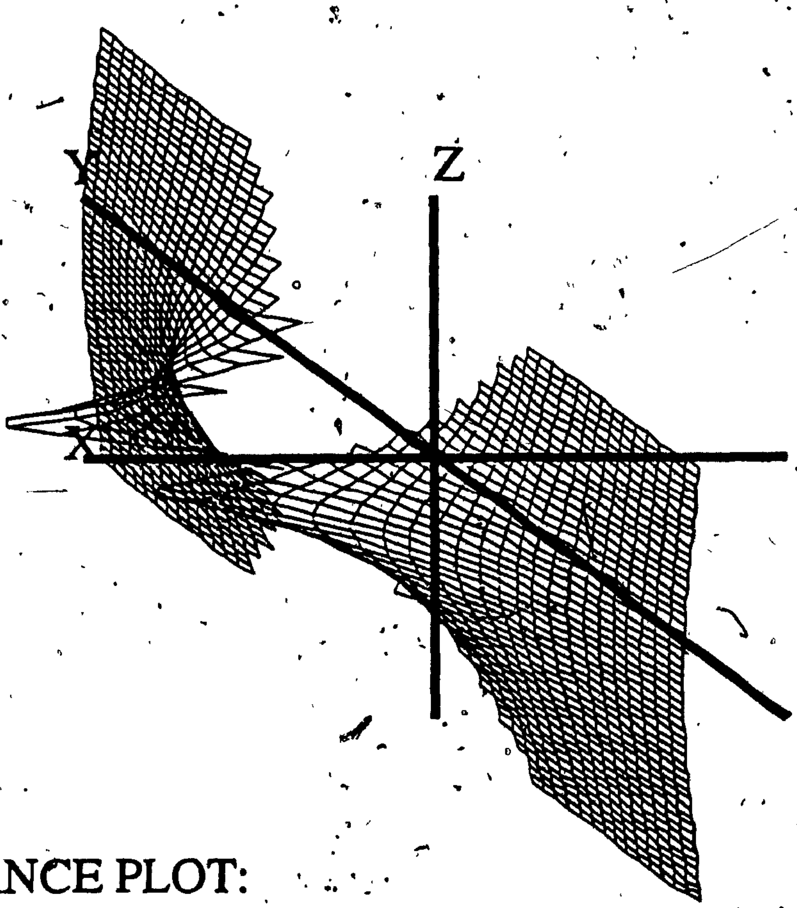
VARIANCE PLOT:
Z-direction variance of XY plane 0.0 for motor 2.
tile size = 2.0 scaling factor = 1000.0
FIG-27



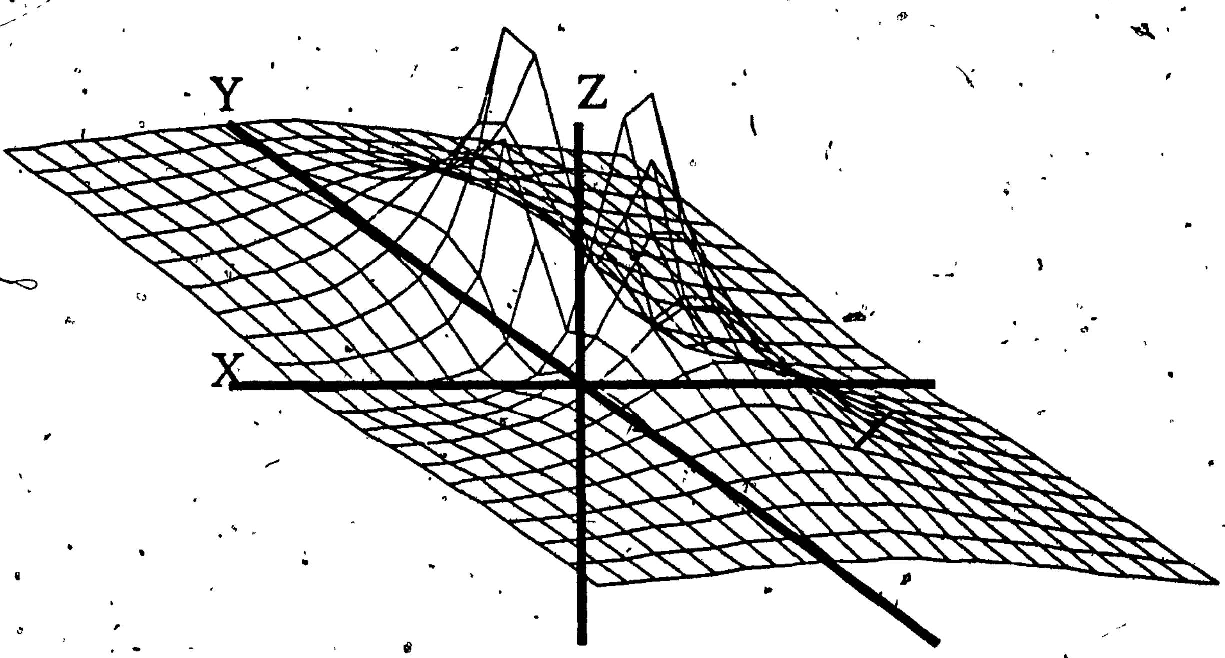
VARIANCE PLOT:
X-direction variance of XZ plane 0.0 for motor 2.
tile size = 2.0 scaling factor = 1000.0
FIG-28



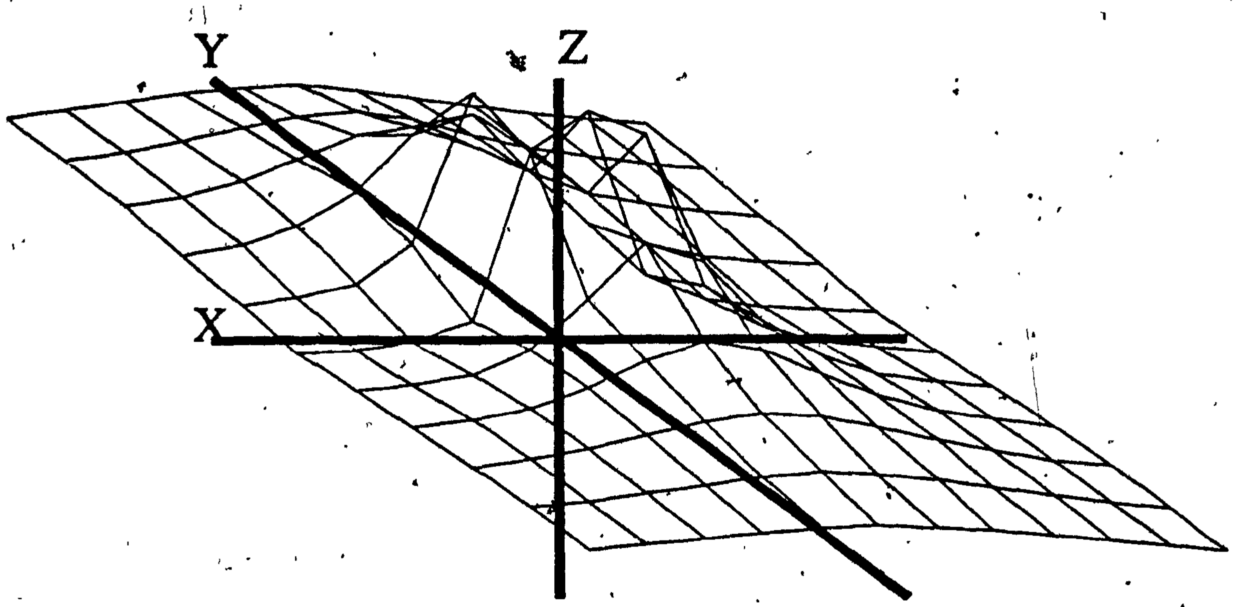
VARIANCE PLOT:
X-direction variance of XZ plane 10.0 for motor 2.
tile size = 2.0 scaling factor = 1000.0
FIG-29



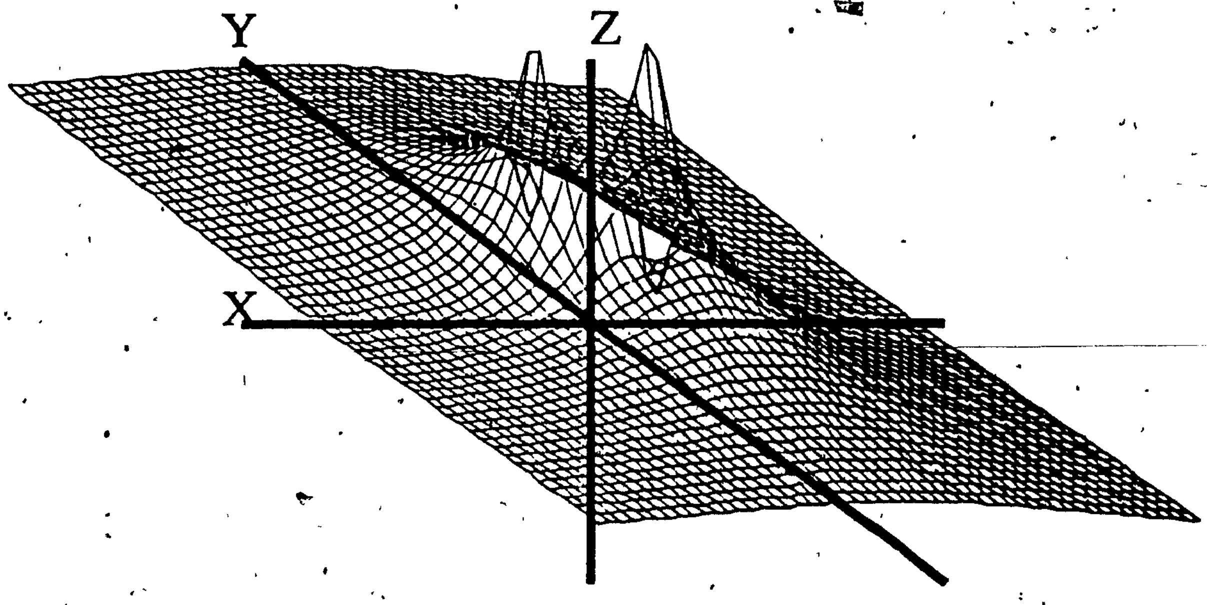
VARIANCE PLOT:
X-direction variance of YZ plane 0.0 for motor 2.
tile size = 2.0 scaling factor = 1000.0
FIG-30



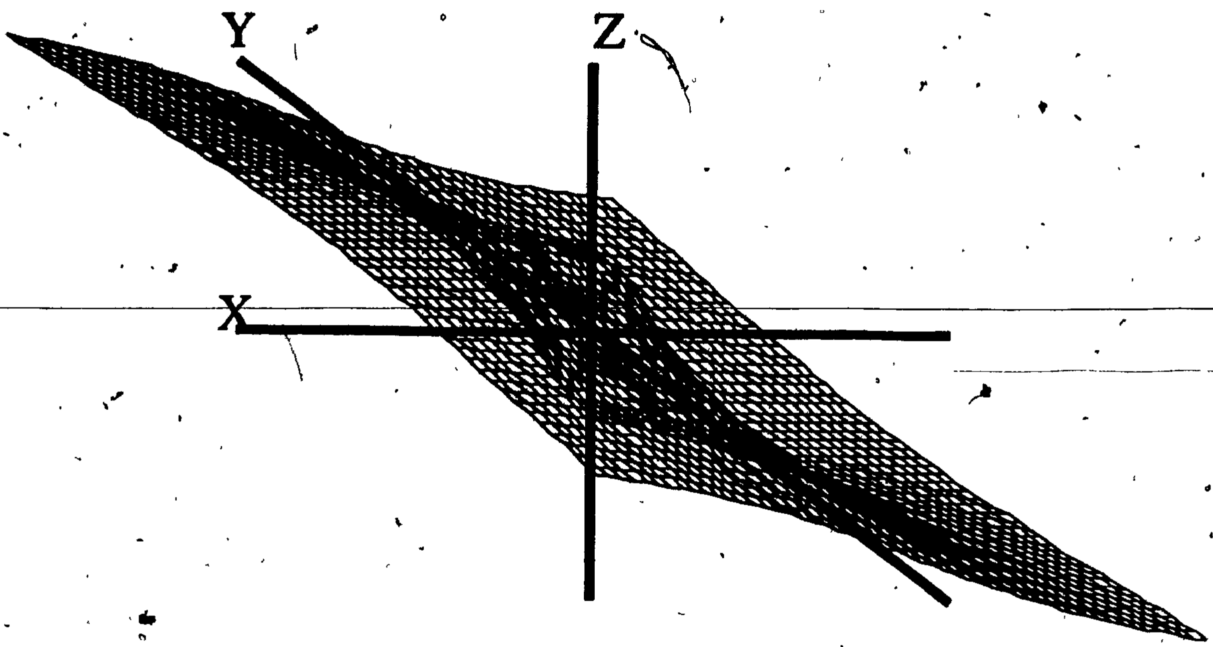
VARIANCE PLOT:
X-direction variance of XY plane 0.0 for motor 2.
tile size = 5.0 scaling factor = 500.0
FIG-31



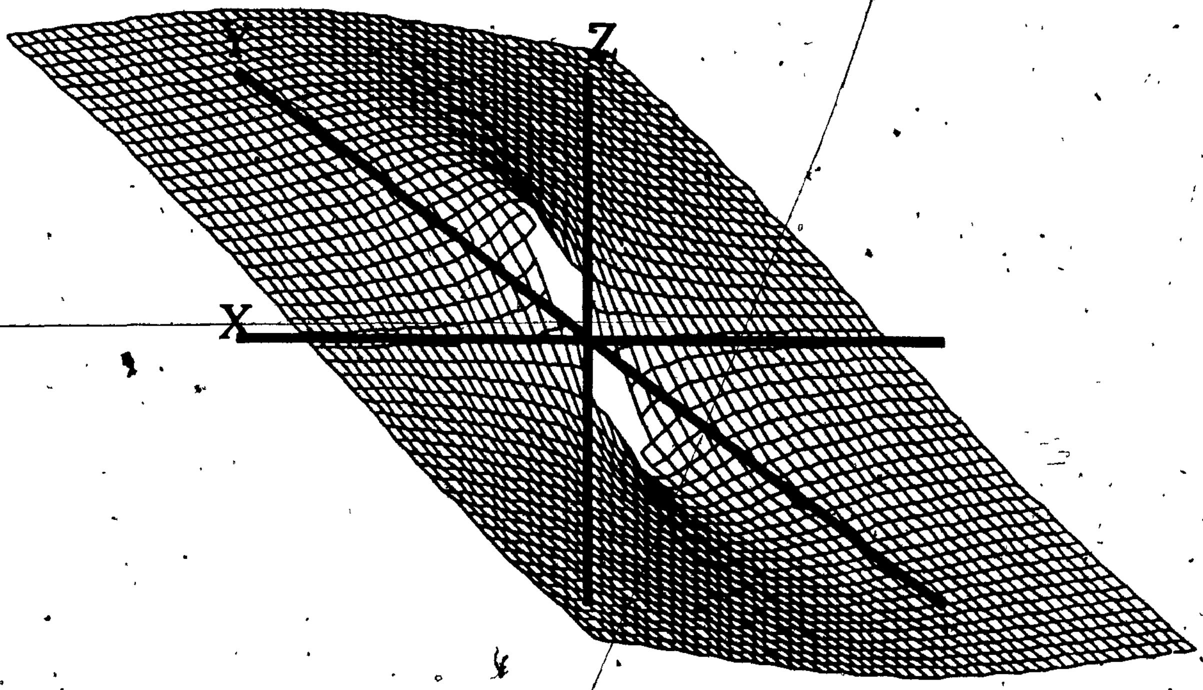
VARIANCE PLOT:
X-direction variance of XY plane 0.0 for motor 2.
tile size = 10.0 scaling factor = 250.0
FIG-32



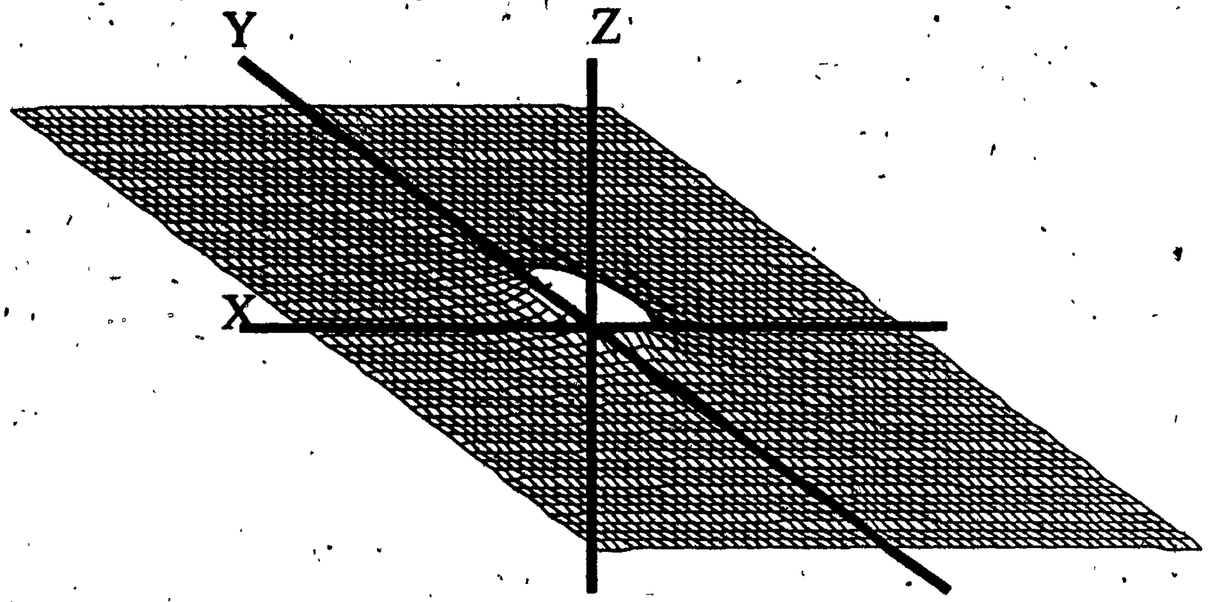
VARIANCE PLOT:
X-direction variance of XY plane 10.0 for motor 2.
tile size = 2.0 scaling factor = 1000.0
FIG-33



COEFFICIENT PLOT:
X-coefficient in XY plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
FIG-34



COEFFICIENT PLOT:
Y-coefficient in XY plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
FIG-35



COEFFICIENT PLOT:
Z-coefficient in XY plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
FIG-36

FIG-31, **FIG-32** and **FIG-33** are all xy-plane variance plots for motor $\mathbf{m_0}$ with varied tile sizes and $\mathbf{z}$ values. Note the scaling factors in all of these plots. As the tile size is increased, the variance increases and the scaling factors must be reduced, but they are still quite large. Also note that even though the variance increases with larger tile sizes, the overall shape of the plot is similar.

6.5 COEFFICIENTS

A set of plots has been generated that represent one of the three components of a coefficient for a motor over a plane of tiles. These plots help to show how the components of the coefficients vary across the tiled space since it is the individual components that are of the most importance in determining the number of pulses to be sent to a motor. These plots were generated by computing the coefficient at the center of the tile and plotting the component that was of interest as in **FIG-34, FIG-35, FIG-36**.

6.6 DESIGN PARAMETERS.

In summary, there are several design parameters that are of importance in implementing the system.

1: Tile Size:

The tile size is of importance since it directly governs the quality of the coefficients that are to be used. If the tile size is large, the current position may be far from the center of the tile where the coefficient represents the actual value. This means that the coefficient may no longer be a good approximation to the actual values at the current position which may make the manipulator's behavior unpredictably. Then, if the tile size is too small, there will be an unnecessarily large storage overhead generated due to the large number of tiles in the manipulator work space. Therefore, the tile size should be chosen so that it is the largest possible value without disrupting the normal movement of the manipulator.

2: Frame Size:

The frame size is important to the performance of the system also. If it is too large, the system will try too move too far in one frame which will result in the manipulator not moving towards the target at the end of the frame. This is due to the nature of the tiling that the algorithm makes use of which is the fact that the initial direction of the movement will be in the general direction of the target. Since most of the joints are rotational, they are non-linear. This means that the movement that has been computed is actually the tangent to the actual movement that a joint will make.

3: Tile Positioning:

The selection of a tile indexing function is important as was discussed previously. The tiling should not be centered over the origin unless it can be guaranteed that that tile will always stay within the unreachable area. Secondly, if a tiling system similar to ours is chosen, then the indexing must not start at zero or multiple tile zeroes will be created. We recommend that the tile indexing start at 1 so that there is no conflict in indices and that the tiles meet at the origin and not be centered over it. Centering the tile over the origin also creates problems in that it covers all quadrants of the coordinate system and the coefficients are likely to be different in each one.

4: Tile vs. Frame Size:

There is some inter-dependence of tile size and frame size. The frame size should be chosen so that an average move will carry the manipulator into an adjacent tile. If it is too small, then there may be too many recomputations made within the same tile which is unnecessary.

The other thing of importance in the system are the exception conditions. These mainly occur at the boundary of the unreachable space.

1: Attempted move into unreachable space:

When the system tries to move into the unreachable space one or more joints will encounter limit switches. This fact is of importance when computing a scaling factor based on the frame size to reduce the maximum number of pulses to the frame size or less. A motor that is against a limit must be excluded from this operation if the motor will stay against the limit after this frame but included if it will be able to move in the indicated direction. This will allow the manipulator to navigate around the unreachable area in some instances.

2: Partially Unreachable Tile:

Some tiles lie on the unreachable boundary. These present no real problem to the system when the center of the tile is in the reachable region but if it is unreachable, the coefficients for the tile must be computed at some other location. The location that we have used in these instances is the corner of the tile that is closest to the xy-plane and furthest from the origin. This position was chosen since it is easy to locate and it is guaranteed to be in the reachable portion of a tile on the boundary.

3. ZERO FACTOR AND VARIANCE SPIKES:

The **ZERO** factor in the simulation is used to determine if floating point results are close enough to zero to be considered a zero. Its primary purpose is to determine if the arm has reached its destination by comparing the displacements on the **x**, **y** and **z** axis from the destination of the current position to **ZERO** to see if the destination has been reached within the tolerance of the system.

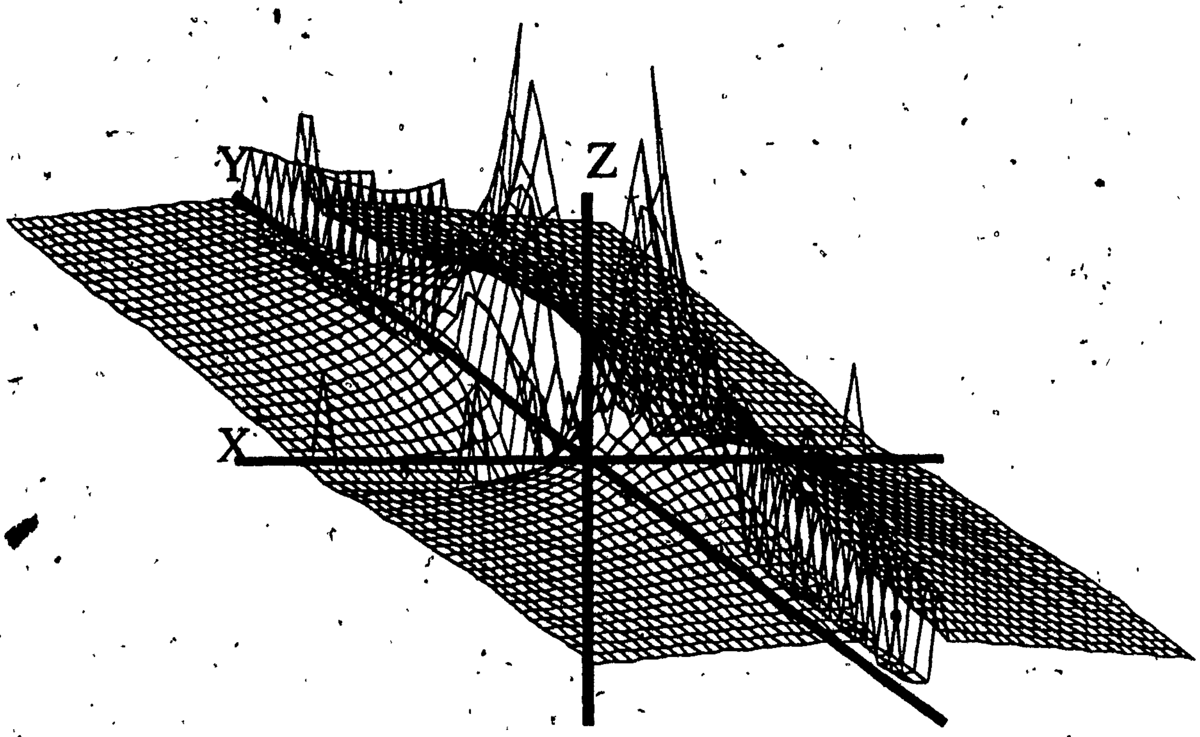
In earlier versions of the simulation, the zero factor was used in determining if a component of a coefficient for a motor was small enough to be called zero. This was done to prevent divide by zero problems due to the lack of precision in some of the calculations and also to make the arm behave properly in areas near the axis. In these areas, the precision of the computations resulted in small coefficients for some motors that should have been zero which meant that the system would try to use these motors in movements where they could not contribute. **ZERO** was then introduced to take care of these inaccuracies because the final pulses that were computed to be sent to the motors in these versions were not scaled to the frame size as they are now but just truncated to the frame size. This meant that **ZERO** could be used to prevent inaccuracies from giving unwanted pulses.

ZERO is no longer used for these purposes on the **SUN**. The greater accuracy in computations and the scaling of the pulses to be sent to the motors and the correction of some small programming errors has removed the need for it. However, if **ZERO** is used in this manner, it does not seem to have any detrimental effect on the operation of the arm but from the error plots, there seems there is the potential for problems.

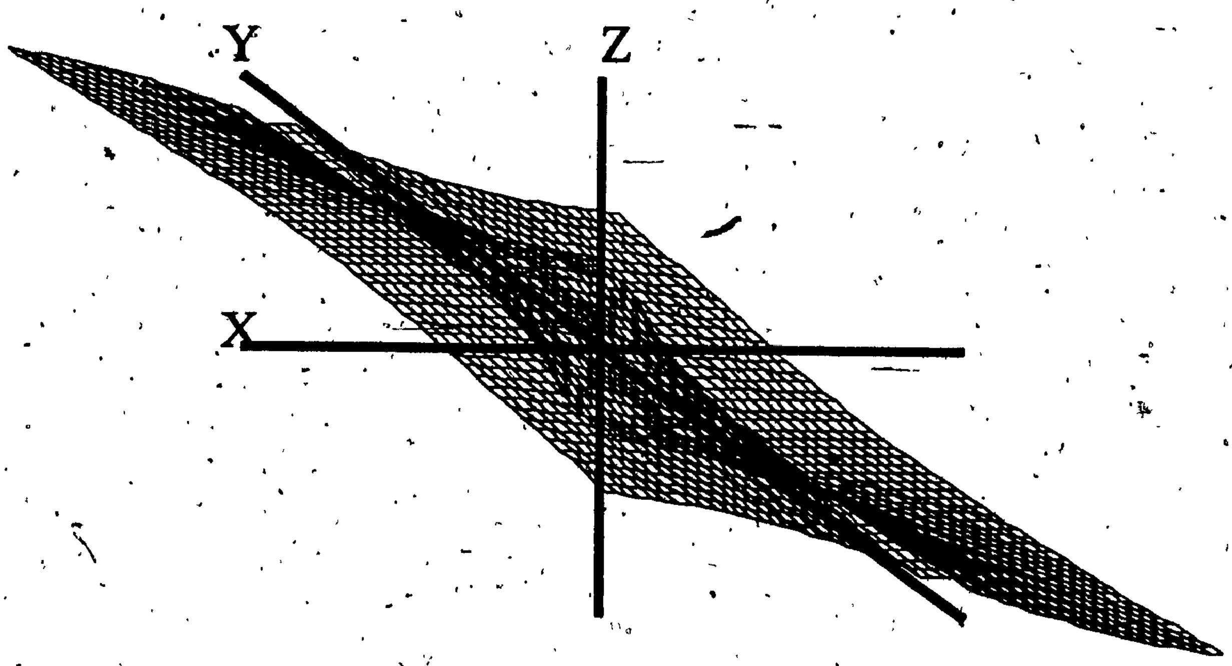
FIG-37 is a coefficient variance plot in the x direction for motor m_2, the extender, in the xy plane with z equal to zero. **ZERO** here has been set to 0.02 and it causes the introduction of the spikes along the x-axis and the deep trench that runs parallel to the y-axis. If the **ZERO** is set much smaller, say to 0.0001, this effect disappears and the plots become smooth surfaces in these areas.

This effect can be traced to the fact that as the coefficients in a particular direction become less than **ZERO**, a break is created in the plot for that component of the coefficient if **ZERO** is too large. This can be seen in the plots of the x and y coefficients (**FIG-38**, **FIG-39**) for motor two that correspond to the plot above. In the area where the spikes occur in the first plot, there are corresponding breaks in the coefficient plots:

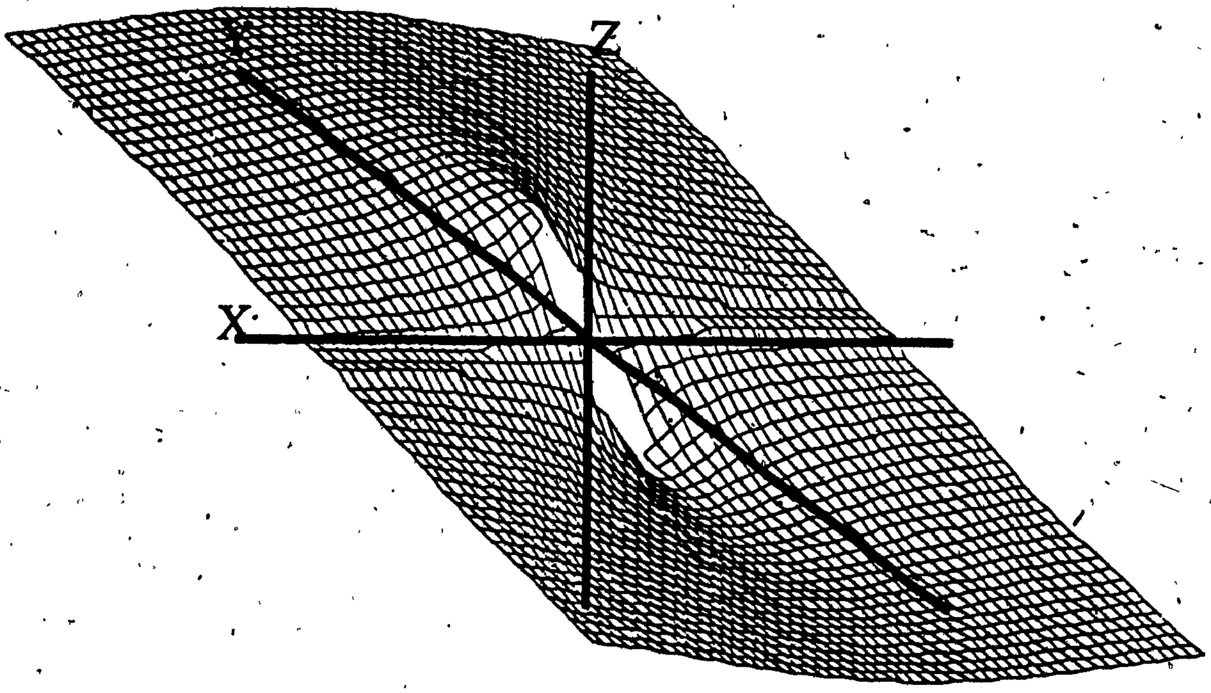
So, if **ZERO** is used for accuracy purposes, it must be very small so that it doesn't create breaks in the coefficients.



VARIANCE PLOT:
X-direction variance of XY plane 0.0 for motor 2.
tile size = 2.0 scaling factor = 1000.0
FIG-37



COEFFICIENT PLOT:
X-coefficient in XY plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
FIG-38



COEFFICIENT PLOT:
Y-coefficient in XY plane 0.0 for motor 2:
tile size = 2.0 scaling factor = 100.0
FIG-39

CHAPTER 7: DISCUSSION.

In this chapter we discuss the significance of our methodology in terms of computational and storage overhead. We discuss the point of the design, which is to reduce the computational overhead involved in manipulator control. This has been accomplished at the expense, of course, of storage overhead.

7.1 COMPUTATION OVERHEAD:

The computational overhead has been determined for for both methods of specifying the orientation of the target and the wrist. Also, the overhead for the forward and reverse kinematics, which is the conventional method of controlling the arm, has been determined in **chapter 2.**

The conventional method of point to point movement of the arm is to evaluate the joint values at the current position and at the target position then take the difference. An additional four multiplications is required to convert the joint differences into pulses for the motors (ignoring the gripper rotation motor since it was not implemented in the simulation). So the computation required by all of the methods for a moving target will be:

[1] = γ_x/γ_z METHOD

[2] = VECTOR NORMAL METHOD

DIFFERENCES ARE [2] - [1].

MOVING TARGET(per frame):

METHOD	ADD	MULT	TRIG	INVERSE TRIG	SQUARE ROOT
Inv. kin.[1]	30	62	12	10	4
Inv. kin.[2]	36	78	4	12	4
Tiled [1]	19	21	4	0	0
Tiled [2]	20	21	0	0	0

DIFFERENCES(between two methods of orientation):

METHOD	ADD	MULT	TRIG	INVERSE TRIG	SQUARE ROOT
Inv. kin.	6	16	-8	2	0
Tiled	1	0	-4	0	0

For the stationary target case, the computation using the inverse kinematics is the computation required for one computation of the inverse kinematics plus this amount for each frame. Also there is an extra expense of 4 additions per frame because of the need to subtract from the target positions of the joints.

STATIONARY TARGET (per frame):

(n = number of frames)

METHOD	ADD	MULT	TRIG	INVERSE TRIG	SQUARE ROOT
Inv. kin.[1]	13+17n	29+33n	6+6n	5+5n	2+2n
Inv. kin.[2]	16+20n	37+41n	2+2n	6+6n	2+2n
Tiled [1]	3+16n	4+17n	4	0	0
Tiled [2]	3+17n	3+18n	0	0	0

DIFFERENCES(between two methods of orientation):

METHOD	ADD	MULT	TRIG	INVERSE TRIG	SQUARE ROOT
Inv. kin.	3+3n	8+8n	-4-4n	1+1n	0
Tiled	1n	-1+1n	-4	0	0

The difference between the conventional method and the tiled method for the two orientation methods are presented in the following table:

DIFFERENCES(tiled - conventional):

MOVING TARGET:

METHOD	ADD	MULT	TRIG	INVERSE TRIG	SQUARE ROOT
[1]	-11	-41	-8	-10	-4
[2]	-16	-57	-12	-12	-4

The tiled method has a large advantage in computation over the conventional method of using the inverse kinematics of the arm. An even greater advantage of the tiled system using the unit vector orientation method is that all of the trigonometric functions have been eliminated leaving only additions and multiplications to be performed. Also, the expense over the method of using orientation angles is only a single addition being traded for four trigonometric functions. The stationary method is similar with the difference being a saving initially of one multiplication but one extra multiplication and addition per frame is required to replace the four trigonometric functions.

7.2 STORAGE OVERHEAD:

The amount of storage that is required in the system to hold the coefficients for all of the tiles that will be reached by the arm is dependent on the number of tiles that are in the system. The number of tiles in the system is a function of the tile sizing as well as the dimensions of the reachable space for which coefficients must be stored. Assuming that the reachable space of the arm is a rectangular cube, since this is the case where the most tiles will exist, the width of the space in the **x** direction in terms of numbers of tiles is the reachable length of the x-axis, $|\mathbf{x}|$, divided by the tile sizing, **S**. Similarly, in the other two directions, the number of tiles is the reachable length of the axis over the tile sizing. The total number of tiles that are in the system is then:

$$\mathbf{T = |x| * |y| * |z| / S^3}$$

Since there are nine values stored for each tile (3 per motor), the total number of values to be stored is **9T**. Since a floating point number requires four bytes of storage, the total amount of storage required will be **36T**. Therefore, as an example, in the workspace examined for our manipulator system with all three axis 100 units long and a tile size of 5 units, would have a total of 8,000 tiles in the work space. The storage for this would be 288,000 bytes in the above format.

By scaling the coefficients, they could be made into integers and thus reduce the number of bytes of storage that is required. This would add to the computational overhead of the system because there would be the need to reverse the scaling process when they were to be used in the system. Also, if the coefficients are stored in inverted form, the values may vary too greatly to fit into the two bytes of an integer without reducing the resolution of the system. If this method was used in the above example, the resulting storage requirements would only be 144,000 bytes which is not much in terms of memory requirements with our present technology.

A second alternative is to leave the coefficients sitting on a secondary storage unit, such as a disk drive, where the size of the storage required is not a problem. With the coefficients sitting on secondary storage, they could be swapped in blocks into main memory when the system need them. By organizing the blocks so they contained a three dimensional cube of tiles, when a block was swapped into memory it would most likely contain several of the next tiles that were to be visited by the arm. Also, by keeping several of these blocks in memory at a time, much as an operating system uses paging, the tiles that were to be accessed next could always be in memory for use by the time the arm reached them by using a suitable "paging" algorithm for swapping them.

7.3 "PAGING" ALGORITHM:

The paging algorithm could be quite simple. By making the block sizes reasonable enough to allow the system to have eight blocks resident in memory at a given time, the next tile that was to be entered (assuming that the frame size is reasonable) would always be in memory. The swapper would look at the current tiles relationship to the edge of the current block of tiles. If the current tile is deep within the block (towards the center), no action need be taken. If the current tile is near one face of the block but towards the center of the face, the block that shares that face should be loaded. The block that is to be swapped out (overwritten) could be the block that is the furthest away from the current tile or, probably more practically, a version of **LRU** could be imposed to get rid of the oldest block in memory.

If the current tile is near the edge of a face but also in the middle of the face, then the three other blocks that share the edge should be loaded. The final case is when the current tile is near the corner of a block. In this case, all of the blocks (this is where the eight blocks comes from) that share the corner should be loaded.

The overhead for this type of arrangement would be very small since the pager would only have to be invoked when the arm passed from one tile to another. By using the tile indexes that are used in the system to access the tiles, the algorithm could be implemented using only some comparison logic to decide what action should be taken.

7.4 REDUCTION OF STORAGE:

It may be possible to reduce the amount of storage that is required for the coefficients by introducing separate tiling for each of the motors. This would allow the use of large tiles for some motors where the coefficients were more stable and smaller tiles for the other motors. Also, it may be possible to have tiles that are not cubic in shape to take advantage of coefficients that change little in one direction but not in another. Then there is the possibility of using tiles that are not square but some other shape such as wedge shaped. If wedge shaped tiles were used for motor m_0, which rotates about the base of the arm, then the tiles could be infinitely long if the coefficients could be scaled by some value before they were used since the change in the coefficients is proportional to the distance from the base of the arm on a line from the base.

Another possibility is to use all integer arithmetic in the system. This could be accomplished by scaling the coefficients so they become integer sized values. It would then be necessary to scale all of the other information used in the algorithm but the cost of this would be small since the computational overhead for integer arithmetic is much smaller than that of floating point.

CHAPTER 8: CONCLUSIONS.

What we have introduced in this work is a new method for controling a robot arm using visual feedback with very low computational overhead. The essence of the concept is to store rather than compute coefficients of the arm movement on a cubic grid of possible locations of the arm. We have demonstrated by a simulation that arm control using this method can be well behaved, and we have investigated the different parameters that can assure this fact.

We have found that the two most important factors in the success of the method are the frame and tile size. The best relationship between frame and tile size appears to be one where an average movement of the arm will move only as far as an adjacent tile during a frame. The exact relationship that will achieve this is difficult to obtain due to the many nonlinear components in the system.

We have presented in **chapter 5** the guidelines for the design of a tiled system. The storage overhead that will be required by the system is a function of the size of the tiling and the workspace. Although the storage overhead may be high, it can be kept to a minimum by coarse tiling since the manipulator is still well behaved for fairly large tile sizes. We have proposed methods that could be used to reduce this overhead such as coefficient storage as integers, paging, and computing only the coefficients that are in the normal work are of the manipulators workspace.

The coefficients can be computed and stored from standard methods of solving the forward and inverse kinematics as has been done in our simulation. As an alternative to this type of computation that requires exact details of the manipulator to provide accurate results, we propose a method where by the system is self-calibrating. Under self-calibration, the system would position (presumably by trial and error or manually by an operator) the arm at the center of each tile and use the information obtained from visual feedback to obtain the coefficients. This would be the preferred method since it would eliminate many of the inaccuracies that would be present in the computational method where no allowance is made for sources of error such as slackness in the mechanical mechanism of the manipulator or nonlinearity in the vision system. The draw back of this method of calibration is that it would be a very time consuming process. There may be the possibility though of using this method to build up the coefficients for the area in which the manipulator works by calibrating and storing the coefficients for a tile when the arm first enters it during its normal operations. This would have the advantage that only the tiles that would be computed and stored and in the process of calibrating the tiles, the manipulator could be doing useful work.

By utilizing storage and visual feedback rather than computation, we have reduced the computation but introduced the necessity of recomputing the trajectory as we proceed towards the target. This recomputation is a function of the frame size that has been selected. The smaller the frame size the more computations to reach the target. This is not a problem where adaptivity is a requirement since the trajectory must continuously be recomputed anyway and this is the intended application of this methodology.

CHAPTER 9: FUTURE RESEARCH.

In the future, there are several areas where research is possible using this control methodology. The first, and most obvious, is the application of this method to the control of a physical arm and vision system. Once this has been accomplished, there is the subject of the self-calibration of the system which could be explored in greater detail as well as the application of the methodology to other areas of automated control. Possible targets for the application of this method include systems for welding irregular seams and automated guided vehicles (AGV).

There is also the possibility of experimenting with irregular or non-cubic tile shapes that could be used in the reduction of the storage overhead that is required by the method.

REFERENCES:

[1] Beni,G.; Hackwood,S.; "Structuring High-Precision Tasks for Intelligent Robots", AT&T Bell Laboratories, Holmdel, New Jersey.

[2] Bisiani,R.; Mauersberg,H.; Reddy,R.; "Task-Oriented Architectures", Proceedings of the IEEE, Vol.71, no.7, July 1983, pp 885-898.

[3] Bozerghi,A; Goldenberg,A.A.; Apkarian,J.; "An Exact Kinematic Model of PUMA 600 Manipulator", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-14, No.3, May/June 1984; pp 483-487.

[4] Brochu,B.; Implementation of a Pick-and-Place Robot with Manipulator and Camera, M.Comp. Sc. Report, Department of Computer Science, Concordia University, August 1985.

[5] Crochetiere, W.J.; "Locating the Wrist of an Elbow–type Manipulator", IEEE Trans. Systems, Man and Cybernetics, SMC-14, 3, 497-499 (1984).

[6] Chen,Chih-Hsin; "Applications of Algebra of Rotations in Robot Kinematics", Mech. Mach. Theory, Vol.22, No.1, 1987, pp 77-83.

[7] Elgazzar, S.; "Efficient kinematic transformation for the PUMA 560 robot", IEEE Journal of Robotics and Automation, RA-1, 3 (1985).

[8] Fancott,T.; Grogono,P.; Polley,H.; "Adaptive Visual Control of a Robot Arm Using Tiled Space", Second International Conference on Robotics and Factories of the Future, ASME, San Diego, July 1987.

[9] Featherstone,R; "Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles", The International Journal of Robotics Research, Vol.2, No.2, Summer 1983; pp 35-45.

[10] Gupta,K.C.; "Kinematic Analysis of Manipulators Using the Zero Reference Position Description", The International Journal of Robotics Research, Vol.5, No.2, Summer 1986; pp 5-13.

[11] Hayward,V.; Paul,R.C.; "Robot Manipulator Control Under Unix Rccl: A Robot Control "C" Library", International Journal Of Robotics Research, Vol. 5, No. 4, Winter 1987, pp 94-111.

[12] Hollerbach,John M.; "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparitive Study of Dynamics Formulation Complexity", IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-10, no.11, November 1980, pp 730-736.

[13] Hollerbach,John M.; Sahar,Gideon; "Wrist Partioned Inverse Kinematic Accelerations and Manipulator Dynamics", The International Journal of Robotics Research, Vol.2, No.4, Winter 1983, pp 61-76.

[14] Kane,T.R.; Levinson,D.A.; "The Use of Kane's Dynamical Equations in Robotics", The International Journal of Robotics Research, Vol.2, No.3, Fall 1983, pp 3-23.

[15] Lee,C.S.George; "Robot Arm Kinematics, Dynamics, and Control", Computer, December 1982, pp 62-80.

[16] Lin,C.D.; Freudenstein,F.; "Optimization of the Workspace of a Three-Link Turning-Pair Connected Robot Arm", The International Journal of Robotics Research, Vol.5, No.2, Summer 1986; pp 104-111.

[17] Low,K.H.; Dubey,R.N.; "A Comparitive Study Of Generalized Coordinates For Solving The Inverse-Kinematics Of A R Robot Manipulator", International Journal Of Robotics Research, Vol. 5, No. 4, Winter 1987, pp 69-88.

[18] Lozano-Perez,T.; "Robot Programming", Proceedings of the IEEE, Vol.71, no.7, July 1983, pp 821-841.

[19] Luh,J.Y.S.; "Conventional Controller Design for Industrial Robots - A Tutorial", IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-13, no.3, May/June 1983, pp 298-316.

[20] Luh,J.Y.S.; Lin, C.S.; "Approximate Joint Trajectories for Control of Industrial Robots Along Cartesian Paths", IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-14, No.3, May/June 1984, pp 444-450.

[21] Luh,J.Y.S.; Waller,M.W.; Paul,R.P.C.; "On-Line Computational Scheme for Mechanical Manipulators", Transactions of the ASME Journal of Dynamic Systems, Measurement, and Control, Vol.102, June 1980, pp 69-76.

[22] Nitzon,David; "Development of Intellegent Robots: Achievements and Issues", IEEE Journal of Robotics and Automation, Vol RA-1, No.1, March 1985; pp 3-13.

[23] Paul,R.P.; Robot Manipulators: mathematics, programming, and control, Cambridge: MIT Press, 1981.

[24] Paul,R.P.; Zhang,H.; "Computationally Efficient Kinematics for Manipulators with Spherical Wrists Based on the Homogeneous Transformation Representation", The International Journal of Robotics Research, Vol.5, No.2, Summer 1986; pp 32-44.

[25] Salisbury,J.K.; Craig,J.J.; "Articulated Hands: Force Control and Kinematic Issues", The International Journal of Robotics Research, Vol.1, No.1, Spring 1982, pp 4-17.

[26] Sanderson,A.C.; Perry,G.; "Sensor-Based Robotic Assembly System: Research and Applications in Electronic Manufacturing", Proceedings of the IEEE, Vol.71, no.7, July 1983, pp 856-871.

[27] Stone,Henry W.;Neuman,Charles P.; "Dynamic Modeling of a Three Degree-of-Freedom Robotic Manipulator", IEEE Transactions on Systems, Man, ad Cybernetics, Vol SMC-14, No.4, July/August 1984, pp 643-654.

[28] Sweet, L.M.; "Sensor-base control systems for arc-welding robots", Robotics and Computer Integrated Manufacturing, 2, 2, 125-133 (1985).

[29] Taylor,R.H.; "Planning and execution of straight line manipulator trajectories", IBM Journal of Research and Development, vol 23, no.4, 424-436 (1979).

[30] Taylor,R.H.; Grossman,D.D.; "An Integrated Robot System Architecture", Proceedings of the IEEE, Vol.71, no.7, July 1983, pp 842-856.

[31] Uchiyama,Masaru; "A Study of Computer Control of Motion of a Mechanical Arm (1st Report, Singular points)", Bulletin of the JSME, Vol.22, no.173, November 1979, pp 1640-1647.

[32] Uchiyama,Masaru; "A Study of Computer Control of Motion of a Mechanical Arm (2nd Report, Control of Coordinate Motion Utilizing a Mathematical Model of the Arm)", Bulletin of the JSME, Vol.22, no.173, November 1979, pp 1648-1656.

[33] Uchiyama,Masaru; "A Study of Computer Control of Motion of a Mechanical Arm (3rd Report, Dynamic Vision and Visual Feedback)", Bulletin of the JSME, Vol.22, no.173, November 1979, pp 1657-1664.

[34] Vaishnav,R.N.; Magrab,E.B.; "A General Procedure to Evaluate Robot Positioning Errors", The International Journal of Robotics Research, Vol.6, No.1, Spring 1987; pp 59-74.

[35] Vukobratovic,M.; Kircanski,M.; "A Dynamic Approach to Nominal Trajectory Synthesis for Redundant Manipulators", IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-14, no.4, July/August 1984, pp 580-586.

[36] Whitney, D.E.; "Resolved motion rate control of manipulators and human prostheses", IEEE Transactions on Man-Machine Systems, vol 10, no.2, pp 47-53 (1969).

APPENDIX:

PLOTTING SOFTWARE:

All of the plotting software makes use of the **SUNCORE** graphics system which is an implementation of the **ACM CORE** graphics standard.

The surface plots were all generated by the same program called "*plot.c*". This program accepted data as an **(x,y,z)** triplet which was a point in 3-dimensional space. There are 2 special data values for the **x** entry that indicated that the program that an exception had occurred in the grid pattern. If the **x** value was equal to ***MAX_VAL***, then the end of a line or a break in the line in the grid has been reached. The next point is then read in and if the comparison function that has been selected returns 1, the end of the line has been reached and the grid is generated. Otherwise, the value was just signaling a hole in the grid. To draw in the grid, the program keeps a buffer of the points in the current line as well as the previous line. The grid generation function then plots the lines between the points in the two buffers with the same value on the coordinate that is being varied to generate a strip of tiles across the plane. If the x value of the next point that has already been read in is equal to ***NO_GRID***, the index to the buffer that contains the previous line is set to -1 to indicate that there was no previous line drawn corresponding to the current line. This causes the program to omit generatation of the grid lines for the next line plotted since a break has occurred between lines.

In the initialization of the program, there are 2 possible arguments that can be accepted from the command line. The first one, "*-a*", is used to indicate to the program to draw the axis on the plot. The default is no axis. The second parameter is "*-f*" which can be used to give the name of the file that the raster image is to be saved under. If no name is given, the program saves the raster as "*/tmp/plot-image.**" where * is the process id (this ensures that the name is unique).

The first argument that is read in by the program is the label. This label is printed on the bottom of the plot. Then, an integer is read in that represents the comparison and the grid generation function that should be used in creating the plot. The difference between the different grid and comparison functions is that they compare on different coordinates in making plotting decisions for the grid generation.

For these plots, the viewing parameters that have been selected create a plot with the z-axis pointing upwards, the x-axis pointing into of the screen and the y-axis pointing to the left of the screen. The projection that has been selected is a parallel projection from the point **(50.0,-50.0,50.0)**. The viewing window has been set to include the **x** range of -100 to 100 and the **y** range of -100 to 100.

Also, the plotting program contains an infinite loop at the end of the plotting. The purpose of this is to keep the process alive so the image remains on the screen for viewing.

A second plotting program has been created to plot the actual positions of the arm in three dimensions at the end of each frame during a move. The form of this plot is selected by command line arguments. The options available are "*-w*" which causes the wrist path to be plotted, "*-e*", which causes the path of the end-effector to be plotted, and "*-a*" which causes the axis to be displayed on the plot. There are two additional arguments, "*-xy*" and "*-xz*", which are used to produce plots as seen by the cameras in the vision system. The "*-xy*" parameter specifies that the projection onto the xy-plane should be plotted. The "*-xz*" selects the projection onto the xz-plane. These plots are representative of the images that the camera would see at the end of each frame of movement.

Both of the camera views can be plotted simultaneously by the use of the pair of utility programs that have been created called "*split*" and "*plug-in*". *Split* connects to a specified number of sockets to which it will duplicate its standard input. *Plug-in*, which must be set running first, creates the socket with the integer extension [illegible] the name given as its argument and reads the socket, placing the information on its standard output. By piping the output of the robot arm simulation into a "*split 2*" command and running 2 copies of the plug-in program ("*plug-in 0*" and "*plug-in 1*") with there outputs piped to this plotting program (one for each camera view), the two plots can be created at the same time.

Also, there is an optional "*-f*" argument that can be specified to give the name of the file to save the plot in. The default for the program is no axis and the plot of the arm at the end of each frame.

The viewing window in this set of plots has been set to -75 to 75 for **x** and -75 to 75 for **y** also. The orientation (except camera views) of the axis has the z-axis pointing upwards, the y axis pointing to the right of the screen and the x-axis pointing out of the screen. The projection is parallel and from the point (50,50,50).

In both of these plotting programs, the **SUNCORE** graphics system has been initialized to the ***DYNAMMICC*** level with ***NOINPUT*** and three dimensional support.

PLOT DATA GENERATION:

There are three programs that have been used to create the data for the above plotting programs, variance, frames and coeff. Variance generated data for the variance plots, frames for the error plots-based on the frame size and coeff generated the data for the coefficient plots. All of these programs accept the same set of command line parameters with the frames program accepting an additional parameter for the frame size and a different set of directional parameters. All of the parameters have default values except for the plane selection and the motor selection. The parameters and there definitions are as follows:

-xy is used to select the xy-plane for the plot and indicates that the next argument is the z-value for the selected xy-plane.

-xz selects the a vertical xz-plane and the next argument is taken as the y-value for this plane.

-yz selects the yz-plane and the next argument is the x-value for this plane.

-m selects the motor that the plot will correspond to and the next parameter will be the number of the motor. Motors are numbered from zero with the base motor being $\mathbf{m_0}$, the shoulder being $\mathbf{m_1}$ and the

extender being $\mathbf{m_2}$. No plots have been created for the motor $\mathbf{m_3}$, the wrist.

-t indicates that the next argument will be the tile size that is to be used in the plots.

-s indicates that the next argument will be the scaling factor used on the error values.

-x selects the direction of interest for the plot. In the variance program, it indicates that the variance is to be computed across the tile in the x-direction. It is not available in the frames program since it has been replaced there by the motor direction parameter. In the coefficient program, coeff, it indicates that the x-component of the coefficient should be plotted.

-y is the same as *-x* but selects the **y** values instead.

-z also is the same but selects **z** values.

-f indicates to the frames program that the next argument is the frame size.

-R is for the frames program only and selects the reverse direction for the motor movements.

-F is also for the frames program and is used to select forward motor movements.

The default values for the variance program are the x-direction with a scaling factor of 1000 and a tile size of 1. For the frames program, it is the forward direction with a scaling factor of 100, tile size of 1 and a frame size of 20. Finally, for the coeff program, it is the x coefficient with a tile size of 1 and a scaling factor of 100.

All of the programs follow the same format for generating the data to be plotted. The only real differences in them is in the actual computation of the data. All of the functions that compute which tile is to be done next etc. are all the same. The first function that they perform is to read the command line arguments and output a report of the arguments that can be placed in a file by the plotting program. This report is terminated with the ***END_LABEL*** character which indicates that to the plotting program that it has received the entire label for the plot. Then, the initialization that is performed by the simulation is also done here to initialize the arm parameters. Then a factor is computed called min_reach which is the square of the minimum radius that the arm is able to retract to (at the wrist). Then the plane generation function that has been selected by the command line argument is called. The command line argument was used to set a pointer to the appropriate function. the same was done with the direction parameter.