# MODELLING AND ANALYSIS OF SYSTEMATIC ERRORS AND THEIR EFFECTS ON POSITIONING ACCURACY OF MANIPULATORS

by

Timothy J. Hahn

Memorandum No. UCB/ERL M87/30

20 May 1987

# MODELLING AND ANALYSIS OF SYSTEMATIC ERRORS AND THEIR EFFECTS ON POSITIONING ACCURACY OF MANIPULATIONS

by

Timothy J. Hahn

Memorandum No. UCB/ERL M87/30

20 May 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

# MODELLING AND ANALYSIS OF SYSTEMATIC ERRORS AND THEIR EFFECTS ON POSITIONING ACCURACY OF MANIPULATIONS

by

Timothy J. Hahn

Memorandum No. UCB/ERL M87/30

20 May 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

# Modelling and Analysis of Systematic Errors and their Effects on Positioning Accuracy of Manipulators

*Timothy J. Hahn* *

Department of Electrical Engineering and Computer Science
Electronics Research Laboratory
University of California, Berkeley CA 94720

*ABSTRACT*

The evaluation of uncertainties in robotic manipulators is important to several areas of robotics. Of special interest is the estimation of the actual kinematic parameters of a manipulator. In this paper, geometric and non-geometric error sources are considered. These error sources can be further subdivided into systematic and non-systematic types. The systematic error sources of a manipulator are those which can be estimated before the manipulator is built. Systematic error sources are modelled, and from these models, bounds on these error sources are determined for a given positioning accuracy. These bounds are determined for two general types of manipulators.

May 20, 1987

* Research supported in part by NSF under grant DMC-8451129

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, in partial satisfaction of the requirements for the degree of Master of Science, Plan II.

# Table of Contents

# Modelling and Analysis of Systematic Errors and their Effects on Positioning Accuracy of Manipulators

*Timothy J. Hahn* *

Department of Electrical Engineering and Computer Science
Electronics Research Laboratory
University of California, Berkeley CA 94720

## 1. Introduction.

Of growing interest to the robotics community today is the representation and evaluation of uncertainties in a robotic workcell. Estimation of these uncertainties is important to three aspects of robotics. In offline programming systems, estimation of uncertainty is necessary in order to evaluate the feasibility of a task. Also, estimation of these uncertainties can result, through better design of fixtures in the environment or use of sensors, in an improvement in the reliability of an action. A recent paper addressing these topics is Brooks [1] in which tasks are broken up into smaller sub-tasks called plans. These plans are then evaluated, changed to include sensing operations if necessary, and a resulting plan is developed. Unfeasible tasks are rejected if increased sensing does not guarantee completion. The estimation of uncertainties for use in offline programming systems is also addressed, in a different fashion, by Foulloy and Kelley [6]. In this scheme, local calibration of an environment is used to improve the reliability of actions performed in that local region.

Another aspect of robotics related to the representation and evaluation of uncertainties online is that which is concerned with online fusion of sensor data. In the estimation of an object's location, a certain uncertainty persists via the uncertainties in sensor data

* Research supported in part by NSF under grant DMC-8451129

coordinate transformations by approximate transformations using random variables as inputs.

In addition to the above areas of robotics, still another area concerned with the design of robotics systems requires the evaluation and representation of uncertainties in robotic workcells. These uncertainties must be accounted for in estimating the performance of the workcell, and in improving it. A special topic in this area of robotics is concerned with finding the actual kinematic parameters of a manipulator. With a precise estimation of these parameters, improved positioning accuracy can be achieved since the kinematic model of the manipulator will more closely resemble the actual manipulator.

In this paper, possible error sources and their effects on manipulator positioning accuracy are considered. These errors stem from errors in the actual kinematic parameters of the manipulator and in the effects of actual motors used in each joint of the manipulator. In addition, the effects of machining tolerances on each of the links of a manipulator is developed. As a result, bounds on machining and joint tolerances can be found for a desired positioning accuracy.

Section 2 of this paper gives a literature review of papers in the area of error sources in a manipulator and their estimation. Both the estimation of the actual kinematic parameters of the manipulator (identification of arm signature) and effects of joint motors are considered. In section 3, error sources and modelling of these is developed. Section 4 deals with the representation of these error sources and how machining and motor tolerances affect positioning accuracy. In section 5, the kinematics and thus the kinematic parameters susceptible to errors of the Intelledex 605T and Adept manipulators are formulated. Also, required machining and motor tolerances for positioning accuracy in several nominal positions for these manipulators is given. Section 6 gives conclusions.

## 2. Past Literature.

Several papers, [5][7][8][14][15][16], have attempted to estimate the actual kinematic parameters of the manipulator and thus determine a manipulator's arm signature. Each of the papers discusses possible error sources in manipulators. In all papers, errors in the links of the manipulator are considered to contribute to the positioning accuracy. In addition to these error sources, errors in the joint motors of the manipulator such as motor compliance and gear backlash are considered.

There are three main topics to be considered in evaluating and estimating the actual kinematic parameters of a manipulator. These are the choice of the parameters to be used, how errors in these parameters relate to positioning accuracy, and how experiments are conducted to estimate these parameters.

Some general remarks on kinematic parameter selection have been noted by Everett et. al. in [5]. In this paper, the notions of completeness, proportionality, and equivalence of kinematic parameters are developed. Completeness refers to the minimal number of parameters needed to completely specify the kinematics of a manipulator. It was shown that four parameters are needed for each revolute joint in the manipulator, two parameters are needed for each prismatic joint, and six parameters are needed to specify the zero configuration of the manipulator. Added to these is the number of joint inputs. Proportionality of these kinematic parameters refers to the quality that small changes in position and orientation correspond to small changes in these parameters. Finally, equivalence is defined as the property that two kinematic parameter models which are complete should be related by some equivalence relation. The modified set of kinematic parameters satisfies the conditions of completeness and proportionality.

In choosing the parameters to model the manipulator, a number of different schemes have been proposed. First, in two papers by Stone and Sanderson [14][15] a set of six parameters termed S parameters defining each link of a manipulator are developed and used to represent coordinate transformations between links in the manipulator. These S

model parameters resemble a more common set of parameters, the Denavit-Hartenberg parameters, which also accomplish the same function. The S model parameters are chosen for a number of reasons. First, small changes in position and orientation are modelled as small changes in these parameters. Second, the S model gives a flexibility in assigning coordinate frames to each link of a manipulator, and third, the Denavit-Hartenberg parameters can be derived from the S model parameters.

In Judd and Knasinski [8] and Veitschegger and Wu [16], a modified set of the Denavit-Hartenberg parameters of a manipulator is used. This modified set of parameters, suggested in [7] by Hayati, uses three, four, or five parameters to define each link. This number is dependent upon the type of joints and their ideal relation.

As with the S parameter representation, it is very desirable that these small positioning errors correspond to small errors in the defining parameters of the manipulator. In this way, the effects of these parameters can be approximated by a linear relation.

In the case of nearly parallel revolute or prismatic joints, small errors in position do not correspond to small errors in the Denavit-Hartenberg parameters defining the manipulator. For this reason, an additional rotation is introduced into the definition of transformations between coordinate frames attached to each link of the manipulator [7]. In this way, the Denavit Hartenberg parameters are modified giving the desirable qualities mentioned in the previous paragraph.

These modified Denavit-Hartenberg parameters represent a well known formulation of the kinematics of a manipulator. In addition, error sources due to machining and motor tolerances can be related to these parameters. For these reasons, the modified Denavit-Hartenberg parameters will be used for analysis in this paper. A detailed review of these parameters will be given in the section 4.

The second topic in evaluating and estimating the actual kinematic parameters of a manipulator is establishing the relation between positioning accuracy and errors in the kinematic parameters.

In Stone and Sanderson [14][15], tests are performed which model the actual S model transformation between each joint of the manipulator. This transformation is modelled by two transformations: one using the nominal S model parameters followed by an additional transformation representing S parameter errors. From this additional transformation, the errors in the S model parameters are determined. These parameters are ultimately related back to the Denavit-Hartenberg parameters of the manipulator. A more direct approach to parameter estimation uses the modified Denavit-Hartenberg parameters to define the manipulator.

In [8], Judd and Knasinski model errors in the modified Denavit-Hartenberg parameters of the manipulator, errors due to gear train inaccuracy between motor and joint encoder, and errors due to gravitational effects of manipulator position. For the modified Denavit-Hartenberg parameters and gear train effects, only linear effects of errors are considered and least square estimation techniques are employed to estimate them. For errors due to gravitational loading, only one joint was considered, and its effect was estimated empirically. This effect was compensated for by changing the transformation relating the effects of this link.

In [8], errors in position and orientation due to the unmodelled effects above are modelled as cyclic functions of the joint angles as in the following equation

$$\begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \\ \gamma_x \\ \gamma_y \\ \gamma_z \end{bmatrix} = a_0 + A_1 \begin{bmatrix} cos(\theta_1) \\ cos(\theta_2) \\ cos(\theta_3) \\ cos(\theta_4) \\ cos(\theta_5) \\ cos(\theta_6) \end{bmatrix} + B_1 \begin{bmatrix} sin(\theta_1) \\ sin(\theta_2) \\ sin(\theta_3) \\ sin(\theta_4) \\ sin(\theta_5) \\ sin(\theta_6) \end{bmatrix} \tag{2.1}$$

The left side of Eq. (2.1) is in twist coordinate form [11]. Using test data, $a_0$, $A_1$, and $B_1$ are estimated via least squares fit techniques. In this way, positioning accuracy is related to errors in the kinematics and motors of the manipulator.

The relation between positioning accuracy and kinematic parameter errors is, in Veitschegger and Wu [16], approximated as a linear correspondence. The linearization is taken as the partial derivative matrix of the forward kinematics representing position and orientation evaluated at the nominal position and orientation of the manipulator. This linearity assumption is justified due to the choice of kinematic parameters. Veitschegger and Wu utilize the modified Denavit-Hartenberg parameters suggested by Hayati [7]. In this case, positioning accuracy is represented by a 6x1 vector which is related to differential changes in position and orientation.

The remaining topic of consideration in the estimation of the actual kinematic parameters of a manipulator is the subject of experiments. The purpose of these experiments is to collect data from two sources. First, joint parameters are read from the joint encoders. Second, the actual position and orientation of the end effector is measured using either an external sensor or special tool. In either case an external system must be used to actually collect data. It is obviously necessary that this system be much more accurate that the manipulator in question. Also, tests must be performed so that all modelled error sources will be detected. This usually corresponds to gathering test data throughout the workspace of the manipulator. Indeed, if experiments do not permit the effect of a certain error source to be seen due to the position in the workspace or other experimental conditions under which the test data was collected, then this error cannot be estimated.

In Stone and Sanderson [14][15] a special test is performed to determine the axis of rotation of each of the revolute joints of a manipulator. Each joint is considered separately. The joint is rotated through its full range of rotation with measurements made in a number of places using ultrasonic range sensors. This test is repeated individually for each joint in the manipulator. In this way, the actual S model transformation matrix can be estimated and errors in the S model parameters are determined. Performance has been shown to improve a PUMA 560 by about a factor of 5 in position and

improve orientation by about a factor of 20.

In Judd and Knasinski [8], approximately 500 "arbitrarily" chosen positions were used to estimate the kinematic parameters and errors of an Automatix AID-900 manipulator. These positions cover the workspace of the manipulator, including the range of each of the manipulator's joints. Improvements in angular offset in the joints of the manipulator are by approximately a factor of 2 for the final estimation. In this paper, compensation is performed for a seemingly endless number of error sources. A level of diminishing returns seems to have been attained, however, in that later compensations do not increase the overall positioning accuracy by very much.

Veitschegger and Wu [16] use a specially designed pointed tool to be placed in the end effector of the manipulator. This tool precisely locates a point in the workspace of the manipulator. The transformation between the end effector and this tool is assumed exact, and measurements are taken on the actual position of the tool point A number of these measurements were made across a plane of the workspace of the manipulator. This plane corresponds to the stand on which the manipulator is mounted. From these measurements, the modified set of Denavit-Hartenberg parameters is estimated. Using this set of modified Denavit-Hartenberg parameters, the greatest measured position error for a PUMA 560 was reduced from 21.7 mm to .3 mm.

In all of the papers reviewed above, improvements in positioning accuracy are attained by estimating the actual kinematic parameters of the manipulator. In the following sections, the effects of machining and motor tolerances are related to these kinematic parameters and hence to the positioning accuracy of the manipulator. Possible error sources and their modelling is now addressed.

## 3. Error Sources and Modelling.

Sources of error in manipulators are numerous. Consider a simple two link planar manipulator: errors may occur in the machining of the links, in the discretization of the joint encoder readings of the joints, and in the compliance and gear backlash in the motors. A natural separation of error sources will be used here. Errors can be thought of as either geometric or non-geometric in nature. Geometric errors correspond to errors in the links of the manipulator. Non-geometric errors include gear backlash, motor compliance, and discretization of joint encoder readings.

A separation can also be made between systematic and non-systematic errors. Systematic errors can generally be estimated with some accuracy before the manipulator is used for applications. Non-systematic errors are those errors in the manipulator which result from actual use of the manipulator in an environment.

### 3.1. Geometric Errors

Geometric errors correspond to errors in the mechanical links, i.e. offsets between the design values of the parameters of each link of the manipulator and the actual values of these parameters. These errors come about in a number of ways. First, machining tolerances give an evaluation of the maximum error which can affect the actual machined link. Tolerances on distance between joint axes and angle between joint axes are evaluations of geometric errors in the links.

In most machining situations, dimensions of parts are defined in units of mils or thousands of an inch. Accuracy of the machines in the shop directly affect the machining tolerance of the manufactured part. Typical accuracy of these machines in on the order of .5 to 1 mil. The accuracy of CNC (numerically controlled) machines is higher. Many times, however, accuracy is given proportional to the design values via some percentage.

While machining tolerances in the links of the manipulator give an evaluation of the systematic geometric errors of a manipulator, other error sources also contribute to these

geometric errors. Over-exertion or misuse of the manipulator may result in the deformation of one or more links of the manipulator. With this deformation, the manipulator no longer corresponds to its defined kinematics and thus, errors result. Also, shipping of the manipulator between factory and plant may result in a deformation in the links of the manipulator. Small deformations in the links may not be noticeable when the manipulator is installed but could become apparent when highly accurate positioning is requested.

All of these factors contribute to geometric errors in the links of the manipulator. Assuming that these situations do not occur on a regular basis, the geometric errors of the manipulator can be thought to be time-invariant. Hence, an estimation of these would be desirable in order to get a kinematic model of the manipulator which is closer to the actual manipulator in question.

In relating the effects of machining errors and misuse on the link parameters as defined above, two assumptions are made: first, machining errors in the links are assumed to be small in comparison to the ideal link parameter values. Second, since misuse and shipping problems do not (hopefully) occur on a planned basis, errors due to them occur randomly between manipulators. It is the purpose of this paper to relate the manufacture of manipulators to their positioning accuracy. For this reason, errors due to misuse and shipping will not be considered. In an actual situation, the best method to account for all geometric errors in manipulators is to perform positioning tests on the manipulator in the environment in which it will be used. As in many of the papers reviewed in section two, this test data can be used to estimate the actual geometric parameters defining the manipulator.

The two machining tolerances which affect this relation most are the tolerance in the distance between joint axes and the tolerance in the angle between these axes. These two quantities will be denoted by $r_n$ and $r_\Delta$ for nominal distance and distance tolerance respectively, and $\gamma_n$ and $\gamma_\Delta$ for nominal angle and angle tolerance respectively.

With this introduction to geometric error sources and notation for defining the systematic part of these sources, their relation can be developed. This will be done in section four. Attention is now turned to non-geometric error sources in manipulators.

### 3.2. Non-geometric Errors.

Non-geometric errors in the manipulator consist of all other error sources which may affect the positioning accuracy and repeatibility of the manipulator. Sources of non-geometric errors are even more numerous than for geometric errors. Non-geometric errors can occur in the manipulator due to motor compliance and gear backlash in the joint motors of the manipulator. Since each motor has a certain compliance, the required input torque to the motor may have to be higher than the nominal desired input torque. These effects are due to unmodelled forces which act on the manipulator but were not accounted for at path generation. These errors are a control problem related to the dynamics of the manipulator. In this paper, only static positioning errors are considered. In addition to motor compliance, gear backlash in the motors can affect the actual joint angle or length. The joint encoder is usually positioned so that the effects of gear backlash are undetectable by the encoder. As shown in figure 3.1, the joint encoder is positioned at the output of the DC motor. With the encoder placed here, any effects of gear backlash would not result in a change in the value returned by the encoder.

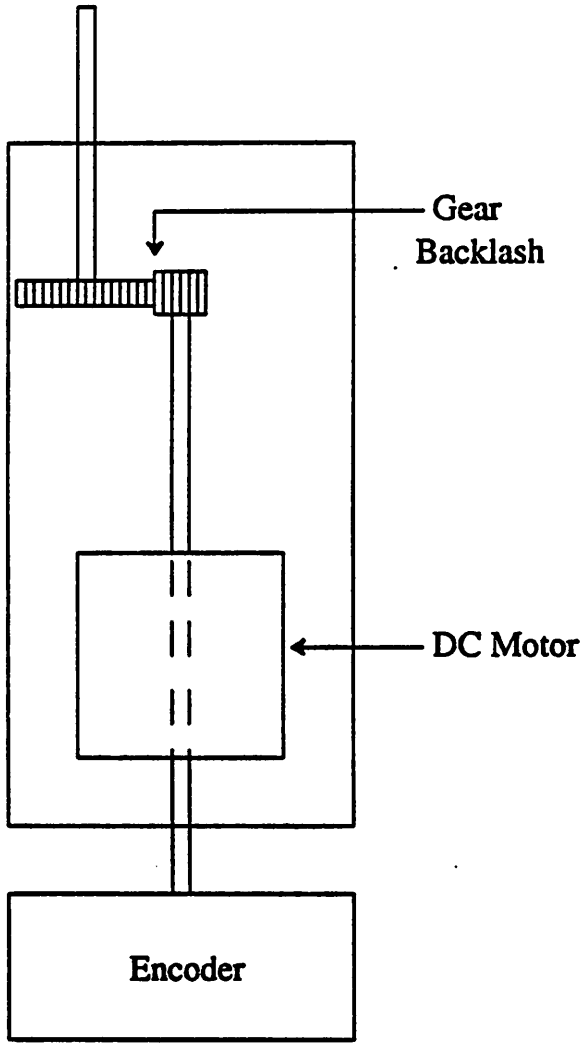


**Figure 3.1**

**Typical joint motor and encoder.**

Depending upon the configuration of the manipulator, the mass of each link of the manipulator may affect the actual joint angle or length. In addition, if the manipulator is holding an object in the end effector, and its mass is large, it too will affect the actual joint angle or length via the motor compliance and gear backlash in the motors. In general, any object of sufficient mass held in the end effector of the manipulator will affect the positioning accuracy.

In addition to these effects, the control loop for each motor may introduce some bias in the actual joint parameter obtained. The input values supplied to each motor of the manipulator are affected by the controller. While this inner, or low-level, control loop may introduce some bias into the actual joint parameters, an outer control loop which depends on the feedback signals from sensors can also introduce biases in the joint variables. These biases will depend both on the outer control loop and the accuracy of the sensors used in the feedback loop. In the computed torque method of control, the inertial effects of the configuration of the manipulator are only estimated. This estimation is done via an adaptive algorithm so that the estimated effects and the actual effects may not be the same. While the adaptive algorithm is guaranteed convergence, inside the convergence time of the algorithm, calculated input values to the joints of the manipulator will not be exactly what they should be thus resulting in an error in the joint parameter. To summarize this point, inputs are determined using only approximate models of the manipulator. In addition, these calculations are not performed in infinite precision. As a result, errors are introduced.

In all of the non-geometric error sources given thus far, the revolute joint angle or prismatic joint length is the affected joint parameter. In each case, the effect will be that the actual joint angle or length will be different than the requested input by some small amount. It should be noted, however, that this effect will, in general, be dependent upon manipulator position and mass held in the end effector.

Non-geometric errors can be divided into systematic and non-systematic types. The non-geometric error sources given above correspond to systematic error sources in a manipulator. They can be estimated before the manipulator is constructed given the estimated mass of the links and the mass in the end effector.

Non-systematic non-geometric error sources from operating conditions in the workcell may affect the manipulator's positioning accuracy. Factors such as temperature, humidity, and vibration of the environment may affect the positioning accuracy.

Indeed, if the required positioning accuracy is very high, the vibration induced by the passing of a truck outside may result in positioning errors.

With these error sources outlined, it is necessary to model these sources so that their effect can be estimated. It is the purpose of this paper to estimate the effects of those error sources which come from the design of the manipulator and can be estimated before the manipulator is built. Hence, non-systematic error sources will be ignored. In an actual situation, positioning tests should be conducted under actual operating conditions so that accurate estimates of all modelled errors will be determined.

The overall effect of these systematic non-geometric error sources is to produce some offset value in each joint parameter of the manipulator. This offset will be represented as $\theta_\Delta(m,\bar{\theta})$ for revolute joints and as $d_\Delta(m,\bar{\theta})$ for prismatic joints. It must be clarified that these values are defined for each individual joint. Hence, $m$ is the effective mass seen by the joint in question and $\bar{\theta}$ is the vector of joint parameters which corresponds to the manipulator's actual position.

With these parameters for systematic geometric and non-geometric error sources developed, their effects on the manipulator kinematics are now addressed.

## 4. Positioning Error and Modelling.

From the discussion above, error sources in manipulators can be divided into two natural categories, namely geometric error sources and non-geometric error sources. Following in a like manner, in this section modelling of geometric error sources is addressed first, and then non-geometric error sources are modelled.

### 4.1. Geometric Errors.

From the last section, it has been determined that systematic geometric error sources have origins in the machining tolerances in the manufacture of the individual links of a manipulator. The modelling of these systematic geometric error effects will be a two-

step process: first, a modified set of the Denavit-Hartenberg parameters which represents small errors in the links as small errors in these parameters will be used to represent error sources in position and orientation. Second, functions relating the machining tolerances of the links to these modified Denavit-Hartenberg parameters will be developed. In this way, machining tolerances can be related to position and orientation accuracy.

### 4.1.1. Modified Denavit-Hartenberg Parameters.

It is desirable that only the linear effects of errors in the defining parameters of a manipulator need be estimated. In order to make this assumption, small errors in position and orientation must correspond to small errors in the defining parameters of the manipulator. By far the most popular method of defining a manipulator is via its Denavit-Hartenberg parameters. For this reason, these parameters are chosen as a starting block in a set of defining parameters satisfying the above conditions.

The Denavit-Hartenberg parameters of a manipulator are a set of parameters, four for each link, which can be used to define the kinematic transformation between the base of the manipulator and the end of the manipulator. These parameters can be derived for any manipulator by proper attachment of coordinate frames to each link of the manipulator. The Denavit-Hartenberg parameters characterize the transformation between these coordinate frames. A crucial rule in using these parameters is that the origins of successive coordinate frames for parallel joints can be chosen at one's discretion along the joint axes. This is possible since the perpendicular distance between parallel joints remains constant. A problem arises, however, when it is desired that small changes in the position of a manipulator of the same structure be reflected as small changes in the defining parameters of each link of the manipulator. This can be seen by the following: if two successive joints are not exactly parallel, the origins of the coordinate frames attached to each link are fixed, no longer arbitrary. Hence, small deviations in the structure of a manipulator can result in large deviations in the defining parameters of the manipulator.

This effect is shown in figure 4.1.

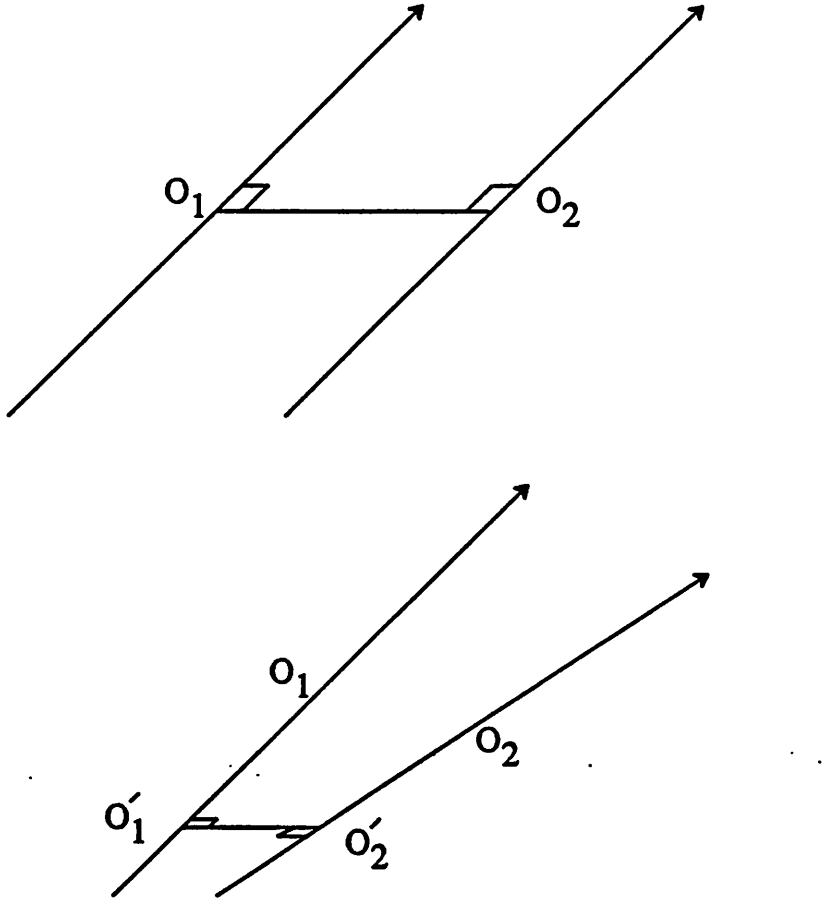


**Figure 4.1**

**Effect of non-parallel joints on defining link parameters.**

This effect has been compensated for by modifying the Denavit-Hartenberg parameters. This modified set of parameters, suggested by Hayati [7], introduces an additional rotational parameter, β, to be defined in the case of nearly parallel joints. Now, coordinate frames are attached to each link of the manipulator by the same convention as for the Denavit-Hartenberg parameters. The transformation between successive links in the manipulator takes the form:

$$ {}_{i}^{i-1}T = Rot(z_{i-1},\theta_i)Trans(0,0,d_i)Trans(a_i,0,0)Rot(x_i,\alpha)Rot(y,\beta_i) \qquad (4.1) $$

as shown by Veitschegger and Wu in [16]. This transformation is explained as follows

(see figure 4.2) the $i^{th}$ coordinate frame is first rotated about its $y_i$ axis by an angle $\beta$ to lineup the $x_i$ axis with the direction of the perpendicular line giving distance between the two joint axes (assuming that the two joint axes were parallel). This new coordinate frame is then rotated about its $x$ axis by an angle $\alpha_i$ to line up the $z_i$ axis with $z_{i-1}$. This frame is translated by $a_i$ in the $x$ direction, then by $d_i$ in the $z_{i-1}$ direction. Finally, this frame is rotated about the $z_{i-1}$ axis by an angle $\theta_i$ to lineup the $x$ axes.

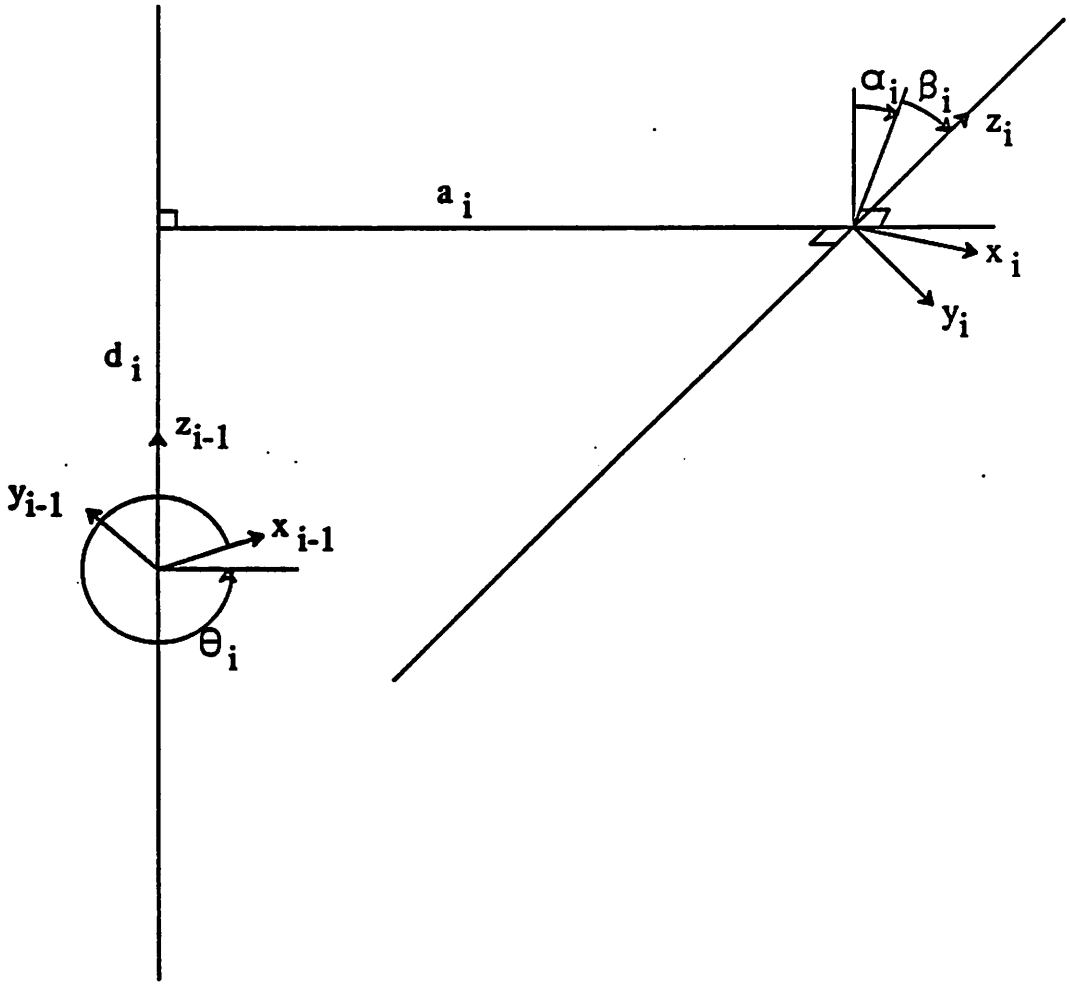


**Figure 4.2**

**Relation between coordinate frames for modified Denavit-Hartenberg parameters.**

**In general, this has the form:**

$$ {}^{i-1}_{i}T = \begin{bmatrix} (C\theta_i C\beta_i - S\theta_i S\alpha_i S\beta_i) & (-S\theta_i C\alpha_i) & (C\theta_i S\beta_i + S\theta_i S\alpha_i C\beta_i) & (a_i C\theta_i) \\ (S\theta_i C\beta_i + C\theta_i S\alpha_i S\beta_i) & (C\theta_i C\alpha_i) & (S\theta_i S\beta_i - C\theta_i S\alpha_i C\beta_i) & (a_i S\theta_i) \\ (-C\alpha_i S\beta_i) & (S\alpha_i) & (C\alpha_i C\beta_i) & (d)_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.2) $$

where $C$ signifies cosine and $S$ signifies sine. This transformation is frequently partitioned as follows

$$ {}^{i-1}_{i}T = \begin{bmatrix} {}^{i-1}_{i}A & {}^{i}\rho \\ 0 & 1 \end{bmatrix} \quad (4.3) $$

where ${}^{i-1}_{i}A$ is a 3x3 rotation matrix, and ${}^{i}\rho$ is a 3x1 position vector.

However, all of these parameters are used only in the case of parallel or nearly parallel prismatic joints. In the three other possible cases: for parallel or nearly parallel revolute joints $a_i$ is assumed zero, for non-parallel revolute joints $\beta_i$ is assumed zero, and for non-parallel prismatic joints $a_i$ and $\beta_i$ are assumed zero. Details of these results can be found in Hayati[7].

### 4.1.2. Differential notation of coordinates.

It has been developed in Paul [10] and by Paden [11] that small changes in position and orientation can be represented by a transformation matrix

$$ g = \begin{bmatrix} \omega X & b \\ 0 & 0 \end{bmatrix} \quad (4.4) $$

where $\omega X$ is a skew-symmetric rotation matrix and $b$ is a position vector. The twist coordinates of this transformation matrix, defined as a vector in $\mathbb{R}^6$, is given as follows

$$ \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = tc \begin{bmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.5) $$

Since the difference between the ideal position and orientation and the actual position and orientation is assumed to be small, this differential notation should give a good parameterization. The following relation holds

$$T_{actual} = (g + I)T_{ideal} \tag{4.6}$$

where $T$ signifies the forward kinematics of the manipulator. For the remainder of this paper, $\varepsilon_w = tc(g)$.

Assuming that the difference between the actual and ideal position and orientation is small, and that small errors in the defining parameters of the manipulator correspond to small errors in position and orientation, only the linear effects of these errors in the defining parameters need be considered. It has been shown by Hayati in [7] that if differential changes in coordinate frames are represented in twist coordinates (Paden [10]), differential changes in the the modified Denavit-Hartenberg link parameters can be related to $\varepsilon_w$ by the following:

$$\varepsilon_w = Ax \tag{4.7}$$

where $A$ is a matrix in $\mathbb{R}^{6xn}$ where $n$ is the number of required defining modified Denavit-Hartenberg parameters and $x$ is a vector in $\mathbb{R}^n$ of errors in each of these parameters. In this way, $\varepsilon_w$ represents a differential position and orientation expressed in the base or world coordinate frame of the manipulator.

As reviewed by Hayati [7] and Paul [10], these differential positions and orientations can be related between coordinate frames by the following

$$\varepsilon_i = \begin{bmatrix} {}^i_jA & ({}^i\rho X){}^i_jA \\ 0 & {}^i_jA \end{bmatrix} \varepsilon_j = {}^i_jJ\,\varepsilon_j \tag{4.8}$$

This relation requires some clarification. $\varepsilon_i$ and $\varepsilon_j$ are the twist coordinate form of a differential position and orientation in terms of coordinate frames $i$ and $j$ respectively. ${}^i_jA$ is a rotation matrix defined by the transformation between coordinate frames $i$ and $j$ as shown in Eq. (4.3). Finally, ${}^i\rho X$ is the position vector of the transformation matrix

between coordinate frames $i$ and $j$ expressed in terms of a skew symmetric matrix (see upper left 3x3 partition of 4x4 matrix in Eq.(4.5)). ${}^i\rho$ is defined as in Eq. (4.3). This operation results in a vector cross-product between ${}^i\rho$ and each of the columns of ${}^j_iA$.

With this relation between the twist coordinates of different coordinate frames, a relation is needed between the differential position and orientation in each link $i$ due to small changes in the link parameters. Since the proportionality property defined in [5] holds for the modified Denavit-Hartenberg parameters of a manipulator, only the linear effects of these parameters will be considered while higher order effects will be assumed to be much smaller and negligible. This relation will be different for each of the four possible combinations of prismatic and revolute joints. In general, this relation can be expressed as

$$\varepsilon_i = G_i x_i \tag{4.9}$$

where $x_i$ is the vector of link parameter changes for link $i$. Depending upon the number of kinematic parameters needed to specify a given link of a manipulator, $G_i$ can have dimension 6x3, 6x4, or 6x5.

In this way, the effects of small errors in each link of a manipulator are related to the positioning accuracy of the manipulator. This relation is given by the following

$$\varepsilon_w = \sum_i {}^w_i J G_i x_i \tag{4.10}$$

where ${}^w_i J$ is defined in Eq. (4.8) and $G_i$ and $x_i$ are given in Eq. (4.9) This relation can be expressed in terms of matrices as follows

$$\varepsilon_w = Ax \tag{4.11}$$

where

$$A = \left[ {}^{w}_{0}JG_1 \; {}^{w}_{1}JG_2 \; \cdots \; {}^{w}_{n}JG_n \; {}^{w}_{t}JG_t \right]$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ x_n \\ x_t \end{bmatrix} \tag{4.12}$$

and $n$ is the number of links of the manipulator and $t$ represents the tool frame of the manipulator. Thus, positioning accuracy is related to errors in the kinematic parameters of the manipulator. The analysis above is a review of the work of Hayati in [7].

Using this representation, the nonlinear relationship between link parameters and manipulator kinematics is approximated by a linear relationship with the assumption that the errors in the link parameters are small in comparison with their actual values. In this way, link parameter errors can be related to positioning errors expressed in twist coordinate form.

### 4.1.3. Machining Tolerances and Geometric Errors.

The second step in geometric error representation will now be developed. Machining errors in the links are assumed to be small in comparison to the ideal link parameter values. As developed in section three, the two machining tolerances which affect the relation between two consecutive joint axes most are the distance and angle between them.

Distance between joint axes is first addressed. This was parameterized as $r_n$ for nominal distance and $r_\Delta$ for distance tolerance. The two link parameters affected by the distance between joint axes are $a_i$ and $d_i$. Since only one link will be considered at a time, the subscript $i$ will be dropped in the interests of notation. In each of the four possible representations of the error sources due to the link parameter errors at least one of

these two link parameters is specified. Now,

$$(r_n + r_\Delta)^2 = (a_n + a_\Delta)^2 + (d_n + d_\Delta)^2 \tag{4.13}$$

In general, both $a$ and $d$ are affected by this machining tolerance and the range of $a_\Delta$ and $d_\Delta$ is shown graphically in the following figure.

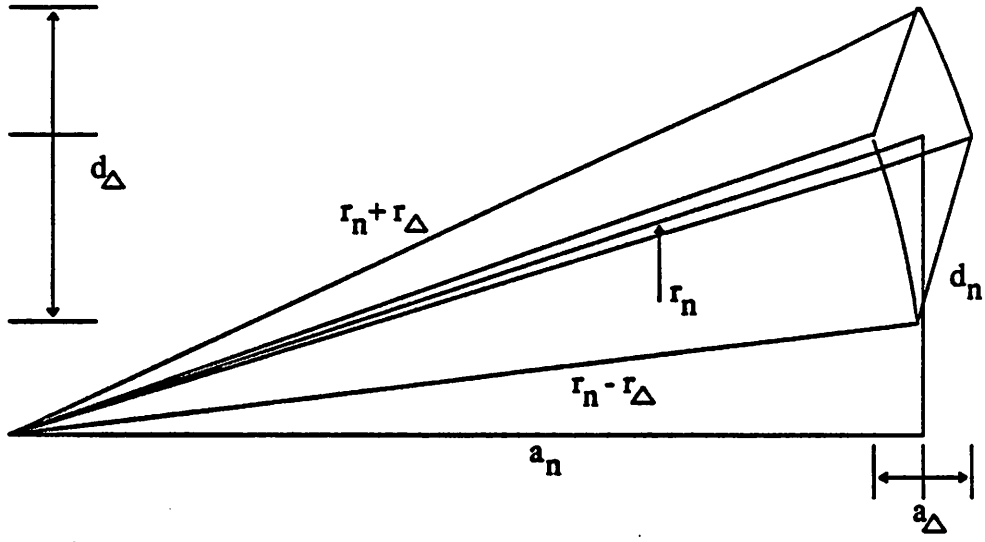


**Figure 4.3**

**Range of possible effects of machining distance tolerance on link parameters.**

It can be shown by straightforward calculation that the range of possible errors for some special cases is as follows: when $d_\Delta = 0$

$$a_\Delta = -a_n + \sqrt{a_n^2 + 2r_n r_\Delta + r_\Delta^2} \tag{4.14}$$

and when $a_\Delta = 0$

$$d_\Delta = -d_n + \sqrt{d_n^2 + 2r_n r_\Delta + r_\Delta^2} \tag{4.15}$$

and the special case that $a_\Delta = d_\Delta$

$$a_\Delta = d_\Delta = -(a_n + d_n) + \sqrt{(a_n + d_n)^2 + 2(2r_n r_\Delta + r_\Delta^2)} \tag{4.16}$$

In general, $a_\Delta$ and $d_\Delta$ satisfy the Eq. (4.13).

For the four possible cases of link representation given above, both $a_\Delta$ and $d_\Delta$ are included in the case of non-parallel revolute joints and parallel or nearly parallel prismatic joints. For non-parallel prismatic joints only $d_\Delta$ is defined, and for parallel or nearly parallel revolute joints, only $a_\Delta$ is defined. The above equations relate machining distance tolerances to link parameter errors.

The other possible error source in machining is the angle between the joint axes and its machining tolerance given by $\gamma_n$ and $\gamma_\Delta$ respectively. This machining tolerance will affect only the link parameters which define the angle between successive joint axes. These parameters are $\alpha_i$ and $\beta_i$. Again, in the interests of notation, the subscript $i$ will be dropped. Also, by the definition, $\gamma_n$ corresponds to $\alpha$. If only the value of the angle tolerance is given, no assumption on the direction of this angle's effect on the joint axis in question can be made. Thus, the possible actual joint axes lie inside a cone as shown by the following figure. In all possible cases of joint pairing, a machining angle tolerance affects only the angles $\alpha$ and $\beta$.

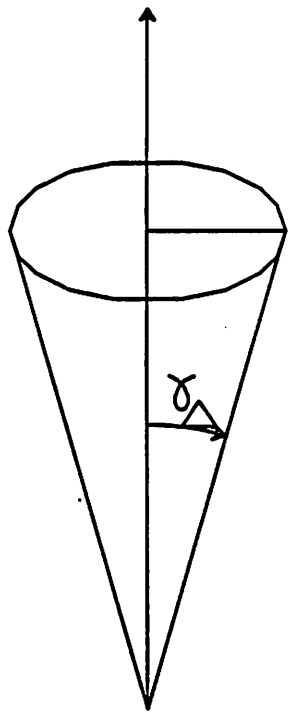


**Figure 4.4**

**Cone of possible joint axes for given machining angle tolerance.**

These effects are as follows: the maximum value of each link parameter angle error will be the machining angle tolerance and the minimum will be zero. This is can be seen by considering the range of possible angle errors as the edge of the cone is traversed (see figure 4.4). Also, a link parameter angle can achieve its maximum error only when the other link parameter error is zero.

If it is assumed that the machining angle tolerance affects both link parameter angles equally, the range of possible link parameter error is

$$\alpha_\Delta = \beta_\Delta \in \left[-tan^{-1}(\frac{1}{\sqrt{2}}tan(\gamma_\Delta)), tan^{-1}(\frac{1}{\sqrt{2}}tan(\gamma_\Delta))\right] \tag{4.17}$$

In general, however, this is not the case, and the actual values of the link parameter errors due to the machining angle tolerance is given by

$$\alpha_\Delta \in \left[-tan^{-1}(cos(\phi)tan(\gamma_\Delta)), tan^{-1}(cos(\phi)tan(\gamma_\Delta))\right] \tag{4.18}$$

$$\beta_\Delta \in \left[-tan^{-1}(sin(\phi)tan(\gamma_\Delta)), tan^{-1}(sin(\phi)tan(\gamma_\Delta))\right] \tag{4.19}$$

where $\phi$ is the angle between the nominal direction of $\alpha_n$ and the direction of the machining angle error $\gamma_\Delta$ (see figure 4.5).

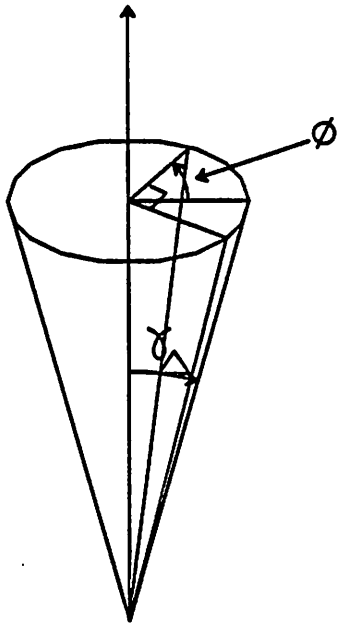


**Figure 4.5**

**Relation between $\gamma_\Delta$ and link defining parameters.**

## 4.2. Non-geometric Errors.

Non-geometric error sources and their effects on the defining link parameters will now be addressed. From the previous section, non-geometric error sources include compliance and gear backlash in the individual joint motors, the effects of the internal and external control loops on the joint input angle or length, and the effects of a mass held in the end effector of the manipulator. These systematic non-geometric error sources will be parameterized.

As discussed in the previous section, the total effect of all of these error sources results in an offset in the actual joint value by some small value. This relation will be

represented as $\theta_n + \theta_\Delta(m,\bar{\theta})$ for revolute joints and $d_n + d_\Delta(m,\bar{\theta})$ for prismatic joints where $\bar{\theta}$ is the vector of all the joint input variables and $m$ is the effective mass for each joint of the manipulator. The effective mass is given by the center of mass of links $i-n$ and the mass of an object held in the end effector. In general, $\theta_\Delta$ will depend on the position of the end effector and the mass held in the end effector.

The effect of gear backlash in a motor is to introduce an offset between the requested and actual joint input. This offset is given by the angular gap between the gear teeth in the motor. The value of the offset is not dependent upon the effective mass seen by the joint motor, but gear backlash is introduced by the effective mass. Hence, in order to examine its effect, it is assumed that the effective mass is large enough to induce its effect on the joint angle or length.

Motor compliance is also induced by the effective mass seen by the joint in question. In the case of a revolute joint, the amount of offset is proportional to the moment induced by the effective mass of the manipulator about the joint axis. For a prismatic joint, this offset is proportional to the force induced by the effective mass in the direction of the joint axis.

The total of these two effects gives the following

$$\theta_\Delta(m,\bar{\theta}) \approx \theta_{\Delta GB} + \theta_{\Delta MC}((l \times m)_z) \quad \text{(revolute)}$$

$$d_\Delta(m,\bar{\theta}) \approx d_{\Delta GB} + d_{\Delta MC}((m)_z) \quad \text{(prismatic)} \qquad (4.20)$$

where $l$ is the vector from the joint axis to the effective center of mass and $(\,)_z$ is the joint axis direction component of the moment and force. This relation is only approximate due to the as yet unmodelled effects of errors induced by the control loop including external sensors.

The effects of the control loop and sensors is modelled as follows: the sensor value returned to the external control loop of the manipulator will normally have some nominal value coupled with a bias and randomness about this bias. Hence, on average, the error

due to sensors can be modelled as just the mean of these values, or the nominal value plus the bias term. So, another constant term should be added to the above equations to model non-geometric error effects on the manipulator.

### 4.3. Establishing Sufficient Machining and Motor Tolerances.

It is certainly interesting to see what error sources most affect the positioning accuracy of a manipulator. This type of analysis is useful in determining which kinematic type of manipulator should be used in a given situation. Another interesting analysis can be done, however, to examine the sufficient conditions on the machining and motor tolerances in order to achieve some specified level of positioning accuracy. This analysis can be performed when designing manipulators for certain tasks.

From Eq. (4.11) it was shown that, assuming only systematic errors,

$$Ax = \varepsilon_w \tag{4.21}$$

If a desired positioning accuracy is specified, hence specifying $\varepsilon_w$, conditions on the tolerances of each of the defining parameters can be determined.

This equation deserves some investigation in itself. Looking at the elements of $\varepsilon_w$ and $x$, this equation has the following form

$$A \begin{bmatrix} a_{\Delta 1} \\ \theta_{\Delta 1} \\ \cdot \\ \cdot \\ \cdot \\ a_{\Delta n} \\ \theta_{\Delta n} \end{bmatrix} = \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \\ \gamma_x \\ \gamma_y \\ \gamma_z \end{bmatrix} \tag{4.22}$$

First, the vector $x$ contains errors in both angles and lengths. It is clear that errors in the length of a link of a manipulator do not affect the orientation error. Thus, if $x$ is reorganized with $A$ reorganized accordingly, Eq. (4.22) takes on a more orderly form. Taking $\bar{r}_\Delta$ as the vector of distance errors and $\bar{\theta}_\Delta$ as the vector of angle errors, Eq. (4.22) can be

expressed as follows

$$\tilde{A}\bar{x} = \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ 0 & \tilde{A}_{22} \end{bmatrix} \begin{bmatrix} \bar{r}_\Delta \\ \bar{\theta}_\Delta \end{bmatrix} = \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \\ \gamma_x \\ \gamma_y \\ \gamma_z \end{bmatrix} \tag{4.23}$$

with $\bar{x} = [\bar{r}_\Delta^T \ \bar{\theta}_\Delta^T]^T$. With this relation, sufficient bounds on both $\bar{r}_\Delta$ values and $\bar{\theta}_\Delta$ values can be found.

In order to determine a sufficient bound on $\bar{x}$ such that the desired positioning accuracy is attainable, the following relation must hold.

$$\| \tilde{A}_j^T \bar{x} \| \leq |(\varepsilon_w)_j| \tag{4.24}$$

for the $j^{th}$ row of $\tilde{A}$. This bound on $\bar{x}$ will guarantee that its effects on position and orientation will be less than or equal to the desired accuracy. This relation will be satisfied in two steps. First, sufficient bounds on $\bar{\theta}_\Delta$ will be found under two constraints. These are

$$\| \tilde{A}_{22j}^T \bar{\theta}_\Delta \| \leq |(\varepsilon_w)_j| \tag{4.25}$$

for the last three rows of $\varepsilon_w$ and

$$\| \tilde{A}_{12j}^T \bar{\theta}_\Delta \| \leq |(\varepsilon_w)_j| - \delta < |(\varepsilon_w)_j| \tag{4.26}$$

for some $0 < \delta < |(\varepsilon_w)_j|$. This condition will, in general, produce more stringent bounds on $\bar{\theta}_\Delta$ than the constraint of Eq. (4.25). This is due to the fact that errors in position due to angle errors are amplified by the length of the link in question. Hence, errors in position will be more affected by joint angle errors than by the joint length errors.

This gives

$$\| \tilde{A}_{12j}^T \bar{\theta}_\Delta \| \leq \| \tilde{A}_{12j}^T \| \ \| \bar{\theta}_\Delta \| \tag{4.27}$$

Now, a sufficient condition on $\bar{\theta}_\Delta$ such that Eq. (4.27) holds is

$$\|\bar{\theta}_\Delta\| \le \min_{j=1,2,3}\left[\frac{|(\varepsilon_w)_{j+3}| - \delta}{\|\bar{A}^T_{12j}\|}\right] \tag{4.28}$$

This is a requirement that all the elements of $\bar{\theta}_\Delta$ stay within a sphere of given radius. Similar conditions can be found from Eq. (4.25). By taking the minimum of these two constraints, a bound on $\bar{\theta}_\Delta$ can be determined.

In order to determine a bound for $\bar{r}_\Delta$, the triangle inequality is used.

$$\|\bar{A}^T_{11j}\bar{r}_\Delta + \bar{A}^T_{12j}\bar{\theta}_\Delta\| \le \|\bar{A}^T_{11j}\bar{r}_\Delta\| + \|\bar{A}^T_{12j}\bar{\theta}_\Delta\| \tag{4.29}$$

With this relation, and the constraints on $\bar{\theta}_\Delta$ found above, constraints on $\bar{r}_\Delta$ are given by

$$\|\bar{A}^T_{11j}\bar{r}_\Delta\| \le |(\varepsilon_w)_j| - \|\bar{A}^T_{12j}\bar{\theta}_\Delta\| \le \delta \tag{4.30}$$

So that

$$\|\bar{r}_\Delta\| \le \min_{j=1,2,3}\left[\frac{|(\varepsilon_w)_j| - \|\bar{A}^T_{12j}\bar{\theta}_\Delta\|}{\|\bar{A}^T_{11j}\|}\right] \tag{4.31}$$

In this way, sufficient conditions on $x$, and hence on the link parameter error tolerances, can be found for a given positioning accuracy. It should be emphasized that these conditions are only sufficient. In order to obtain less conservative bounds, an optimization of $x$ could be performed. For instance, maximize $\|\bar{r}_\Delta\|$ and $\|\bar{\theta}_\Delta\|$ under the constraints that each element of $\bar{r}_\Delta$ and $\bar{\theta}_\Delta$ be positive and also that Eq. (4.24) is satisfied for $j = 1,2,\ldots,6$.

To relate these sufficient conditions in terms of link parameter errors to sufficient conditions on machining and motor tolerances, the same relations as developed in the first part of this section are used.

For each link in the manipulator, sufficient conditions on the defining parameters for link distance can be found from Eq. (4.31). From this, the following relation gives a sufficient condition for these distances in terms of the machining distance tolerances:

$$r_\Delta \le -r_n + \sqrt{r_n^2 + (2a_n + a_\Delta)a_\Delta + (2d_n + d_\Delta)d_\Delta} \tag{4.32}$$

For machining angle tolerances,

$$\gamma_\Delta \leq min(\alpha_\Delta, \beta_\Delta) \tag{4.33}$$

These two relations give sufficient conditions on the machining tolerances of each link of the manipulator needed in order to obtain the desired positioning accuracy.

Bounds on the motor tolerances required to obtain the desired positioning accuracy are directly related to the bounds given for the link parameter errors found by Eq. (4.28). Using these bounds on the link parameter errors sufficient bounds on the motor compliances and gear backlashes are given by

$$\theta_{\Delta GB} + \theta_{\Delta MC}((l \times m)_z) \leq \theta_\Delta \quad \text{(revolute)}$$

$$d_{\Delta GB} + d_{\Delta MC}((m)_z) \leq d_\Delta \quad \text{(prismatic)} \tag{4.34}$$

with all variables defined as in Eq. (4.20).

Thus, sufficient conditions have been derived for machining and motor tolerances in order to achieve a given positioning accuracy. By formulating this matrix $A$ for a number of nominal configurations of the two manipulators in question and supplying desired positioning accuracy, sufficient bounds can be determined for machining and motor tolerances for the manipulators. This analysis is now done for the Intelledex 605T and the Adept manipulators.

## 5. Manipulator Kinematics and Data.

### 5.1. Manipulator Kinematics.

The Denavit-Hartenberg parameters of each link of the manipulator can be determined to give the ideal manipulator kinematics. As discussed in the last section, a slightly modified version of this method proves to be better for the purposes of error representation in the links of the manipulator. Hence, the kinematics both the Intelledex 605T and the Adept will be defined using these modified Denavit-Hartenberg parameters.

By concatenating each of these link transformations, the total forward kinematics of the manipulator can be found. For the case of the Intelledex 605T, the nominal, or ideal, modified Denavit-Hartenberg parameters are given as

| Parameters for link i of Intelledex 605T | | | | | |
|---|---|---|---|---|---|
| i | $\alpha_i$ | $a_i$ | $d_i$ | $\theta_i$ | $\beta_i$ |
| 1 | -90° | 0 | 0 | $\theta_1$ | $\bar{0}$ |
| 2 | 90° | 0 | 0 | $\theta_2$ | $\bar{0}$ |
| 3 | 0 | $l_2$ | $\bar{0}$ | $\theta_3$ | 0 |
| 4 | 0 | $l_3$ | $\bar{0}$ | $\theta_4$ | 0 |
| 5 | 90° | 0 | 0 | $\theta_5$ | $\bar{0}$ |
| 6 | 0 | 0 | 0 | $\theta_6$ | $\bar{0}$ |

**Figure 5.1**

**Ideal modified parameters for Intelledex 605T.**

In the table, $\bar{0}$ signifies values which are identically zero and are not used in modelling errors in the transformation. These kinematic parameters can be shown to define the transformation between the properly defined coordinate frames for each link of the manipulator. These coordinate frames are shown in the following figure.

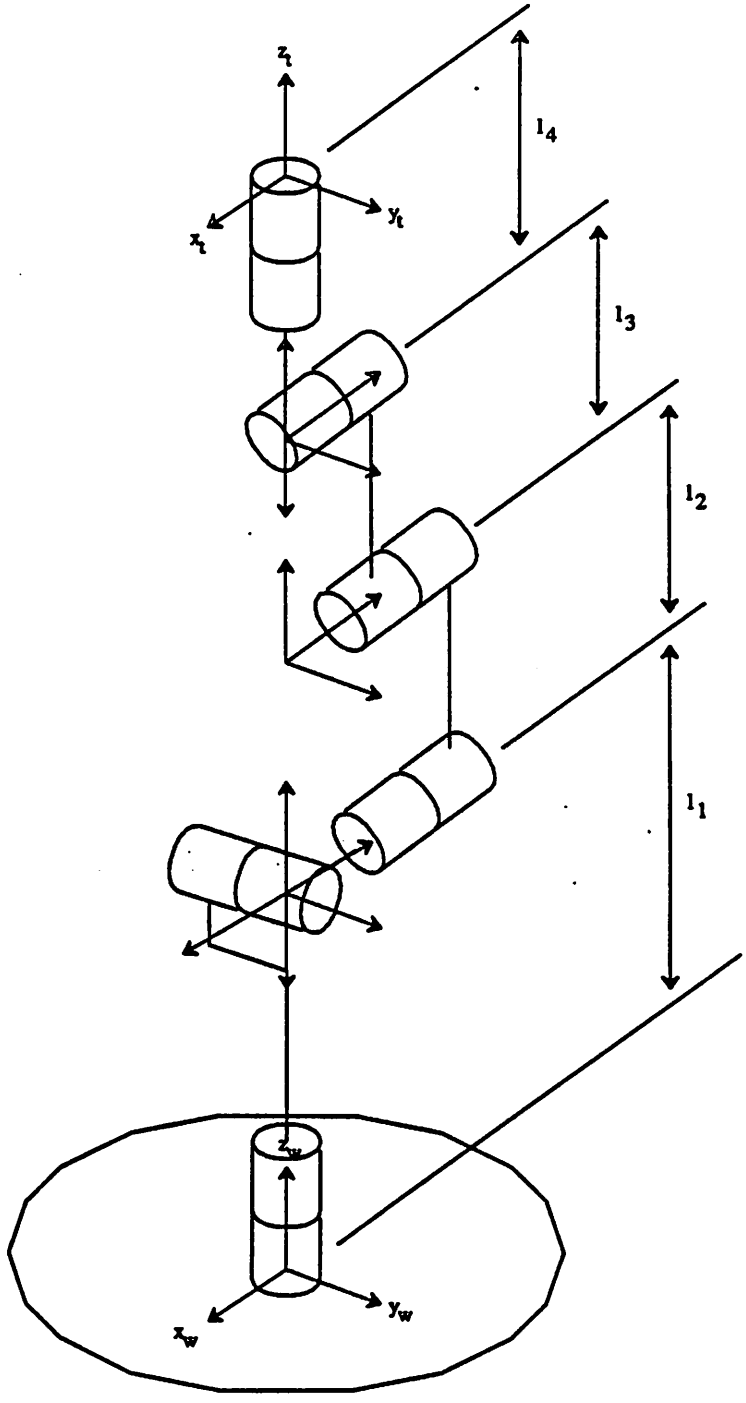


**Figure 5.2**

**Link coordinate frames for Intelledex 605T.**

In addition, two transformations are needed to relate the base or world to frame 0 and frame 6 to the tool frame. These are given by

$$ {}^{w}_{0}T = Trans(0,0,l_1) \tag{5.1} $$

$$ {}^{6}_{t}T = Rot(z_t,90^\circ)Trans(0,0,l_4) \tag{5.2} $$

Errors in these transformations are also modelled in a similar fashion. Thus, the forward kinematics of the manipulator are determined in terms of the modified Denavit-Hartenberg parameters. These are used to model geometric error effects in the manipulator.

In a likewise fashion to the above derivation, the forward kinematics of the Adept in terms of the modified Denavit-Hartenberg parameters are determined. These parameters are given by

| Parameters for link i of Adept | | | | | |
|---|---|---|---|---|---|
| i | $\alpha_i$ | $a_i$ | $d_i$ | $\theta_i$ | $\beta_i$ |
| 1 | 0 | 0 | $\bar{0}$ | $\theta_1$ | 0 |
| 2 | 0 | $l_2$ | $\bar{0}$ | $\theta_2$ | 0 |
| 3 | 0 | $l_3$ | $\bar{0}$ | $\theta_3$ | 0 |
| 4 | 0 | 0 | $d_4$ | 0 | 0 |

**Figure 5.3**

**Ideal modified parameters for Adept.**

These parameters define the coordinate transformations between each link coordinate frame. The link coordinate frames are defined in the following figure. As in the case of the Intelledex, two additional transformations are required. These are given by

$$ {}^{w}_{0}T = Rot(z_0,180^\circ)Trans(0,0,l_1) \tag{5.3} $$

$$ {}^{4}_{t}T = Rot(z_t,180^\circ)Trans(0,0,l_4) \tag{5.4} $$

Again, error sources in these transformations are modelled in a similar fashion.

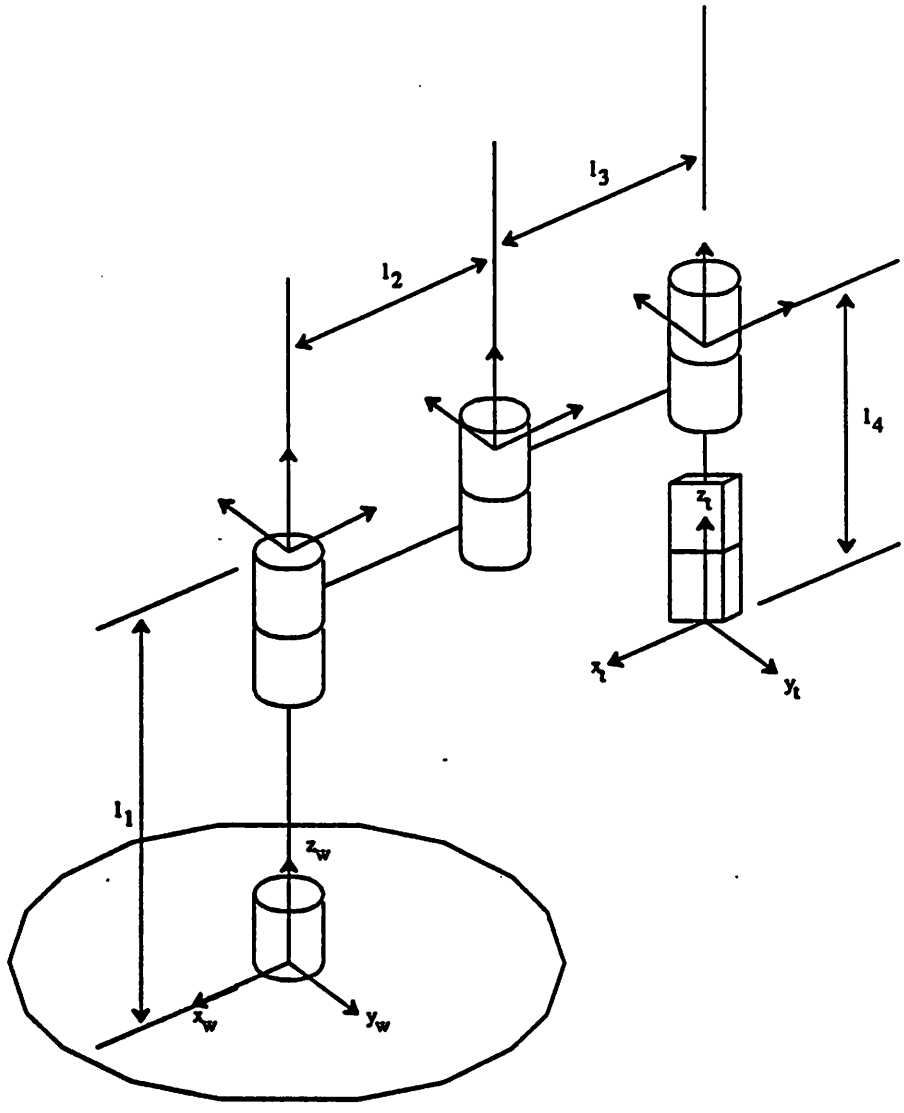


**Figure 5.4**

**Link coordinate frames for Adept.**

With the kinematics of these manipulators defined, the effects of errors in the defining link parameters can be determined.

## 5.2. Data.

In this section, data representing sufficient bounds on machining and motor tolerances is presented. This data has been found for two positions of both the Intelledex and the Adept manipulators. In order that a comparison can be made between the two manipulators, these positions were chosen as the same for both manipulators.

The first position corresponds to a place on the work surface of the manipulators and could be a desired position in pick-and-place operations. This position is described by

$$\begin{aligned} x &= 200.000000 & \theta_1 &= 1.571237 \\ y &= 200.000000 & \theta_2 &= 1.063532 \\ z &= 0.000000 & \theta_3 &= -0.944167 \\ \gamma &= 0.000000 & \theta_4 &= 1.754183 \\ \beta &= 1.570000 & \theta_5 &= -0.810926 \\ \alpha &= 0.000000 & \theta_6 &= -0.507265 \end{aligned}$$

for the Intelledex 605T where $x$, $y$, and $z$ are in units of millemeters and $\gamma$, $\beta$, and $\alpha$ are euler angles in units of radians. The corresponding position for the Adept is given by

$$\begin{aligned} x &= 200.000000 & \theta_1 &= -0.262798 \\ y &= 200.000000 & \theta_2 &= 2.429293 \\ z &= 0.000000 & \theta_3 &= -2.166495 \\ \theta &= 0.000000 & d_4 &= -203.199997 \end{aligned}$$

with $x$, $y$, and $z$ in units of millemeters and $\theta$ in units of radians.

For this position, sufficient bounds are determined for the link parameter error tolerances by the procedure described in Eqs. (4.23)-(4.31). For the Intelledex manipulator, the values of $\|\bar{\theta}_\Delta\|$ and $\|\bar{r}_\Delta\|$ are given by

$$\|\bar{\theta}_\Delta\| \leq 0.000370 \ rad$$

$$\|\bar{r}_\Delta\| \leq 0.035243 \ mm$$

For the Adept manipulator, these values are given by

$$\|\bar{\theta}_\Delta\| \leq 0.000508 \ rad$$

$$\|\bar{r}_\Delta\| \leq 0.043655 \ mm$$

These values were computed using tolerances of

$$\varepsilon_w = \begin{bmatrix} 0.500000 \\ 0.500000 \\ 0.500000 \\ 0.100000 \\ 0.100000 \\ 0.100000 \end{bmatrix}$$

and $\delta$ given by

$$\delta = 0.100000$$

Sufficient machining and motor tolerances were computed for this position of both manipulators. In order to do this, it was assumed that machining tolerances were equal for all distance measurements ($a_\Delta = d_\Delta$). For the Intelledex, it was found that sufficient machining tolerances were given by

$$r_\Delta \leq \begin{bmatrix} 0.050177 \\ 0.070956 \\ 0.070956 \\ 0.050177 \\ 0.050178 \\ 0.070956 \\ 0.070956 \\ 0.050186 \end{bmatrix} \ mm$$

with $r_\Delta$ the vector of machining distances. Also,

$$\gamma_\Delta \leq 0.004537 \ rad$$

for all links of the Intelledex manipulator. This gives sufficient bounds on the machining tolerances of the Intelledex for the specified positioning accuracy.

Sufficient bounds on the motor tolerances are the same as for the angle tolerances and are given by

$$\theta_\Delta \leq 0.004537 \ rad$$

For the same tolerances in position and orientation, sufficient bounds on machining and motor tolerances for the Adept manipulator are given by

$$r_\Delta \leq \begin{bmatrix} 0.055848 \\ 0.078971 \\ 0.055844 \\ 0.055845 \\ 0.055833 \\ 0.078971 \end{bmatrix} \; mm$$

$$\gamma_\Delta \leq 0.005313 \; rad$$

$$\theta_\Delta \leq 0.005313 \; rad$$

for the first three joints, and

$$d_\Delta \leq 0.055833 \; mm$$

for the fourth (prismatic) joint.

It is noticed that the required bounds on the motor tolerances are much higher than the bounds on the machining distance tolerance. This is due to the amplification of angle errors by the length of the link in question.

In observing the data for each manipulator, the derived bounds on distance tolerance vary for each link. This is due to the fact that as $a_n$ and $d_n$ increase, the effect of a machining tolerance becomes smaller. Thus, for different size links, the machining tolerances can be different. In comparing the two manipulators, it is noticed that the required machining and motor tolerances are much more strict for the Intelledex. This could be due to both the requested position and increased complexity of the Intelledex over the Adept.

In the same fashion, this procedure was carried out for a second position and orientation given by

| | |
|---|---|
| $x$ = -200.000000 | $\theta_1$ = 1.571408 |
| $y$ = 200.000000 | $\theta_2$ = 0.914246 |
| $z$ = 100.000000 | $\theta_3$ = 1.416886 |
| $\gamma$ = 0.000000 | $\theta_4$ = -1.666827 |
| $\beta$ = 1.570000 | $\theta_5$ = 0.248936 |
| $\alpha$ = 0.000000 | $\theta_6$ = -0.656550 |

for the Intelledex manipulator, and by

| | |
|---|---|
| $x$ = -200.000000 | $\theta_1$ = 1.307998 |
| $y$ = 200.000000 | $\theta_2$ = 2.429293 |
| $z$ = 100.000000 | $\theta_3$ = -3.737291 |
| $\theta$ = 0.000000 | $d_4$ = -103.199997 |

for the Adept.

With the same desired position and orientation tolerances, sufficient bounds on angle and distance parameters are given by

$$\|\bar{\theta}_\Delta\| \le 0.000440 \ rad$$

$$\|\bar{r}_\Delta\| \le 0.041592 \ mm$$

for the Intelledex, and for the Adept manipulator

$$\|\bar{\theta}_\Delta\| \le 0.000666 \ rad$$

$$\|\bar{r}_\Delta\| \le 0.045870 \ mm$$

As above, these parameters are used to compute sufficient bounds on the machining and motor tolerances of each manipulator. These are given by

$$r_\Delta \le \begin{bmatrix} 0.054510 \\ 0.077083 \\ 0.077083 \\ 0.054510 \\ 0.054511 \\ 0.077083 \\ 0.077083 \\ 0.054521 \end{bmatrix} mm$$

with $r_\Delta$ the vector of machining distances. Also,

$$\gamma_\Delta \le 0.004946 \ rad$$

for all links of the manipulator. This gives sufficient bounds on the machining tolerances of the Intelledex for the specified positioning accuracy.

Sufficient bounds on the motor tolerances are the same as for the angle tolerances and are given by

$$\theta_\Delta \leq 0.004946 \ rad$$

Sufficient bounds on machining and motor tolerances for the Adept manipulator are given by

$$r_\Delta \leq \begin{bmatrix} 0.057248 \\ 0.080950 \\ 0.057244 \\ 0.057245 \\ 0.057224 \\ 0.080950 \end{bmatrix} mm$$

$$\gamma_\Delta \leq 0.006081 \ rad$$

$$\theta_\Delta \leq 0.006081 \ rad$$

for the first three joints, and

$$d_\Delta \leq 0.057224 \ mm$$

Again, sufficient machining and motor tolerances for the Adept are less strict than for the Intelledex. While two positions cannot give a full picture, it seems that there is a trade off. This being between flexibility of the manipulator's possible configurations, and tolerances in parts to give positioning accuracy.

In each case, the machining tolerances are within the accuracy of machining equipment. This is encouraging in that sophisticated machining equipment is not necessary to obtain the machining accuracy needed to get the specified positioning accuracy. The motor tolerances in all four cases are very strict. This was expected, however, and shows that most problems in manipulator accuracy probably arise out of their inaccuracies. An explanation is given as follows: errors in joint angles are amplified by the length of the

links whereas errors in joint lengths are not. This results in angle errors having a much greater effect on the accuracy of the manipulator. A possible approach to compensating for this is the use of direct drive motors. In this way, gear backlash does not affect the actual angle of the joint and errors stem only from the discretization of the joint encoder readings and the control loop of the joint. With asymptotic tracking controllers, the errors in the joint angles of the manipulator can be greatly reduced. In fact, the Adept uses direct drive motors.

In this section, it has been shown that the calculation of sufficient machining motor tolerances given a desired positioning accuracy is possible and that these tolerances are in all likelihood attainable. With this information, more accurate manipulators can be manufactured.

## 6. Conclusion.

The representation and evaluation of uncertainty is important to many areas of robotics. Accounting for uncertainty is needed by offline programming systems to estimate the feasibility of tasks during the planning process. Also, estimation of uncertainties in manipulators is important to online robotic systems in order to raise reliability. In addition to these, the evaluation of uncertainties in manipulators is important in the evaluation and improvement of performance of robotic workcells.

A special topic in this last area is concerned with estimating the actual kinematic parameters of a manipulator. This paper addresses the topic of how certain error sources affect these kinematic parameters and how these parameters affect the positioning accuracy of the manipulator.

In this paper, many possible error sources are noted for manipulators. These error sources can be divided into geometric and non-geometric types. Under these two types, error sources can be categorized further as systematic or non-systematic.

In designing of manipulators, systematic error sources' effects on positioning accuracy can be estimated before the manipulator is built. This paper has given a relation by which, given a desired positioning accuracy, sufficient bounds on the systematic errors of the manipulator are determined. This relation is used to determine sufficient bounds on systematic error sources (machining tolerances and joint motor tolerances) for two types of manipulators.

In summary, this paper has given a method to determine sufficient bounds on systematic error sources of manipulators in order to attain a desired positioning accuracy.

## 7. References

[1] Brooks, Rodney A., "Symbolic Error Analysis and Robot Planning," The International Journal of Robotics Research, Winter 1982.

[2] Chen, J., and L. M. Chao, "Positioning error analysis for robot manipulators with all rotary joints," IEEE International Conference on Robotics and Automation, San Francisco, California, April 7-10, 1986.

[3] Chen, J., and Y. F. Chen, "Estimation of Coordinate Measuring Machine Error Parameters," IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March 31 - April 3, 1987.

[4] Craig, John J. Introduction to Robotics, Mechanics & Control. Addison-Wesley. Reading, Massachusetts. 1986.

[5] Everett, L. J., M. Driels, and B. W. Mooring, "Kinematic Modelling For Robot Calibration," IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March 31 - April 3, 1987.

[6] Foulloy, Laurent P. and Robert B. Kelley, "Improving the Precision of a Robot," IEEE 1st International Conference on Robotics, Atlanta, Georgia, March 13-15, 1984.

[7] Hayati, Samad A., "Robot Arm Geometric Link Parameter Estimation," IEEE Conference on Decision and Control, San Antonio, Texas, December 14-16, 1983.

[8] Judd, Robert P., and Al. B. Knasinski, "A Technique to Calibrate Industrial Robots With Experimental Verification," IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March 31 - April 3, 1987.

[9] Mooring, B. W. and T. J. Pack, "Determination and Specification of Robot Repeatibility," IEEE International Conference on Robotics and Automation, San Francisco, California, April 7-10, 1986.

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

[11] Paden, B. E., "Kinematics and Control of Robot Manipulators," Memorandum No. UCB/ERL M86/5, Electronics Research Laboratory, College of Engineering, University of California, January, 1986.

[12] Puskorius, G. V., and L. A. Feldkamp, "Global Calibration of a Robot/Vision System," IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March 31 - April 3, 1987.

[13] Smith, Randall C. and Peter Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," The International Journal of Robotics Research, Winter 1987.

[14] Stone, Henry W., Arthur C. Sanderson, and Charles P. Neuman, "Arm Signature Identification," IEEE International Conference on Robotics and Automation, San Francisco, California, April 7-10, 1986.

[15] Stone, Henry W., Arthur C. Sanderson, and Charles P. Neuman, "A Prototype Arm Signature Identification System," IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March 31 - April 3, 1987.

[16] Veitschegger, W. K., and Chi-haur Wu, "A Method for Calibrating and Compensating Robot Kinematic Errors," IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March 31 - April 3, 1987.

## 8. Appendix. Source Code.

intella.c

```
/* This program develops the linear approximating A matrix which
   relates small errors in the defining modified Denavit-Hartenberg
   parameters (from Paul [10]) to the twist coordinate representation
   of position and orientation error.
   The routines in intelledex.h use Denavit-Hartenberg parameters
   as defined in Craig [4].
   These are used to find the inverse kinematics of the manipulator.
   These two parameterizations give the same kinematic map for
   the IDEAL form of the manipulator.                          * /
/* output from this program is put into two files, intin.* which
   gives the requested input position of the manipulator, and
   inta.* which gives the A matrix transposed.                 * /
#include <stdio.h>
#include "intelledex.h"
#include "kinematic2.h"
/*modified Denavit-Hartenberg parameters of Intelledex 605T* /
#define OUTPUT      "inta.1"
#define OUTPUT2     "intat.1"
#define OUTPUT3     "inttol.1"
#define OUTPUT4     "intmach.1"
#define PI          3.14159
#define MA0         (0.)
#define MA1         (0.)
#define MA2         (0.)
#define MA3         (373.4)
#define MA4         (279.4)
#define MA5         (0.)
#define MA6         (0.)
#define MA7         (0.)
#define MALPHA0     (0.)
#define MALPHA1     (-PI /2.)
#define MALPHA2     (PI /2.)
#define MALPHA3     (0.)
#define MALPHA4     (0.)
#define MALPHA5     (PI /2.)
#define MALPHA6     (0.)
#define MALPHA7     (0.)
#define MD0         (359.7)
#define MD1         (0.)
#define MD2         (0.)
#define MD3         (0.)
#define MD4         (0.)
#define MD5         (0.)
#define MD6         (0.)
#define MD7         (100.)
#define MBETA0      (0.)
#define MBETA1      (0.)
#define MBETA2      (0.)
#define MBETA3      (0.)
#define MBETA4      (0.)
#define MBETA5      (0.)
#define MBETA6      (0.)
#define MBETA7      (0.)

main(){                                                        main

float       a[8],
            alpha[8],
            d[8],
            delta,
            tol[6],
            tolo[6],
            norm[6],
            mint,
```

```
            minr,
            ad,
            dd,
            gd,
            md,
            ggd[8],
            td[6],
            rd[8],
            x[6],
            theta[8],
            beta[8],
            itheta[6],
            tclose[6],
            check[6],
            t00[4][4],
            tw0[4][4],
            t01[4][4],
            t12[4][4],
            t23[4][4],
            t34[4][4],
            t45[4][4],
            t56[4][4],
            t6t[4][4],
            jw0[6][6],
            j01[6][6],
            j12[6][6],
            j23[6][6],
            j34[6][6],
            j45[6][6],
            j56[6][6],
            j6t[6][6],
            jw1[6][6],
            jw2[6][6],
            jw3[6][6],
            jw4[6][6],
            jw5[6][6],
            jw6[6][6],
            jwt[6][6],
            g0[6][4],
            g1[6][4],
            g2[6][4],
            g3[6][4],
            g4[6][4],
            g5[6][4],
            g6[6][4],
            gt[6][4],
            a0[6][4],
            a1[6][4],
            a2[6][4],
            a3[6][4],
            a4[6][4],
            a5[6][4],
            a6[6][4],
            at[6][4],
            aat[6][32],
            aa[6][32];
int         i,
            j,
            k;
FILE        *file,*fopen();

a[0] = MA0;
a[1] = MA1;
a[2] = MA2;
```

```
a[3] = MA3;
a[4] = MA4;
a[5] = MA5;
a[6] = MA6;
a[7] = MA7;
alpha[0] = MALPHA0;
alpha[1] = MALPHA1;
alpha[2] = MALPHA2;
alpha[3] = MALPHA3;
alpha[4] = MALPHA4;
alpha[5] = MALPHA5;
alpha[6] = MALPHA6;
alpha[7] = MALPHA7;
d[0] = MD0;
d[1] = MD1;
d[2] = MD2;
d[3] = MD3;
d[4] = MD4;
d[5] = MD5;
d[6] = MD6;
d[7] = MD7;
beta[0] = MBETA0;
beta[1] = MBETA1;
beta[2] = MBETA2;
beta[3] = MBETA3;
beta[4] = MBETA4;
beta[5] = MBETA5;
beta[6] = MBETA6;
beta[7] = MBETA7;
 /*get input thetas for desired position orientation. * /
for(i=0;i<6;i++) tclose[i] = 0.;
fprintf(stderr,"enter desired position: ");
fscanf(stdin,"%f %f %f %f %f %f",&x[0],&x[1],&x[2],&x[3],&x[4],&x[5]);
invintell(x,tclose,itheta);
for(i=0;i<6;i++) fprintf(stderr,"itheta[%d] = %f\n",i,itheta[i]);
intelledex(itheta,t00);
geulerzyx(t00,check);
for(i=0;i<6;i++) fprintf(stderr,"check[%d] = %f\n",i,check[i]);
theta[0] = 0.;
for(i=0;i<6;i++){
            theta[i+1] = itheta[i];
            }
theta[7] = PI /2.;

 /*form transformations between each coordinate frame * /
mtransf(a[0],alpha[0],d[0],theta[0],beta[0],tw0);
mtransf(a[1],alpha[1],d[1],theta[1],beta[1],t01);
mtransf(a[2],alpha[2],d[2],theta[2],beta[2],t12);
mtransf(a[3],alpha[3],d[3],theta[3],beta[3],t23);
mtransf(a[4],alpha[4],d[4],theta[4],beta[4],t34);
mtransf(a[5],alpha[5],d[5],theta[5],beta[5],t45);
mtransf(a[6],alpha[6],d[6],theta[6],beta[6],t56);
mtransf(a[7],alpha[7],d[7],theta[7],beta[7],t6t);

 /*form g transformations for each transformation* /
gnpr(a[0],alpha[0],g0);
gnpr(a[1],alpha[1],g1);
gnpr(a[2],alpha[2],g2);
gppr(alpha[3],d[3],beta[3],g3);
gppr(alpha[4],d[4],beta[4],g4);
gnpr(a[5],alpha[5],g5);
gnpr(a[6],alpha[6],g6);
gnpr(a[7],alpha[7],gt);
```

```
/*form J's */
getj(tw0,jw0);
getj(t01,j01);
getj(t12,j12);
getj(t23,j23);
getj(t34,j34);
getj(t45,j45);
getj(t56,j56);
getj(t6t,j6t);

/*form partitions of A matrix*/
mpy664(jw0,g0,a0);
mpy666(jw0,j01,jw1);
mpy664(jw1,g1,a1);
mpy666(jw1,j12,jw2);
mpy664(jw2,g2,a2);
mpy666(jw2,j23,jw3);
mpy664(jw3,g3,a3);
mpy666(jw3,j34,jw4);
mpy664(jw4,g4,a4);
mpy666(jw4,j45,jw5);
mpy664(jw5,g5,a5);
mpy666(jw5,j56,jw6);
mpy664(jw6,g6,a6);
mpy666(jw6,j6t,jwt);
mpy664(jwt,gt,at);

/*now form A*/
for(i=0;i<6;i++){
        k=0;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a0[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a1[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a2[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a3[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a4[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a5[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a6[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = at[i][j];
                }
        }
```

...main

```
/* output A matrix */
file = fopen(OUTPUT,"w");
for(i=0;i<32;i++){
        fprintf(file,"%11.6f %11.6f %11.6f %11.6f %11.6f %11.6f\n",aa[0][i], aa[1][i],aa[2][i],aa[3][i],aa[4][i],aa[5][i]);
        }
fclose(file);

/* now rearrange the columns of A to give proper form */
for(i=0;i<6;i++){
        aat[i][14] = aa[i][0];
        aat[i][6] = aa[i][1];
        aat[i][0] = aa[i][2];
        aat[i][22] = aa[i][3];
        aat[i][15] = aa[i][4];
        aat[i][7] = aa[i][5];
        aat[i][1] = aa[i][6];
        aat[i][23] = aa[i][7];
        aat[i][16] = aa[i][8];
        aat[i][8] = aa[i][9];
        aat[i][2] = aa[i][10];
        aat[i][24] = aa[i][11];
        aat[i][17] = aa[i][12];
        aat[i][9] = aa[i][13];
        aat[i][25] = aa[i][14];
        aat[i][30] = aa[i][15];
        aat[i][18] = aa[i][16];
        aat[i][10] = aa[i][17];
        aat[i][26] = aa[i][18];
        aat[i][31] = aa[i][19];
        aat[i][19] = aa[i][20];
        aat[i][11] = aa[i][21];
        aat[i][3] = aa[i][22];
        aat[i][27] = aa[i][23];
        aat[i][20] = aa[i][24];
        aat[i][12] = aa[i][25];
        aat[i][4] = aa[i][26];
        aat[i][28] = aa[i][27];
        aat[i][21] = aa[i][28];
        aat[i][13] = aa[i][29];
        aat[i][5] = aa[i][30];
        aat[i][29] = aa[i][31];
}
file = fopen(OUTPUT2,"w");
for(i=0;i<32;i++){
        fprintf(file,"%11.6f %11.6f %11.6f %11.6f %11.6f %11.6f\n",aat[0][i], aat[1][i],aat[2][i],aat[3][i],aat[4][i],aat[5][i]
        }
fclose(file);

/* now find bounds on link parameter tolerances */

fprintf(stderr,"enter desired position and orientation tolerances: ");
fscanf(stdin,"%f %f %f %f %f %f",&tol[0],&tol[1],&tol[2],&tol[3],&tol[4],&tol[5]);
fprintf(stderr,"enter desired delta: ");
fscanf(stdin,"%f",&delta);
for(i=0;i<6;i++){
        norm[i] = 0.;
        tolo[i] = tol[i];
        }
for(i=0;i<6;i++){
        for(j=14;j<32;j++){
                norm[i] += aat[i][j]*aat[i][j];
                }
        norm[i] = sqrt(norm[i]);
        }
```

```
for(i=0;i<3;i++) tol[i] = tol[i] - delta;
mint = tol[0] /norm[0];
for(i=1;i<6;i++){
        if(tol[i] /norm[i] < mint) mint = tol[i] /norm[i];
        }
for(i=0;i<6;i++) norm[i] = 0.;
for(i=0;i<3;i++){
        for(j=0;j<14;j++){
                norm[i] += aat[i][j]*aat[i][j];
                }
        norm[i] = sqrt(norm[i]);
        }
for(i=0;i<3;i++) tol[i] = delta;
minr = tol[0] /norm[0];
for(i=1;i<3;i++){
        if(tol[i] /norm[i] < minr) minr = tol[i] /norm[i];
        }
file = fopen(OUTPUT3,"w");
fprintf(file,"For the Intellidex 605T\n\n");
fprintf(file,"with input:\n");
fprintf(file,"        x = %11.6f        theta[1] = %11.6f\n",x[0],itheta[0]);
fprintf(file,"        y = %11.6f        theta[2] = %11.6f\n",x[1],itheta[1]);
fprintf(file,"        z = %11.6f        theta[3] = %11.6f\n",x[2],itheta[2]);
fprintf(file,"        g = %11.6f        theta[4] = %11.6f\n",x[3],itheta[3]);
fprintf(file,"        b = %11.6f        theta[5] = %11.6f\n",x[4],itheta[4]);
fprintf(file,"        a = %11.6f        theta[6] = %11.6f\n\n",x[5],itheta[5]);
fprintf(file,"A bound on the error in distance parameters = %11.6f\n\n",minr);
fprintf(file,"A bound on the error in angle parameters =      %11.6f\n\n",mint);
fprintf(file,"for tolerances of:\n");
fprintf(file,"        dx =   %11.6f\n",tolo[0]);
fprintf(file,"        dy =   %11.6f\n",tolo[1]);
fprintf(file,"        dz =   %11.6f\n",tolo[2]);
fprintf(file,"        dwx =  %11.6f\n",tolo[3]);
fprintf(file,"        dwx =  %11.6f\n",tolo[4]);
fprintf(file,"        dwx =  %11.6f\n\n",tolo[5]);
fprintf(file,"and\n");
fprintf(file,"     delta =  %11.6f\n",delta);
fclose(file);

 /* now relate these bounds to machining tolerances */
 /* assume ad and dd equal */
ad = dd = sqrt(minr /14.);
gd = md = sqrt(mint /18.);
for(i=0;i<8;i++){
        mach(ad,dd,a[i],d[i],&rd[i]);
        ggd[i] = gd;
        }
for(i=0;i<6;i++){
        td[i] = md;
        }
file = fopen(OUTPUT4,"w");
fprintf(file,"For the Intellidex 605T\n\n");
fprintf(file,"machining distance tolerances:\n");
for(i=0;i<8;i++){
        fprintf(file,"          rd[%1d] = %11.6f\n",i,rd[i]);
        }
fprintf(file,"\nmachining angle tolerances:\n");
for(i=0;i<8;i++){
        fprintf(file,"          ggd[%1d] = %11.6f\n",i,ggd[i]);
        }
fprintf(file,"\nmotor angle tolerances:\n");
for(i=0;i<6;i++){
        fprintf(file,"          td[%1d] = %11.6f\n",i,td[i]);
        }
```

*...main*

```
}
```

*mach*

```
/* function to relate parameter tolerances to machining tolerances */
mach(a,b,c,d,e)
float       a,
            b,
            c,
            d,
            *e;
{
float       dummy;
dummy = sqrt(c*c + d*d);
*e = -dummy + sqrt(dummy*dummy + 2.*c*a + 2.*d*b + a*a + b*b);
}
```

```
/* This program develops the linear approximating A matrix which
   relates small errors in the defining modified Denavit-Hartenberg
   parameters (from Paul [10]) to the twist coordinate representation
   of position and orientation error.
   The routines in intelledex.h use Denavit-Hartenberg parameters
   as defined in Craig [4].
   These are used to find the inverse kinematics of the manipulator.
   These two parameterizations give the same kinematic map for
   the IDEAL form of the manipulator.                    */
/* output from this program is put into two files, intin.* which
   gives the requested input position of the manipulator, and
   inta.* which gives the A matrix transposed.           */
#include <stdio.h>
#include "adept.h"
#include "kinematic2.h"
/*modified Denavit-Hartenberg parameters of Adept AdeptOne*/
#define OUTPUT      "adpta.1"
#define OUTPUT2     "adptat.1"
#define OUTPUT3     "adpttol.1"
#define OUTPUT4     "adptmach.1"
#define PI          3.14159
#define MA0         (0.)
#define MA1         (0.)
#define MA2         (425.)
#define MA3         (375.)
#define MA4         (0.)
#define MA5         (0.)
#define MALPHA0     (0.)
#define MALPHA1     (0.)
#define MALPHA2     (0.)
#define MALPHA3     (0.)
#define MALPHA4     (0.)
#define MALPHA5     (PI /2.)
#define MD0         (203.2)
#define MD1         (0.)
#define MD2         (0.)
#define MD3         (0.)
#define MD4         (0.)
#define MD5         (0.)
#define MBETA0      (0.)
#define MBETA1      (0.)
#define MBETA2      (0.)
#define MBETA3      (0.)
#define MBETA4      (0.)
#define MBETA5      (0.)

main(){                                                    main

float       a[6],
            alpha[6],
            d[6],
            delta,
            tol[6],
            tolo[6],
            norm[6],
            minr,
            mint,
            ad,
            dd,
            gd,
            md,
            td[4],
            ggd[6],
            rd[6],
```

```
                x[4],
                theta[6],
                beta[6],
                itheta[4],
                tclose[4],
                check[4],
                t00[4][4],
                tw0[4][4],
                t01[4][4],
                t12[4][4],
                t23[4][4],
                t34[4][4],
                t4t[4][4],
                jw0[6][6],
                j01[6][6],
                j12[6][6],
                j23[6][6],
                j34[6][6],
                j4t[6][6],
                jw1[6][6],
                jw2[6][6],
                jw3[6][6],
                jw4[6][6],
                jwt[6][6],
                g0[6][4],
                g1[6][4],
                g2[6][4],
                g3[6][4],
                g4[6][5],
                gt[6][4],
                a0[6][4],
                a1[6][4],
                a2[6][4],
                a3[6][4],
                a4[6][5],
                at[6][4],
                aat[6][25],
                aa[6][25];
int             i,
                j,
                k;
FILE            *file,*fopen();

a[0] = MA0;
a[1] = MA1;
a[2] = MA2;
a[3] = MA3;
a[4] = MA4;
a[5] = MA5;
alpha[0] = MALPHA0;
alpha[1] = MALPHA1;
alpha[2] = MALPHA2;
alpha[3] = MALPHA3;
alpha[4] = MALPHA4;
alpha[5] = MALPHA5;
d[0] = MD0;
d[1] = MD1;
d[2] = MD2;
d[3] = MD3;
d[4] = MD4;
d[5] = MD5;
beta[0] = MBETA0;
beta[1] = MBETA1;
beta[2] = MBETA2;
```

```
beta[3] = MBETA3;
beta[4] = MBETA4;
beta[5] = MBETA5;

/*get input thetas for desired position orientation. */
for(i=0;i<4;i++) tclose[i] = 0.;
fprintf(stderr,"enter desired position: ");
fscanf(stdin,"%f %f %f %f",&x[0],&x[1],&x[2],&x[3]);
invadept(x,tclose,itheta);
adept(itheta,t00);
check[0] = t00[0][3];
check[1] = t00[1][3];
check[2] = t00[2][3];
check[3] = atan2(t00[1][0],t00[0][0]);
for(i=0;i<4;i++) fprintf(stderr,"itheta[%d] = %f\n",i,itheta[i]);
for(i=0;i<4;i++) fprintf(stderr,"check[%d] = %f\n",i,check[i]);
theta[0] = PI;
for(i=0;i<4;i++){
            theta[i+1] = itheta[i];
            }
theta[5] = PI;
/*but joint 4 is prismatic* /
theta[4] = 0.;
d[4] = itheta[3];

/*form transformations between each coordinate frame * /
mtransf(a[0],alpha[0],d[0],theta[0],beta[0],tw0);
mtransf(a[1],alpha[1],d[1],theta[1],beta[1],t01);
mtransf(a[2],alpha[2],d[2],theta[2],beta[2],t12);
mtransf(a[3],alpha[3],d[3],theta[3],beta[3],t23);
mtransf(a[4],alpha[4],d[4],theta[4],beta[4],t34);
mtransf(a[5],alpha[5],d[5],theta[5],beta[5],t4t);

/*form g transformations for each transformation* /
gnpr(a[0],alpha[0],g0);
gppr(alpha[1],d[1],beta[1],g1);
gppr(alpha[2],d[2],beta[2],g2);
gppr(alpha[3],d[3],beta[3],g3);
gppp(a[4],alpha[4],beta[4],g4);
gnpr(a[5],alpha[5],gt);

/*form J's * /
getj(tw0,jw0);
getj(t01,j01);
getj(t12,j12);
getj(t23,j23);
getj(t34,j34);
getj(t4t,j4t);

/*form partitions of A matrix* /
mpy664(jw0,g0,a0);
mpy666(jw0,j01,jw1);
mpy664(jw1,g1,a1);
mpy666(jw1,j12,jw2);
mpy664(jw2,g2,a2);
mpy666(jw2,j23,jw3);
mpy664(jw3,g3,a3);
mpy666(jw3,j34,jw4);
mpy665(jw4,g4,a4);
mpy666(jw4,j4t,jwt);
mpy664(jwt,gt,at);
```

```
/*now form A* /
for(i=0;i<6;i++){
        k=0;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a0[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a1[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a2[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k] = a3[i][j];
                }
        k++;
        for(j=0;j<5;j++){
                aa[i][j + 4*k] = a4[i][j];
                }
        k++;
        for(j=0;j<4;j++){
                aa[i][j + 4*k + 1] = at[i][j];
                }
        }

/* output A matrix * /
file = fopen(OUTPUT,"w");
for(i=0;i<25;i++){
        fprintf(file,"%11.6f %11.6f %11.6f %11.6f %11.6f %11.6f\n",aa[0][i], aa[1][i],aa[2][i],aa[3][i],aa[4][i],aa[5][i]);
        }
fclose(file);

/* now rearrange the columns of A to give proper form * /
for(i=0;i<6;i++){
        aat[i][9] = aa[i][0];
        aat[i][3] = aa[i][1];
        aat[i][0] = aa[i][2];
        aat[i][15] = aa[i][3];
        aat[i][10] = aa[i][4];
        aat[i][4] = aa[i][5];
        aat[i][16] = aa[i][6];
        aat[i][21] = aa[i][7];
        aat[i][11] = aa[i][8];
        aat[i][5] = aa[i][9];
        aat[i][17] = aa[i][10];
        aat[i][22] = aa[i][11];
        aat[i][12] = aa[i][12];
        aat[i][6] = aa[i][13];
        aat[i][18] = aa[i][14];
        aat[i][23] = aa[i][15];
        aat[i][7] = aa[i][16];
        aat[i][13] = aa[i][17];
        aat[i][1] = aa[i][18];
        aat[i][19] = aa[i][19];
        aat[i][24] = aa[i][20];
        aat[i][14] = aa[i][21];
        aat[i][8] = aa[i][22];
        aat[i][2] = aa[i][23];
        aat[i][20] = aa[i][24];
}
file = fopen(OUTPUT2,"w");
```

```
for(i=0;i<25;i++){
        fprintf(file,"%11.6f %11.6f %11.6f %11.6f %11.6f %11.6f\n",aat[0][i], aat[1][i],aat[2][i],aat[3][i],aat[4][i],aat[5][i]
        }
fclose(file);

/* now find bounds on link parameter tolerances */

fprintf(stderr,"enter desired position and orientation tolerances: ");
fscanf(stdin,"%f %f %f %f %f %f",&tol[0],&tol[1],&tol[2],&tol[3],&tol[4],&tol[5]);
fprintf(stderr,"enter desired delta: ");
fscanf(stdin,"%f",&delta);
for(i=0;i<6;i++){
        norm[i] = 0.;
        tolo[i] = tol[i];
        }
for(i=0;i<6;i++){
        for(j=9;j<25;j++){
                norm[i] += aat[i][j]*aat[i][j];
                }
        norm[i] = sqrt(norm[i]);
        }
for(i=0;i<3;i++) tol[i] = tol[i] - delta;
mint = tol[0] /norm[0];
for(i=1;i<6;i++){
        if(tol[i] /norm[i] < mint) mint = tol[i] /norm[i];
        }
for(i=0;i<6;i++) norm[i] = 0.;
for(i=0;i<3;i++){
        for(j=0;j<9;j++){
                norm[i] += aat[i][j]*aat[i][j];
                }
        norm[i] = sqrt(norm[i]);
        }
for(i=0;i<3;i++) tol[i] = delta;
minr = tol[0] /norm[0];
for(i=1;i<3;i++){
        if(tol[i] /norm[i] < minr) minr = tol[i] /norm[i];
        }
file = fopen(OUTPUT3,"w");
fprintf(file,"For the Adept AdeptOne\n\n");
fprintf(file,"with input:\n");
fprintf(file,"        x = %11.6f        theta[1] = %11.6f\n",x[0],itheta[0]);
fprintf(file,"        y = %11.6f        theta[2] = %11.6f\n",x[1],itheta[1]);
fprintf(file,"        z = %11.6f        theta[3] = %11.6f\n",x[2],itheta[2]);
fprintf(file,"        t = %11.6f        d[4]     = %11.6f\n\n",x[3],itheta[3]);
fprintf(file,"A bound on the error in distance parameters = %11.6f\n\n",minr);
fprintf(file,"A bound on the error in angle parameters =      %11.6f\n\n",mint);
fprintf(file,"for tolerances of:\n");
fprintf(file,"        dx =   %11.6f\n",tolo[0]);
fprintf(file,"        dy =   %11.6f\n",tolo[1]);
fprintf(file,"        dz =   %11.6f\n",tolo[2]);
fprintf(file,"        dwx =  %11.6f\n",tolo[3]);
fprintf(file,"        dwx =  %11.6f\n",tolo[4]);
fprintf(file,"        dwx =  %11.6f\n\n",tolo[5]);
fprintf(file,"and\n");
fprintf(file,"    delta =  %11.6f\n",delta);
fclose(file);

/* now relate these bounds to machining tolerances */
/* assume ad and dd equal */
ad = dd = sqrt(minr /14.);
gd = md = sqrt(mint /18.);
for(i=0;i<6;i++){
        mach(ad,dd,a[i],d[i],&rd[i]);
```

*...main*

```
            ggd[i] = gd;
            }
for(i=0;i<4;i++){
            td[i] = md;
            }
td[3] = rd[4];
file = fopen(OUTPUT4,"w");
fprintf(file,"For the Adept AdeptOne\n\n");
fprintf(file,"machining distance tolerances:\n");
for(i=0;i<6;i++){
            fprintf(file,"          rd[%1d] = %11.6f\n",i,rd[i]);
            }
fprintf(file,"\nmachining angle tolerances:\n");
for(i=0;i<6;i++){
            fprintf(file,"          ggd[%1d] = %11.6f\n",i,ggd[i]);
            }
fprintf(file,"\nmotor angle tolerances:\n");
for(i=0;i<4;i++){
            fprintf(file,"          td[%1d] = %11.6f\n",i,td[i]);
            }
}

/* function to relate parameter tolerances to machining tolerances */
mach(a,b,c,d,e)                                                    mach
float       a,
            b,
            c,
            d,
            *e;
{
float       dummy;
dummy = sqrt(c*c + d*d);
*e = -dummy + sqrt(dummy*dummy + 2.*c*a + 2.*d*b + a*a + b*b);
}
```

```
#include <stdio.h>
#include <math.h>
#include "kinematic.h"
 /* these are the ideal D-H parameters of the manipulator */
 /* as defined by Craig [4] */
#define PI (3.14159)
#define A0 (0.)
#define A1 (425.)
#define A2 (375.)
#define A3 (0.)
#define ALPHA0 (0.)
#define ALPHA1 (0.)
#define ALPHA2 (0.)
#define ALPHA3 (0.)
#define D0 (203.2)
#define D1 (0.)
#define D2 (0.)
#define D3 (0.)
#define THETA4 (0.)

adept(itheta,tw4)                                                    adept
 /* performs ideal kinematics of intelledex manipulator */
 /* itheta[0-2] input joint angles 1-3, itheta[3] input joint length */
float       itheta[4],
            tw4[4][4];
{
float       a[4],
            alpha[4],
            d[4],
            dummy,
            tw0[4][4],
            t01[4][4],
            t12[4][4],
            t23[4][4],
            t34[4][4],
            tw1[4][4],
            tw2[4][4],
            tw3[4][4];
int         i,
            j;
 /* define parameters to be used */
a[0] = A0;
a[1] = A1;
a[2] = A2;
a[3] = A3;
alpha[0] = ALPHA0;
alpha[1] = ALPHA1;
alpha[2] = ALPHA2;
alpha[3] = ALPHA3;
d[0] = D1;
d[1] = D2;
d[2] = D3;
d[3] = itheta[3];
dummy = THETA4;
 /* get transformations */
for(i=0;i<4;i++){
            for(j=0;j<4;j++){
                        tw0[i][j] = 0.;
                        }
            tw0[i][i] = 1.;
            }
tw0[2][3] = D0;
transf(a[0],alpha[0],d[0],itheta[0],t01);
transf(a[1],alpha[1],-d[1],itheta[1],t12);
```

adept.h



```
transf(a[2],alpha[2],d[2],itheta[2],t23);
transf(a[3],alpha[3],d[3],dummy,t34);
 /* get forward kinematics */
mpynn(tw0,t01,tw1);
mpynn(tw1,t12,tw2);
mpynn(tw2,t23,tw3);
mpynn(tw3,t34,tw4);
}
```

```
#include <stdio.h>
#include <math.h>
#include "kinematic.h"
 /* these are the ideal D-H parameters of the manipulator * /
 /* based on D-H definitions in Craig [4] * /
#define PI (3.14159)
#define A0 (0.)
#define A1 (0.)
#define A2 (0.)
#define A3 (373.4)
#define A4 (279.4)
#define A5 (0.)
#define ALPHA0 (0.)
#define ALPHA1 (-PI /2.)
#define ALPHA2 (PI /2.)
#define ALPHA3 (0.)
#define ALPHA4 (0.)
#define ALPHA5 (PI /2.)
#define D0 (359.7)
#define D1 (0.)
#define D2 (0.)
#define D3 (0.)
#define D4 (0.)
#define D5 (0.)
#define D6 (100.)

intelledex(itheta,tw6)                                    intelledex
 /* performs ideal kinematics of intelledex manipulator * /
float       itheta[6],
            tw6[4][4];
{
float       a[6],
            alpha[6],
            d[6],
            tw0[4][4],
            t01[4][4],
            t12[4][4],
            t23[4][4],
            t34[4][4],
            t45[4][4],
            t56[4][4],
            tw1[4][4],
            tw2[4][4],
            tw3[4][4],
            tw4[4][4],
            tw5[4][4];
int         i,
            j;
 /* define parameters to be used * /
a[0] = A0;
a[1] = A1;
a[2] = A2;
a[3] = A3;
a[4] = A4;
a[5] = A5;
alpha[0] = ALPHA0;
alpha[1] = ALPHA1;
alpha[2] = ALPHA2;
alpha[3] = ALPHA3;
alpha[4] = ALPHA4;
alpha[5] = ALPHA5;
d[0] = D1;
d[1] = D2;
d[2] = D3;
```

```
d[3] = D4;
d[4] = D5;
d[5] = D6;
 /* get transformations */
for(i=0;i<4;i++){
        for(j=0;j<4;j++){
                tw0[i][j] = 0.;
                }
        tw0[i][i] = 1.;
        }
tw0[2][3] = D0;
transf(a[0],alpha[0],d[0],itheta[0],t01);
transf(a[1],alpha[1],d[1],itheta[1],t12);
transf(a[2],alpha[2],d[2],itheta[2],t23);
transf(a[3],alpha[3],-d[3],itheta[3],t34);
transf(a[4],alpha[4],-d[4],itheta[4],t45);
transf(a[5],alpha[5],d[5],itheta[5],t56);
 /* get forward kinematics */
mpynn(tw0,t01,tw1);
mpynn(tw1,t12,tw2);
mpynn(tw2,t23,tw3);
mpynn(tw3,t34,tw4);
mpynn(tw4,t45,tw5);
mpynn(tw5,t56,tw6);
}
```

```
#include <stdio.h>
#include <math.h>
#include "matrix4.h"
transf(ka,kalpha,kd,ktheta,out)                                    transf
 /* forms link transformation matrix based on Denavit-Hartenberg parameters */
 /* as defined by Craig [4] */
float           ka,
                kalpha,
                kd,
                ktheta,
                out[4][4];
{
float           n1,
                n2,
                n3,
                n4;
n1 = sin(kalpha);
n2 = cos(kalpha);
n3 = sin(ktheta);
n4 = cos(ktheta);
out[0][0] = n4;
out[0][1] = -n3;
out[0][2] = 0;
out[0][3] = ka;
out[1][0] = n3*n2;
out[1][1] = n4*n2;
out[1][2] = -n1;
out[1][3] = -n1*kd;
out[2][0] = n3*n1;
out[2][1] = n4*n1;
out[2][2] = n2;
out[2][3] = n2*kd;
out[3][0] = 0;
out[3][1] = 0;
out[3][2] = 0;
out[3][3] = 1;
}

invtrf(tin,tout)                                                   invtrf
 /* inverts transformation matrices using properties of these matrices */
float           tin[4][4],
                tout[4][4];
{
int             i,
                j;
for(i=0;i<4;i++){
                for(j=0;j<4;j++){
                                tout[i][j] = 0.;
                                }
                }
for(i=0;i<3;i++){
                for(j=0;j<3;j++){
                                tout[i][j] = tin[j][i];
                                }
                tout[3][i] = 0.;
                }
for(i=0;i<3;i++){
                for(j=0;j<3;j++){
                                tout[i][3] += (-tout[i][j])*tin[j][3];
                                }
                }
tout[3][3] = 1.;
}
```

*geulerzyx*

```
geulerzyx(t,x)
 /* extracts zyx euler angles from a transformation matrix */
 /* order is 0-5:x,y,z,gammaa,beta,alpha */
float       t[4][4],
            x[6];
{
            x[0] = t[0][3];
            x[1] = t[1][3];
            x[2] = t[2][3];
            x[3] = atan2(t[2][1],t[2][2]);
            x[4] = atan2(-t[2][0],sqrt(t[0][0]*t[0][0] + t[1][0]*t[1][0]));
            x[5] = atan2(t[1][0],t[0][0]);
}
```

*eulerzyx*

```
eulerzyx(x,t)
 /* forms transformation matrix from offsets and zyx euler angles */
 /* order of inputs is 0-5:x,y,z,gammaa,beta,alpha */
float       x[6],
            t[4][4];
{
float       cg,
            sg,
            cb,
            sb,
            ca,
            sa;
cg = cos(x[3]);
sg = sin(x[3]);
cb = cos(x[4]);
sb = sin(x[4]);
ca = cos(x[5]);
sa = sin(x[5]);
t[0][0] = ca*cb;
t[0][1] = ca*sb*sg - sa*cg;
t[0][2] = ca*sb*cg + sa*sg;
t[0][3] = x[0];
t[1][0] = sa*cb;
t[1][1] = sa*sb*sg + ca*cg;
t[1][2] = sa*sb*cg - ca*sg;
t[1][3] = x[1];
t[2][0] = -sb;
t[2][1] = cb*sg;
t[2][2] = cb*cg;
t[2][3] = x[2];
t[3][0] = 0.;
t[3][1] = 0.;
t[3][2] = 0.;
t[3][3] = 1.;
}
```

*solve*

```
solve(px,py,d,tclose,theta)
 /* solves for theta in eq. px*cos(theta) + py*sin(theta) = d */
 /* in addition, it picks theta closest to tclose */
float       px,
            py,
            d,
            tclose,
            *theta;
{
float       theta1,
            theta2,
            t1,
            sq,
            dist1,
```

kinematic.h

*...solve*

```
            dist2;
if(d == 0.)
            {
            *theta = atan2(px,(-py));
            }
else
            {
t1 = atan2(px,(-py));
sq = sqrt(py*py + px*px - d*d);
theta1 = atan2(px,(-py)) - atan2(d,sq);
theta2 = atan2(px,(-py)) - atan2(d,(-sq));
dist1 = (tclose - theta1)*(tclose - theta1);
dist2 = (tclose - theta2)*(tclose - theta2);
if(dist1 <= dist2 )
            {
            *theta = theta1;
            }
else
            {
            *theta = theta2;
            }
}
}
```

*mtransf*

```
#include <stdio.h>
#include <math.h>
mtransf(ka,kalpha,kd,ktheta,kbeta,out)
 /* forms link transformation matrix based on Denavit-Hartenberg parameters */
 /* as defined by Paul [10] */
float       ka,
            kalpha,
            kd,
            ktheta,
            kbeta,
            out[4][4];
{
float       n1,
            n2,
            n3,
            n4,
            n5,
            n6;
n1 = cos(kalpha);
n2 = sin(kalpha);
n3 = cos(ktheta);
n4 = sin(ktheta);
n5 = cos(kbeta);
n6 = sin(kbeta);
out[0][0] = n3*n5 - n4*n2*n6;
out[0][1] = -n4*n1;
out[0][2] = n3*n6 + n4*n2*n5;
out[0][3] = ka*n3;
out[1][0] = n4*n5 + n3*n2*n6;
out[1][1] = n3*n1;
out[1][2] = n4*n6 - n4*n2*n5;
out[1][3] = ka*n4;
out[2][0] = -n1*n6;
out[2][1] = n2;
out[2][2] = n1*n5;
out[2][3] = kd;
out[3][0] = 0;
out[3][1] = 0;
out[3][2] = 0;
out[3][3] = 1;
}
```

*gnpr*

```
gnpr(ka,kalpha,out)
 /* forms 6x4 matrix which relates errors in link parameters to
    twist coordinates for non-parallel revolute joints    */
float       ka,
            kalpha,
            out[6][4];
{
float       n1,
            n2;
int         i,
            j;
n1 = cos(kalpha);
n2 = sin(kalpha);
for(i=0;i<6;i++){
            for(j=0;j<4;j++){
                        out[i][j] = 0.;
                        }
            }
out[0][2] = 1.;
out[1][0] = ka*n1;
out[1][1] = n2;
out[2][0] = -ka*n2;
```

*...gnpr*

```
out[2][1] = n1;
out[3][3] = 1.;
out[4][0] = n2;
out[5][0] = n1;
}
```

*gnpp*

```
gnpp(kalpha,out)
 /* forms 6x3 matrix which relates errors in link parameters to
    twist coordinates for non-parallel prismatic joints    * /
float       kalpha,
            out[6][3];
{
float       n1,
            n2;
int         i,
            j;
n1 = cos(kalpha);
n2 = sin(kalpha);
for(i=0;i<6;i++){
            for(j=0;j<3;j++){
                        out[i][j] = 0.;
                        }
            }
out[1][1] = n2;
out[2][1] = n1;
out[3][2] = 1.;
out[4][0] = n2;
out[5][0] = n1;
}
```

*gppr*

```
gppr(kalpha,kd,kbeta,out)
 /* forms 6x4 matrix which relates errors in link parameters to
    twist coordinates for parallel or nearly parallel revolute joints * /
float       kalpha,
            kd,
            kbeta,
            out[6][4];
{
float       n1,
            n2,
            n3,
            n4;
int         i,
            j;
n1 = cos(kalpha);
n2 = sin(kalpha);
n3 = cos(kbeta);
n4 = sin(kbeta);
for(i=0;i<6;i++){
            for(j=0;j<3;j++){
                        out[i][j] = 0.;
                        }
            }
out[0][0] = -kd*n2*n4;
out[0][1] = n3;
out[1][0] = kd*n1;
out[2][0] = -kd*n2*n3;
out[2][1] = -n4;
out[3][0] = n1*n4;
out[3][2] = n3;
out[4][0] = n2;
out[4][3] = -1.;
out[5][0] = n1*n3;
out[5][2] = -n4;
```

kinematic2.h

*...gppr*

```
}
```

*gppp*

```
gppp(ka,kalpha,kbeta,out)
 /* forms 6x5 matrix which relates errors in link parameters to
    twist coordinates for parallel or nearly parallel prismatic joints * /
float           ka,
                kalpha,
                kbeta,
                out[6][5];
{
float           n1,
                n2,
                n3,
                n4;
int             i,
                j;
n1 = cos(kalpha);
n2 = sin(kalpha);
n3 = cos(kbeta);
n4 = sin(kbeta);
for(i=0;i<6;i++){
                for(j=0;j<3;j++){
                                out[i][j] = 0.;
                                }
                }
out[0][0] = n1*n4;
out[0][1] = -kalpha*n2*n4;
out[0][2] = n3;
out[1][0] = n2;
out[1][1] = kalpha*n1;
out[2][0] = n1*n3;
out[2][1] = -kalpha*n2*n3;
out[2][2] = -n4;
out[3][1] = n1*n4;
out[3][3] = n3;
out[4][1] = n2;
out[4][4] = -1.;
out[5][1] = n1*n3;
out[5][3] = -n4;
}
```

*getj*

```
getj(in,out)
 /* function to get J from T* /
float           in[4][4],
                out[6][6];
{
float           skew[3][3],
                rot[3][3],
                temp[3][3];
int             i,
                j;
for(i=0;i<6;i++){
                for(j=0;j<6;j++){
                                out[i][j] = 0;
                                }
                }
for(i=0;i<3;i++){
                for(j=0;j<3;j++){
                                rot[i][j] = in[i][j];
                                out[i][j] = rot[i][j];
                                out[3+i][3+j] = rot[i][j];
                                }
                }
skew[0][1] = -in[2][3];
```

...getj

```
skew[0][2] = in[1][3];
skew[1][0] = in[2][3];
skew[1][2] = -in[0][3];
skew[2][0] = -in[1][3];
skew[2][1] = in[0][3];
mpy333(skew,rot,temp);
for(i=0;i<3;i++){
        for(j=0;j<3;j++){
                out[i][3+j] = temp[i][j];
                }
        }
}
```

```
#include <stdio.h>
#define NN 4
 /* multiplys two NNxNN matrices * /
mpynn(in1,in2,out)
float       in1[NN][NN],
            in2[NN][NN],
            out[NN][NN];
{
int       i,
          j,
          n;
for(i=0;i<NN;i++){
          for(j=0;j<NN;j++){
                    out[i][j] = 0.;
                    }
          }
for(i=0;i<NN;i++){
          for(j=0;j<NN;j++){
                    for(n=0;n<NN;n++){
                              out[i][j] += in1[i][n]*in2[n][j];
                              }
                    }
          }
}

 /* multiplies an NNxNN matrix by an NNx1 matrix * /
mpyn1(in1,in2,out)
float       in1[NN][NN],
            in2[NN],
            out[NN];
{
int         i,
            j;
for(i=0;i<NN;i++){
            out[i] = 0.;
            }
for(i=0;i<NN;i++){
            for(j=0;j<NN;j++){
                        out[i] += in1[i][j]*in2[j];
                        }
            }
}

 /* multiplies 6x6 by 6x6 matrices * /
mpy666(in1,in2,out)
float       in1[6][6],
            in2[6][6],
            out[6][6];
{
int       i,
          j,
          n;
for(i=0;i<6;i++){
          for(j=0;j<6;j++){
                    out[i][j] = 0.;
                    }
          }
for(i=0;i<6;i++){
          for(j=0;j<6;j++){
                    for(n=0;n<6;n++){
                              out[i][j] += in1[i][n]*in2[n][j];
                              }
                    }
          }
```

*mpynn*

*mpyn1*

*mpy666*

*...mpy666*

```
}
```

*mpy664*

```
/* multiplies 6x6 by 6x4 matrices */
mpy664(in1,in2,out)
float       in1[6][6],
            in2[6][4],
            out[6][4];
{
int         i,
            j,
            n;
for(i=0;i<6;i++){
            for(j=0;j<4;j++){
                        out[i][j] = 0.;
                        }
            }
for(i=0;i<6;i++){
            for(j=0;j<4;j++){
                        for(n=0;n<6;n++){
                                    out[i][j] += in1[i][n]*in2[n][j];
                                    }
                        }
            }
}
```

*mpy665*

```
/* multiplies 6x6 by 6x5 matrices */
mpy665(in1,in2,out)
float       in1[6][6],
            in2[6][5],
            out[6][5];
{
int         i,
            j,
            n;
for(i=0;i<6;i++){
            for(j=0;j<5;j++){
                        out[i][j] = 0.;
                        }
            }
for(i=0;i<6;i++){
            for(j=0;j<5;j++){
                        for(n=0;n<6;n++){
                                    out[i][j] += in1[i][n]*in2[n][j];
                                    }
                        }
            }
}
```

*mpy663*

```
/* multiplies 6x6 by 6x3 matrices */
mpy663(in1,in2,out)
float       in1[6][6],
            in2[6][3],
            out[6][3];
{
int         i,
            j,
            n;
for(i=0;i<6;i++){
            for(j=0;j<3;j++){
                        out[i][j] = 0.;
                        }
            }
for(i=0;i<6;i++){
            for(j=0;j<3;j++){
```

*...mpy663*

```
                for(n=0;n<6;n++){
                        out[i][j] += in1[i][n]*in2[n][j];
                        }
                }
        }
}
```

*mpy333*

```
/* multiplies 3x3 by 3x3 matrices */
mpy333(in1,in2,out)
float   in1[3][3],
        in2[3][3],
        out[3][3];
{
int     i,
        j,
        n;
for(i=0;i<3;i++){
        for(j=0;j<3;j++){
                out[i][j] = 0.;
                }
        }
for(i=0;i<3;i++){
        for(j=0;j<3;j++){
                for(n=0;n<3;n++){
                        out[i][j] += in1[i][n]*in2[n][j];
                        }
                }
        }
}
```