Efficient Inverse Kinematics for General 6R Manipulators

Dinesh Manocha and John F. Canny

Abstract—**The inverse kinematics of serial manipulators is a central problem in the automatic control of robot manipulators. The main interest has been in inverse kinematics of a six revolute ($6R$) jointed manipulator with arbitrary geometry. It has been recently shown that the joints of a general $6R$ manipulator can orient themselves in 16 different configurations (at most), for a given pose of the end-effector. However, there are no good practical solutions available that give a level of performance expected of industrial manipulators. In this paper, we present an algorithm and implementation for efficient inverse kinematics for a general $6R$ manipulator. When stated mathematically, the problem reduces to solving a system of multivariate equations. We make use of the algebraic properties of the system and the symbolic formulation used for reducing the problem to solving a univariate polynomial. However, the polynomial is expressed as a matrix determinant and its roots are computed by reducing to an eigenvalue problem. The other roots of the multivariate system are obtained by computing eigenvectors and substitution. The algorithm involves symbolic preprocessing, matrix computations and a variety of other numerical techniques. The average running time of the algorithm, for most cases, is 11 *milliseconds* on an IBM RS/6000 workstation. This approach is applicable to inverse kinematics of all serial manipulators.**

I. Introduction

THE INVERSE kinematics problem for general serial manipulators is fundamental for computer controlled robots. Given the pose of the end effector (the position and orientation), the problem corresponds to computing the joint displacements for that pose. The most interesting case has been that of serial manipulators with six joints. The complexity of inverse kinematics of a general six jointed manipulator is a function of its geometry. While the solution can be expressed in closed form for a variety of special cases, such as when three consecutive axes intersect in a common point, no such formulation is known for the general case. The main interest has been in a $6R$ manipulator, that has six revolute joints, the links are of arbitrary length and no constraints are imposed on the geometry of various links. Iterative solutions (based on numerical techniques) to the inverse kinematics for general $6R$ manipulators have been known for quite some time. However, they suffer from two drawbacks. Firstly, they are slow for practical applications, and secondly they are unable to find all the solutions. As a result, most industrial manipulators are designed sufficiently simply so that a closed form solution exists.

In the absence of a closed form solution, [26] claim that the problem of inverse kinematics for a general $6R$ manipulator is considered solved under the following conditions.

1) A tight upper bound on the number of solutions has been established.
2) An efficient, numerically sound method for computing all solutions has been developed.

At the same time, we feel it is important that the solution be able to provide a level of performance expected of industrial manipulators.

The need for fast algorithms for inverse kinematics of general manipulators has been felt in *kinematic design, kinematic calibration* and *goal-directed computer animation* as well. Kinematic design corresponds to generating appropriate configuration of manipulators given a set of kinematic task specifications [16], [17]. Given the kinematic requirements as workspace volume, maximum reach and maximum positional error the problem is reduced to manipulating algebraic equations, whose variables are the manipulator parameters [17]. The current solutions are restricted to $6R$ manipulators with closed form solutions, which limits the class of manipulators that can be used for kinematic design [17].

The need for kinematic calibration arises due to manufacturing errors in machining and assembly of manipulators. This results in discrepancies between the design parameters and physical structure and can produce significant errors between the actual and predicted positions and orientations of the end effector. The solution to this problem involves identification of the individual kinematic parameters and incorporating them into manipulator's controller to improve positional accuracy. Given the accurate kinematic parameters, a number of methods have been proposed to calibrate and compensate for the kinematic errors in robot manipulators with closed form solutions [8], [25]. However, a practical solution for the inverse kinematics of general manipulators eliminates the need for any algorithms for compensation of kinematic errors.

The inverse kinematics problem for six revolute joints has been studied for more than two decades. The earlier work includes that of Pieper [18] and Roth *et al.* [22]. The first constructive solution to the problem was given by [1], in the form of determinant of a 12×12 matrix, whose entries were quartic polynomials in the tangent of the half-angle of one of

Manuscript received April 13, 1993; revised October 12, 1993. This work was supported in part by an IBM Graduate Fellowship, under ONR Contract N00014-94-1-0738, a NSF Grant CCR-9319957, ARPA Contract DABT63-93-C-0048, and a grant from Mitsubishi Electric Research Lab, as well as a David and Lucile Packard Fellowship, and a NSF Presidential Young Investigator Award (# IRI-8958577).

D. Manocha is with the Department of Computer Science at the University of North Carolina, Chapel Hill, NC 27599-3175 USA.

J. Canny is with the Computer Science Division, University of California, Berkeley CA 94720 USA.

IEEE Log Number 9403325.

the joint variables. Later [5] provided a 32 degree polynomial in the tangent of the half-angle of one of the joint variables. Tsai and Morgan used a *higher-dimensional approach* to the inverse kinematics problem [24]. In particular, they cast the problem as eight second-degree equations and solved them numerically using polynomial continuation. This is in contrast with the earlier approaches, where a single polynomial in the tangent of the half-space of one of the joint variables was derived (referred as the *lower dimensional approach*). Based on their implementation, [24] conjectured that this problem has at most 16 solutions. The first conclusive proof of the fact that the problem can have at most 16 solutions was given by [19], based on the fact that the remaining 16 solutions to the 32 degree polynomial in [5] have purely imaginary parts. Finally, [9], [10] gave the exact solution in lower dimensions by reducing the problem to a 16 degree polynomial. More recently, [20], [21] used dialytic elimination to derive a 16 degree polynomial in the tangent of the half-angle of a joint variable. In [18], [15], examples of a $6R$ manipulator and a pose of the end effector are given such that the inverse kinematics problem has 16 solutions. As a result, 16 is a tight bound on the number of solutions.

Algorithms based on the higher as well lower dimensional approach have been implemented. It turns out that the problem of computing roots of polynomials of degree 16 can be ill-conditioned [27]. As a result, in many cases extra precision is required to accurately compute the solutions to the inverse kinematics problem. Moreover, implementations based on continuation methods are rather slow for practical applications. In particular, the best known algorithm takes about 10 seconds on an average of CPU time on an IBM $370 - 3090$ using double precision arithmetic [26], which falls short of what is expected of industrial manipulators.

In this paper we present an algorithm and implementation for efficient inverse kinematics for a general $6R$ manipulator. The algorithm makes use of symbolic manipulation used in deriving a univariate polynomial and matrix computations. In particular, we use the symbolic formulation presented by Raghavan and Roth [20]. However, the algorithm can also be used along with the formulations given in [1], [9]. The main contribution of our algorithm lies in the fact that we use matrix operations and reduce the problem to an eigenvalue problem as opposed to finding roots. These matrix operations correspond to manipulating matrix polynomials, constructing equivalent companion matrices and computing their eigendecomposition. The main advantage of this technique lies in its *efficiency and numerical stability*. The algorithms for computing eigenvalues and eigenvectors of a matrix are *backward stable*[1] and fast implementations are available as part of linear algebra packages [2], [7]. This is in contrast with expanding a symbolic determinant to compute a degree 16 polynomial and thereby, computing its roots. For almost all instances of the problem we are able to compute accurate solutions using 64 bit IEEE floating point arithmetic. The average running time of the algorithm is 11 *milliseconds* on an IBM RS/6000.

[1] An eigendecomposition algorithm is backward stable if it computes the exact eigendecomposition of a slightly perturbed matrix.

The rest of the paper is organized in the following manner. In Section II, we review the inverse kinematics problem and reduce the problem to solving a system of multivariate polynomials. Section III introduces matrix polynomials and discusses their properties, which are used in finding solutions of non-linear polynomial equations. In Section IV we discuss the algorithm for real time inverse kinematics for general $6R$ manipulators. We highlight the symbolic-numeric interface in the implementation of the algorithm. The symbolic preprocessing is performed once for a given class of manipulators and the numeric computation is performed in real time for a given pose of the end-effector. The numerical accuracy, implementation and performance of the algorithm are discussed in Section V. In Section VI we discuss extensions of the algorithm to general serial manipulators. A preliminary version of this paper had appeared in [13].

II. Inverse Kinematics

A. Problem Formulation

We use Denavit-Hartenberg formalism, [4], to model a $6R$ manipulator. Each link is represented by the line along its joint axis and the common normal to the next joint axis. In the case of parallel joints, any of the common normals can be chosen. The links of the $6R$ manipulator are numbered from 1 to 7. The base link is 1, and the outermost link or hand is 7. A coordinate system is attached to each link for describing the relative arrangements among the various links. The coordinate system attached to the ith link is numbered i. More details of the model are given in [23], [24]. The 4×4 transformation matrix relating $i+1$ coordinate system to i coordinate system is [23]:

$$\boldsymbol{A}_i = \begin{pmatrix} c_i & -s_i\lambda_i & s_i\mu_i & a_ic_i \\ s_i & c_i\lambda_i & -c_i\mu_i & a_is_i \\ 0 & \mu_i & \lambda_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

where

$s_i = \sin\theta_i$, $c_i = \cos\theta_i$, θ_i is the ith joint rotation angle,

$\mu_i = \sin\alpha_i$, $\lambda_i = \cos\alpha_i$, α_i : twist angle,

a_i : length of link $i+1$, d_i : offset distance at joint i.

For a given robot with revolute joints we are given the a_i's, d_i's, μ_i's and λ_i's and the pose of the end-effector, attached to link 7. This pose is described with respect to the base link. We represent it as:

$$\boldsymbol{A}_{hand} = \begin{pmatrix} l_x & m_x & n_x & q_x \\ l_y & m_y & n_y & q_y \\ l_z & m_z & n_z & q_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The problem of inverse kinematics corresponds to computing the joint angles, $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$ and θ_6 such that

$$\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_3\boldsymbol{A}_4\boldsymbol{A}_5\boldsymbol{A}_6 = \boldsymbol{A}_{hand}. \tag{2}$$

The left hand side entries of the matrix equation given above are functions of the sines and cosines of the joint angles. Furthermore, this matrix equation corresponds to 12 scalar equations. Since the matrix formed by the first 3 rows and 3 columns of A_{hand} is orthonormal, only 6 of the 12 equations are independent. Thus, the problem of inverse kinematics of general $6R$ manipulators corresponds to solving 6 equations for 6 unknowns.

B. Raghavan and Roth Solution

We briefly describe the lower dimensional approach described by Raghavan and Roth [20]. They reduce the multivariate system to a degree 16 polynomial in $tan(\frac{\theta_3}{2})$, such that the joint angle θ_3 can be computed from its roots. The other joint angles are computed from substitution and solving for some intermediate equations.

Raghavan and Roth rearrange the matrix equation, (2), as

$$A_3A_4A_5 = A_2^{-1}A_1^{-1}A_{hand}A_6^{-1}. \tag{3}$$

As a result the entries of the left hand side matrix are functions of θ_3, θ_4 and θ_5 and the entries of the right hand side matrix are functions of θ_1, θ_2 and θ_6. This lowers their degrees and reduces the symbolic complexity of the resulting expressions. On equating the corresponding entries of the matrix equation, (3), and after simplification, these equations are expressed in a linear formulation given as:

$$(Q)\begin{pmatrix} s_1s_2 \\ s_1c_2 \\ c_1s_2 \\ c_1c_2 \\ s_1 \\ c_1 \\ s_2 \\ c_2 \end{pmatrix} = (P)\begin{pmatrix} s_4s_5 \\ s_4c_5 \\ c_4s_5 \\ c_4c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \\ 1 \end{pmatrix} \tag{4}$$

where Q is a 14×8 matrix, whose entries are functions of the parameters of the manipulator and the pose of the end effector. P is a 14×9 matrix, whose entries are linear functions of s_3 and c_3 and their coefficients are functions of the manipulator parameters and the pose. The relationship expressed in (4) helps us in eliminating four of the five variables.

Raghavan and Roth use 8 of the 14 equations in (4) to eliminate the left hand side terms, expressed as functions of θ_1 and θ_2, in terms of the right hand side, expressed as functions of θ_3, θ_4 and θ_5. After substituting

$$s_i = \frac{2x_i}{1+x_i^2}, \; c_i = \frac{1-x_i^2}{1+x_i^2}$$

where $x_i = tan(\frac{\theta_i}{2})$, and taking power products, the system is represented as:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & \mathbf{o} \\ A_{21} & A_{22} & A_{23} & \mathbf{o} \\ \mathbf{o} & A_{11} & A_{12} & A_{13} \\ \mathbf{o} & A_{21} & A_{22} & A_{23} \end{pmatrix}\begin{pmatrix} x_4^3x_5^2 \\ x_4^3x_5 \\ x_4^3 \\ x_4^2x_5^2 \\ x_4^2x_5 \\ x_4^2 \\ x_4x_5^2 \\ x_4x_5 \\ x_4 \\ x_5^2 \\ x_5 \\ 1 \end{pmatrix} = 0 \tag{5}$$

where A_{ij} is a 3×3 matrix and $\mathbf{o}$ is the 3×3 null matrix. The entries of A_{ij} are quadratic polynomial in x_3. Let us represent the left hand side 12×12 matrix by Σ. Its determinant is a polynomial of degree 24 in x_3. It turns out that $(1+x_3^2)^4$ divides the determinant and the rest of the 16 roots corresponding to x_3 component of the inverse kinematics solution. In the next section we show the equivalence between this formulation and the non-linear eigenvalue problem. Given θ_3, other angles are computed by solving a system of linear equations [20].

III. Matrix Polynomials

In this section, we review some literature on matrix polynomials and present techniques to solve the non-linear eigenvalue problem. If $A_0, A_1, \cdots, A_k$ are $m \times m$ numeric matrices, then the matrix-valued function defined by

$$L(\lambda) = \Sigma_{i=0}^{k} A_i\lambda^i$$

is called a *matrix polynomial* of degree k. When $A_k = I$, the identity matrix, the matrix polynomial is said to be *monic*. More details on matrix polynomials and their properties are given in [6]. In our application we will be dealing with matrix polynomials in the context of solving non-linear polynomial equations (as shown in (5)). Our main interest is in finding roots of the polynomial equation

$$P(\lambda) = \text{Determinant}(L(\lambda)) = 0. \tag{6}$$

A simple solution to this problem is expand the determinant and compute the roots of the resulting polynomial. However, the resulting approach is numerically unstable and expensive in practice.

Let us consider the case when A_k is a non-singular and well conditioned matrix. As a result computation of A_k^{-1} does not introduce severe numerical errors. Let

$$\overline{L}(\lambda) = A_k^{-1}L(\lambda), \text{ and } \overline{A}_i = A_k^{-1}A_i, \quad 0 \le i < k.$$

$\overline{L}(\lambda)$ is a monic matrix polynomial. Its determinant has the same roots as that of $P(\lambda)$. Let $\lambda = \lambda_0$ be a root of the equation

$$\text{Determinant}(\overline{L}(\lambda)) = 0.$$

As a result $\overline{\boldsymbol{L}}(\lambda_0)$ is a singular matrix and there is at least one non trivial vector in its kernel. Let us denote that $m \times 1$ vector as $\boldsymbol{v}$. That is

$$\overline{\boldsymbol{L}}(\lambda_0)\boldsymbol{v} = \mathbf{o}, \tag{7}$$

where $\mathbf{o}$ is a $m \times 1$ null vector.

Theorem 1: Given the matrix polynomial, $\overline{\boldsymbol{L}}(\lambda)$ the roots of the polynomial corresponding to its determinant are the eigenvalues of the matrix

$$\boldsymbol{C} = \begin{bmatrix} \mathbf{o} & \boldsymbol{I}_m & \mathbf{o} & \cdots & \mathbf{o} \\ \mathbf{o} & \mathbf{o} & \boldsymbol{I}_m & \cdots & \mathbf{o} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \mathbf{o} & \mathbf{o} & \mathbf{o} & \cdots & \boldsymbol{I}_m \\ -\overline{\boldsymbol{A}}_0 & -\overline{\boldsymbol{A}}_1 & -\overline{\boldsymbol{A}}_2 & \cdots & -\overline{\boldsymbol{A}}_{k-1} \end{bmatrix} \tag{8}$$

where $\mathbf{o}$ and $\boldsymbol{I}_m$ are $m \times m$ null and identity matrices, respectively. Furthermore, the eigenvectors of $\boldsymbol{C}$ corresponding to the eigenvalue $\lambda = \lambda_0$ are of the form:

$$[\boldsymbol{v}\ \lambda_0\boldsymbol{v}\ \lambda_0^2\boldsymbol{v}\ \cdots\ \lambda_0^{k-1}\boldsymbol{v}]^T$$

where $\boldsymbol{v}$ is the vector in the kernel of $\overline{\boldsymbol{L}}(\lambda_0)$ as highlighted in (7).

Proof: The eigenvalues of $\boldsymbol{C}$ correspond to the roots of

$$\text{Determinant}(\boldsymbol{C} - s\boldsymbol{I}) = 0.$$

$\boldsymbol{C}$ is a matrix of order mk. Let $s = s_0$ be an eigenvalue of $\boldsymbol{C}$. As a result there is a non-trivial vector $\boldsymbol{V}$ in the kernel of $\boldsymbol{C} - s_0\boldsymbol{I}$. Furthermore, we represent $\boldsymbol{V}$ as

$$\boldsymbol{V} = [\boldsymbol{v}_1^T\ \boldsymbol{v}_2^T\ \cdots\ \boldsymbol{v}_k^T]$$

and each $\boldsymbol{v}_i$ is an $m \times 1$ vector. The relationship between $\boldsymbol{C}$, s_0 and $\boldsymbol{V}$ can be represented as

$$\begin{bmatrix} \mathbf{o} & \boldsymbol{I}_m & \cdots & \mathbf{o} \\ \mathbf{o} & \mathbf{o} & \cdots & \mathbf{o} \\ \vdots & \cdots & \vdots & \vdots \\ \mathbf{o} & \mathbf{o} & \cdots & \boldsymbol{I}_m \\ -\overline{\boldsymbol{A}}_0 & -\overline{\boldsymbol{A}}_1 & \cdots & -\overline{\boldsymbol{A}}_{k-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_{k-1} \\ \boldsymbol{v}_k \end{bmatrix} = s_0 \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_{k-1} \\ \boldsymbol{v}_k \end{bmatrix}. \tag{9}$$

Multiplying the submatrices of $\boldsymbol{C}$ with the vectors in $\boldsymbol{V}$ and equating them with the vectors on the right hand side results in:

$$\boldsymbol{v}_2 = s_0\boldsymbol{v}_1;\quad \boldsymbol{v}_3 = s_0\boldsymbol{v}_2;\ \cdots\ \boldsymbol{v}_k = s_0\boldsymbol{v}_{k-1}$$

and

$$-\overline{\boldsymbol{A}}_0\boldsymbol{v}_1 - \overline{\boldsymbol{A}}_1\boldsymbol{v}_2 - \overline{\boldsymbol{A}}_2\boldsymbol{v}_3 - \cdots - \overline{\boldsymbol{A}}_{k-1}\boldsymbol{v}_k = s_0\boldsymbol{v}_k.$$

These relations imply

$$\boldsymbol{v}_i = s_0^{i-1}\boldsymbol{v}_1, \quad \text{for } 1 \leq i \leq k.$$

and

$$-(\overline{\boldsymbol{A}}_0 + s_0\overline{\boldsymbol{A}}_1 + s_0^2\overline{\boldsymbol{A}}_2 + \cdots + s_0^{k-1}\overline{\boldsymbol{A}}_{k-1} + s_0^k\boldsymbol{I}_k)\boldsymbol{v}_1 = \mathbf{o}.$$

Equating the above relation with (7) results in the fact that s_0 is a solution of $\overline{\boldsymbol{L}}(\lambda) = 0$ and $\boldsymbol{v}_1$ is a vector in the kernel of $\overline{\boldsymbol{L}}(s_0) = 0$. Thus, every eigenvalue of $\boldsymbol{C}$ is a root of $P(\lambda)$. Since the leading matrix of $\boldsymbol{L}(\lambda)$ is non-singular, $P(\lambda)$ is a polynomial of degree mk. Furthermore, $\boldsymbol{C}$ is a matrix of order mk and therefore, has mk eigenvalues. Thus, all the roots of $P(\lambda)$ correspond to the eigenvalues of $\boldsymbol{C}$. Q.E.D.

The matrix polynomials have been used to solve general systems of non-linear polynomial equations. More details are given in [11], [12]. The relationship between the eigenvalues of $\boldsymbol{C}$ and the roots of $P(\lambda)$ has also been proved using similarity transformations in [6]. Many a time the leading matrix $\boldsymbol{A}_k$ is singular or close to being singular (due to high condition number). It may still be possible to reduce the problem to an eigenvalue problem using linear transformations (explained in detail in Section IV-C). However, this technique may not work at times and in these cases we reduce the problem to a generalized eigenvalue problem.

Theorem 2: Given the matrix polynomial, $\overline{\boldsymbol{L}}(\lambda)$ the roots of the polynomial corresponding to its determinant are the eigenvalues of the generalized system $\boldsymbol{C}_1\lambda - \boldsymbol{C}_2$, where

$$\boldsymbol{C}_1 = \begin{bmatrix} \boldsymbol{I}_m & \mathbf{o} & \cdots & \mathbf{o} \\ \mathbf{o} & \boldsymbol{I}_m & \cdots & \mathbf{o} \\ \vdots & \cdots & \vdots & \vdots \\ \mathbf{o} & \cdots & \boldsymbol{I}_m & \mathbf{o} \\ \mathbf{o} & \cdots & \mathbf{o} & \boldsymbol{A}_k \end{bmatrix}$$

$$\boldsymbol{C}_2 = \begin{bmatrix} \mathbf{o} & \boldsymbol{I}_m & \cdots & \mathbf{o} \\ \mathbf{o} & \mathbf{o} & \cdots & \mathbf{o} \\ \vdots & \cdots & \vdots & \vdots \\ \mathbf{o} & \mathbf{o} & \cdots & \boldsymbol{I}_m \\ -\boldsymbol{A}_0 & -\boldsymbol{A}_1 & \cdots & -\boldsymbol{A}_{k-1} \end{bmatrix}$$

where $\mathbf{o}$ and $\boldsymbol{I}_m$ are $m \times m$ null and identity matrices, respectively.

The proof of this theorem is similar to that of Theorem 1.

IV. Algorithm

In this section we describe our algorithm in detail. The initial steps in our algorithm make use of the results presented in [20]. However, we perform symbolic preprocessing and make certain checks for condition numbers and degeneracy to improve the accuracy of the overall algorithm. The overall algorithm proceeds in the following manner:

1) *Symbolic Computation:* For any class of serial manipulators we perform symbolic preprocessing for simplification, minimizing numerical errors and the computation at run time. In particular, we treat the a_i's, d_i's, λ_i's, μ_i's and the entries of the right hand side matrix $\boldsymbol{A}_{hand}$ as symbolic constants. As a result, express the entries of the 14×9 matrix $\boldsymbol{P}$ and 14×8 matrix $\boldsymbol{Q}$, as shown in equation (4), as functions of these symbolic constants. Many geometric properties of manipulators can be interpreted from the linear algebra structure of these matrices. It corresponds to symbolic elimination and is performed using the properties highlighted in [20]. However, it is performed only once for general $6R$ manipulators. An equivalent symbolic elimination can be performed for a serial manipulator with prismatic and revolute joints.
2) *Substitution of Manipulator Parameters:* Given a particular $6R$ manipulator, substitute the numerical values

corresponding to the link lengths, offset distances and twist angles in the symbolic formulations derived above. The substitution results in numerical matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$, as shown in (4).

3) *Numerical Conditioning:* Compute the rank of $\boldsymbol{Q}$ using SVD (singular value decomposition). If $\boldsymbol{Q}$ has rank 8 then this manipulator can have up to 16 solutions for any pose of the end-effector. However, the rank may be less than 8 and as a result we obtain an over-constrained system. In this case the upper bound on the number of solutions may be less than 16. For example, a PUMA manipulator has a total of at most 8 solutions for any pose of the end-effector [23].
4) *Numeric Elimination:* Eliminate the variables θ_1 and θ_2 from (4). This elimination is performed by computing a minor of maximum rank of Q and using that minor to represent θ_1 and θ_2 as functions of θ_4 and θ_5.
5) *Rank Computation:* After eliminating θ_1 and θ_2, we obtain a matrix $\boldsymbol{\Sigma}$. The actual number of rows in $\boldsymbol{\Sigma}$ is equal to $R = (14 - \text{rank}(\boldsymbol{Q})) \geq 6$. Take any of the 6 rows of $\boldsymbol{\Sigma}$ (among R) and substitute for sines and cosines of θ_3, θ_4 and θ_5 in terms of x_3, x_4 and x_5, respectively. In case there are more than 6 rows we recommend taking 6 distinct linear combinations.
6) *Reduction to Eigenvalue Problem:* Reduce the problem of computing roots of Determinant($\boldsymbol{\Sigma}$) $= 0$ to an eigenvalue problem. The eigenvalues of the resulting 24×24 matrix correspond to the root x_3 and the corresponding eigenvectors are used to compute the values of x_4 and x_5. Substitute these relations in (4) and (3) to compute the joint angles θ_1, θ_2 and θ_6. The algorithm also involves clustering eigenvalues to accurately compute eigenvalues of multiplicity greater than one. Depending upon the condition number of the matrices involved, the problem may be reduced to a generalized eigenvalue problem.
7) *Improving the Accuracy:* Compute the condition number of the eigenvalues. In case the condition number is high, improve the accuracy of resulting solution by Newton's method. The solutions computed above are the starting points for Newton's method and its quadratic convergence gives us high accuracy in a few steps.

These steps are explained in detail in the following sections.

A. Symbolic Preprocessing

The algorithm performs symbolic preprocessing for the inverse kinematics solution. It treats the Denavit-Hartenberg parameters and the entries of $\boldsymbol{A}_{hand}$ as symbolic constants. These symbolic constants along with the variables θ_i are used in the symbolic derivation of the equations. We use the computer algebra system, MAPLE, for the derivation and simplification of the expressions. The coefficients of the equations are used to compute the entries of the matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$. As a result, we are able to express the entries of $\boldsymbol{P}$ and $\boldsymbol{Q}$ as polynomial functions of the symbolic constants. In the case of $\boldsymbol{P}$, each entry is of the form $\beta\sin(\theta_3) + \gamma\cos(\theta_3) + \delta$, where β, γ and δ are functions of the symbolic constants.

The matrix $\boldsymbol{Q}$ has a special structure. In particular many of its entries are zero and as a result the system of equations, (4), can be expressed as two different systems of equations of the form [20]:

$$(\boldsymbol{Q}_1)\begin{pmatrix} s_1 \\ c_1 \end{pmatrix} = (\boldsymbol{P}_1)\begin{pmatrix} s_4 s_5 \\ s_4 c_5 \\ c_4 s_5 \\ c_4 c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \end{pmatrix} \tag{10}$$

$$(\boldsymbol{Q}_2)\begin{pmatrix} s_1 s_2 \\ s_1 c_2 \\ c_1 s_2 \\ c_1 c_2 \\ s_2 \\ c_2 \end{pmatrix} = (\boldsymbol{P}_2)\begin{pmatrix} s_4 s_5 \\ s_4 c_5 \\ c_4 s_5 \\ c_4 c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \\ 1 \end{pmatrix} \tag{11}$$

where $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \boldsymbol{P}_1, \boldsymbol{P}_2$ are $6 \times 2, 8 \times 6, 6 \times 9, 8 \times 9$ matrices, respectively. In particular, we break the set of the 14 equations into sets of 6 and 8 equations. $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$ are submatrices of $\boldsymbol{Q}$ and $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ are submatrices of $\boldsymbol{P}$.

B. Numerical Substitution and Rank Computation

The symbolic preprocessing is performed offline. Given the manipulator geometry and the pose of the end-effector, the numerical computations are performed online. In particular, given the Denavit-Hartenberg parameters of a manipulator, we substitute the a_i's, d_i's, λ_i's and μ_i's into the functions used to represent the entries of $\boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{Q}_1, \boldsymbol{Q}_2$. These computations are only performed once for a manipulator and are independent of the pose of the end-effector. As a result, they are categorized under pre-processing computation. Given the pose of the end-effector, we substitute them to compute the entries of $\boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{Q}_1, \boldsymbol{Q}_2$. Let the corresponding numerical matrices (obtained after substitution) be $\overline{\boldsymbol{P}_1}, \overline{\boldsymbol{P}_2}, \overline{\boldsymbol{Q}_1}, \overline{\boldsymbol{Q}_2}$.

We use singular value decompositions to compute the ranks of $\overline{\boldsymbol{Q}_1}$ and $\overline{\boldsymbol{Q}_2}$ [7]. The singular vectors obtained are also used to eliminate θ_1 and θ_2 from (10) and (11). In particular, let the singular value decomposition of $\overline{\boldsymbol{Q}_1}$ be expressed as:

$$\overline{\boldsymbol{Q}_1} = \boldsymbol{U}\boldsymbol{\Sigma}'\boldsymbol{V}^T$$

where $\boldsymbol{U}, \boldsymbol{\Sigma}'$ and $\boldsymbol{V}^T$ are 6×2, 2×2 and 2×2 matrices, respectively. Initially we compute the singular values, σ_1, σ_2 of $\overline{\boldsymbol{Q}_1}$. If both the singular values are non-zero, $\overline{\boldsymbol{Q}_1}$ has full rank and let $\overline{\boldsymbol{Q}_1}' = \overline{\boldsymbol{Q}_1}$. If either of the singular values, σ_i is close to 0.0, we conclude that $\overline{\boldsymbol{Q}_1}$ does not have full rank. In this case we represent

$$\sigma_i' = \begin{cases} \sigma_i & \sigma_i \geq \epsilon \\ 0 & \sigma_i < \epsilon \end{cases}$$

where ϵ is a user defined constant to test the rank deficiency of the matrix. Furthermore we compute the elements of $\boldsymbol{U}$ and

$\boldsymbol{V}$, and represent

$$\overline{\boldsymbol{Q}}'_{1_{ij}} = \Sigma_{k=1}^{2} \sigma'_k \boldsymbol{U}_{ik} \boldsymbol{V}_{jk}.$$

$\overline{\boldsymbol{Q}}'_1$ has the property that a small perturbation does not decrease the rank of the matrix. It turns out that this property has significant impact on the accuracy of the rest of the algorithm. We use $\overline{\boldsymbol{Q}}'_1$ for eliminating θ_1, θ_2 in the system of equations (10) to obtain

$$(\overline{\boldsymbol{Q}}'_1)\begin{pmatrix} s_1 \\ c_1 \end{pmatrix} = (\boldsymbol{P}_1)\begin{pmatrix} s_4 s_5 \\ s_4 c_5 \\ c_4 s_5 \\ c_4 c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \end{pmatrix}. \tag{12}$$

We perform Gaussian elimination with complete pivoting on $\overline{\boldsymbol{Q}}'_1$ and corresponding row and column operations are carried on the elements of $\boldsymbol{P}$. Depending on the rank of $\overline{\boldsymbol{Q}}'_1$, whether 0.1 or 2, we obtain 6.5 or 4 equations, respectively, in sines and cosines of θ_4, θ_5.

In a similar fashion we compute the rank of $\overline{\boldsymbol{Q}}_2$, as represented in (11). In case either of the singular values is close to 0.0, we recompute the matrix $\overline{\boldsymbol{Q}}'_2$ from the singular value decomposition of $\overline{\boldsymbol{Q}}_2$. Otherwise $\overline{\boldsymbol{Q}}_2 = \overline{\boldsymbol{Q}}_2$. The modified matrix is used in eliminating θ_1, θ_2 from (11). Depending on the rank of $\overline{\boldsymbol{Q}}'_2$, we may obtain anywhere from 2 to 8 equations after elimination.

C. Reduction to Eigenvalue Problem

In this section, we reduce the problem of root finding to an eigenvalue problem. Moreover, we exploit the structure of the resulting matrix for efficiently computing its eigenvalues.

We are given a 12×12 matrix, $\boldsymbol{\Sigma}$, whose entries are quadratic polynomials in x_3. Our problem is to solve the system of equations

$$\boldsymbol{\Sigma v} = \boldsymbol{\Sigma}\begin{pmatrix} x_4^3 x_5^2 \\ x_4^3 x_5 \\ x_4^3 \\ x_4^2 x_5^2 \\ x_4^2 x_5 \\ x_4^2 \\ x_4 x_5^2 \\ x_4 x_5 \\ x_4 \\ x_5^2 \\ x_5 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{13}$$

We express the matrix as

$$\boldsymbol{\Sigma} = \boldsymbol{A} x_3^2 + \boldsymbol{B} x_3 + \boldsymbol{C} \tag{14}$$

where $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$ are 12×12 matrices consisting of numerical entries. We compute the condition number of $\boldsymbol{A}$. If the matrix is singular, its condition number is infinity. Let us consider the case, when the matrix $\boldsymbol{A}$ is well conditioned. We take the matrix equation, (14), and multiply it by A^{-1}. Let

$$\overline{\boldsymbol{\Sigma}} = \boldsymbol{I} x_3^2 + \boldsymbol{A}^{-1} \boldsymbol{B} x_3 + \boldsymbol{A}^{-1} \boldsymbol{C}.$$

In practice $\boldsymbol{A}^{-1}\boldsymbol{B}$ and $\boldsymbol{A}^{-1}\boldsymbol{C}$ are computed by linear equation solvers. Given $\overline{\boldsymbol{\Sigma}}$, we use Theorem 1 to construct a 24×24 matrix $\boldsymbol{M}$ of the form

$$\boldsymbol{M} = \begin{pmatrix} \mathbf{o} & \boldsymbol{I} \\ -\boldsymbol{A}^{-1}\boldsymbol{C} & -\boldsymbol{A}^{-1}\boldsymbol{B} \end{pmatrix}.$$

It follows from the structure of $\boldsymbol{M}$ that the eigenvalues of $\boldsymbol{M}$ correspond exactly to the roots of Determinant$(\boldsymbol{\Sigma}) = 0$. Furthermore, the eigenvectors of $\boldsymbol{M}$, corresponding to the eigenvalue x_3 have the structure

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{v} \\ x_3 \boldsymbol{v} \end{pmatrix} \tag{15}$$

where $\boldsymbol{v}$ is the vector corresponding to the variables in (13). Thus, the eigenvectors of $\boldsymbol{M}$ can be used to compute the roots of the equations in (13).

In many instances the matrix $\boldsymbol{A}$ in (14) may be ill-conditioned. One example of such a case occurs, when one of the solution of inverse kinematics has $\theta_3 \approx 180$. As a result, $x_3 = tan(\frac{\theta_3}{2}) \approx \infty$. Therefore, $\boldsymbol{A}$ is nearly singular. We take the matrix equation, (14), and reduce it to a generalized eigenvalue problem by constructing two matrices, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$

$$\boldsymbol{M}_1 = \begin{pmatrix} \boldsymbol{I} & \mathbf{o} \\ \mathbf{o} & \boldsymbol{A} \end{pmatrix}, \boldsymbol{M}_2 = \begin{pmatrix} \mathbf{o} & \boldsymbol{I} \\ -\boldsymbol{C} & -\boldsymbol{B} \end{pmatrix}$$

where $\mathbf{o}$ and $\boldsymbol{I}$ are 12×12 null and identity matrices, respectively. Furthermore, the roots of Determinant $(\boldsymbol{\Sigma}) = 0$, correspond exactly to the eigenvalues of the generalized eigenvalue problem $\boldsymbol{M}_1 - x_3 \boldsymbol{M}_2$, according to Theorem 2. The eigenvectors have the same structure as (15).

Computing the eigendecomposition of a generalized eigenvalue problem is costlier than the eigenvalue problem by a factor of 2.5 to 3. In most cases, we can perform a linear transformation and reduce the problem to an eigenvalue problem. In particular, we perform a transformation of the form

$$x_3 = \frac{a\overline{x}_3 + b}{c\overline{x}_3 + d} \tag{16}$$

where a, b, c, d are random numbers. As a result of this transformation, (14) transforms into

$$\begin{aligned} \boldsymbol{\Sigma}_1 = (a^2\,\boldsymbol{A} + ac\,\boldsymbol{B} + c^2\,\boldsymbol{C})\overline{x}_3^2 + (2ab\,\boldsymbol{A} + \\ (ad + bc)\,\boldsymbol{B} + 2cd\,\boldsymbol{C})\overline{x}_3 + (b^2\,\boldsymbol{A} + bd\,\boldsymbol{B} + d^2\,\boldsymbol{C}). \end{aligned} \tag{17}$$

Let $\overline{\boldsymbol{A}} = a^2\,\boldsymbol{A} + ac\,\boldsymbol{B} + c^2\,\boldsymbol{C}$. In most cases $\overline{\boldsymbol{A}}$ is well conditioned. The only exceptions arise when

$$\begin{pmatrix} \boldsymbol{I} & \mathbf{o} \\ \mathbf{o} & \boldsymbol{A} \end{pmatrix} - \lambda \begin{pmatrix} \mathbf{o} & \boldsymbol{I} \\ -\boldsymbol{C} & -\boldsymbol{B} \end{pmatrix}$$

is a singular pencil. $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ may have common singular pencils. In the latter case, $\overline{\boldsymbol{A}}$ is ill conditioned for all choices of a, b, c, d.

We try this transformations for a few choices of a, b, c, d and compute the condition number of $\overline{\boldsymbol{A}}$. The cost of estimating condition number is rather small as compared to computing the eigendecomposition of the matrix. If $\overline{\boldsymbol{A}}$ is well conditioned, solve for Determinant$(\boldsymbol{\Sigma}_1) = 0$ by reducing it to an eigenvalue problem. Given $\overline{x}_3$, apply the inverse transformation to compute x_3. The eigenvectors have the same structure as (15), except that x_3 is replaced by $\overline{x}_3$.

V. Implementation

We have implemented the algorithm on an IBM RS/6000. We have used many routines from EISPACK and LAPACK for matrix operations [2]. These routines are available in Fortran and we interfaced them with our C programs. Many of the algorithms for matrix computations have been specialized to our application. The details are given below.

A. Eigendecomposition

In the previous section we reduced the problem of root finding to an eigenvalue problem. The 24×24 matrix, M, has 24 eigenvalues. However, following the properties of the symbolic formulation in [20], 8 of the eigenvalues correspond to the roots of the polynomial $(1 + x_3^2)^4 = 0$. In other words, ι and $-\iota$ are eigenvalues of $\boldsymbol{M}$ of multiplicity 4 each, where $\iota = \sqrt{-1}$. If we transform the variable x_3, as shown in (16), these eigenvalues are suitably transformed. We make use the structure of $\boldsymbol{M}$ along with the QR algorithm for eigenvalue computation [7]. In the double shift QR algorithm we chose the shift value for the first few iterations corresponding to ι and $-\iota$. It uses at most four iterations of the double shift algorithm to reduce the problem to computing the eigenvalues of a 16×16 matrix.

B. Clustering Eigenvalues

In many instances the solution has a root of multiplicity greater than one. Such cases arise when the manipulator is at a singular configuration. As such the problem of computing multiple roots of polynomial equations can be ill-conditioned. In other words the condition numbers for such eigenvalues can be high and the solution therefore, is not accurate. In most instances of the problem, we have noticed that there is a symmetric perturbation in the multiple roots. For example, let $x_3 = \alpha$ be a root of multiplicity k of the given equation. The floating point errors cause the roots to be perturbed and the algorithm computes k different roots $\alpha_1, \cdots, \alpha_k$. Moreover, $\mid \alpha - \alpha_j \mid$ may be relatively high. Let $\alpha_m = \frac{\alpha_1 + \alpha_2 + \cdots + \alpha_k}{k}$. In many cases $\mid \alpha - \alpha_m \mid$ is relatively small and α_m is very close to the multiple roots. It turns out that each of the perturbed eigenvalues, α_i, can be ill-conditioned; however, the arithmetic mean of the perturbed eigenvalues, α_m is well-conditioned [3]. We actually verify the accuracy of these computations by computing the condition number of the eigenvalue and the condition number of a cluster of eigenvalues. Routines to compute this condition number of a cluster are available as part of LAPACK.

C. Eigenvector Computation

The eigenvector corresponding to a real eigenvalue is computed by solving a quasi-upper triangular system [7]. Given an eigenvector $\boldsymbol{V}$, we use its structure, (15), to accurately compute x_4 and x_5 from it. However, due to floating point errors each component of the eigenvector undergoes a slight perturbation. Each term of the vector has the same bound on the maximum error occurred due to perturbation [28]. As a result, terms of maximum magnitude generally have the minimum amount of relative error. We use this property in accurate computation of x_4 and x_5. Given the eigenvector $\boldsymbol{V}$, let

$$\boldsymbol{v}_1 = \begin{cases} \boldsymbol{v} & \mid x_3 \mid \leq 1 \\ x_3 \boldsymbol{v} & \mid x_3 \mid > 1 \end{cases}.$$

Thus, $\boldsymbol{v}_1$ corresponds to elements of $\boldsymbol{V}$, whose relative error is low. x_4 and x_5 can be computed from $\boldsymbol{v}_1$ by solving for

$$\boldsymbol{v}_1 = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \end{pmatrix} = \begin{pmatrix} x_4^3 x_5^2 \\ x_4^3 x_5 \\ x_4^3 \\ x_4^2 x_5^2 \\ x_4^2 x_5 \\ x_4^2 \\ x_4 x_5^2 \\ x_4 x_5 \\ x_4 \\ x_5^2 \\ x_5 \\ 1 \end{pmatrix}. \tag{18}$$

Therefore, x_4 and x_5 corresponds to ratio of two terms of $\boldsymbol{v}_1$. Initially, we decide whether $\mid x_4 \mid \geq 1$ or $\mid x_4 \mid < 1$ by comparing the magnitude of v_1 and v_2. A similar computation is performed for determining the magnitude of x_5. Depending upon their magnitudes, we tend to use terms of maximum magnitude such that their ratios correspond to x_4 and x_5. As a result we minimize the error.

D. Computing All Joint Angles

Given a triple (x_3, x_4, x_5) corresponding to a solution of the 12 equations represented as the 12×12 matrix $\boldsymbol{\Sigma}$, as shown in (5). We use these solutions to compute the rest of the joint angles. Given the values of $s_3, c_3, s_4, c_4, s_5, c_5$, solve for the unknowns s_1, c_1 based on the linear relationship shown in (10). Similarly solve the linear system (11) for the unknowns s_2, c_2. These five joints angles, $\theta_1, \cdots, \theta_5$ are substituted into (3) to compute θ_6.

E. Improving the Accuracy

With each eigenvalue, we have the knowledge of its condition number and therefore, the accuracy of the resulting solution. If we desire further accuracy, we use these solutions as start points for Newton's iterations on the algebraic equations obtained from (2). In most instances we have been able to compute the joint angles up to 10 digits of accuracy, by using one or two Newton iterations (as it has local quadratic convergence).

TABLE I
THE DENAVIT HARTENBERG PARAMETERS OF A $6R$ MANIPULATOR

Number	Link length	Offset	Twist angle
i	a_i	d_i	α_i
1	0.3	0.0	90.0
2	1.0	0.0	1.0
3	0.0	0.2	90.0
4	1.5	0.0	1.0
5	0.0	0.0	90.0
6	0.0	0.0	1.0

F. Performance

We have applied our algorithm to many examples. In particular, we used it on 21 problem instances given in [26] and verified the accuracy of our algorithm. All these problems can be accurately solved using double precision arithmetic. In many cases we are able to compute solutions up to 11–12 digits of accuracy.

For most problems, the algorithm takes about 11 milliseconds on an average on an IBM RS/6000. About 75–80% of the time is spent in the QR algorithms for computing the eigendecomposition. Thus, better algorithms and implementations for eigendecomposition can improve the running time even further.

In a few cases the algorithm takes as much as 25 milliseconds on the IBM RS/6000. In these instances the matrices A, B, C in (14) are ill-conditioned and have singular pencils. As a result we reduce the resulting problem to a generalized eigenvalue problem, which slows down the algorithm.

Example: Let us consider the manipulator presented in [26] along with a pose of the end effector. This is problem 6 in [26] and corresponds to a slight variation of the manipulator presented in [15]. For this configuration the problem of inverse kinematics has 16 real solutions. The robot parameters are given in Table I.

The position and orientation of the end effector and is given by the matrix

$$A_{hand} = \begin{pmatrix} -0.7601 & -0.6416 & 0.1022 & -1.1401 \\ 0.1333 & 0.0 & 0.9910 & 0.0 \\ -0.6359 & 0.7669 & -0.0855 & 0.0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

After substitution into the symbolic matrices, we obtain

$$\overline{Q}_1 = \begin{pmatrix} -1.140 & -0.0 \\ 0.0910 & -0.990 \\ -0.0 & 0.684 \\ -0.297 & -0.027 \\ 0.0 & 0.112 \\ -0.110 & -1.377 \end{pmatrix},$$

$$\overline{Q}_2 = \begin{pmatrix} 0.0 & 0.0 & 0.0 & -1.14 & -0.0 & -0.30 \\ 0.0 & 0.0 & -1.14 & -0.0 & -0.30 & 0.0 \\ 0.0 & 0.99 & 0.0 & 0.09 & 0.099 & 0.0 \\ 0.99 & 0.0 & 0.091 & 0.0 & 0.0 & -0.099 \\ -0.027 & -0.113 & 0.297 & -0.0 & 1.129 & 0.0 \\ -0.113 & 0.027 & -0.0 & -0.297 & 0.0 & -1.129 \\ 0.0 & 1.20 & 0.068 & -0.127 & 0.138 & -0.062 \\ 1.199 & 0.0 & -0.126 & -0.067 & -0.062 & -0.138 \end{pmatrix}.$$

The entries of P_1 and P_2 are functions of $s3$ and $c3$. P_1 is an 6×9 matrix,

$$\overline{P}_1 = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & .3e\text{-}4 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 3.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.99 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -.003 & -.003 & 0.0 & 0.0 & 0.0 & 0.0 & -0.043 & 0.0 \\ 0.0 & -0.39 & -0.39 & 0.0 & 0.0 & 0.0 & 0.0 & -3.0 & 0.0 \end{pmatrix} c3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.03 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.017 & 0.017 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.75 & 0.0 \\ 0.0 & -1.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 6.1e\text{-}5 & 0.0 \\ 0.0 & 0.11 & -0.02 & 0.0 & 0.0 & 0.0 & 0.0 & 0.017 & 0.0 \end{pmatrix} s3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 1.5 & 0.0 & 0.0 & 0.0 & 0.2 \\ 1.0 & 0.0 & 0.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.6 & 0.0 & 0.0 & 0.0 & 1.9 \\ 0.2 & 0.0 & 0.0 & -0.20 & 0.0 & 0.0 & 1.5 & 0.0 & 0.10 \\ 0.017 & 0.0 & 0.0 & -0.044 & 0.0 & 0.0 & 0.0 & 0.0 & -0.03 \\ -1.29 & 0.0 & 0.0 & -3.21 & 0.0 & 0.0 & -0.6 & 0.0 & 0.68 \end{pmatrix}.$$

Similarly, P_2 is a 8×9 matrix

$$\overline{P}_2 = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.5 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.017 & 0.0 \\ 0.0 & 0.026 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 3.5e\text{-}3 & 0.0 \\ 0.0 & 0.2 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 & 1.5 & 0.0 \\ 0.0 & 1.29 & -3.21 & 0.0 & 0.0 & 0.0 & 0.0 & 0.6 & 0.0 \\ 0.0 & -0.017 & -6.9e\text{-}3 & 0.0 & 0.0 & 0.0 & 0.0 & -0.11 & 0.0 \end{pmatrix} c3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.5 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.02 & 0.0 \\ 0.0 & -1.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.02 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 & 1.5 & 0.0 \\ 0.0 & -0.044 & -0.017 & 0.0 & 0.0 & 0.0 & 0.0 & -3.5e\text{-}3 & 0.0 \\ 0.0 & -0.01 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.023 & 0.0 \\ 0.0 & 3.29 & 1.21 & 0.0 & 0.0 & 0.0 & 0.0 & -0.6 & 0.0 \end{pmatrix} s3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.03 & 0.0 & 0.0 & 0.0 & 3.5e\text{-}3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.017 & 0.0 & 0.0 & -0.017 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.1 & 0.0 & 0.0 & 0.0 & -3.49e\text{-}3 & 0.0 \\ -0.40 & 0.0 & 0.0 & 0.4 & 0.0 & 0.0 & -3.0 & 0.0 & 0.0 \\ -0.02 & 0.0 & 0.0 & -0.11 & 0.0 & 0.0 & -0.01 & 0.0 & 0.0 \end{pmatrix}.$$

TABLE II
EIGENVALUES AND THEIR CONDITION NUMBERS

Number	Eigenvalue	Condition number
1	3679.99	9.32215
2	-123.591	11.3508
3	-35.0237	7.71049
4	-50.794	8.82256
5	-3.45709	10.4068
6	3.33357	9.10936
7	-1.56894	7.09899
8	1.48377	6.83255
9	-0.673961	6.83255
10	0.637372	7.09899
11	-0.299978	9.10936
12	0.289261	10.4068
13	0.0285521	7.71049
14	-0.00027174	9.32215
15	0.00809121	11.3508
16	0.0196874	8.82256

TABLE III
THE JOINT ANGLES CORRESPONDING TO THE SOLUTIONS

i	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
1	-96.28	-6.27	179.96	38.48	52.55	-39.40
2	-120.78	172.33	-179.07	31.33	-146.71	142.82
3	88.67	-176.72	-176.72	-63.24	157.19	140.43
4	113.84	5.30	-177.74	-55.92	-62.98	-43.37
5	-178.12	108.19	-147.73	-5.69	-164.67	179.58
6	168.32	-103.89	146.60	-17.24	-171.87	98.16
7	-12.94	-105.09	-114.97	3.02	7.41	-79.42
8	2.51	108.07	112.04	-10.52	0.00	-0.10
9	2.51	108.07	-67.95	-169.47	179.99	179.89
10	-12.94	-105.09	65.02	176.97	172.58	100.57
11	168.32	-103.89	-33.39	-162.75	-8.12	-81.83
12	-178.12	108.19	32.26	-174.30	-15.32	-0.41
13	88.67	-176.72	3.27	-116.75	22.80	-39.56
14	-96.28	-6.27	-0.03	141.51	127.44	140.59
15	-120.78	172.33	0.92	148.66	-33.28	-37.17
16	113.84	5.30	2.25	-124.07	-117.01	136.62

The matrices $\overline{\boldsymbol{Q}}_1$ and $\overline{\boldsymbol{Q}}_2$ have no singular values close to zero. In other words they are full rank matrices. As a result after numerical elimination we obtain a 6×9 matrix $\boldsymbol{\Sigma}$. $\boldsymbol{\Sigma}$ is converted into a matrix polynomial using the transformation $x_3 = tan(\frac{\theta_3}{2})$ and obtaining the 12×12 matrix $\boldsymbol{\Sigma}''$, expressed as a matrix polynomial in x_3. The estimated condition number of the leading matrix is 5000.0.[2] As a result, we reduce it to an eigenvalue problem of a 24×24 square matrix. The eigenvalues are computed using LAPACK routines. The real eigenvalues and their condition numbers are given in Table II.

Thus, we see that all the 16 eigenvalues are real. Furthermore, they are computed up to 15 digits of accuracy. This follows from the fact that the machine constant for IEEE floating point arithmetic is of the order of 10^{-16} and the maximum condition number is of the order of 11. As a result, the eigenvalues have a relative error bounded by 10^{-15}. Given the eigenvalues, the rest of the algorithm involves computation of rest of the corresponding eigenvectors and joint angles. Let's illustrate the process for the first eigenvalue, $x_3 = 3679.99$. As a result,

$$s3 = 0.00054348, \qquad c3 = -0.999999.$$

Since $\mid x_3 \mid > 1$, we make $\boldsymbol{v}_1$ equal to the last 12 elements of $\boldsymbol{V}$ the eigenvector, as shown in (18). Analyzing the elements of $\boldsymbol{v}_1$ results in $\mid x_4 \mid < 1$ and $\mid x_5 \mid < 1$. Elements of maximum magnitude of $\boldsymbol{v}_1$ are used to compute x_4 and x_5 to the best possible accuracy. It results in $x_4 = 0.34907$ and $x_5 = 0.49368$. These are used to compute $s1, s2, c1, c2, s6, c6$ by solving a system of linear equations.

Given the sines and cosines of the joint angles, s_i and c_i, their accuracy is improved by using a few iterations of the Newton's method. As a result, it is possible to obtain solutions to 12 digits of accuracy on this example. The 16 solutions for this position and orientation of the end-effector are given in Table III.

[2] In practice we have been able to linearize matrix polynomials with leading matrices of condition number up to $1e05$ to eigenvalue problems.

VI. GENERAL SERIAL MANIPULATORS

The techniques presented have been extended to all serial manipulators with a finite number of solutions by making use of the matrix structures [14]. The joints may be prismatic or revolute. In particular, Raghavan and Roth have shown that for many cases of manipulators with six joints (revolute or prismatic) the problem of inverse kinematics reduces to finding roots of a univariate polynomial [21]. It involves taking suitable minors of matrix and reduction to an eigenvalue problem.

VII. CONCLUSION

In this paper we presented an efficient algorithm for inverse kinematics of a $6R$ manipulator of general geometry. The algorithms performs symbolic preprocessing, matrix computations and reduces the problem to computing the eigendecomposition of a matrix. The numerical accuracy of the operations used in the algorithm is well understood. For most instances of the problem the solution can be accurately computed using double precision arithmetic. The algorithm has been tested on a variety of instances and the average running time is 11 ms on an IBM RS/6000. We believe that this algorithm gives us a level of performance expected of industrial manipulators. This approach can be directly extended to all serial manipulators with a finite number of solutions.

ACKNOWLEDGMENT

We are grateful to J. Demmel for productive discussions on matrix computations.

REFERENCES

[1] H. Albala and J. Angeles, "Numerical solution to the input output displacement equation of the general 7R spatial mechanism," in *Proc. Fifth World Cong. Theory Mach. Mechanisms,* 1979, pp. 1008–1011.
[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen, *LAPACK User's Guide, Release 1.0,* Philadelphia, PA: SIAM, 1992.
[3] Z. Bai, J. Demmel, and A. McKenney, "On the conditioning of the nonsymmetric eigenproblem: Theory and software," Computer Science

Dept. Technical Report 469, Courant Institute, New York, NY, October, 1989 (LAPACK Working Note #13).
[4] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based upon matrices," *J. App. Mechanics*, 77, pp. 215–221, 1955.
[5] J. Duffy and C. Crane, "A displacement analysis of the general spatial 7R mechanism," *Mechanisms Mach. Theory*, vol. 15, pp. 153–169, 1980.
[6] I. Gohberg, P. Lancaster, and L. Rodman, *Matrix Polynomials*. New York: Academic Press, 1982.
[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: John Hopkins Press, 1989.
[8] S. A. Hayati. "Robot arm geometric link calibration," in *IEEE Contr. Decision Conf.*, 1983, pp. 1477–1483.
[9] H. Y. Lee and C. G. Liang, "Displacement analysis of the general spatial 7-link 7R mechanism," *Mechanisms and Machine Theory*, vol. 23, no. 3, pp. 219–226, 1988.
[10] H. Y. Lee and C. G. Liang, "A new vector theory for the analysis of spatial mechanisms," *Mechanisms Mach. Theory*, vol. 23, no. 3, pp. 209–217, 1988.
[11] D. Manocha, "Algebraic and numeric techniques for modeling and robotics," Ph.D. thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.
[12] D. Manocha, Solving systems of polynomial equations, *IEEE Comput. Graph. Applicat.*, Special Issue on Solid Modeling, pp. 46–55, March 1994.
[13] D. Manocha and J. F. Canny, "Real time inverse kinematics of general 6R manipulators," in *Proc. IEEE Conf. Robotics Automat.*, pp. 383–389, 1992.
[14] D. Manocha and Y. Zhu, "A fast algorithm and system for inverse kinematics of general serial manipulators," in *Proc. IEEE Conf. Robotics Automat.*, 1994.
[15] R. Manseur and K. L. Doty, "A robot manipulator with 16 real inverse kinematic solution set," *Int. J. Robotics Res.*, vol. 8, no. 5, pp. 75–79, 1989.
[16] B. Paden and S. Sastry, "Optimal kinematic design of 6R manipulators," *Int. J. Robotics Res.*, vol. 7, no. 2, pp. 43–61, 1988.
[17] C. J. Paredis and P. K. Khosla, "An approach for mapping kinematic task specification into a manipulator design," in *Fifth Int. Conf. Advanced Robotics*, Pisa, Italy, June 1991.
[18] D. Pieper, "The kinematics of manipulators under computer control," Ph.D. thesis, Stanford University, 1968.
[19] E. J. F. Primrose, "On the input-output equation of the general 7R-mechanism," *Mechanisms and Machine Theory*, vol. 21, pp. 509–510, 1986.
[20] M. Raghavan and B. Roth, "Kinematic analysis of the 6R manipulator of general geometry," in *Int. Symp. Robotics Res.*, pp. 314–320, Tokyo, 1989.
[21] M. Raghavan and B. Roth, "Inverse kinematics of the general 6R manipulator and related linkages," *Trans. ASME J. Mech. Des.*, to appear.
[22] B. Roth, J. Rastegar, and V. Scheinman, "On the design of computer controlled manipulators," in *On the Theory and Practice of Robots and Manipulators*, First CISM IFToMM Symposium, 1973, pp. 93–113.
[23] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control*. New York: John Wiley and Sons, 1989.
[24] L. W. Tsai and A. P. Morgan, "Solving the kinematics of the most general six and five-degree-of-freedom manipulators by continuation methods," *Trans. ASME J. Mech. Transmissions Automat. Des.*, 107, pp. 189–200, 1985.
[25] W. K. Veitschegger and C. Wu, "A method for calibrating and compensating robot kinematic errors," in *IEEE Conf. Robotics Automat.*, pp. 39–43, 1987.
[26] C. Wampler and A. P. Morgan, "Solving the 6R inverse position problem using a generic-case solution methodology," *Mechanisms Mach. Theory*, vol. 26, no. 1, pp. 91–106, 1991.
[27] J. H. Wilkinson, "The evaluation of the zeros of ill-conditioned polynomials—Parts i and ii," *Numer. Math.*, vol. 1, pp. 150–180, 1959.
[28] J. H. Wilkinson, *The algebraic eigenvalue problem*. Oxford: Oxford University Press, 1965.

Dinesh Manocha received the B.Tech. degree in computer science from Indian Institute of Technology, Delhi, in 1987, and the M.S. and Ph.D. degrees in computer science from the University of California at Berkeley in 1990 and 1992, respectively. He received an Alfred and Chella D. Moore fellowship and an IBM graduate fellowship in 1988 and 1991, respectively, and a junior faculty award in 1992. He is currently an assistant professor of computer science at the University of North Carolina at Chapel Hill. His research interests include geometric and solid modeling, virtual environments, geometric constraint systems and symbolic and scientific computation.

John F. Canny photograph and biography unavailable.