Proceedings of the 1992 IEEE
International Conference on Robotics and Automation
Nice, France - May 1992

Real Time Inverse Kinematics for General 6R Manipulators

Dinesh Manocha and **John F. Canny**

Computer Science Division
University of California
Berkeley, California 94720

Abstract: 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, which give a level of performance expected of industrial manipulators. In this paper, we present an algorithm and implementation for real time inverse kinematics for a general $6R$ manipulator. The algorithm involves symbolic preprocessing, matrix computations and a variety of numerical techniques. The numerical accuracy of these operations is well understood and for most cases we are able to compute accurate solutions using 64 bit IEEE floating point arithmetic available on most workstations. The average running time of the algorithm, for most cases, is 11 *milliseconds* on an IBM RS/6000 workstation.

1 Introduction

The inverse kinematics problem for six-link manipulators is a central problem in automatic robot control. Given the pose of the end effector (the position and orientation), the problem is to compute the joint parameters for that pose. The complexity of this problem is a function of the geometry of the manipulator. 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, which has six revolute joints, the links are of arbitrary length and no constraints are imposed on the geometry of various links. It is not clear whether the solutions for such a manipulator can be expressed in closed form. Iterative solutions (based on numerical techniques) to the inverse kinematics for general $6R$ manipulators have been known for quite some time. However, they are slow for practical applications and unable to find all the solutions. As a result, most industrial manipulators are designed sufficiently simply so that a closed form exists.

In the absence of a closed form solution, it is widely beleived that the problem of inverse kinematics for a general manipulator is considered solved when

- A tight upper bound on the number of solutions has been established.
- 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 inverse kinematics problems has been studied for at least three decades. The earliest systematic attempt on the inverse kinematics problem appears to have been by Pieper [10]. The work on the general version of the problem for $6R$ manipulator includes that of Roth, Rastegar and Scheinman [13], Albala and Angeles [1] and Duffy and Crane [4]. Tsai and Morgan used a ***higher dimensional approach*** to the inverse kinematics problem [15]. In particular, they cast the problem as eight second-degree equations and solved them numerically using polynomial continuation. Complementing this approach is the *lower dimensional approach*, where a single polynomial in the tangent of the half-angle of one of the joint variables is derived. After trying many configurations Tsai and Morgan conjectured that 16 is an upper bound on the number of solutions. The first conclusive proof of the fact that the problem can have at most 16 solutions was given by Primrose, [11], using Duffy and Crane's formulation. Finally, Lee and Liang, [7], gave the exact solution in lower dimension by reducing the problem to a 16 degree polynomial. Moreover, Raghavan and Roth, [12], used dialytic elimination to derive a 16 degree polynomial as well. Complementing these results [9] presented an example consisting of a manipulator and a pose of the end effector such that the inverse kinematics problem has 16 real solutions and thereby, establishing the fact that 16 is a tight bound on the number of solutions.

As far as implementations of these algorithms are concerned, only continuation methods have been able to solve the problem for a variety of cases [15, 16]. According to [16], lower dimensional methods like the one in [12] are impractical due to numerical problems. At the same time algorithms based on continuation methods are rather slow. The best known algorithm takes about 10 seconds of CPU time on an IBM 370 –

0-8186-2720-4/92 $3.00 ©1992 IEEE

3090 using double precision arithmetic [16], which falls short of what is expected of industrial manipulators. As a result no good practical solutions are available for the inverse kinematics of a $6R$ manipulator.

In this paper we present an algorithm and implementation for real time inverse kinematics for a general $6R$ manipulator. We make use of the results presented in [12]. However, we perform matrix operations and reduce the problem to computing eigenvalues and eigenvectors of a matrix as opposed to computing a univariate polynomial in the tangent of a half-angle of a joint variable. 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* and fast implementations are available [6, 2]. This is in contrast with expanding a symbolic determinant to compute a degree 16 polynomial and then computing its roots. The latter method is relatively slower and the problem of computing roots of such polynomials can be ill-conditioned [17]. The numerical stability of the operations used in our algorithm is well understood and we are able to come up with tight bounds on the accuracy of the solution. Furthermore, for almost all instances of the problem we are able to compute accurate solutions using 64 bit IEEE floating point arithmetic. Moreover, the average running time of the algorithm is 11 *milliseconds* on an IBM RS/6000. In a few cases we need to use sophisticated techniques like solving generalized eigenvalue system and the resulting algorithm may take up to 25 milliseconds on the IBM RS/6000.

The rest of the paper is organized in the following manner. In Section 2, we review the inverse kinematics problem and reduce the problem to solving a system of multivariate polynomials. We also give a brief preview of the lower dimensional approach presented in [12]. The algorithm has been presented in Section 3 and we discuss its accuracy, implementation and performance in Section 4.

2 Inverse Kinematics

We use Denavit–Hartenberg formalism, [3], to model the $6R$ manipulator. Each link is represented by the line along its joint axis and the common normal to the next joint axis. 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 [14, 15]. The 4×4 transformation matrix relating $i+1$th coordinate system to ith coordinate system is [14]:

$$A_i = \begin{pmatrix} c_i & -s_i\lambda_i & s_i\mu_i & a_ic_i \\ s_i & c_i\lambda_i & -c_i\mu_i & a_is_i \\ 0 & \mu_i & \lambda_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad (1)$$

where $s_i = \sin\theta_i$, $c_i = \cos\theta_i$, $\mu_i = \sin\alpha_i$, $\lambda_i = \cos\alpha_i$, and θ_i is the *ith* joint angle and α_i is the *ith* twist angle between the axes *ith* and $i+1$. Moreover, a_i is the length of link $i+1$ and d_i is the 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. For the inverse kinematics problem we are also given the pose of the end-effector, attached to link 7. This pose is described with respect to the base link or link 1. We represent this pose as:

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

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

$$A_1A_2A_3A_4A_5A_6 = A_{hand}. \qquad (2)$$

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 corresponds to solving 6 equations in 6 unknowns.

2.1 Raghavan and Roth Solution

In this section, we briefly describe the lower dimensional approach of Raghavan and Roth [12] 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

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

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. The entries of columns 3 and 4 of the right hand side matrix in (3) are independent of θ_6. As a result, comparing the entries of the 3rd and 4th column results in 6 equations in 5 variables:

$$\begin{array}{ll} EQ1: & c_3f_1 + s_3f_2 = c_2h_1 + s_2h_2 - a_2 \\ EQ2: & s_3f_1 - c_3f_2 = -\lambda_2(s_2h_1 - c_2h_2) + \mu_2(h_3 - d_2) \\ EQ3: & f_3 = \mu_2(s_2h_1 - c_2h_2) + \lambda_2(h_3 - d_2) \\ EQ4: & c_3r_1 + s_3r_2 = c_2n_1 + s_2n_2 \\ EQ5: & s_3r_1 - c_3r_3 = -\lambda_2(s_2n_1 - c_2n_2) + \mu_2n_3 \\ EQ6: & r_3 = \mu_3(s_2n_1 - c_2n_2) + \lambda_2n_3, \end{array} \qquad (4)$$

where f_i's, h_i's, r_i's and n_i's are functions of the input variables [12]. The equations, EQ1-EQ6 are rearranged, in terms of variables, to obtain 6 new equations, $p_1, p_2, p_3, l_1, l_2, l_3$ as functions of h_i's, f_i's,n_i's and r_i's. After the arrangement the left hand side of p_i and l_i is a linear combination of $1, c_2, s_2, c_1, s_1, c_1c_2, c_1s_2, s_1c_2, s_1s_2$ and the right hand side is a linear combination of $1, c_5, s_5, c_4, s_4, c_4c_5, c_4s_5, s_4c_5, s_4s_5$. However, the coefficients used to express the right hand side as a linear combination are functions of s_3 and c_3.

Consider $\mathbf{p} = (p_1\ p_2\ p_3)^T$ and $\mathbf{l} = (l_1\ l_2\ l_3)^T$ as 3×1 vectors. According to [12], the left and right hand sides of the following equations have the same power products as the left and right hand sides of p_i and l_i:

$$\mathbf{p} \cdot \mathbf{p},\ \mathbf{p} \cdot \mathbf{l},\ \mathbf{p} \times \mathbf{l},\ (\mathbf{p} \cdot \mathbf{p})\mathbf{l} - 2(\mathbf{p} \cdot \mathbf{l})\mathbf{p}. \qquad (5)$$

In all we get 14 equations and they can be expressed as:

$$(Q)\,(s_1s_2\ s_1c_2\ c_1s_2\ c_1c_2\ s_1\ c_1\ s_2\ c_2)^T$$
$$= (P)\,(s_4s_5\ s_4c_5\ c_4s_5\ c_4c_5\ s_4\ c_4\ s_5\ c_5\ 1)^T, \qquad (6)$$

where Q is a 14×8 matrix, whose entries are all constants. Furthermore, these entries are obtained from the left hand sides of p_i's, l_i's and the equations (5). P is a 14×9 matrix, whose entries are linear functions of s_3 and c_3 and they are obtained from the right hand sides of p_i's, l_i's and the equations, (5). The relationship expressed in (6) helps us in eliminating four of the five variables.

Raghavan and Roth use 8 of the 14 equations in (6) 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. As a result, they obtain the relation:

$$(\Sigma)\,(s_4s_5\ s_4c_5\ c_4s_5\ c_4c_5\ s_4\ c_4\ s_5\ c_5\ 1)^T = \mathbf{0}, \qquad (7)$$

where Σ is 6×9 matrix, whose entries are linear combinations of s_3, c_3 and 1. It is possible that Q's rank is less than 8. More details to handle such cases are given in Section 3. Given (7), substitute $s_i = \frac{2x_i}{1+x_i^2}$, $c_i = \frac{1-x_i^2}{1+x_i^2}$, for $i = 3,4,5$, where $x_i = \tan(\frac{\theta_i}{2})$. After the substitution clear out the denominators and the system of equations, (7) can, therefore, be expressed as:

$$(\Sigma')\left(x_4^2x_5^2\ x_4^2x_5\ x_4^2\ x_4x_5^2\ x_4x_5\ x_4\ x_5^2\ x_5\ 1\right)^T = \mathbf{0}, \qquad (8)$$

where (Σ') is 6×9 matrix, whose entries are quadratic polynomial in x_3. The system given above is not a square system and it is converted into a square system using dialytic elimination. In particular, the equations expressed in (8) are multiplied by x_4 to obtain a square system of the form

$$\Sigma''\left(x_4^3x_5^2\ x_4^3x_5\ x_4^3\ x_4^2x_5^2\ x_4^2x_5\ x_4^2\ x_4x_5^2\ x_4x_5\ x_4\ x_5^2\ x_5\ 1\right)^T, \qquad (9)$$

where Σ'' is a 12×12 matrix of the form

$$\Sigma'' = \begin{pmatrix} \Sigma' & \mathbf{0} \\ \mathbf{0} & \Sigma' \end{pmatrix}.$$

$\mathbf{0}$ is a 6×3 null matrix and the other entries of Σ'' quadratic polynomials in x_3. Therefore, its determinant is a polynomial of degree 24 in x_3. Let us represent its determinant as $R(x_3)$.

Lemma I: $(1 + x_3^2)^4$ divides $R(x_3)$.
Proof: [12].

As a result, the degree 16 polynomial,

$$Q(x_3) = \frac{R(x_3)}{(1 + x_3^2)^4}, \qquad (10)$$

is the input–output polynomial, whose roots are used to compute the joint angle θ_3. Raghavan and Roth suggest expanding the determinant and use a root solver for computing the values of θ_3. Given θ_3, they usee the 11 linear independent equation in (9) to solve for θ_4 and θ_5. Finally they use the equations, (6) and (3) to solve for θ_1, θ_2 and θ_6.

2.2 Numerical Problems in the Raghavan and Roth Solution

Although Raghavan and Roth present a constructive solution to the inverse kinematics problem, their algorithm suffers from efficiency and numerical accuracy problems. For example, the algebraic manipulations in generating $p_1, p_2, p_3, l_1, l_2, l_3$ can introduce errors due to floating point arithmetic. Furthermore, the determinant expansion of Σ'' can introduce significant numerical errors such that $(1 + x_3^2)^4$ may not exactly divide the determinant. The symbolic expansion of the determinant is relatively expensive for real time performance. Finally, the computation of real roots of polynomials of degree 16 can be ill conditioned [17]. The floating point errors accumulated in the intermediate steps of the computation, and therefore in the coefficients of the degree 16 polynomial, can have a significant impact on the accuracy of the roots of the resulting polynomial.

3 Algorithm

In this section we describe our algorithm in detail. The initial steps in our algorithm make use of the results presented in [12]. 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. Treat the a_i's, d_i's, λ_i's, μ_i's and the entries of the right hand side matrix A_{hand} as symbolic constants. As a result, express the entries of the 14×9 matrix P and 14×8 matrix Q, as shown in equation (6), as functions of these symbolic constants. This computation is performed using the properties highlighted in [12] and only once for general $6R$ manipulators.

2. Given an instance of the problem, substitute the numerical values of the symbolic constants highlighted above, to compute the numerical entries of the matrices P and Q, as shown in (6), for this instance.

3. Compute the rank of Q using singular value decomposition. If 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 [14].

4. Eliminate the variables θ_1 and θ_2 from (6). 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. After eliminating θ_1 and θ_2, we obtain a matrix Σ, as shown in (7). The actual number of rows in Σ is equal to $R = (14-\text{rank(Q)}) \geq 6$. Take any of the 6 rows of Σ (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. As a result, we obtain a matrix of the form Σ', as shown in (8). After using dialytic elimination we compute the 12×12 matrix, Σ'', whose entries are quadratic polynomial in x_3.

6. Reduce the problem of computing roots of the equation, determinant$(\Sigma'') = 0$, 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 (6) 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. Compute the condition number of the eigenvalue computation. If the condition number is high, improve the accuracy of the resulting solution by Newton's method.

These steps arr explained in detail in the following sections.

3.1 Symbolic Preprocessing

Many properties of the ideal generated by the equations, EQ1–EQ6, may not hold in practice due to floating point arithmetic. As a result we treat the known parameters of a $6R$ manipulator, the a_i's, d_i's, α_i's and the entries of A_{hand} (like l_x, l_y, q_x, q_y) as symbolic constants. These symbolic constants along with the variables θ_i are used in the symbolic derivation of the equations highlighted in (5). We use the computer algebra system, MAPLE, for the derivation and simplification of the equations. The left and right hand side of the 8 equations, shown in (5), are computed separately. After computing the 14 equations, EQ1–EQ6 and 8 equations shown in (5), collect the terms as functions of sines and cosines of the joint angles θ_1 and θ_2 for the left hand side of the equations and of the joint angles θ_3, θ_4 and θ_5 for the right hand side of the equations. The coefficients of the equation are used to compute the entries of the matrices P and Q. As a result, we are able to express the entries of P and Q as polynomial functions of the symbolic constants a_i's, d_i's, λ_i's, μ_i's, p, q, r, u, v, w. In case of 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 Q has a special structure. In particular many of its entries are zero and as a result the system of equations, (6), can be expressed as two different system of equations of the form:

$$(Q_1)\mathbf{w}_1 = (P_1)(s_4 s_5 \; s_4 c_5 \; c_4 s_5 \; c_4 c_5 \; s_4 \; c_4 \; s_5 \; c_5)^T, \quad (11)$$
$$(Q_2)\mathbf{w}_2 = (P_2)(s_4 s_5 \; s_4 c_5 \; c_4 s_5 \; c_4 c_5 \; s_4 \; c_4 \; s_5 \; c_5 \; 1)^T, (12)$$

where

$$\mathbf{w}_1 = (s_1 \; c_1), \quad \mathbf{w}_2 = (s_1 s_2 \; s_1 c_2 \; c_1 s_2 \; c_1 c_2 \; s_2 \; c_2).$$

and Q_1, Q_2, P_1, P_2 are $6 \times 2, 8 \times 6, 6 \times 9, 8 \times 9$ matrices, respectively. The details of this formulation are given in [12]. In particular, we break the set of the 14 equations into sets of 6 and 8 equations. Q_1, Q_2 are minors of Q and P_1, P_2 are minors of P.

The symbolic complexity of the entries of P_1, P_2, Q_1, Q_2 corresponding to the equations $\mathbf{p} \times \mathbf{p}$, $(\mathbf{p}\cdot\mathbf{p})\mathbf{l} - 2(\mathbf{p}\cdot\mathbf{l})\mathbf{p}$ is high. Simplifying these entries symbolically by collecting terms with common subexpressions increases the efficiency and numerical accuracy of subsequent computations.

3.2 Numerical Substitution and Rank Computation

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 P_1, P_2, Q_1, Q_2. Although we have never encountered numerical problems in substitution computations, the accuracy of these operations can be improved by using higher precision arithmetic. These substitutions are performed only once for a given manipulator. Given the pose of the end–effector, we substitute it to compute the entries of P_1, P_2, Q_1, Q_2. Let the corresponding numerical matrices (obtained after substitution) be $\overline{P_1}, \overline{P_2}, \overline{Q_1}, \overline{Q_2}$.

We use singular value decomposition to compute the ranks of $\overline{Q_1}$ and $\overline{Q_2}$ [6]. The singular vectors obtained are also used to eliminate θ_1 and θ_2 from (11) and (12). In particular, let the singular value decomposition of $\overline{Q_1}$ be expressed as:

$$\overline{Q_1} = U' \Sigma' V'^T,$$

where U', Σ' and V'^T are 6×2, 2×2 and 2×2 matrices, respectively. Initially compute the singular values, σ_1, σ_2 of $\overline{Q_1}$. If both the singular values are non-zero, $\overline{Q_1}$ has full rank and let $\overline{Q_1}' = \overline{Q_1}$. If either of the singular values is close to 0.0, we conclude that $\overline{Q_1}$ 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, ϵ is a function of the accuracy of the input data. Compute the new matrix $\overline{Q}_1$, whose (i,j) element is given as:

$$\overline{Q}'_{1_{ij}} = \Sigma_{k=1}^{2} \sigma_k' U_{ik} V_{jk}.$$

$\overline{Q}_1'$ 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. Use $\overline{Q}_1'$ for eliminating θ_1, θ_2 in the system of equations (11) to obtain

$$(Q_1')\,(s_1\ c_1)^T = (P_1)\,(s_4s_5\ s_4c_5\ c_4s_5\ c_4c_5\ s_4\ c_4\ s_5\ c_5)^T. \quad (13)$$

Perform Gauss elimination with complete pivoting on $\overline{Q}_1'$ and corresponding row and column operations are carried on to the elements of P. Depending on the rank of $\overline{Q}_1'$, whether $0, 1$ or 2, we obtain 6, 5 or 4 equations, respectively, in sines and cosines of θ_4, θ_5. Each equation corresponds to a row of Σ in (7).

In a similar fashion we compute the rank of $\overline{Q}_2$, as represented in (12). Let the modified matrix computed after singular value decomposition be $\overline{Q}_2'$. It is used in eliminating θ_1, θ_2 from (12). Depending on the rank of $\overline{Q}_2'$, we may obtain anywhere from 2 to 8 equations after elimination. Each equation corresponds to a row of Σ in (7).

The matrix Σ is a $p \times 9$ matrix, where $6 \le p \le 14$. Furthermore each entry is a function of $\sin(\theta_3)$ and $\cos(\theta_3)$. We choose any 6 of the p rows and break up the resulting matrix into Σ_1 and Σ_2 consisting of 6 and $p-6$ rows, respectively. The algorithm finds the solutions of the equations corresponding to Σ_1 and back substitutes the solution into equations corresponding to Σ_2. As a result, we solve for the system of equations represented by Σ.

Given the 6×9 matrix Σ_1, we substitute the sines and cosines of $\theta_3, \theta_4, \theta_5$ in terms of x_3, x_4 and x_5, perform dialytic elimination and obtain a 12×12 matrix, Σ'', whose entries are quadratic polynomials in x_3.

3.3 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 the eigenvalue algorithm.

Given the 12×12 matrix, Σ'', each of its entries is a quadratic polynomial in x_3. The problem has been reduced to solving the system of linear equations

$$\begin{aligned}\Sigma''\mathbf{v} = \Sigma''\,&(x_4^3x_5^2\ x_4^3x_5\ x_4^3\ x_4^2x_5^2\ x_4^2x_5\ x_4^2\ x_4x_5^2\ x_4x_5\ x_4\ x_5^2\ x_5\ 1)^T \quad (14)\\ =\ &(0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0)^T.\end{aligned}$$

We express the matrix as

$$\Sigma'' = Ax_3^2 + Bx_3 + C, \quad (15)$$

where A, B and C are 12×12 matrices consisting of numerical entries and compute the condition number of A. The actual computation of a condition takes $O(n^3)$ time. However, good estimators of complexity $O(n^2)$ are available and are available in LINPACK and LAPACK [2]. If the matrix is singular, its condition number is infinity. Lets consider the case when A is well conditioned. Multiply the matrix equation, (15), A^{-1} to obtain:

$$\overline{\Sigma}'' = Ix_3^2 + A^{-1}Bx_3 + A^{-1}C,$$

where I is a 12×12 identity matrix. Given $\overline{\Sigma}''$, we use Theorem 1.1 [5] to construct a 24×24 matrix M of the form

$$M = \begin{pmatrix} \mathbf{0} & I \\ -A^{-1}C & -A^{-1}B \end{pmatrix},$$

where $\mathbf{0}, I$ are 12×12 null and identity matrices, respectively. It follows from the structure of M that the eigenvalues of M correspond exactly to the roots of determinant$(\Sigma'') = 0$. Furthermore, the eigenvectors of M, corresponding to the eigenvalue x_3 have the structure

$$\mathbf{V} = \begin{pmatrix} \mathbf{v} \\ x_3\mathbf{v} \end{pmatrix}, \quad (16)$$

where $\mathbf{v}$ is the 12×1 vector corresponding to the variables in (14). Thus, the eigenvectors of M can be used to compute x_4 and x_5.

We now consider the case, when A in (15) is 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, $x_3 = tan(\frac{\theta_3}{2}) \approx \infty$ and A is almost singular. The matrix equation, (15), is reduced to a *generalized eigenvalue problem* by constructing two matrices, M_1 and M_2

$$M_1 = \begin{pmatrix} I & \mathbf{0} \\ \mathbf{0} & A \end{pmatrix}, M_2 = \begin{pmatrix} \mathbf{0} & I \\ -C & -B \end{pmatrix},$$

where $\mathbf{0}, I$ are 12×12 null and identity matrices, respectively [5, 6]. Furthermore, the roots of determinant$(\Sigma'') = 0$, correspond to the eigenvalues of the generalized eigenvalue problem $M_1 - x_3M_2$ [5]. The eigenvectors have the same structure as (16).

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 $x_3 = \frac{a\overline{x}_3+b}{c\overline{x}_3+d}$, where a, b, c, d are random numbers. As a result of this transformation, the resulting matrix polynomial is:

$$\Sigma_1'' = \overline{A}\overline{x}_3^2 + \overline{B}\overline{x}_3 + \overline{C}, \quad (17)$$

where $\overline{A} = a^2\ A + ac\ B + c^2\ C$, $\overline{B} = 2abA + (ad + bc)B + 2cdC$ and $\overline{C} = b^2A + bdB + d^2C$. For most cases $\overline{A}$ is well conditioned. The only exceptions arise when A, B, C may have common singular pencils. In the latter case, $\overline{A}$ is ill conditioned for all choices of a, b, c, d.

We suggest trying this transformation for a few choices of a, b, c, d and compute the condition number of $\overline{A}$. The cost of estimating condition number is rather small as compared to computing the eigendecomposition. If $\overline{A}$ is well conditioned, we solve for determinant$(\Sigma_1'') = 0$ 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 (16), except that x_3 is replaced by $\overline{x}_3$.

4 Implementation

We have implemented the algorithm on an IBM RS/6000. We have used many routines from EISPACK and LAPACK for matrix operations. 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.

4.1 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, according to Lemma I, 8 of the eigenvalues correspond to the roots of the polynomial $(1 + x_3^2)^4 = 0$. In other words, $\imath$ and $-\imath$ are eigenvalues of M of multiplicity 4 each, where $\imath = \sqrt{-1}$. If we transform the variable x_3, these eigenvalues are suitably modified.

The QR algorithm for eigenvalue computation transforms M into a Hessenberg matrix by a series of orthogonal symmetric transformations [6]. The rest of the algorithm consists of performing orthogonal symmetric transformations such that the matrix reduces to its real Schur form. These transformations correspond to choosing shifts and computing the QR decomposition of the resulting matrix. More details are given in [6]. Since we know 8 of the eigenvalues apriori, we perform the shifts corresponding to these eigenvalues. As the complex eigenvalues occur in conjugates, the shifts can be performed using real arithmetic (termed as a double shift). As a result, after 4 double shifts, we obtain a 16×16 Hessenberg matrix. The eigenvalues of the latter matrix correspond exactly to the roots of $Q(x_3)$ in (10).

Given the real Schur form we are only interested in computing the eigenvectors corresponding to real eigenvalues. These eigenvalues can be easily identified by 1×1 diagonal matrices R_{ii}. To account for numerical errors, we test whether the imaginary part of the eigenvalue is less than ϵ. For such eigenvalues, we set the imaginary part equal to zero and it becomes a real eigenvalue of multiplicity two. In other words, the 2×2 matrix, R_{jj} corresponding to the complex eigenvalues is converted into a diagonal matrix.

4.1.1 Clustering Eigenvalues

In many instances the solution has a root of multiplicity greater than one. As such the problem of computing multiple roots 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 $\alpha_1, \ldots, \alpha_k$. Moreover, $\mid \alpha - \alpha_j \mid$ may be relatively high. Let $\alpha_m = \frac{\alpha_1 + \alpha_2 + \ldots + \alpha_k}{k}$. It turns out $\mid \alpha - \alpha_m \mid$ is relatively small and α_m is very close to the multiple roots. We can actually verify the accuracy of these computations by computing the condition number of the eigenvalue and the condition number of a cluster of eigenvalues. The eigendecomposition routines in LAPACK have implementation of these condition numbers.

4.1.2 Eigenvector computation

The eigenvector corresponding to a real eigenvalue is computed by solving a quasi–upper triangular system [6]. Given an eigenvector $\mathbf{V}$, we use its structure, (16), 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 [18]. As a result, terms of maximum magnitude have the minimum amount of relative error. We use this property in accurate computation of x_4 and x_5. Given the eigenvector $\mathbf{V}$, let

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

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

$$\mathbf{v}_1 = (v_1\ v_2\ v_3\ v_4\ v_5\ v_6\ v_7\ v_8\ v_9\ v_{10}\ v_{11}\ v_{12})^T$$

$$= \left(x_4^3x_5^2\ x_4^3x_5\ x_4^3\ x_4^2x_5^2\ x_4^2x_5\ x_4^2\ x_4x_5^2\ x_4x_5\ x_4\ x_5^2\ x_5\ 1\right)^T.$$

Therefore, x_4 and x_5 corresponds to ratio of two terms of $\mathbf{v}_1$. 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.

4.1.3 Computing all Joint Angles

Given a triple (x_3, x_4, x_5) corresponding to a solution of the 6 equations represented as the 6×9 matrix Σ_1. We substitute these solutions into the equation corresponding to the matrix Σ_2. The triple is classified as a solution of the original system if it satisfies all the equations obtained after eliminating θ_1 and θ_2. These equations are represented by the matrix Σ.

Given a solution of Σ, solve for s_1, c_1, s_2, c_2 from Q_1' and Q_2', as shown in (13). These solutions are substituted into (3) to compute θ_6.

4.2 Improving the Accuracy

The solution obtained above are back substituted into the equations EQ1–EQ6, (4). The residues obtained are used to check the accuracy of the given solutions. To improve the accuracy we use Newton's method. If the given solution has multiplicity one, the residual quickly converges to zero.

We apply the Newton's method on the equations. We represent each equation in terms of x_i, where $x_i = tan(\frac{\theta_i}{2})$. As a result each equation is quadratic polynomial in x_i. The solution computed from the algorithm highlighted above is used as the initial guess. At each step we evaluate the functions and compute the Jacobian. This process is repeated till the residual is almost zero.

4.3 Performance

We have applied our algorithm to many examples. In particular, we used it on 21 problem instances given in [16] 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. The actual time varies between 9.5 milliseconds to 14 milliseconds. 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 (15) 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.

5 Conclusions

In this paper we have presented a real time algorithm for the inverse kinematics of general $6R$ robot manipulators. In the process we used symbolic techniques, matrix computations and numerical methods. We believe that this algorithm gives us a level of performance expected of industrial manipulators. More details of the algorithm are given in [8].

6 Acknowledgements

We are grateful to J. Demmel for productive discussions on matrix computations and numerical analysis. This research was supported in part by David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (# IRI-8958577). The first author is also supported by IBM Graduate Fellowship.

References

[1] H. Albala and J. Angeles. Numerical solution to the input output displacement equation of the general 7r spatial mechanism. In *Proceedings of the Fifth World Congress on Theory of Machines and Mechanisms*, pages 1008-1011, 1979.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. Lapack: A portable linear algebra library. Computer Science Technical Report CS-90-105, University of Tennessee, 1990.

[3] J. Denavit and R.S. Hartenberg. A kinematic notation for lower-pair mechanisms based upon matrices. *Journal of Applied Mechanics*, 77:215-221, 1955.

[4] J. Duffy and C. Crane. A displacement analysis of the general spatial 7r mechanism. *Mechanisms and Machine Theory*, 15:153-169, 1980.

[5] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials.* Academic Press, New York, 1982.

[6] G.H. Golub and C.F. Van Loan. *Matrix Computations.* John Hopkins Press, Baltimore, 1989.

[7] H.Y. Lee and C.G. Liang. A new vector theory for the analysis of spatial mechanisms. *Mechanisms and Machine Theory*, 23(3):209-217, 1988.

[8] D. Manocha and J.F. Canny. Real time inverse kinematics for general 6r manipulators. Technical Report ESRC 92-2, RAMP 92-1, Engineering System Research Center, University of California, Berkeley, 1992.

[9] R. Manseur and K.L. Doty. A robot manipulator with 16 real inverse kinematic solution set. *International Journal of Robotics Research*, 8(5):75-79, 1989.

[10] D. Pieper. *The kinematics of manipulators under computer control.* PhD thesis, Stanford University, 1968.

[11] E.J.F. Primrose. On the input-output equation of the general 7r-mechanism. *Mechanisms and Machine Theory*, 21:509-510, 1986.

[12] M. Raghavan and B. Roth. Kinematic analysis of the 6r manipulator of general geometry. In *International Symposium on Robotics Research*, pages 314-320, Tokyo, 1989.

[13] B. Roth, J. Rastegar, and V. Scheinman. On the design of computer controlled manipulators. In *On the Theory and Practice of Robots and Manipulators*, pages 93-113. First CISM IFToMM Symposium, 1973.

[14] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley and Sons, 1989.

[15] L.W. Tsai and A.P. Morgan. Solving the kinematics of the most general six and five-degree-of-freedom manipulators by continuation methods. *Transactions of the ASME, Journal of Mechanisms, Transmissions and Automation in Design*, 107:189-200, 1985.

[16] C. Wampler and A.P. Morgan. Solving the 6r inverse position problem using a generic-case solution methodology. *Mechanisms and Machine Theory*, 26(1):91-106, 1991.

[17] J.H. Wilkinson. The evaluation of the zeros of ill-conditioned polynomials. parts i and ii. *Numer. Math.*, 1:150-166 and 167-180, 1959.

[18] J.H. Wilkinson. *The algebraic eigenvalue problem.* Oxford University Press, Oxford, 1965.